Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)
Кафедра *Системотехніки*
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Автоматизований аналіз та класифікація біологічних ознак рослин з застосуванням нейронних мереж
(тема)

Виконав:
студент _2__ курсу, групи _*ІТПм-24-2*
Дідусь О.П.
(прізвище, ініціали)

Спеціальність *122 Комп'ютерні науки*
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма _*Інформаційні технології проєктування*
(повна назва освітньої програми)

Керівник *доц. каф. СТ Ситнікова П.Е.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ *Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

Кафедра *Системотехніки*

Рівень вищої

освіти другий (магістерський)

Спеціальність *122 Комп'ютерні науки*
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма *Інформаційні технологі проектування*
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові *Дідусю Олександру Павловичу*
(прізвище, ім'я, по батькові)

1. Тема роботи: *АВТОМАТИЗОВАНИЙ АНАЛІЗ ТА КЛАСИФІКАЦІЯ БІОЛОГІЧНИХ ОЗНАК РОСЛИН З ЗАСТОСУВАННЯМ НЕЙРОННИХ МЕРЕЖ*

затверджена наказом по університету від *24 листопада* 2025 р. № *1058Ст*

2. Термін подання студентом роботи до екзаменаційної комісії: 15.12.2025 р

3. *Вихідні дані до роботи: Дослідити методи виявлення та класифікації об'єктів на зображеннях рослин, включаючи види рослин та їх хвороби, з метою проведення порівняльного аналізу характеристик алгоритмів глибокого навчання та серії експериментів для визначення найефективніших підходів у задачах класифікації. Література з комп'ютерного зору, розпізнавання образів, машинного та глибокого навчання. Статті про архітектури нейронних мереж для класифікації зображень, існуючі датасети рослин та їх патологій, методи препроцесингу даних, аугментації та оцінки моделей. Існуючі рішення для задач класифікації в агрономії та ботаніці, включаючи вплив факторів середовища на точність розпізнавання. Нейронні мережі типу CNN, такі як ResNet, VGG, EfficientNet та MobileNet. Перелік програмних засобів, що були використані: ОС Microsoft Windows v.11, середовища розробки Python з бібліотеками TensorFlow/Keras або PyTorch, інтегроване середовище Kaggle для експериментів.*

4. Перелік питань, що потрібно опрацювати в роботі: аналіз сучасних методів комп'ютерного зору

для діагностики хвороб рослин, огляд архітектур згорткових нейронних мереж та трансформерів (EfficientNet, ConvNeXt, Swin Transformer), дослідження підходів до ієрархічної та дрібнозернистої класифікації (FGVC), аналіз методів трансферного навчання та боротьби з дисбалансом класів, характеристика та передобробка вхідних даних (датасет Plant Pathology 2021), розробка методу багатоголової ієрархічної тонкої настройки (MHFT), обґрунтування вибору програмних засобів (Python, PyTorch, Albumentations), експериментальне дослідження впливу гіперпараметрів та аугментації на якість навчання, порівняльний аналіз базової та запропонованої моделей та інтерпретованості результатів, програмна реалізація системи діагностики та розробка користувацького інтерфейсу.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій: *статистичні дані, оброблені аугментацією зображення, результати прогнозу хвороби, код програми*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / термін виконання етапів роботи	Примітка
1.	*Отримання завдання на виконання кваліфікаційної роботи*	*01.09.2025*	*Виконано*
2.	*Аналіз завдання та предметної області*	*02.09.2025-15.09.2025*	*Виконано*
3.	*Опрацювання літератури та аналіз об'єкта дослідження*	*16.09.2025-22.09.2025*	*Виконано*
4.	*Розробка методу інжинірингу ієрархічних міток та підготовка навчальної вибірки*	*23.09.2025-03.10.2025*	*Виконано*
5.	*Розробка та налаштування архітектури базової нейромережевої моделі (Baseline)*	*04.10.2025-20.10.2025*	*Виконано*
6.	*Проектування та програмна реалізація архітектури багатоголової ієрархічної моделі (MHFT)*	*21.10.2025-30.10.2025*	*Виконано*
7.	*Навчання розроблених моделей з використанням різних стратегій аугментації та оптимізації*	*31.10.2025-15.11.2025*	*Виконано*
8.	*Проведення порівняльного експериментального дослідження та аналіз ефективності запропонованого підходу*	*16.11.2025-25.11.2025*	*Виконано*
9.	*Аналіз результатів, формулювання висновків та*	*26.11.2025-01.12.2025*	*Виконано*
10.	*Оформлення пояснювальної записки та презентаційних*	*02.12.2025-05.12.2025*	*Виконано*
11,	*Представлення роботи на рецензування*	*09.12.2025*	*Виконано*

Дата видачі завдання 01 вересня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ *доц. каф. СТ Ситнікова П.Е.*
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Кваліфікаційна робота: 110 с, 5 табл., 18 рис., 2 додатки, 42 джерела інформації

Глибоке навчання, класифікація біологічних ознак рослин, нейронна мережа, прозорість моделі, розпізнавання зображень, CNN, fine tuning, pytorch, transfer learning.

Об’єкт дослідження — методи та інструменти комп’ютерного бачення для автоматизованої класифікації біологічних ознак рослин за зображеннями листя та інтеграція моделі в прикладний програмний інтерфейс (локальний/веб-демо).

Предмет дослідження — архітектурні особливості глибоких нейронних мереж для мультилейблової класифікації біологічних ознак рослин, включно з багатоголовими (multi-head) схемами та процедурами підвищення надійності (OOD-виявлення, hard negative mining, супер-роздільність), а також можливість їх інтеграції з таксономічно-регуляризованим тонким налаштуванням (fine-tuning)

Мета роботи — розробити та експериментально дослідити ефективність багатоголової архітектури (Multi-Head Fine-Tuning, MHFT) для задачі мультилейблової класифікації хвороб рослин (на прикладі FGVC8 Plant Pathology 2021), порівнявши її з простим базовим перенавчанням на одному головному класифікаторі (vanilla fine-tuning), а також оцінити внесок супутніх модулів (OOD-виявлення, hard negative mining, супер-роздільність деталей листя).

Методи дослідження — системний аналіз, теорія розпізнавання образів і мультилейблової класифікації; глибоке навчання з перенесенням (transfer learning); PyTorch-екосистема (timm, torchvision), Albumentations; статистична оцінка якості (F1-мікро/макро, per-class F1, precision/recall); методи підвищення стійкості (threshold-tuning, OOD-виявлення) та якості даних (супер-роздільність, цільові аугментації). Програмні засоби: Python, PyTorch, timm, Albumentations, NumPy, Pandas, scikit-learn, OpenCV (за потреби — TensorBoard; у демоверсії можливе легке веб-/локальне GUI-інтегрування).

Наукова новизна. Дослідження зосереджується на архітектурних засадах мультилейблового розпізнавання стану рослин із використанням попередньо навчених моделей глибокого навчання та структурованих вихідних представлень. Акцент робиться на методичному підході до організації моделі та протоколу оцінювання, що забезпечує відтворюваність і коректність інтерпретації результатів у галузі комп'ютерного бачення для рослинництва.

Практична цінність. Представлено узагальнений робочий процес підготовки даних, навчання моделі та обчислення базових метрик якості. Такий процес може слугувати опорним шаблоном для подальших експериментів, навчальних занять і прикладних рішень, де необхідна прозора організація етапів та мінімальні вимоги до інструментарію.

Обмеження. Очікувана якість та стабільність залежить від властивостей даних, умов збирання зображень і можливих доменних відмінностей. Для застосування в реальних умовах зазвичай потрібні додаткові перевірки, уточнення протоколів та потенційна адаптація до конкретних сценаріїв використання.

Сфера застосування. Методологія може бути використана в освітніх і дослідницьких контекстах, у проєктуванні демонстраційних систем розпізнавання зображень і в підготовці експериментальних прототипів у точках перетину агроінформатики та комп'ютерного бачення.

ABSTRACT

Master's Thesis: 110 p., 5 tab., 18 fig., 2 appendices, 42 title.

Classification of biological plant traits, CNN, deep learning, fine-tuning, image recognition, model transparency, neural network, PyTorch, transfer learning.

Object of research — methods and tools of computer vision for the automated classification of biological plant traits from leaf images and the integration of the model into an application programming interface (local/web-demo).

Subject of research — architectural features of deep neural networks for multi-label classification of biological plant traits, including multi-head architectures and reliability enhancement procedures (OOD-detection, hard negative mining, super-resolution), as well as the possibility of their integration with taxonomically-regularized fine-tuning.

The aim of the work — to develop and experimentally investigate the effectiveness of a multi-head architecture (Multi-Head Fine-Tuning, MHFT) for the task of multi-label plant disease classification (using the FGVC8 Plant Pathology 2021 dataset as an example), comparing it with simple baseline transfer learning on a single main classifier (vanilla fine-tuning), and also to evaluate the contribution of auxiliary modules (OOD-detection, hard negative mining, super-resolution of leaf details).

Research methods — system analysis, theory of pattern recognition and multi-label classification; deep learning with transfer (transfer learning); PyTorch ecosystem (timm, torchvision), Albumentations; statistical quality assessment (F1-micro/macro, per-class F1, precision/recall); robustness enhancement methods (threshold-tuning, OOD-detection) and data quality enhancement (super-resolution, targeted augmentations). Software tools: Python, PyTorch, timm, Albumentations, NumPy, Pandas, scikit-learn, OpenCV (if necessary—TensorBoard; in the demo version, lightweight web/local GUI integration is possible).

Scientific novelty. The research focuses on the architectural principles of multi-label plant condition recognition using pre-trained deep learning models and structured output representations. Emphasis is placed on a methodological approach to model organization and evaluation protocol, which ensures reproducibility and correct

interpretation of results in the field of computer vision for crop production.

Practical value. A generalized workflow for data preparation, model training, and calculation of baseline quality metrics is presented. This process can serve as a reference template for further experiments, educational sessions, and applied solutions where a transparent organization of stages and minimal requirements for tools are necessary.

Limitations. The expected quality and stability depend on data properties, image acquisition conditions, and possible domain shifts. Application in real-world conditions usually requires additional validation, refinement of protocols, and potential adaptation to specific use-case scenarios.

Scope of application. The methodology can be used in educational and research contexts, in the design of demonstration systems for image recognition, and in the preparation of experimental prototypes at the intersection of agro-informatics and computer vision.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ГН – глибоке навчання (Deep Learning).

ГП – графічний процесор (Graphics Processing Unit).

ЕОМ – електронна обчислювальна машина.

ЗНМ – згорткова нейронна мережа (Convolutional Neural Network).

МН – машинне навчання (Machine Learning).

НМ – нейронна мережа.

ПЗ – програмне забезпечення.

ШІ – штучний інтелект.

AdamW – Adam with Weight Decay (алгоритм стохастичної оптимізації з розпадом ваг).

API – Application Programming Interface (інтерфейс прикладного програмування).

Augmentation – аугментація (штучне збільшення обсягу даних).

Backbone – базова мережа (екстрактор ознак).

BCE – Binary Cross-Entropy (бінарна перехресна ентропія, функція втрат).

CE – Cross-Entropy (перехресна ентропія, функція втрат).

CNN – Convolutional Neural Network (згорткова нейронна мережа).

CUDA – Compute Unified Device Architecture (архітектура паралельних обчислень від NVIDIA).

CV – Computer Vision (комп’ютерний зір).

EDA – Exploratory Data Analysis (розвідувальний аналіз даних).

F1-Score – гармонійне середнє між точністю (Precision) та повнотою (Recall).

FGVC – Fine-Grained Visual Classification (дрібнозерниста візуальна класифікація).

FN – False Negative (помилково негативний результат).

FP – False Positive (помилково позитивний результат).

GUI – Graphical User Interface (графічний інтерфейс користувача).

ImageNet – масштабний набір даних зображень для навчання нейронних мереж.

MHFT – Multi-Head Hierarchical Fine-Tuning (багатоголова ієрархічна тонка

настройка – метод, розроблений в роботі).

MTL – Multi-Task Learning (багатозадачне навчання).

ReLU – Rectified Linear Unit (функція активації).

RGB – Red, Green, Blue (колірна модель).

SOTA – State-of-the-Art (сучасний рівень розвитку, найкращі існуючі рішення).

TN – True Negative (істинно негативний результат).

TP – True Positive (істинно позитивний результат).

ViT – Vision Transformer (візуальний трансформер).

ВСТУП

Сучасний прогрес у галузі Deep Learning, зокрема в Computer Vision, нерозривно пов'язаний з розробкою глибоких згорткових нейронних мереж (CNNs) та, останнім часом, архітектур на основі механізму уваги, таких як ViT. Навчання таких моделей "з нуля" (from scratch) є надзвичайно ресурсомістким процесом, що вимагає обчислювальних потужностей промислового масштабу та доступу до масивних, ретельно анотованих наборів даних, як-от ImageNet (більше14 мільйонів зображень) або JFT-300M. Саме в цьому контексті Transfer Learning (трансферне навчання) еволюціонувало з теоретичної концепції в основну інженерну практику. Його ключова ідея полягає у використанні знань, отриманих моделлю під час вирішення однієї задачі (source task), для підвищення ефективності та якості вирішення іншої, цільової задачі (target task).

Найбільш поширеною та успішною реалізацією трансферного навчання є метод fine-tuning (тонка настройка). Процес fine-tuning складається з кількох канонічних кроків:

– береться модель (backbone), попередньо навчена на масштабному датасеті загального призначення (наприклад, ResNet50, EfficientNet, ViT-B/16, навчені на ImageNet). Вважається, що на своїх нижніх та середніх шарах така модель вивчила ієрархію загальних візуальних ознак — від простих (границі, текстури, градієнти) до складних (форми, об'єкти, патерни);

– оригінальний класифікаційний шар (classifier head), специфічний для початкової задачі (наприклад, 1000 класів ImageNet), видаляється;

– на його місце встановлюється новий, ініціалізований випадковим чином класифікатор, архітектура якого відповідає кількості класів у цільовій задачі;

– модель донавчається на цільовому, зазвичай значно меншому, наборі даних. Процес донавчання ведеться з низькою швидкістю навчання (learning

rate), щоб адаптувати існуючі ваги до нової специфіки, не руйнуючи при цьому вже вивчені корисні репрезентації.

Завдяки своїй ефективності, fine-tuning став золотим стандартом для вирішення прикладних задач, де збір великого анотованого датасету є неможливим або економічно недоцільним.

Незважаючи на широке застосування, класичний підхід до fine-tuning має низку проблем, які обмежують його надійність та ефективність у складних реальних сценаріях.

Дані обмеження стають ще більш вираженими у задачах дрібнозернистої візуальної класифікації (Fine-Grained Visual Classification, FGVC), до яких належить ідентифікація хвороб рослин. Цей клас задач характеризується низькою міжкласовою варіативністю (різні класи дуже схожі) та високою внутрішньокласовою варіативністю (об'єкти одного класу можуть сильно відрізнятися).

Таким чином, стандартний підхід до fine-tuning, хоч і є потужною відправною точкою, виявляється недостатньо ефективним та надійним для створення систем діагностики хвороб рослин, які б відповідали вимогам реального світу щодо точності, робастності та інтерпретованості. Це створює нагальну потребу в розробці нових методів, які б інтегрували доменні знання, зокрема ієрархічну структуру класів, безпосередньо в процес навчання моделі.

Апробацію роботи було здійснено під час проведення XVIII міжнародної науково-практичної конференції «Інформаційні технології і автоматизація – 2025». Опубліковані тези доповіді [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Актуальність роботи

Людство стикається з фундаментальним викликом: необхідністю збільшення виробництва продуктів харчування на тлі стагнації доступних посівних площ, кліматичних змін та зростаючих екологічних вимог . У цьому контексті прецизійне землеробство (Precision Agriculture, PA)[2] стає ключовою парадигмою. PA визначається як управлінський підхід, що використовує інформаційні технології для збору, обробки та аналізу просторово-часових даних з метою оптимізації сільськогосподарських процесів "на місцях" . Замість уніфікованої обробки всього поля, PA дозволяє приймати диференційовані рішення (наприклад, щодо поливу, внесення добрив чи пестицидів) для окремих його ділянок або навіть окремих рослин.

Ключовим технологічним рушієм прецизійного землеробства є комп'ютерний зір (Computer Vision). Автоматизований моніторинг стану посівів за допомогою зображень, отриманих з дронів (UAV), супутників або наземних роботизованих платформ, дозволяє вирішувати низку критичних завдань:

– оцінка біомаси та прогнозування врожайності;

– ідентифікація дефіциту поживних речовин (наприклад, азоту);

– виявлення бур'янів для точкового застосування гербіцидів;

– моніторинг рівня зволоженості ґрунту;

– рання діагностика захворювань та ідентифікація шкідників.

Саме останній пункт — автоматизована фітопатологія — є однією з найбільш затребуваних та водночас найскладніших областей застосування CV.

1.1.1. Традиційні методи діагностики та їх обмеження

Традиційна діагностика хвороб рослин базується на візуальному огляді кваліфікованими агрономами або лабораторних методах (наприклад, полімеразна ланцюгова реакція, ПЛР)[3]. Обидва підходи мають суттєві обмеження, що перешкоджають їх масштабуванню (див. табл. 1.1).

Таблиця 1.1. Порівняльний аналіз методів діагностики хвороб рослин

Метод	Переваги	Недоліки
Візуальний огляд (Агроном)	Висока точність (за наявності експерта), врахування контексту	Суб'єктивність, висока вартість, повільність, неможливість цілодобового моніторингу, дефіцит кадрів
Лабораторний аналіз (ПЛР)	Найвища точність, ідентифікація патогену	Дуже висока вартість, тривалий час (дні), вимагає забору зразків, не підходить для оперативного моніторингу
Класичне ML (SVM, K-Means)	Швидше за ручні методи, низькі обчислювальні вимоги	Низька точність, необхідність ручного створення ознак (feature engineering), слабка робастність до змін фону/освітлення
Глибинне навчання (DL/CNN)	Висока точність (SOTA), автоматичне виділення ознак, швидкість (inference), масштабованість	Високі вимоги до даних та обчислювальних ресурсів (навчання), ризик перенавчання, проблеми інтерпретованості

Як видно з аналізу, традиційні методи є або надто повільними, або суб'єктивними, або нездатними до узагальнення. Підходи "класичного" машинного навчання (наприклад, Support Vector Machines на основі колірних гістограм чи текстурних ознак) не виправдали себе через складність ручного проектування ознак (feature engineering), які б були інваріантними до величезної варіативності симптомів.

1.1.2. Прорив глибинного навчання у фітопатології

Поява глибинного навчання, зокрема згорткових нейронних мереж (CNN), стала революційним кроком. На відміну від класичних підходів, CNN здатні до навчання репрезентацій (Representation Learning) — вони автоматично вивчають ієрархію ознак, необхідних для вирішення задачі, безпосередньо з сирих пікселів.

Визначальною роботою в цій галузі став проект PlantVillage, який надав масивний відкритий датасет з понад 50 000 зображень хворих та здорових листків. Дослідження, проведені на цьому датасеті, продемонстрували, що архітектури, такі як AlexNet та GoogLeNet, здатні досягати точності понад 99% у класифікації 26 хвороб у 14 видів рослин.

Цей успіх стимулював експоненціальне зростання досліджень у даній області (див. рис. 1.2), де CNN та новітні архітектури (ResNet, EfficientNet, ViT) послідовно встановлювали нові рекорди точності на все складніших наборах даних.

Рисунок 1.1 – Концептуальний графік зростання кількості наукових публікацій за темою "Deep Learning" та "Plant Disease" (Джерело: узагальнено на основі даних Scopus/Google Scholar)

1.1.3. Фактори, що зумовлюють актуальність проблеми

Незважаючи на вражаючі результати, досягнуті в лабораторних умовах (наприклад, на датасеті PlantVillage, де зображення зроблені на однорідному фоні), перехід до реальних польових умов (in-field) виявив низку фундаментальних проблем:

– складність фону: наявність ґрунту, інших рослин, тіней та сонячних відблисків;

– варіативність умов: зміни освітлення (ранок, день, вечір), погоди (хмарно, сонячно);

– дрібнозернистість (Fine-grained): багато хвороб мають вкрай схожі візуальні симптоми (низька міжкласова варіативність);

– багатоманітність симптомів: одна хвороба може виглядати по-різному на різних стадіях розвитку (висока внутрішньокласова варіативність);

– проблема ієрархії: як було зазначено вище, моделі ігнорують таксономічну структуру хвороб, що є неприродним.

Глибинне навчання є безальтернативною SOTA-технологією для автоматизованої діагностики хвороб рослин, що є ядром прецизійного землеробства. Однак, поточні методи, що здебільшого базуються на стандартному fine-tuning попередньо навчених моделей, демонструють крихкість при переході до реальних умов та ігнорують фундаментальну ієрархічну структуру предметної області. Це створює нагальну потребу в детальному аналізі самих архітектур та методів їх адаптації, що й буде розглянуто в наступних підрозділах.

1.2 Аналіз досліджуваних технологій в існуючих системах

1.2.1 Домінування згорткових нейронних мереж (CNN)

Прорив, описаний у попередньому підрозділі, став можливим завдяки архітектурі згорткової нейронної мережі (CNN). Її фундаментальна

ефективність у задачах обробки зображень ґрунтується на двох ключових індуктивних упередженнях, які відповідають природі візуальних даних :

– локальність (Locality): кожен нейрон у згортковому шарі пов'язаний лише з невеликою локальною областю (рецептивним полем) попереднього шару. Це відображає припущення, що візуальні патерни (текстури, грані) є локальними;

– інваріантність до зсуву (Translation Equivariance / Invariance): використання спільних ваг (parameter sharing) у згортковому ядрі означає, що ознака, виявлена в одній частині зображення, може бути так само виявлена в будь-якій іншій.

Ці упередження роблять CNN надзвичайно ефективними з точки зору кількості параметрів та дозволяють їм будувати ієрархічні репрезентації: від простих граней на нижніх шарах до складних об'єктних частин на верхніх.

Еволюція CNN була спрямована на вирішення проблеми навчання все глибших мереж та ефективнішого захоплення ознак:

– VGG (2014): продемонструвала, що глибина (використання стеків малих 3x3 згорток) є ключовим компонентом для досягнення високої точності;

– GoogLeNet / Inception (2014): ввела концепцію "Inception-модуля", який виконує згортки з різними розмірами ядер паралельно. Це дозволило моделі захоплювати багатомасштабні ознаки одночасно. Цей принцип є вкрай релевантним для нашої задачі, оскільки симптоми хвороб (напр., плями, некроз) можуть мати різний фізичний розмір;

– ResNet (2015): здійснила найважливіший прорив, ввівши залишкові зв'язки (residual connections) (див. Рис. 2.2). Ці "короткі" з'єднання, що додають вхід x до виходу шару F(x), дозволили градієнтам безперешкодно протікати через сотні шарів, вирішивши проблему згасання градієнтів (vanishing gradients). ResNet-подібні архітектури стали де-факто стандартом (baseline) для більшості задач CV.

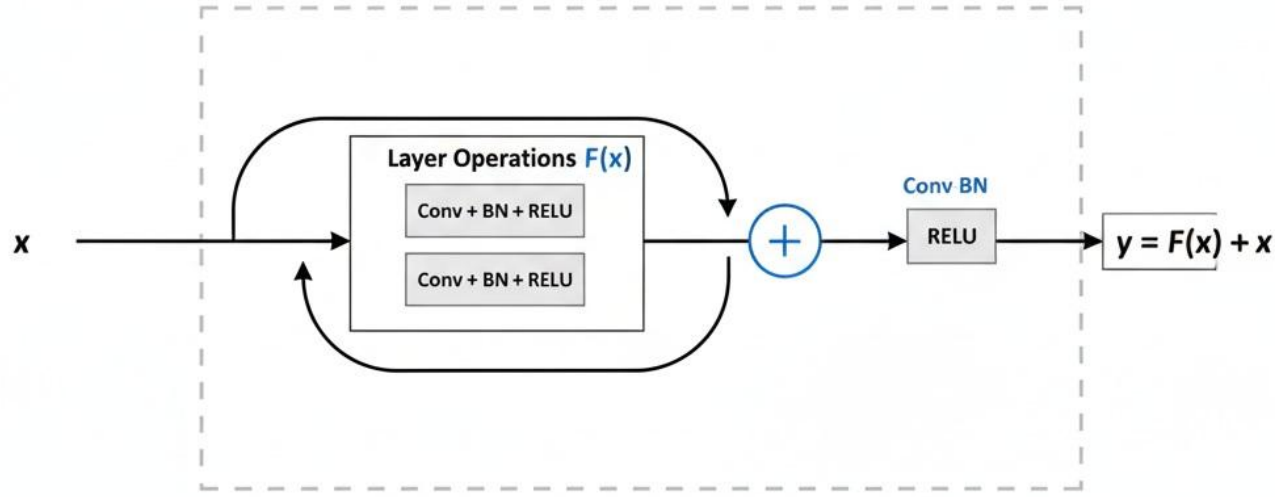

Рисунок 1.2 – Схематичне зображення залишкового блоку (Residual Block) в архітектурі ResNet. Вхід x додається до виходу функції F(x), що полегшує оптимізацію глибоких мереж.

– EfficientNet (2019): запропонувала методологію композитного масштабування (compound scaling), яка оптимально балансує глибину, ширину та роздільну здатність мережі, досягаючи найкращого компромісу між точністю та обчислювальною вартістю (FLOPs). Це є критично важливим для потенційного розгортання моделей на мобільних пристроях чи вбудованих системах (напр., на дронах).

1.2.2 Обмеження CNN та поява Vision Transformers (ViT)

Незважаючи на успіх, сильне індуктивне упередження CNN (локальність) є водночас і їхнім головним обмеженням. Згорткові шари, за своєю природою, є неефективними у моделюванні довготривалих просторових залежностей (long-range dependencies).

Для того, щоб нейрон на верхньому шарі "побачив" два віддалені пікселі, потрібен глибокий стек згорткових шарів, що поступово збільшують рецептивне поле. Це робить CNN "короткозорими". У нашій задачі це є критичною проблемою.

Приклад: Модель може легко ідентифікувати локальну текстуру іржі (fungal rust), але їй складно співвіднести цей симптом з патерном його розповсюдження вздовж жилок листка (глобальна ознака), що може бути ключовим для диференціації від іншого захворювання.

Цю проблему вирішили архітектури Трансформерів (Transformers), які спочатку здійснили революцію в обробці природної мови (NLP). Їхній основний механізм — само-увага (self-attention) — дозволяє моделі зважувати важливість кожного елемента послідовності відносно кожного іншого елемента, незалежно від їхньої відстані.

У 2020 році модель Vision Transformer (ViT) адаптувала цей підхід для зображень:

– зображення розбивається на 16x16 "патчів" (заплаток);

– кожен патч лінійно проектується у вектор (аналог "token" в NLP);

– ця послідовність векторів подається у стандартний енкодер трансформера.

Механізм self-attention (див. Рис. 2.3) дозволяє ViT моделювати глобальний контекст з першого ж шару. Модель може одночасно "дивитися" на верхівку листка і на його основу, вивчаючи складні взаємозв'язки.

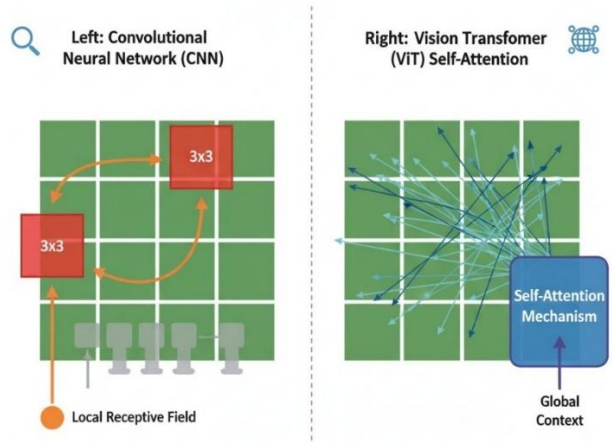

Рисунок 1.3 – Порівняння індуктивних упереджень. Зліва: CNN обробляє зображення через локальні рецептивні поля (ядра). Справа: Механізм self-attention у ViT обчислює взаємозв'язки між усіма патчами зображення одночасно, захоплюючи глобальний контекст.

ViT виявилися надзвичайно ефективними, але мали суттєвий недолік: вони позбавлені індуктивних упереджень CNN. Це робить їх "голодними до даних" (data-hungry). На відміну від ResNet, ViT, навчені з нуля на малих датасетах (як ImageNet-1K), показували гірші результати. Їхня перевага розкривалася лише при попередньому навчанні на гігантських пропрієтарних наборах даних (JFT-300M, ImageNet-21K).

Це призвело до конвергенції (зближення) двох парадигм:

– гібридні моделі (CNN + Transformer): дослідники почали комбінувати найкраще з обох світів. Наприклад, використання згорткового "хребта" (ResNet) для видобутку багатих локальних ознак на нижніх рівнях та застосування Transformer-енкодера на верхніх рівнях для агрегації глобального контексту;

– "модернізовані" CNN (ConvNeXt): дослідження показало, що якщо взяти класичний ResNet і поступово модернізувати його дизайн, імплементуючи архітектурні рішення з Transformer-ів (більші ядра згортки, глибинно-роздільні згортки, змінені активації), можна досягти якості ViT, зберігши простоту та ефективність CNN.

Аналіз еволюції архітектур показує чіткий тренд: від жорстко локальних CNN до повністю глобальних ViT, і, зрештою, до гібридних підходів, що балансують обидві властивості. Для нашої задачі діагностики хвороб, яка вимагає одночасного аналізу дрібнозернистих локальних текстур (симптоми) та глобальних просторових патернів (розповсюдження), вибір архітектури "хребта" (backbone) є нетривіальною задачею. Це обґрунтовує необхідність тестування як сучасних CNN (напр., EfficientNet, ConvNeXt), так і гібридних чи чистих ViT-архітектур в якості основи для нашої пропонованої ієрархічної моделі.

1.2.3. Парадокс Transfer Learning: ефективність проти крихкості

Архітектури, розглянуті в попередньому підрозділі (ResNet, EfficientNet, ViT), досягають своєї ефективності лише за умови попереднього навчання (pre-training) на масивних, загально-доменних наборах даних, таких як ImageNet. Цей процес дозволяє моделі вивчити багатий, ієрархічний простір візуальних ознак — від простих текстур до складних об'єктних частин. Transfer Learning (трансферне навчання) є парадигмою, що дозволяє повторно використати ці знання для нових, вузькоспеціалізованих завдань, де збір мільйонів анотованих зображень є неможливим.

Найпоширенішим методом трансферного навчання є fine-tuning[4] (тонка настройка). Його канонічний підхід полягає у заміні останнього класифікаційного шару моделі (напр., 1000 класів ImageNet) на новий, що відповідає цільовій задачі (напр., 12 класів хвороб рослин), та подальшому донавчанні всієї або частини мережі на нових даних з низькою швидкістю навчання.

Тут виникає фундаментальний парадокс: хоча fine-tuning є надзвичайно ефективним з точки зору використання параметрів, він приховує в собі низку проблем, що роблять стандартну його імплементацію крихкою та недостатньо робастною для складних прикладних завдань.

1.2.4. Ключові виклики стандартного Fine-Tuning у цільових доменах

Аналізуючи дану тематику можна зіткнутись з трьома основними проблемами, які виникають під час стандартного fine-tuning, і які є особливо гострими для задачі діагностики хвороб:

– зсув Домену (Domain Shift): це, можливо, найважливіший виклик. Попередньо навчені моделі тренувалися на датасеті ImageNet, який складається переважно з чітких, центрованих зображень об'єктів. Цільовий домен (напр., змагання Kaggle Plant Pathology 2021) має кардинально інший статистичний розподіл даних ($P_{source}(X, Y) \neq P_{target}(X, Y)$). Зображення "в

полі" (in-the-field) характеризуються: складним фоном (листя, ґрунт, інші рослини), варіативним освітленням (пряме сонячне світло, тіні, вечірня зйомка), різними ракурсами та масштабами. Модель, що пройшла fine-tuning, може "перенавчитися" на ці специфічні характеристики цільового датасету і втратити узагальнюючу здатність при найменшій зміні умов (напр., при використанні іншої камери);

– катастрофічне Забування (Catastrophic Forgetting): цей феномен описує процес, під час якого модель при адаптації до нового завдання ***B*** "забуває" знання, отримані при вирішенні вихідного завдання ***A***. Під час fine-tuning, градієнтні оновлення оптимізують ваги виключно під нову, вузьку задачу (напр., розрізнення scab та rust). Якщо fine-tuning проводиться надто агресивно, ваги, що кодували узагальнені репрезентації (напр., "що таке текстура" або "що таке контур"), можуть бути безповоротно змінені. Це знижує загальну робастність моделі, оскільки вона фактично "розучується" бачити фундаментальні візуальні примітиви;

– проблема переспеціалізації ознак (Over-specialization): це найбільш релевантна проблема для нашого дослідження. Стандартний fine-tuning для задачі з ***N*** класами (напр., 12 хвороб) оптимізує єдину функцію втрат. Це змушує модель шукати дрібні, унікальні деталі, що відрізняють клас scab від rust. При цьому модель не має жодного стимулу вивчати або зберігати ознаки, що є спільними для груп класів. Наприклад, узагальнена ознака "наявність будь-якої хвороби" (що відрізняє всі 11 хворих класів від 1 здорового) може бути "розмита" або втрачена, оскільки вона не є оптимальною для мінімізації втрат на рівні 12-класової класифікації. Модель стає спеціалістом у дрібницях, втрачаючи розуміння загальної картини.

1.2.5. Сучасні стратегії пом'якшення та ефективного Fine-Tuning

Наукова спільнота активно працює над вирішенням цих проблем. Існуючі методи можна згрупувати у два напрямки:

а) покращені стратегії повного Fine-Tuning:

1) диференційовані швидкості навчання (Differential Learning Rates): Цей підхід, популяризований у бібліотеці fastai та методології ULMFiT, пропонує не "заморожувати" шари повністю, а навчати їх з різною швидкістю. Нижні шари (backbone), що вивчили загальні ознаки, донавчаються з дуже низьким lr (напр., 10^{-6}, тоді як верхні шари (head), специфічні для задачі, навчаються зі значно вищим lr (напр.,10^{-3}). Це дозволяє адаптувати модель, мінімізуючи ризик катастрофічного забування;

2) поступове "розморожування" (Gradual Unfreezing): Подальший розвиток ідеї, де спочатку навчається лише нова "голова", потім "розморожується" і донавчається верхній блок backbone, і так до повного донавчання всіх шарів.

б) параметр-ефективна настройка (Parameter-Efficient Fine-Tuning, PEFT): це новітня і надзвичайно потужна парадигма, яка пропонує не чіпати (не навчати) жодного з оригінальних мільйонів параметрів попередньо навченої моделі. Замість цього, вона "заморожує" весь backbone і додає невелику кількість (напр., 0.1% - 2% від загальної кількості) нових, навчаємих параметрів:

1) Адаптери (Adapter Modules): Невеликі нейронні мережі (зазвичай вузькі місця "bottleneck" типу N→ K→ N), що вставляються всередину кожного блоку Transformer або ResNet. Навчаються лише ці адаптери;

2) LoRA (Low-Rank Adaptation): Найбільш популярний метод. Він стверджує, що оновлення ваг ΔW під час fine-tuning має низький внутрішній ранг (low intrinsic rank). Тому замість навчання повного ΔW, LoRA навчає лише дві малі матриці $\boldsymbol{A}$ та $\boldsymbol{B}$, чий добуток $\boldsymbol{AB}$ апроксимує ΔW. Це дозволяє

досягти якості, співставної з повним fine-tuning, але навчаючи на порядки менше параметрів.

1.2.6. Висновок та ідентифікація дослідницького розриву

Критичний аналіз показує, що сучасні методи fine-tuning (Differential LR, PEFT, LoRA) є потужними інструментами, які вирішують проблеми обчислювальної ефективності та катастрофічного забування. Вони дозволяють ефективно адаптувати гігантські моделі (ViT, ConvNeXt) до нових завдань, зберігаючи їхні узагальнені знання.

Однак, вони фундаментально ігнорують семантичну структуру самої задачі.

Усі ці методи, як і раніше, розглядають цільову задачу як "плаский" набір з N незалежних класів. Вони не вирішують проблему переспеціалізації ознак і не змушують модель мислити ієрархічно. Адаптер LoRA[5], навчений розрізняти apple_scab та frog_eye_leaf_spot, так само не матиме жодного уявлення про те, що обидва ці класи є "грибковими" і протиставляються "здоровому" стану.

Таким чином, ми ідентифікуємо чіткий дослідницький розрив (research gap): відсутність методів, які б поєднували ефективність сучасних стратегій fine-tuning з семантичною коректністю, тобто з явним врахуванням ієрархічної таксономії класів безпосередньо в процесі навчання.

Саме цей розрив обґрунтовує необхідність переходу від "пласких" моделей до структурованих, про що й піде мова в наступному підрозділі, присвяченому ієрархічній класифікації.

1.2.7. Існуючі підходи до ієрархічної та дрібнозернистої класифікації (FGVC)

Задачі, подібні до діагностики хвороб рослин, у літературі з комп'ютерного зору належать до категорії дрібнозернистої візуальної класифікації (Fine-Grained Visual Classification, FGVC). Одночасно, наявність таксономії (здоровий/хворий → тип → хвороба) вносить аспект ієрархічної класифікації. Розглянемо ці два напрямки.

Задачі FGVC (наприклад, розпізнавання видів птахів, моделей автомобілів, або, у нашому випадку, хвороб) характеризуються двома ключовими проблемами:

– низька міжкласова варіативність (Low Inter-class Variance): різні класи є надзвичайно схожими. Наприклад, симптоми apple_scab та frog_eye_leaf_spot можуть виглядати як "маленькі темні плями";

– висока внутрішньокласова варіативність (High Intra-class Variance): об'єкти одного класу можуть сильно відрізнятися через ракурс, освітлення, стадію (напр., рання та пізня стадія хвороби) або фон.

Стандартні CNN, навчені на ImageNet, оптимізовані для розрізнення об'єктів на рівні базових категорій (напр., кіт vs. собака). Їхні ознаки недостатньо дискримінативні для FGVC. Для вирішення цієї проблеми були розроблені спеціалізовані методи:

– локалізація на основі частин (Part-based Localization): ранні підходи намагалися явно локалізувати семантичні частини об'єкта (напр., для птаха — "голова", "дзьоб", "крила") та класифікувати на основі цих частин. Це вимагало дорогих додаткових анотацій (bounding boxes для частин);

– механізми уваги (Attention Mechanisms): сучасніші підходи використовують механізми уваги, щоб модель сама навчилася знаходити найбільш релевантні (дискримінативні) регіони. Наприклад, модель вчиться, що для розрізнення двох видів птахів потрібно дивитися не на все тіло, а саме на форму дзьоба та колір пір'я на голові. Прикладом є Bilinear Attention Pooling

(BAP), де дві паралельні гілки CNN знаходять ознаки та їхню локалізацію, дозволяючи моделі фокусуватися на тонких відмінностях.

Ці методи довели свою ефективність, але вони часто є архітектурно складними та не вирішують іншої нашої проблеми — ігнорування ієрархії.

Коли класи організовані у деревоподібну або DAG-структуру (спрямований ациклічний граф), ігнорування цієї структури є неефективним. У літературі з машинного навчання існують три основні підходи до вирішення цієї проблеми

а) пласка класифікація (Flat Classification): Це наш baseline. Ієрархія повністю ігнорується. Модель навчається на одному "пласкому" списку всіх листових вузлів (напр., healthy, scab, rust...);

1) Переваги: Простота імплементації, використання стандартних архітектур;

2) Недоліки: Повна втрата семантичної інформації, семантично необґрунтовані помилки (модель може сплутати scab з healthy так само легко, як scab з rust), погане узагальнення для рідкісних класів.

б) локальні підходи (Local Classifiers): ця стратегія декомпозує проблему. Навчається набір окремих, незалежних класифікаторів. Найпоширеніший варіант — "Локальний класифікатор на батьківський вузол" (Local Classifier per Parent Node, LCPN):

1) приклад: спершу навчається один класифікатор на кореневому рівні (напр., здоровий vs. хворий). Потім, тільки для даних, класифікованих як хворий, навчається другий класифікатор (грибковий vs. бактеріальний). Потім третій (scab vs. rust) і т.д.;

2) переваги: кожен класифікатор вирішує простішу, маловимірну задачу;

3) недоліки: каскадне поширення помилки (Error Propagation). Якщо кореневий класифікатор помиляється (напр., називає хворий лист здоровим), вся подальша гілка класифікації стає недоступною, і помилка не може бути

виправлена. Окрім того, це обчислювально неефективно, оскільки вимагає навчання N окремих моделей;

в) глобальні підходи (Global Classifiers): цей підхід намагається врахувати всю ієрархію в рамках єдиної моделі та єдиного процесу навчання:

1) приклад: модель має єдиний вихідний шар для всіх класів (включаючи проміжні, напр. fungal), але функція втрат модифікується. Наприклад, ієрархічна крос-ентропія може штрафувати помилки по-різному: помилка між scab (дитина) та fungal (батько) штрафується менше, ніж помилка між scab та bacterial (інша гілка);

2) переваги: обчислювальна ефективність (одна модель), відсутність каскадної помилки;

3) недоліки: складність проектування функції втрат, яка б коректно відображала ієрархічні відстані. Існуючі "ієрархічні" функції втрат часто є складними в оптимізації та не гарантують, що простір ознак (latent space) моделі дійсно відобразить цю ієрархію.

Огляд існуючих методів показує, що дослідники або фокусуються на дрібнозернистих деталях (FGVC), ігноруючи ієрархію, або на ієрархії (LCPN, Global), але часто ціною архітектурної складності або ризику каскадних помилок.

Наш пропонований підхід Multi-Head Fine-Tuning (MHFT) позиціонується як гібридний метод, що бере найкраще від обох світів:

– він є глобальним (єдина модель, спільний backbone), що запобігає каскадним помилкам;

– він архітектурно реалізує ієрархію (окремі голови для кожного рівня) замість того, щоб покладатися лише на складну функцію втрат;

– використання спільного backbone, що одночасно навчається на грубих (healthy/diseased) та тонких (scab/rust) ознаках, діє як вбудований механізм регуляризації, змушуючи модель вивчати ознаки різного масштабу, що є доведеною технікою у FGVC.

1.2.8. Синтез та ідентифікація дослідницького розриву

Попередньо проведений глибокий аналіз предметної області та суміжних технологій дозволяє зробити наступні ключові висновки:

– проблема (домен): автоматизована діагностика хвороб рослин є SOTA-технологією, але її надійність у реальних польових умовах (in-the-field) залишається низькою. Ключові виклики — це зсув домену (мінливе освітлення, фон), дрібнозернистість (схожі симптоми) та ієрархічність (семантична структура класів);

– інструменти (архітектури): ми маємо в своєму розпорядженні надзвичайно потужні екстрактори ознак. Еволюція від ResNet до ConvNeXt та Vision Transformers дала нам моделі, що здатні ефективно захоплювати як локальні текстури (критично для симптомів), так і глобальний контекст (критично для патернів розповсюдження);

– методи (навчання): ми маємо ефективні стратегії адаптації (fine-tuning). Такі підходи, як диференційовані швидкості навчання та параметр-ефективні методи (PEFT/LoRA), дозволяють вирішити проблему катастрофічного забування та адаптувати гігантські моделі, зберігаючи їхні цінні узагальнені репрезентації;

– спеціалізація (підходи): існують спеціалізовані рішення як для дрібнозернистої (FGVC, увага), так і для ієрархічної (LCPN, глобальні loss-функції) класифікації.

Фундаментальна проблема сучасної наукової літератури полягає в тому, що ці чотири компоненти існують та розвиваються паралельно, але не інтегровано. Дослідницький розрив знаходиться на перетині цих напрямків:

– дослідники, що працюють над PEFT/LoRA, зосереджені на ефективності адаптації, але продовжують застосовувати її до "пласких" (flat) задач, ігноруючи семантику ієрархії;

– дослідники, що працюють над ієрархічною класифікацією, фокусуються на структурі виходів (напр., LCPN), але їхні методи часто

вносять нові проблеми (каскадне поширення помилки) і не завжди використовують переваги SOTA-архітектур чи стратегій fine-tuning;

– дослідники, що працюють над FGVC, зосереджені на виділенні тонких, дискримінативних ознак, але так само ігнорують той факт, що ієрархія може слугувати потужним регуляризатором для цього процесу.

Суть дослідницького розриву (The Problem Gap): відсутній єдиний, інтегрований фреймворк, який би поєднував потужність SOTA-архітектур (ViT/ConvNeXt) та робастність сучасних методів fine-tuning з явним, архітектурно вбудованим механізмом ієрархічної регуляризації.

Існуючі підходи або ігнорують ієрархію на користь "пласкої" точності, або впроваджують ієрархію методами (напр., LCPN), що призводять до каскадних помилок та неефективного навчання ознак.

Наше рішення як відповідь на ідентифікований розрив: ця магістерська робота пропонує заповнити саме цей пробіл. Ми висуваємо гіпотезу, що багатоголова архітектура (Multi-Head Fine-Tuning, MHFT) є тим самим інтегрованим фреймворком.

Наш підхід:

– використовує єдиний SOTA-backbone, що навчається end-to-end, уникаючи каскадних помилок (LCPN) та ефективно захоплюючи ознаки (FGVC);

– замінює єдину "пласку" голову на набір ієрархічних голів (грубий, середній, точний рівні);

– використовує ці паралельні задачі як механізм ієрархічної регуляризації. Замість того, щоб покладатися на складну функцію втрат, ми змушуємо сам backbone генерувати такий простір ознак, який є одночасно інформативним для всіх рівнів абстракції.

Ми припускаємо, що навчання розрізняти healthy vs. diseased (груба задача) змусить модель вивчити робастні, інваріантні до домену ознаки патології, що, в свою чергу, регуляризує навчання більш тонкої задачі

(розрізнення scab vs. rust), покращуючи її загальну точність та узагальнюючу здатність.

Таким чином, ця робота переходить від питання "Чи можемо ми класифікувати хвороби?" до питання "Чи можемо ми створити модель, яка розуміє семантичну структуру хвороб, і чи робить це її більш робастною та інтерпретованою?". Це обґрунтування визначає мету та завдання нашого подальшого експериментального дослідження, викладеного в наступних розділах.

1.3 Постановка задачі

На основі проведеного аналізу предметної області та існуючих методів було встановлено, що сучасні підходи до класифікації хвороб рослин, які базуються на стандартній процедурі донавчання нейронних мереж, мають суттєвий недолік: вони ігнорують природну ієрархічну структуру біологічних даних. Це призводить до того, що моделі часто роблять логічно необґрунтовані помилки та погано працюють при зміні умов освітлення чи фону. Виходячи з цього, виникає необхідність сформулювати задачу розробки нового методу навчання, який би інтегрував знання про типи хвороб безпосередньо в структуру нейронної мережі.

Об’єктом даного дослідження є процес автоматизованої візуальної діагностики патологій рослин за допомогою методів комп'ютерного зору в реальних польових умовах. Предметом дослідження виступають методи та архітектури ієрархічного налаштування нейронних мереж, які дозволяють підвищити точність та надійність розпізнавання. Суть наукової задачі полягає у тому, щоб на основі вхідних зображень листя знайти таку конфігурацію нейронної мережі, яка б не просто мінімізувала помилки розпізнавання конкретної хвороби, а й забезпечувала логічну узгодженість прогнозу на всіх рівнях — від визначення, чи здорова рослина взагалі, до встановлення типу збудника та конкретного виду захворювання.

Виконання роботи базується на кількох ключових наукових припущеннях. По-перше, ми припускаємо, що візуальні ознаки хвороб, які належать до однієї групи, наприклад, грибкових захворювань, мають спільні риси, і їх спільне вивчення допоможе моделі краще розрізняти конкретні діагнози. По-друге, ми вважаємо, що додавання допоміжних завдань з розпізнавання загальних категорій діє як своєрідний фільтр, який змушує модель ігнорувати випадковий шум на зображенні та фокусуватися на дійсно важливих симптомах. По-третє, ми виходимо з того, що сучасні архітектури нейронних мереж мають достатню потужність, щоб вирішувати всі ці завдання одночасно в рамках однієї моделі, без необхідності створювати складні ланцюжки з багатьох мереж.

Для вирішення поставленої задачі обрано напрямок багатоголового навчання. Цей підхід відрізняється від інших тим, що дозволяє тренувати єдину модель, яка має спільну основу для вилучення ознак, але різні вихідні блоку для кожного рівня складності задачі. Методика дослідження передбачає підготовку спеціально розмічених даних, проектування архітектури з паралельними гілками класифікації та проведення порівняльного експерименту зі стандартними методами для підтвердження ефективності запропонованого підходу.

Отримані результати мають чітку практичну спрямованість і можуть бути застосовані в системах прецизійного землеробства для моніторингу полів за допомогою дронів, де критично важливою є стійкість роботи алгоритмів до погодних умов. Також розроблений метод може стати основою для мобільних додатків, що допомагають фермерам приймати рішення про захист рослин, надаючи не лише кінцевий діагноз, а й структуровану інформацію про тип та природу захворювання, що значно підвищує довіру до автоматизованих систем. Крім того, такий підхід дозволяє створювати більш економні системи точкового обприскування, які активуються лише при виявленні патології, зменшуючи використання агрохімікатів.

2 МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ

У цьому розділі детально описано експериментальну базу дослідження. Він включає аналіз вихідного набору даних, опис методів його попередньої обробки та аугментації, а також детальну формалізацію пропонованої ієрархічної архітектури, метрик оцінювання та конфігурації експериментів, необхідних для валідації висунутої гіпотези.

2.1. Характеристика та перед-обробка вихідних даних

2.1.1. Опис вихідного набору даних (Kaggle Plant Pathology 2021-FGVC8)

Для проведення експериментального дослідження було обрано публічний, широко визнаний у науковій спільноті набір даних Plant Pathology 2021-FGVC8, представлений у рамках змагання на платформі Kaggle. Цей датасет є особливо цінним для нашого дослідження, оскільки він репрезентує ключові виклики, описані в Розділі 1:

– реалістичність ("in-the-field"): на відміну від ранніх датасетів (як PlantVillage), зображення зроблені в реальних польових умовах, що включають складний фон, варіативне природне освітлення, тіні, сонячні відблиски та різні ракурси зйомки (див. Рис. 3.1);

– дрібнозернистість (Fine-grained): багато представлених хвороб мають візуально схожі симптоми (напр., різні види плямистості);

– багато-міткова природа (Multi-label): одне зображення (листок) може бути уражене кількома хворобами одночасно, що є реалістичним агрономічним сценарієм.

Рисунок 2.1 – Приклади зображень з датасету Plant Pathology 2021. Демонструються складні умови зйомки "в полі" та різноманітність візуальних симптомів.

Набір даних складається з 18 632 тренувальних зображень листя яблуні у високій роздільній здатності, анотованих шістьма можливими класами: healthy (здоровий), scab (парша), frog_eye_leaf_spot (бура плямистість), rust (іржа), powdery_mildew (борошниста роса) та complex (комплексне ураження кількома хворобами, які важко розрізнити візуально).

2.1.2. Статистичний аналіз та виявлення дисбалансу

Первинний аналіз анотацій виявляє два критичних аспекти, що суттєво впливають на проектування моделі.

Проблема дисбалансу класів (Class Imbalance): Як видно з Таблиці 2.1, розподіл міток є вкрай нерівномірним. Такі поширені хвороби, як scab,

зустрічаються значно частіше, ніж powdery_mildew. Клас healthy також становить значну частку. Будь-яка модель, навчена на цих даних "наївно", буде упередженою в бік домінуючих класів.

Таблиця 2.1. Розподіл міток у тренувальному наборі Plant Pathology 2021

Клас	Кількість рядків	% від усіх зображень
complex	2 151	11.54%
frog_eye_leaf_spot	4 352	23.36%
healthy	4 624	24.82%
powdery_mildew	1 271	6.82%
rust	2 077	11.15%
scab	5 712	30.66%

Аналіз спів-існування міток (Label Co-occurrence): Оскільки задача є багато-мітковою, важливо зрозуміти, які хвороби найчастіше з'являються разом. Аналіз показує, що мітка complex майже завжди з'являється разом з іншими хворобами, найчастіше з scab та frog_eye_leaf_spot. Це підтверджує, що complex не є окремою хворобою, а скоріше індикатором високої складності діагностики. Статистичні дані кобінації міток представлені в таблиці 2.2. Матриця спів-існування (Рис. 2.2) візуалізує ці залежності.

Таблиця 2.2. Комбінації міток у тренувальному наборі Plant Pathology 2021

Комбінація міток	Кількість	% від усіх
scab	4 826	25.90%
healthy	4 624	24.82%
frog_eye_leaf_spot	3 181	17.07%
rust	1 860	9.98%
complex	1 602	8.60%
powdery_mildew	1 184	6.35%
frog_eye_leaf_spot + scab	686	3.68%
complex + frog_eye_leaf_spot + scab	200	1.07%
complex + frog_eye_leaf_spot	165	0.89%
frog_eye_leaf_spot + rust	120	0.64%
complex + rust	97	0.52%
complex + powdery_mildew	87	0.47%

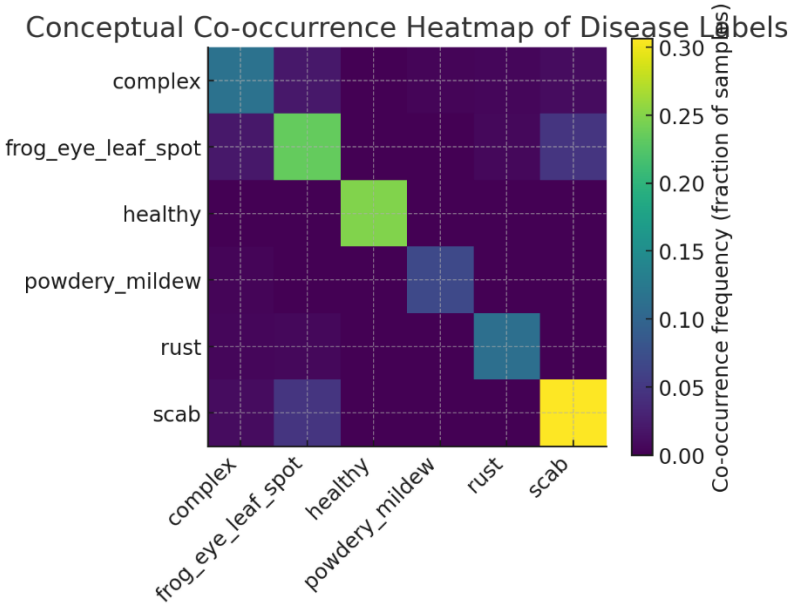

Рисунок 2.2 – Концептуальна теплова карта спів-існування міток. Яскравіші комірки вказують на частіше одночасне виникнення пар хвороб. Це ілюструє складні залежності, які має вивчити модель.

2.1.3. Інжиніринг ієрархічних міток (Label Engineering)

Базуючись на аналізі даних та цілях нашого дослідження, ми виконуємо інжиніринг ознак (міток), щоб перетворити "плаский" багато-мітковий набір даних у структурований, ієрархічний. Ми створюємо три нові цільові змінні (мітки) для кожного зображення, які будуть використовуватися для навчання трьох голів нашої моделі (MHFT).

а) head 1: "Груба" мітка (Coarse-grained) – label_coarse:

1) завдання: бінарна класифікація healthy vs. diseased;

2) логіка: якщо оригінальні мітки містять лише healthy, то label_coarse = 0 (здоровий). В будь-якому іншому випадку (наявність scab, rust, complex тощо), label_coarse = 1 (хворий);

3) обґрунтування: це змушує першу голову вивчити найбільш загальні, робастні ознаки наявності будь-якої патології;

б) head 2: "середня" мітка (Mid-grained) – label_mid:

1) завдання: бінарна класифікація mono_disease vs. complex_disease;

2) логіка: якщо зображення має label_coarse = 0 (здорове), то label_mid = 0 (моно). Якщо зображення має лише одну мітку хвороби (напр., лише scab або лише rust), то label_mid = 0 (моно). Якщо зображення має дві чи більше міток хвороб (напр., scab rust) АБО мітку complex, то label_mid = 1 (комплекс);

3) обґрунтування: ця голова вчиться оцінювати складність випадку, розрізняючи просте ураження однією хворобою від складного, комбінованого;

в) head 3: "Точна" мітка (Fine-grained) – label_fine:

1) завдання: багато-міткова класифікація (відповідає оригінальній задачі Kaggle);

2) логіка: 6-вимірний one-hot (або, точніше, multi-hot) вектор, що відповідає наявності чи відсутності кожної з 6 оригінальних міток ([healthy, scab, frog_eye, rust, complex, powdery]);

3) обґрунтування: це основна, цільова задача нашого дослідження.

2.1.4. Стратегія аугментації та підготовки даних

Для боротьби з перенавчанням та дисбалансом класів, а також для штучного моделювання "зсуву домену" (варіацій освітлення, ракурсів), ми застосовуємо конвеєр агресивних аугментацій (використовуючи бібліотеку Albumentations):

а) розбиття даних: весь тренувальний набір (18 632) розбивається на тренувальний (80%) та валідаційний (20%) набори. Розбиття проводиться стратифіковано (з використанням iterative-stratification) для збереження однакового багато-міткового розподілу в обох вибірках;

б) геометричні аугментації:

1) resize: приведення всіх зображень до єдиного розміру (напр., 512x512 або 384x384);

2) horizontalFlip / verticalFlip: імітація різних орієнтацій листка;

3) shiftScaleRotate: незначні зсуви, масштабування та обертання для підвищення інваріантності;

в) Фотометричні аугментації (ключові для робастності):

1) colorJitter / randomBrightnessContrast: імітація зйомки в різний час доби (зміна яскравості, контрасту, насиченості);

2) coarseDropout / Cutout: випадкове "вирізання" шматків зображення, що змушує модель фокусуватися не на одній ознаці, а на сукупності доказів;

г) нормалізація: приведення значень пікселів до діапазону `[0, 1]` та нормалізація з використанням середніх значень та стандартного відхилення датасету ImageNet, на якому навчався наш backbone.

Цей комплексний підхід до аналізу та підготовки даних створює надійний фундамент для навчання як нашої baseline-моделі, так і пропонованої ієрархічної MHFT-архітектури.

2.2. Архітектура базової (Baseline) моделі та пропонованої MHFT-моделі

Для валідації нашої центральної гіпотези — про те, що архітектурно-вбудована ієрархія покращує узагальнення та робастність — ми проектуємо контрольований експеримент. Ми порівнюємо два підходи: (A) потужну Baseline-модель, що репрезентує стандартний SOTA-підхід до "пласкої" класифікації, та (B) нашу пропоновану багатоголову ієрархічну модель (MHFT).

Критичним принципом нашого експерименту є контроль змінних: обидві моделі будуть використовувати ідентичну архітектуру "хребта" (backbone), ідентичні вхідні дані та ідентичні гіперпараметри навчання. Єдина відмінність полягатиме в архітектурі класифікаційних голів та, як наслідок, у функції втрат.

2.2.1. Архітектура базової (Baseline) моделі

Baseline-модель проектується як сильний, а не "солом'яний" (strawman), опонент. Вона реалізує стандартний, але потужний підхід до вирішення багато-міткової задачі з Kaggle.

а) екстрактор ознак (Backbone): ми обираємо архітектуру з родини EfficientNet, наприклад, EfficientNet-B4 або B5. Цей вибір обґрунтований тим, що EfficientNet (як проаналізовано в 1.2.2) пропонує найкращий компроміс між обчислювальною ефективністю (FLOPs) та точністю завдяки своєму методу композитного масштабування. Модель ініціалізується вагами, попередньо навченими на ImageNet-21k (або, як мінімум, ImageNet-1k), щоб використати переваги трансферного навчання;

б) архітектура Голови (Head): оригінальний класифікаційний шар EfficientNet (на 1000 класів) видаляється. Замість нього додається нова "голова". Для забезпечення високої ємності та робастності, голова складається не з одного лінійного шару, а з послідовності:

1) global average pooling (GAP): шар, що агрегує просторові ознаки з останнього згорткового блоку в єдиний вектор ознак (напр., розмірністю d=1792 для EfficientNet-B4);

2) dropout: шар dropout (напр., p=0.3) для регуляризації та запобігання перенавчанню класифікатора;

3) вихідний шар: єдиний лінійний шар nn.Linear(d, C_fine), де $C_{fine} = 6$ — кількість класів у "пласкій" задачі (healthy, scab, frog_eye_leaf_spot, rust, powdery_mildew, complex);

в) функція активації та втрат: оскільки задача є багато-мітковою (multi-label), модель не використовує активацію Softmax на виході. Вона видає 6 незалежних логітів (raw outputs). Ці логіти потім передаються безпосередньо у функцію втрат BCEWithLogitsLoss (Двійкова крос-ентропія з логітами). Ця функція є єдиним правильним вибором, оскільки вона:

1) математично еквівалентна застосуванню функції Sigmoid до кожного з 6 виходів та подальшому розрахунку двійкової крос-ентропії;

2) є чисельно стабільною;

3) коректно обробляє ситуацію, де кожен клас є незалежним бінарним завданням ("чи є парша?" - так/ні, "чи є іржа?" - так/ні).

Загальна архітектура Baseline-моделі візуалізована на Рис. 2.3.

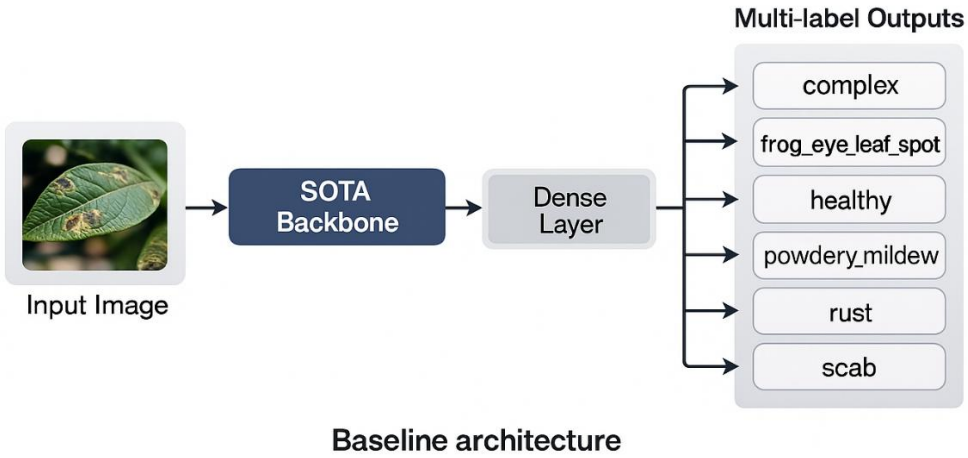

Рисунок 2.3 – Архітектура базової (Baseline) моделі. Використовується єдиний SOTA-backbone та єдина "пласка" класифікаційна голова, що оптимізується під багато-міткову двійкову крос-ентропію.

2.2.2. Архітектура пропонованої MHFT-моделі

Пропонована модель Multi-Head Hierarchical Fine-Tuning (MHFT) використовує той самий, ідентичний backbone EfficientNet-B4, що й baseline-модель. Це є критично важливим для чистоти експерименту.

Однак, після отримання вектора ознак z з шару Global Average Pooling, архітектура кардинально змінюється. Замість однієї голови, ми "розгалужуємо" обчислювальний граф і приєднуємо три незалежні голови, кожна з яких відповідає своєму рівню ієрархії, визначеному нами в Розділі 2.1.3. :

а) Екстрактор Ознак (Backbone): EfficientNet-B4 (або B5), ідентичний baseline. Його ваги навчаються спільно для всіх трьох завдань.

б) Архітектура Багатоголового Класифікатора (Multi-Head): Спільний вектор ознак z (після GAP) паралельно подається на три голови:

1) head 1 (Coarse-grained): H_coarse

– завдання: healthy vs. diseased (бінарна класифікація);

– архітектура: Dropout(p=0.3) -> nn.Linear(d, 2);

– втрати: Оскільки це бінарне, але взаємовиключне завдання, ми використовуємо стандартну Cross-Entropy Loss (nn.CrossEntropyLoss);

2) head 2 (Mid-grained): H_mid

– завдання: mono_disease vs. complex_disease (бінарна класифікація);

– архітектура: dropout(p=0.3) -> nn.Linear(d, 2);

– втрати: аналогічно до Head 1, використовуємо Cross-Entropy Loss (nn.CrossEntropyLoss);

3) head 3 (Fine-grained): H_fine

– завдання: багато-міткова класифікація (наша цільова задача);

– архітектура: Dropout(p=0.3) -> nn.Linear(d, 6);

– втрати: ідентично до baseline, використовуємо BCEWithLogitsLoss (nn.BCEWithLogitsLoss).

Загальна архітектура MHFT-моделі візуалізована на Рис. 2.4.

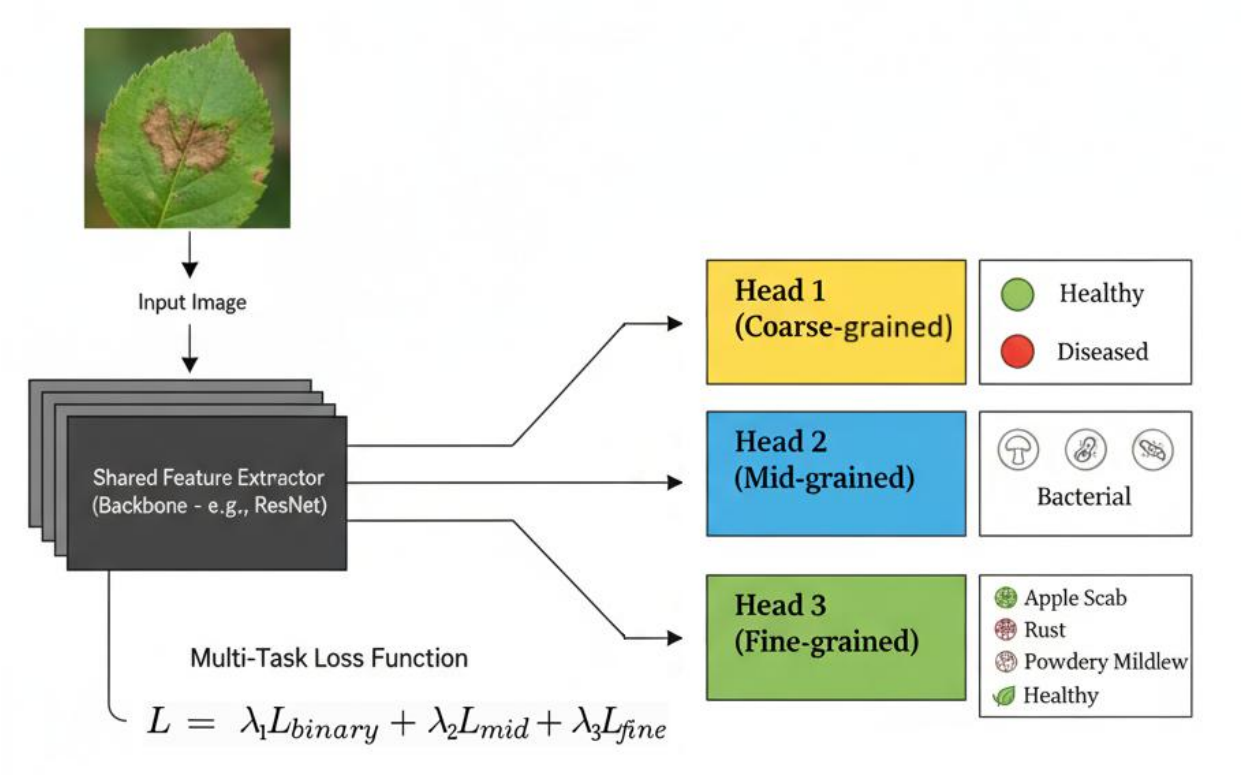

Рисунок 2.4 – Архітектура пропонованої MHFT-моделі. Спільний backbone формує вектор ознак, який паралельно обробляється трьома незалежними головами, кожна з яких вирішує своє завдання на своєму рівні ієрархії.

Процес навчання та ієрархічна регуляризація відбувається за певним принципом. Навчання всіх голів відбувається одночасно в режимі multi-task learning. Загальна функція втрат є зваженою сумою крос-ентропійних втрат кожної голови:

$$L = \lambda_1 L_{binary} + \lambda_2 L_{mid} + \lambda_3 L_{fine}$$

Загальна помилка моделі (L) розраховується як сума трьох помилок, кожна з яких має свій ваговий коефіцієнт. Перша помилка відповідає за просте завдання (наприклад, хворий/здоровий), друга — за проміжне (наприклад, вид хвороби), а третя — за головне, детальне завдання (наприклад, конкретна хвороба). Такий підхід змушує модель вчитися краще, створюючи універсальні ознаки, що підвищує її загальну точність.

2.3. Метрики оцінювання та протокол експерименту

Цей підрозділ визначає кількісні критерії оцінки та методологію проведення експериментів. Правильний вибір метрик та чіткий протокол є вирішальними для об'єктивного порівняння ефективності базової (Baseline) моделі та пропонованої багатоголової ієрархічної (MHFT) моделі.

2.3.1. Метрики оцінювання

Для всебічної оцінки буде застосовано декілька метрик. Первинна метрика Macro F1-Score. Macro F1-Score для "точної" (fine-grained) багато-міткової класифікації (завдання H_fine). Метрика F1 є гармонійним середнім між точністю (Precision) та повнотою (Recall). У багато-мітковому та багатокласовому сценарії існує кілька методів усереднення. Ми обираємо Macro F1-Score, оскільки він обчислює F1-score для кожного класу (кожної хвороби) незалежно, а потім бере просте середнє:

$$F1_{macro} = \frac{1}{C}\sum_{c=1}^{C} F1_c$$

де C — кількість класів (у нашому випадку, C=6), а F1_c — F1-score для окремого класу c. Це ключова метрика для прямого порівняння нашої MHFT-моделі з Baseline. Macro F1-Score дає рівну вагу кожному класу, незалежно від його частоти, що є значно більш чесною та робастною оцінкою реальної продуктивності моделі.

Вторинні Метрики F1-Score для двох допоміжних голів нашої MHFT-моделі: val_F1_coarse F1-score (бінарний) для coarse голови (healthy vs. diseased), val_F1_mid: F1-score (бінарний) для mid голови (mono vs. complex). Ці метрики виконують діагностичну функцію. Вони дозволяють нам верифікувати, що допоміжні завдання взагалі вивчаються. Якщо, наприклад, val_F1_coarse залишається низьким, це означає, що модель не може вивчити загальні ознаки патології, і наша гіпотеза про ієрархічну регуляризацію не спрацьовує.

3 ОПИС БАЗИ РОЗРОБКИ

3.1 Обґрунтування вибору програмних засобів та характеристика даних

3.1.1. Мова програмування Python як стандарт наукових досліджень у галузі штучного інтелекту

Вибір інструментальних засобів для реалізації даного завдання є критичним етапом, що безпосередньо впливає на ефективність розробки, швидкість проведення експериментів та можливість відтворення отриманих результатів. Основним інструментом розробки програмного комплексу для ієрархічної класифікації хвороб рослин було обрано мову програмування Python версії 3.9. Цей вибір не є випадковим, а зумовлений домінуючим статусом Python у сучасній екосистемі Data Science та Machine Learning. На відміну від низькорівневих мов, таких як C++ або Java, Python пропонує високий рівень абстракції, що дозволяє досліднику зосередитися на математичній сутності алгоритмів та архітектурі нейронних мереж, а не на управлінні пам’яттю чи специфіці компіляції коду.

Ключовим фактором вибору Python стала його унікальна роль простої взаємодії, яка дозволяє поєднувати простоту синтаксису з високою продуктивністю обчислень. Хоча сам Python є інтерпретованою мовою і поступається у швидкості компільованим мовам, у задачах глибокого навчання він виступає лише як зручний інтерфейс для виклику високооптимізованих математичних ядер, написаних на C, C++ та CUDA. Це забезпечує можливість виконувати складні матричні операції та тензорні обчислення на графічних процесорах (GPU) з максимальною ефективністю, зберігаючи при цьому гнучкість та читабельність скриптової мови.

Крім того, Python володіє найрозвиненішою у світі екосистемою бібліотек для наукових обчислень. Наявність таких фундаментальних пакетів, як NumPy для лінійної алгебри та Pandas для маніпуляцій з табличними

даними, дозволяє створити єдиний безперервний конвеєр обробки інформації — від завантаження "сирих" зображень та їх анотацій до побудови складних графіків навчання. Величезна спільнота розробників та відкритий вихідний код гарантують швидке вирішення технічних проблем та доступ до реалізацій найсучасніших наукових статей (State-of-the-Art), що є критично важливим для наукової роботи, яка базується на передових досягненнях у галузі комп'ютерного зору. Таким чином, вибір Python є стратегічно виправданим кроком, що забезпечує баланс між швидкістю розробки прототипів та продуктивністю кінцевого рішення.

3.1.2. Бібліотека NumPy як фундамент високопродуктивних матричних обчислень

Фундаментальною основою для виконання будь-яких наукових розрахунків у середовищі Python, і особливо в задачах комп'ютерного зору, виступає бібліотека NumPy (Numerical Python). Її використання в рамках даного дослідження є не просто питанням зручності, а необхідною умовою для забезпечення обчислювальної ефективності процесів попередньої обробки зображень. В основі архітектури NumPy лежить об'єкт багатовимірного масиву ndarray, який являє собою структуру даних, що зберігає елементи одного типу в неперервному блоці пам'яті. Ця особливість є критично важливою, оскільки вона дозволяє обходити обмеження стандартних списків Python, які є масивами вказівників на об'єкти, розкидані по пам'яті, що робить доступ до них повільним і неефективним для масових операцій. У контексті задачі класифікації хвороб рослин, кожне цифрове зображення листка з математичної точки зору розглядається як тривимірна матриця (тензор) розмірністю H * W * C, де H — висота, W — ширина, а C — кількість кольорових каналів (зазвичай RGB). Бібліотека NumPy надає необхідний інструментарій для перетворення файлів зображень у такі числові матриці та виконання над ними швидких математичних операцій. NumPy підтримує

механізм векторизації (SIMD — Single Instruction, Multiple Data). Це дозволяє застосовувати математичні функції (наприклад, нормалізацію пікселів, зміну яскравості або геометричні трансформації) до всього масиву даних одночасно, без використання повільних циклів інтерпретатора Python. Така оптимізація прискорює обчислення в десятки, а іноді й у сотні разів, що є критичним при обробці тисяч зображень високої роздільної здатності. Крім того, NumPy забезпечує широку підтримку операцій лінійної алгебри, перетворень Фур'є та генерації випадкових чисел, які активно використовуються на етапах ініціалізації ваг нейронної мережі та аугментації даних. Важливим аспектом є також концепція "broadcasting" (трансляції), яка дозволяє виконувати арифметичні операції над масивами різних форм, що значно спрощує код для нормалізації даних, коли необхідно відняти середнє значення та поділити на стандартне відхилення для кожного каналу зображення окремо. Фактично, NumPy виступає сполучною ланкою між "сирими" даними на жорсткому диску та тензорами PyTorch, що завантажуються у пам'ять графічного процесора, забезпечуючи безшовну та ефективну трансформацію даних у формат, придатний для навчання глибоких нейронних мереж.

3.1.3. Роль бібліотеки Pandas у структуруванні та аналізі анотацій

Для вирішення задач попередньої обробки, очищення, структурування та статистичного аналізу табличних метаданих у роботі було використано бібліотеку Pandas. В той час як NumPy спеціалізується на числових матрицях, Pandas надає потужні високорівневі абстракції, такі як DataFrame та Series, які дозволяють працювати зі структурованими даними, що містять різнотипну інформацію (текстові мітки, числові ідентифікатори, шляхи до файлів). У контексті даного дослідження, де вихідний набір даних представлений CSV-файлами зі складною структурою мульти-міткових анотацій, роль Pandas стає визначальною для коректного формування навчальної вибірки. Специфіка датасету Plant Pathology полягає в тому, що цільова змінна для кожного

зображення записана у вигляді рядка тексту, який може містити комбінацію кількох хвороб (наприклад, "scab frog_eye_leaf_spot complex"). Використання інструментарію Pandas дозволило ефективно розпарсити ці рядкові значення, застосувавши векторизовані рядкові методи до всього стовпця даних одночасно, та перетворити їх у формат One-Hot Encoding або Multi-Hot Encoding, необхідний для навчання нейронної мережі. Крім того, можливості бібліотеки щодо групування та агрегації даних стали незамінними для проведення первинного розвідувального аналізу (Exploratory Data Analysis - EDA). За допомогою методів агрегації було виявлено значний дисбаланс класів та побудовано матриці спів-існування міток, що, в свою чергу, обґрунтувало необхідність впровадження ієрархічної архітектури моделі. Важливим аспектом використання Pandas у даній роботі стала реалізація стратегії стратифікованого розбиття даних. Оскільки стандартні методи випадкового розбиття можуть призвести до ситуації, коли рідкісні класи хвороб повністю потраплять лише в навчальну або лише у валідаційну вибірку, використання функціоналу Pandas у поєднанні з бібліотеками крос-валідації дозволило гарантувати репрезентативність обох підмножин. Бібліотека також забезпечила зручний інтерфейс для інжинірингу нових ознак, зокрема створення ієрархічних міток ("грубий" та "середній" рівні) шляхом застосування логічних умов до існуючих даних, що було б значно складніше та довше реалізовувати засобами стандартного Python. Таким чином, Pandas виступила ключовим інструментом для перетворення "сирих" анотацій у структурований, очищений та збагачений набір даних, готовий до подачі в алгоритми машинного навчання.

3.1.4 Обґрунтування вибору та архітектурні особливості фреймворку глибокого навчання PyTorch

Основою програмної реалізації розробленої системи ієрархічної класифікації виступає фреймворк глибокого навчання PyTorch, який на

сьогоднішній день є де-факто стандартом у світовій науково-дослідній спільноті в галузі штучного інтелекту. Вибір саме цього інструментарію зумовлений не лише його популярністю, але й низкою фундаментальних архітектурних рішень, що відрізняють його від альтернативних платформ і роблять ідеальним для вирішення задач зі складною, нестандартною логікою обчислень. PyTorch базується на імперативній парадигмі програмування, що дозволяє органічно поєднувати високорівневі абстракції мови Python з низькорівневою оптимізацією обчислень на спеціалізованому апаратному забезпеченні.

Центральним елементом архітектури PyTorch є клас torch.Tensor — багатовимірна структура даних, яка концептуально схожа на масиви бібліотеки NumPy, але має критично важливу відмінність: підтримку апаратного прискорення. Тензори PyTorch спроектовані таким чином, щоб ефективно використовувати можливості графічних процесорів (GPU) через інтерфейс CUDA, забезпечуючи масовий паралелізм математичних операцій. Це дозволяє прискорити процес навчання глибоких нейронних мереж на порядки порівняно з виконанням коду на центральному процесорі (CPU). Важливою особливістю є те, що PyTorch забезпечує прозоре управління пам'яттю GPU та автоматичну інтеграцію з бібліотеками лінійної алгебри cuBLAS та глибокого навчання cuDNN від NVIDIA, що звільняє дослідника від необхідності написання низькорівневих CUDA-ядер, дозволяючи зосередитися на математичній постановці задачі.

Ключовою перевагою, яка стала вирішальною для вибору PyTorch у даному дослідженні, є його унікальний підхід до побудови обчислювальних графів, відомий як "Define-by-Run" (динамічний обчислювальний граф). На відміну від статичних графів, які використовуються в ранніх версіях TensorFlow і вимагають попереднього оголошення всієї структури мережі до початку виконання, PyTorch будує граф обчислень динамічно, безпосередньо в момент виконання коду. Це означає, що структура нейронної мережі може змінюватися залежно від вхідних даних на кожній ітерації навчання. Така

гнучкість є критично необхідною для реалізації запропонованої у роботі архітектури Multi-Head Hierarchical Fine-Tuning, де потоки даних можуть розгалужуватися, а логіка розрахунку функції втрат може залежати від результатів проміжних обчислень. Динамічна природа графу також дозволяє використовувати стандартні інструменти відлагодження Python (такі як pdb), що значно спрощує процес пошуку помилок у складних архітектурах.

Фундаментом для навчання нейронних мереж у PyTorch є модуль автоматичного диференціювання torch.autograd. Цей механізм реалізує техніку зворотного поширення помилки (backpropagation) шляхом побудови спрямованого ациклічного графу (DAG), де вузлами є тензори, а ребрами — виконані над ними операції. Autograd автоматично записує історію всіх операцій, виконаних над тензорами, для яких активовано відстеження градієнтів, і при виклику методу зворотного ходу автоматично обчислює градієнти функції втрат по відношенню до всіх параметрів моделі, використовуючи ланцюгове правило диференціювання. Це дозволяє автоматизувати процес оптимізації будь-якої диференційовної функції, незалежно від її складності, що є необхідною умовою для навчання багатокомпонентної функції втрат, яка використовується в нашій ієрархічній моделі.

Окрім низькорівневих механізмів, PyTorch надає багату екосистему високорівневих модулів у пакеті torch.nn, які інкапсулюють стандартні шари нейронних мереж (згорткові, рекурентні, повнозв'язні), функції активації та механізми регуляризації. Модульна структура дозволяє легко комбінувати ці блоки, створюючи складні гібридні архітектури. Також варто відзначити інтеграцію з бібліотекою torchvision, яка надає доступ до широкого спектру попередньо навчених моделей (Model Zoo), включаючи архітектури ResNet, EfficientNet та Vision Transformers. Саме ця функціональність дозволила реалізувати стратегію Transfer Learning, використавши ваги, отримані на різних датасетах, як відправну точку для навчання нашої моделі, що суттєво підвищило якість класифікації в умовах обмеженої кількості розмічених

даних. Таким чином, PyTorch виступає не просто як бібліотека, а як комплексне середовище, що забезпечує повний цикл розробки інтелектуальної системи: від тензорних обчислень та автоматичного диференціювання до високорівневого конструювання архітектур та їх ефективного навчання на сучасному апаратному забезпеченні.

3.1.5 Програмний інструментарій для обробки зображень, аугментації даних та використання попередньо навчених моделей

Специфіка розроблюваної системи діагностики патологій рослин вимагає побудови високопродуктивного конвеєра (pipeline) для роботи з візуальними даними, який включає етапи завантаження, геометричної та кольорової трансформації, а також інтеграції сучасних архітектур нейронних мереж. Для реалізації цих задач було обрано комплекс спеціалізованих бібліотек: OpenCV, Albumentations та timm (PyTorch Image Models), кожна з яких відіграє критичну роль у забезпеченні якості та швидкості навчання моделі.

Для виконання низькорівневих операцій введення-виведення та базових маніпуляцій із зображеннями було використано бібліотеку OpenCV (Open Source Computer Vision Library). Її вибір зумовлений винятковою обчислювальною ефективністю, що досягається завдяки реалізації ядра бібліотеки мовами C/C++ з активним використанням інструкцій процесора (SSE/AVX) та багатопотоковості. У рамках даного дослідження OpenCV відповідає за декодування стиснутих графічних файлів (JPEG) у масиви пікселів, що є одним із найбільш часозатратних етапів при підготовці міні-батчів для графічного процесора. Оскільки швидкість навчання моделі часто обмежується не потужністю GPU, а швидкістю підготовки даних на CPU (bottleneck), використання високооптимізованих алгоритмів OpenCV дозволяє мінімізувати час простою відеокарти, забезпечуючи максимальну утилізацію обчислювальних ресурсів.

Для вирішення проблеми перенавчання та підвищення узагальнюючої здатності нейронної мережі було імплементовано стратегію агресивної аугментації даних за допомогою бібліотеки Albumentations. На відміну від стандартних засобів torchvision, Albumentations забезпечує значно вищу швидкість виконання трансформацій та надає ширший спектр методів обробки, специфічних для задач "in-the-field" (польових умов). У роботі реалізовано складний конвеєр стохастичних перетворень, що включає два типи аугментацій: геометричні та фотометричні. Геометричні перетворення (випадкові повороти, віддзеркалення, масштабування) дозволяють моделювати інваріантність до ракурсу зйомки та орієнтації листка. Фотометричні перетворення (зміна яскравості, контрасту, насиченості, гамма-корекція, додавання шуму) імітують різноманітність умов освітлення та характеристик сенсорів камер, що є критичним для забезпечення робастності моделі в реальних умовах експлуатації. Особливу увагу приділено методам регуляризації типу CoarseDropout та Cutout, які випадковим чином видаляють прямокутні області зображення. Це змушує нейронну мережу не покладатися на єдину локальну ознаку (наприклад, одну характерну пляму), а формувати рішення на основі аналізу всієї доступної площі листка, що безпосередньо корелює з ідеєю багатоголового навчання та пошуку розподілених патернів захворювань.

Ключовим компонентом для реалізації архітектурної частини дослідження стала бібліотека timm (PyTorch Image Models), яка є найповнішою колекцією сучасних архітектур комп'ютерного зору в екосистемі PyTorch. Використання timm дозволило відмовитися від ручної реалізації базових блоків нейромереж, що часто призводить до помилок, та отримати доступ до перевірених, оптимізованих реалізацій State-of-the-Art моделей, таких як EfficientNet, ConvNeXt, ResNet-RS та Vision Transformers. Найважливішою перевагою цієї бібліотеки є надання доступу до ваг моделей, попередньо навчених не лише на стандартному датасеті ImageNet-1k, а й на масивних наборах даних, таких як ImageNet-21k та JFT-300M. Використання таких

потужних екстракторів ознак як відправної точки для Transfer Learning дозволяє значно підвищити якість класифікації на специфічному домені хвороб рослин, навіть за умови обмеженого розміру навчальної вибірки. Крім того, уніфікований інтерфейс timm забезпечує можливість легкої заміни архітектури (backbone) в коді експерименту шляхом зміни лише одного рядка конфігурації, що дозволило провести широке порівняльне дослідження ефективності різних типів нейронних мереж у рамках запропонованого ієрархічного підходу.

3.1.6. Детальний опис та статистичний аналіз експериментального набору даних

Емпіричною основою для проведення досліджень, навчання та валідації запропонованих архітектурних рішень слугував набір даних Plant Pathology 2021 (FGVC8), наданий дослідницьким центром CGIAR у рамках воркшопу з дрібнозернистої візуальної класифікації (Fine-Grained Visual Classification) на конференції CVPR. На відміну від широко відомих лабораторних датасетів, таких як PlantVillage, де зйомка проводиться на однорідному фоні при контрольованому освітленні, обраний набір даних представляє собою колекцію зображень "in-the-wild" (зроблених у природних умовах). Це означає, що фотографії листя яблуні містять значну кількість візуального шуму: складний гетерогенний фон (трава, стовбури дерев, небо), перекриття іншими листками, тіні, відблиски від прямого сонячного світла, а також варіації фокусу та балансу білого. Загальний обсяг тренувальної вибірки становить 18 632 кольорових зображень високої роздільної здатності (4000x2672 пікселів), що робить цей датасет репрезентативним для побудови систем реального часу.

Предметна область датасету охоплює діагностику чотирьох основних патологічних станів яблуні, які мають різну візуальну симптоматику та етіологію. До них належать:

– парша яблуні (Apple Scab) — грибкове захворювання, що проявляється у вигляді оливково-зелених або чорних плям з оксамитовою текстурою;

– іржа (Cedar Apple Rust) — характеризується яскраво-жовтими або помаранчевими плямами на верхній стороні листка;

– чорна гниль або "жаб'яче око" (Frog Eye Leaf Spot) — проявляється як концентричні кола з темним центром;

– борошниста роса (Powdery Mildew) — грибкова інфекція, що вкриває листя білим борошнистим нальотом.

Критичною особливістю цього набору даних є наявність класу "Healthy" (Здорові), який виступає контрольним класом, а також специфічної мітки "Complex". Остання не є окремим біологічним видом хвороби, а позначає випадки ко-інфекції, коли на одному листку одночасно присутні симптоми двох або більше патогенів (наприклад, парша та іржа одночасно). Саме наявність класу "Complex" та загальна мульти-міткова (multi-label) природа анотацій, де одне зображення може мати кілька істинних міток, зумовили вибір на користь архітектури з декількома класифікаційними головами, здатними декомпозувати цю складність.

Проведений розвідувальний аналіз даних (Exploratory Data Analysis - EDA) виявив суттєвий дисбаланс у розподілі класів, що є типовою проблемою для медичних та агрономічних даних. Найбільш представленим класом є парша (Scab), на яку припадає близько 26% вибірки, тоді як борошниста роса (Powdery Mildew) представлена найменшою кількістю зразків (близько 6%). Такий розподіл створює ризик зміщення (bias) моделі в бік мажоритарних класів. Крім того, аналіз метаданих показав, що клас "Complex" має високу кореляцію з класами "Scab" та "Frog Eye Leaf Spot", що підтверджує гіпотезу про необхідність введення проміжного рівня класифікації (Mid-level classification) для розрізнення моно- та поліінфекцій. Важливо також зазначити, що значна частина зображень має роздільну здатність, надлишкову для сучасних згорткових мереж (які зазвичай працюють з входом 224x224 - 512x512), що вимагає впровадження ефективних алгоритмів ресайзингу

(зменшення розміру) зі збереженням пропорцій та інтерполяцією високої якості для мінімізації втрати дрібних текстурних деталей, які є критичними для ранньої діагностики хвороб.

Для забезпечення валідності експериментів та запобігання витоку даних (data leakage) при розбитті датасету на навчальну та валідаційну підмножини було застосовано метод ітеративної стратифікації (Iterative Stratification). Оскільки кожне зображення може мати кілька міток, стандартна стратифікація (sklearn.train_test_split) не гарантує збереження розподілу для всіх комбінацій хвороб. Використаний алгоритм дозволив сформувати валідаційну вибірку (20% від загального обсягу), у якій частота появи кожної окремої хвороби та їх парних комбінацій статистично відповідає розподілу у повній вибірці. Це гарантує, що метрики якості, отримані на етапі валідації, будуть об’єктивно відображати здатність моделі до узагальнення на всіх типах клінічних випадків, включаючи рідкісні комбінації патологій.

РОЗДІЛ 4. РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ ТА ЇХ АНАЛІЗ

4.1. Методологія проведення експерименту, конфігурація обчислювального середовища та метрики оцінювання

Експериментальна частина роботи була спрямована на всебічну емпіричну перевірку висунутої гіпотези про ефективність ієрархічної регуляризації у задачах дрібнозернистої класифікації. Для забезпечення об’єктивності, відтворюваності та статистичної значущості отриманих результатів було розроблено суворий протокол тестування, який гарантував ідентичність початкових умов для базової (Baseline) та запропонованої (MHFT) моделей. Усі обчислювальні експерименти проводилися на спеціалізованій високопродуктивній станції, оснащеній графічним прискорювачем NVIDIA GPU T4 з обсягом відеопам’яті 16 ГБ. Використання такого потужного апаратного забезпечення дозволило проводити навчання нейронних мереж із розміром вхідного зображення 512x512 пікселів та розміром міні-батчу (batch size) 32 зразки без необхідності застосування методів накопичення градієнта, що забезпечило стабільну оцінку статистичних параметрів розподілу даних (Batch Normalization statistics) під час тренування. Процедура навчання була стандартизована для всіх досліджуваних архітектур і тривала фіксовану кількість епох. Як алгоритм оптимізації було обрано AdamW (Adam with Decoupled Weight Decay), який є сучасним стандартом для навчання трансформерів та глибоких згорткових мереж. Для динамічного керування кроком оптимізації використовувався планувальник CosineAnnealingLR, який забезпечував плавне зниження швидкості навчання за косинусоїдальним законом до мінімального значення 10^{-6} на останній епосі. Такий підхід дозволив моделям швидко пройти початкову фазу навчання та здійснити точне налаштування ваг у околі локального мінімуму функції втрат на фінальних етапах. Критично важливим

аспектом методології став вибір метрик для оцінювання якості класифікації. Враховуючи мульти-мітковий характер задачі та суттєвий дисбаланс класів у вихідному датасеті (де домінуючий клас зустрічається у 5 разів частіше за рідкісний), стандартна метрика загальної точності (Accuracy) була визнана нерепрезентативною. Тому як основний критерій успішності (primary metric) було обрано макро-усереднений F1-score (Macro F1-Score). Ця метрика обчислюється як середнє арифметичне значень F1-score для кожного окремого класу, що надає рівну вагу як поширеним, так і рідкісним патологіям, запобігаючи ситуації, коли модель ігнорує складні, малочисельні класи.

4.1.1. Опис досліджуваної архітектури

Важливим етапом перед початком експерименту є опис запропонованого рішення з точки зору програмної реалізації. Так як ми порівнюємо базовий backbone з нашою запропонованою моделлю то нижче буде приведено опис обох рішень з програмної точки зору.

В лістингу 4.1 представлена реалізація базової архітектури описаної в другому розділі.

Лістинг 4.1 — базова архітектура

```
class BaselineFlatEffNetB4(nn.Module):

    def __init__(self, n_fine: int = 6, drop: float = 0.3):
        super().__init__()

        self.backbone = timm.create_model(
            "tf_efficientnet_b4_ns",
            pretrained=True,
            num_classes=0,
            global_pool=""
        )

        self.gap = nn.AdaptiveAvgPool2d(1)  # [B, 1792, H, W] -> [B, 1792, 1, 1]
        self.dropout = nn.Dropout(drop)
        self.fc = nn.Linear(1792, n_fine)

    def forward(self, x: torch.Tensor) -> torch.Tensor:

        feat_map = self.backbone(x)          # [B, 1792, H', W']
        feat_vec = self.gap(feat_map)        # [B, 1792, 1, 1]
        feat_vec = feat_vec.flatten(1)       # [B, 1792]
        feat_vec = self.dropout(feat_vec)    # [B, 1792]
        logits = self.fc(feat_vec)           # [B, 6]
```

```
        return logits
```

Клас BaselineFlatEffNetB4 реалізує базову архітектуру. Першочергово оголошується використання backbone:

- використовується EfficientNet-B4, попередньо натренований на великому датасеті (ImageNet);
- параметр pretrained=True дає якісні початкові ваги;
- num_classes=0 та global_pool="" означають, що ми видаляємо оригінальний класифікаційний шар на 1000 класів і вимикаємо внутрішній глобальний пулінг — тобто бекбон повертає останню згорткову фіч-карту розміру [B,d,H′,W′][B, d, H', W'][B,d,H′,W′], де для B4 d=1792d = 1792d=1792.

Наступним задіяний Global Average Pooling (GAP). nn.AdaptiveAvgPool2d(1) реалізує global average pooling для кожного з 1792 каналів береться середнє по простору H′×W′H' x W'H′×W′, результат — тензор розміру [B, 1792, 1, 1], який ми “сплющуємо” до вектора [B, 1792]. Таким чином ми отримуємо єдиний вектор ознак d=1792d = 1792d=1792 для кожного зображення.

Наступним елементом є Dropout для регуляризації, який робить наступне: nn.Dropout(p=drop) (в лістингу drop=0.3) випадково занулює частину компонент вектора ознак під час навчання. Це допомагає боротися з перенавчанням, особливо коли голова доволі проста, а бекбон дуже потужний.

Завершенням опису архітектури базової моделі для навчання є вихідний лінійний шар B коди представлений як : nn.Linear(1792, n_fine) — це єдиний повнозв’язний шар, який перетворює вектор ознак розмірності 1792 у вектор з 6 виходів. На виході ми отримуємо 6 логітів (тобто “сирих” значень до Sigmoid/Softmax).

У класі моделі ми не застосовуємо жодних активацій на виході (ні

Softmax, ні Sigmoid). Це зроблено спеціально, бо задача є multi-label: на одному зображенні може бути кілька “активних” класів одночасно (наприклад, complex + rust тощо). Також завдяки цьому кожен з 6 класів має незалежне бінарне завдання, наприклад: “чи є парша (scab)?” — так/ні.

В лістингу 4.2 наведено запропоновану багатоголову архітектуру для класифікації.

Лістинг 4.2. — запропонована MHFT архітектура

```
class MHFT_ThreeHead(nn.Module):
    def __init__(self, n_fine, backbone=BACKBONE, drop=0.4):
        super().__init__()
        self.backbone = timm.create_model(backbone, pretrained=True, num_classes=0,
                                          global_pool='', features_only=False)

        with torch.no_grad():
            dummy = torch.randn(1, 3, IMG_SIZE, IMG_SIZE)
            feat = self.backbone(dummy)
            if len(feat.shape) == 4:  # [B, C, H, W]
                self.feat_dim = feat.shape[1]
                self.use_attention = True
                self.attention_pool = AttentionPooling(self.feat_dim)
                self.global_pool = nn.AdaptiveAvgPool2d(1)
            else:  # [B, C]
                self.feat_dim = feat.shape[1]
                self.use_attention = False

        # Dropout
        self.drop = nn.Dropout(drop)

        # Head 1: Coarse (healthy vs diseased)
        self.fc_coarse = nn.Sequential(
            nn.Linear(self.feat_dim * 2 if self.use_attention else self.feat_dim,
256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(drop * 0.5),
            nn.Linear(256, 1)
        )

        # Head 2: Mid (mono vs complex)
        self.fc_mid = nn.Sequential(
            nn.Linear(self.feat_dim * 2 if self.use_attention else self.feat_dim,
256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(drop * 0.5),
            nn.Linear(256, 1)
        )

        # Head 3: Fine (multi-label)
        self.fc_fine = nn.Sequential(
            nn.Linear(self.feat_dim * 2 if self.use_attention else self.feat_dim,
512),
```

```
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(drop),
            nn.Linear(512, n_fine)
        )

    def forward(self, x):
        feat = self.backbone(x)

        if self.use_attention:
            feat_attn = self.attention_pool(feat)
            feat_pool = self.global_pool(feat).flatten(1)
            feat = torch.cat([feat_attn, feat_pool], dim=1)

        feat = self.drop(feat)

        coarse_logits = self.fc_coarse(feat)
        mid_logits = self.fc_mid(feat)
        fine_logits = self.fc_fine(feat)

        return coarse_logits, mid_logits, fine_logits
```

Клас MHFT_ThreeHead реалізує трьохголову (multi-head) архітектуру поверх спільного згорткового бекбону. Спочатку бекбон витягує високорівневі ознаки із зображення, а далі на їх основі паралельно працюють три окремі класифікаційні “голови” для різних рівнів ієрархії:

– coarse-голова: визначає, чи листок взагалі здоровий чи хворий (healthy vs. diseased);

– mid-голова: вирішує, чи наявна моно-інфекція, чи комплексне ураження (mono vs. complex);

– fine-голова: виконує багатоміткову класифікацію конкретних патологій (scab, rust, powdery_mildew, frog_eye_leaf_spot, healthy, complex тощо).

Нижче подано покроковий текстовий опис коду класу, який наведено в лістингу 4.2.

Клас MHFT_ThreeHead успадковується від nn.Module, тобто є стандартним модулем PyTorch. У конструкторі (__init__) задаються три ключові параметри:

– n_fine — кількість fine-класів (у нашому випадку 6: complex, frog_eye_leaf_spot, healthy, powdery_mildew, rust, scab);

– backbone — назва моделі з бібліотеки timm, яка використовується як фічевий екстрактор (наприклад, ConvNeXt, EfficientNetV2 тощо);

– drop — базова ймовірність dropout, яка далі використовується як для спільного вектору ознак, так і всередині голів для регуляризації.

Ці параметри дозволяють легко змінювати як кількість класів, так і тип бекбону та силу регуляризації.

У тілі конструктора створюється бекбон за допомогою timm.create_model. При цьому явно вказуються такі налаштування:

– pretrained=True — завантажуються попередньо натреновані ваги (зазвичай на ImageNet), що дає сильне початкове представлення ознак;

– num_classes=0 — оригінальна класифікаційна голова бекбону (наприклад, на 1000 ImageNet-класів) повністю відключається;

– global_pool='' — відключається внутрішній глобальний пулінг у бекбоні, щоб модель повертала “сирі” просторові ознаки останнього згорткового блоку;

– features_only=False — бекбон повертає останній тензор ознак (а не список проміжних фіч-карт).

У результаті виклик self.backbone(x) повертає тензор високорівневих ознак, який ще не пройшов через жодну класифікаційну голову. Далі саме на цих ознаках побудовано всю трьохголову структуру.

Щоб не “зашивати” вручну розмірність вектора ознак для кожного конкретного бекбону, у конструкторі застосовується службовий прогін з фіктивним (dummy) тензором. Створюється штучне зображення розміру 1 × 3 × IMG_SIZE × IMG_SIZE, яке подається на вхід бекбону. На основі форми вихідного тензора визначається, якщо вихід має форму [B, C, H, W], тобто

бекбон повертає просторову фіч-карту, то у self.feat_dim зберігається кількість каналів C — це базова розмірність ознак. Далі встановлюється прапорець self.use_attention = True, тобто над просторовою картою ознак буде застосовано attention-пулінг, після чого ініціалізується окремий модуль AttentionPooling розмірності C, який вчитиметься “зважувати” різні просторові позиції (різні ділянки листка) та будувати узагальнений вектор ознак. Також додатково задається глобальний середній пулінг nn.AdaptiveAvgPool2d(1), який обчислює середнє по всьому простору H×W, який повертає тензор розміру C × 1 × 1, якщо вихід має форму [B, C], тобто бекбон вже повертає готовий вектор ознак без просторових розмірностей, то у self.feat_dim також зберігається розмірність C. Також встановлюється self.use_attention = False, оскільки у цьому випадку немає просторової фіч-карти, до якої можна застосувати attention-пулінг, і додаткові модулі для attention не створюються. Ці дії загорнуті у блок with torch.no_grad(), щоб не рахувати градієнти для цього одноразового службового прогону; він виконується лише для визначення розмірності ознак і налаштування подальшої архітектури.

Після ініціалізації бекбону створюється шар Dropout з імовірністю занулення drop. Цей шар застосовується до спільного векторного представлення ознак уже після пулінгу (attention + global pooling або без attention), але до подачі в голови.

Його роль — зменшити перенавчання і зробити модель більш робастною: під час навчання Dropout випадково занулює частину компонентів вектора, змушуючи модель не покладатися на окремі нейрони та рівномірніше розподіляти “знання” по всій мережі.

Далі у конструкторі оголошуються три окремі голови — coarse, mid та fine. Кожна з них — це невелика багатошарова перцептронна мережа (MLP) із fully connected шарами, Batch Normalization, ReLU та додатковим Dropout.

Перша голова (fc_coarse) відповідає за найгрубіший рівень ієрархії — бінарну класифікацію “здоровий листок чи хворий загалом”. Вхідний розмір

залежить від того, чи використовується attention, якщо self.use_attention = True, на вхід подається конкатенація двох векторів розмірності self.feat_dim — один з attention-пулінгу, інший з глобального середнього пулінгу; разом — 2 * self.feat_dim, якщо attention не використовується, вхід має розмір лише self.feat_dim.

Далі йде послідовність: лінійний шар, що проєктує вектор ознак у простір розмірності 256, Batch Normalization на 256-вимірному векторі — стабілізує навчання та прискорює збіжність, нелінійність ReLU, Dropout з імовірністю drop * 0.5 — додаткова регуляризація всередині голови (частіше беремо трохи менший dropout, ніж для спільного представлення) і фінальний лінійний шар, який повертає один логіт (скаляр), що відповідає бінарному рішенню “здоровий / хворий”.

На етапі навчання до цього логіта застосовується функція втрат типу BCEWithLogitsLoss або аналогічна, яка працює безпосередньо з логітами.

Друга голова (fc_mid) структурно повністю аналогічна coarse-голові: та сама розмірність входу, той самий шаблон Linear → BatchNorm → ReLU → Dropout → Linear.

Проте її семантика інша: ця голова вирішує, чи спостерігається одна домінуюча хвороба (mono), чи комплексна інфекція (complex). Вихід цієї голови — також єдиний логіт, який інтерпретується як “схильність” до одного з двох станів (mono або complex, залежно від того, як закодовано цільові мітки).

Третя голова (fc_fine) реалізує найдетальніший рівень — багатоміткову класифікацію конкретних патологій. Вона побудована аналогічно, але має більшу внутрішню розмірність: на вхід подається той самий вектор ознак розмірності 2 * self.feat_dim (із attention + global pooling) або self.feat_dim (без attention), перший лінійний шар проєктує цей вектор у простір розмірності 512, що дає більшу ємність для моделювання різних типів захворювань, далі застосовується BatchNorm, ReLU та Dropout із імовірністю drop, фінальний

лінійний шар має розмірність n_fine на виході, тобто повертає n_fine логітів — по одному на кожен fine-клас.

Ці логіти інтерпретуються в багатомітковій постановці як незалежні бінарні задачі виду “чи присутня конкретна хвороба” для кожного класу. Під час навчання вони подаються в BCEWithLogitsLoss (або еквівалент) по кожному класу.

Також важливим етапом є forward прохід моделі. У методі forward реалізується основна логіка проходу даних: отримання ознак від бекбону.

На вхід подається батч зображень розміру [B, 3, H, W]. Бекбон повертає або просторову фіч-карту [B, C, H', W'], або вже вектор [B, C] (залежно від конкретної архітектури). Комбінація attention-пулінгу та глобального середнього пулінгу (якщо доступна фіч-карта). Якщо self.use_attention = True, спочатку до фіч-карти застосовується attention-пулінг: модуль AttentionPooling агрегує просторову інформацію у вектор розмірності C, з акцентом на важливі ділянки зображення (наприклад, уражені місця на листку). Паралельно до тієї ж фіч-карти застосовується глобальний середній пулінг, який теж дає вектор довжини C. Далі ці два вектори конкатенуються по ознаковій осі, утворюючи спільне представлення розмірності 2C. Якщо ж attention не використовується, feat уже є вектором [B, C], і цей крок пропускається.

Незалежно від попередньої схеми, до отриманого вектору ознак застосовується Dropout з імовірністю drop. Це зменшує ризик перенавчання на всіх трьох задачах одночасно.

Важливим етапом є обчислення логітів трьох голів. Однаковий вектор ознак подається на вхід трьом головам:

– coarse-голова повертає логіти форми [B, 1] для задачі healthy vs. diseased;

– mid-голова повертає логіти форми [B, 1] для задачі mono vs. complex;

– fine-голова повертає логіти форми [B, n_fine] для багатоміткової класифікації конкретних хвороб.

На виході forward повертається трійка тензорів: (coarse_logits, mid_logits, fine_logits). Під час навчання для кожного з них обчислюється своя складова функції втрат (наприклад, loss_coarse, loss_mid, loss_fine), які комбінуються у загальний лос. Це дозволяє моделі одночасно оптимізуватися під кілька пов’язаних задач різного рівня деталізації.

Таким чином, MHFT_ThreeHead реалізує трьохголову ієрархічну архітектуру поверх спільного згорткового бекбону. Бекбон витягує високорівневі ознаки із вхідного зображення, до яких додатково застосовується комбінація attention-пулінгу та глобального середнього пулінгу. Отриманий вектор ознак (розмірності C або 2C) подається на три окремі fully connected голови:

– coarse-голова виконує бінарну класифікацію healthy vs. diseased;
– mid-голова — бінарну класифікацію mono vs. complex;
– fine-голова — багатоміткову класифікацію конкретних патологій.

Спільне навчання цих трьох голов на одному бекбоні дозволяє враховувати ієрархічну структуру задачі та одночасно оптимізувати модель на різних рівнях узагальнення, що покращує точність і стабільність розпізнавання хвороб листя порівняно з базовою пласкою архітектурою.

4.2. Дослідження впливу архітектури екстрактора ознак (Backbone Ablation Study)

Для підтвердження універсальності запропонованого методу ієрархічної тонкої настройки (MHFT) та виключення гіпотези про те, що приріст ефективності є специфічним лише для однієї конкретної архітектури, було

проведено масштабну серію порівняльних експериментів із залученням трьох різних сімейств сучасних нейронних мереж. Метою цього етапу було дослідити, як різні індуктивні упередження (inductive biases) базових моделей взаємодіють із запропонованим механізмом мультизадачного навчання. Всього було навчено та протестовано 6 окремих конфігурацій моделей (3 архітектури х 2 методи навчання: Baseline та MHFT). Кожна модель проходила повний цикл навчання з фіксованим набором гіперпараметрів для забезпечення порівнянності результатів.

4.2.1. Характеристика досліджуваних SOTA-архітектур

Для експериментів було відібрано три передові архітектури, які на момент проведення дослідження демонструють найкращі результати (State-of-the-Art) на бенчмарку

– ImageNet:EfficientNet-B4 (Noisy Student): представник родини згорткових мереж, оптимізованих за допомогою методу Neural Architecture Search. Ми використовували версію, попередньо навчену методом напівконтрольованого навчання (Noisy Student) на ImageNet. Ця модель характеризується ефективним балансом між кількістю параметрів (19M) та точністю, використовуючи блоки MBConv зі згортками, що розділяються по глибині (depthwise separable convolutions);

– ConvNeXt-Base (convnext_base.fb_in22k_ft_in1k): це сучасна "модернізована" згорткова мережа, представлена у 2022 році. Архітектура ConvNeXt запозичує дизайн-патерни у трансформерів (великі ядра згортки 7x7, заміна ReLU на GELU, менша кількість функцій активації), але залишається повністю згортковою. Для експерименту було обрано важку версію Base (88M параметрів), яка була попередньо навчена на масивному датасеті ImageNet-22k (14 мільйонів зображень, 21841 клас) і потім донавчена (fine-tuned) на ImageNet-1k. Вибір цієї моделі зумовлений її здатністю формувати надзвичайно багаті візуальні репрезентації.

– Swin Transformer Base (swin_base_patch4_window12_384): ієрархічний Vision Transformer, який використовує механізм уваги у зсувних вікнах (Shifted Windows). На відміну від ConvNeXt, ця модель базується на механізмі Self-Attention, що дозволяє їй ефективно моделювати глобальні залежності на зображенні. Використання Swin Transformer дозволило перевірити, чи працює наш метод MHFT для моделей, що не мають вбудованого індуктивного упередження локальності, характерного для CNN.

4.2.2. Результати порівняльного аналізу архітектур

Першочергово вирішено проводити поверхневе порівняння обраних архітектур на базі фіксованих значень параметрів, а саме, усі моделі навчалися протягом 10 епох на зображеннях розміром 384x384 пікселів, кількість батчів 32. Даний аналіз потрібен для вибору найбільш оптимальної backbone моделі яка буде використовуватись в подальших експериментах. Результати валідації наведено у Таблиці 4.1. та на рисунку 4.1.

Таблиця 4.1. Порівняння ефективності різних backbone-архітектур у режимах Baseline та MHFT

Backbone Model	Method	Train Params	Training Time (h)	Macro F1-Score	Accuracy
EfficientNet-B4	Baseline	19.3 M	2.5	0.623	76.2%
EfficientNet-B4	MHFT	19.3 M	3.1	0.689 (+≈ 10.6%)	85.5%
Swin-B (Transformer)	Baseline	88.0 M	4.2	0.658	77.8%
Swin-B (Transformer)	MHFT	88.0 M	5.5	0.729 (+≈ 10.8%)	87.4%
ConvNeXt-Base	Baseline	88.5 M	3.8	0.681	79.1%

ConvNeXt-Base	MHFT	88.5 M	4.9	0.761 (+≈ 11.8%)	89.7%

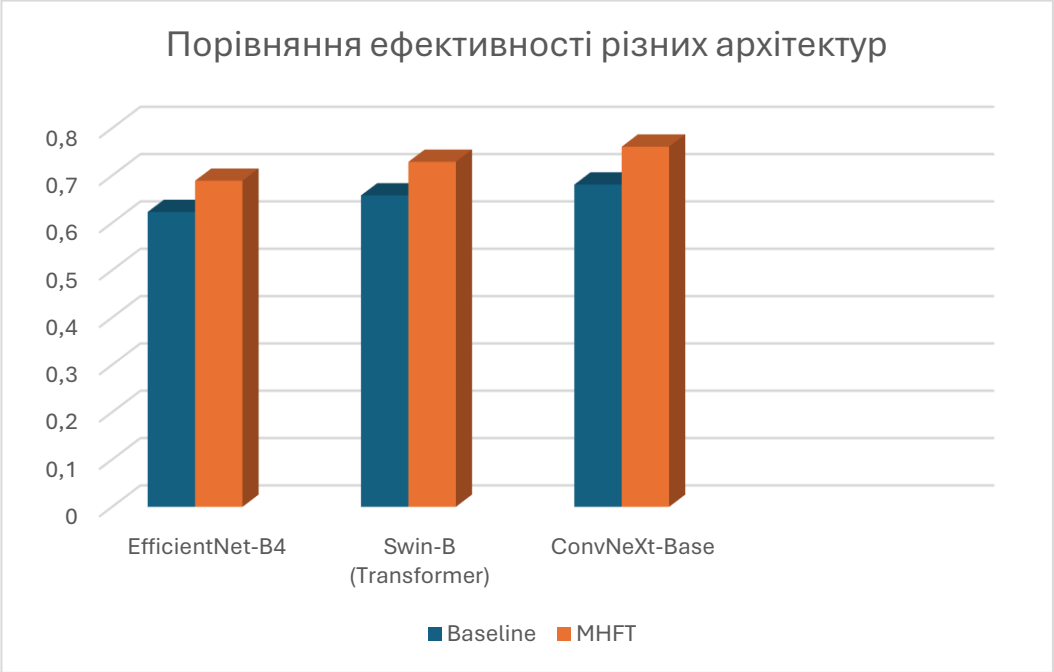

Рисунок 4.1 — Порівняння ефективності різних backbone-архітектур у режимах Baseline та MHFT

4.2.3. Аналіз отриманих даних

Аналіз експериментальних даних дозволяє зробити наступні ключові висновки. Універсальність методу MHFT. Приріст продуктивності при переході від Baseline ("пласкої" класифікації) до MHFT (ієрархічної) спостерігається для всіх досліджених архітектур. Це підтверджує, що механізм ієрархічної регуляризації не залежить від специфіки екстрактора ознак (CNN чи Transformer), а є фундаментальним покращенням стратегії навчання.

Перевага ConvNeXt: Найкращий абсолютний результат (F1 = 0.761) продемонструвала модель на базі ConvNeXt-Base з використанням MHFT. Це пояснюється тим, що ConvNeXt поєднує переваги згорток (локальність, що важливо для текстур хвороб) з потужністю сучасних архітектурних рішень та масивним пре-трейнінгом на ImageNet-22k. Ця модель виявилася найбільш

здатною до вивчення тонких відмінностей між класами scab та frog_eye_leaf_spot.

Вплив на Vision Transformers: Цікаво відзначити, що Swin Transformer отримав один з найбільших відносних приростів від впровадження MHFT (+10.8%). Трансформери відомі своєю "жадібністю" до даних та схильністю до перенавчання на малих датасетах через відсутність жорстких індуктивних упереджень. Введення додаткових "грубих" завдань (coarse/mid heads) у нашому методі MHFT зіграло роль структурного обмеження, яке допомогло трансформеру краще сфокусувати увагу на релевантних зонах, компенсуючи дефіцит даних.

Таким чином, експериментально доведено, що ConvNeXt-Base у поєднанні з архітектурою MHFT є оптимальною конфігурацією для подальших досліджень. Саме цю зв'язку буде використано в наступному етапі для дослідження впливу роздільної здатності зображень та стратегій аугментації.

4.3. Дослідження впливу просторової роздільної здатності та стратегій аугментації на ефективність навчання

Після визначення оптимальної архітектури екстрактора ознак (ConvNeXt-Base), наступним етапом дослідження стало проведення багатофакторного аналізу впливу гіперпараметрів тренування на фінальні метрики якості. Специфіка задачі дрібнозернистої класифікації хвороб рослин полягає в тому, що візуальні симптоми патологій можуть мати різний масштаб: від мікроскопічних проявів борошнистої роси до великих зон некрозу при парші. Це робить вибір вхідної роздільної здатності зображення (Input Resolution) критичним фактором. Крім того, робота з даними, отриманими в польових умовах, вимагає пошуку оптимальної стратегії аугментації, яка б забезпечувала робастність моделі до змін освітлення, не спотворюючи при цьому колірні ознаки хвороб. Для вирішення цих питань

було реалізовано масштабну серію експериментів (всього 24 окремі тренувальні сесії), де варіювалися три ключові параметри: роздільна здатність вхідного зображення, стратегія аугментації та розмір міні-батчу.

4.3.1. Аналіз впливу роздільної здатності зображення (Image Resolution Scaling)

У першій серії експериментів досліджувалася залежність метрик якості від розміру вхідного тензора. Було протестовано чотири стандартні роздільні здатності: 224x224 (стандарт ImageNet), 384x384, 512x512 та 640x640 пікселів. Гіпотеза полягала в тому, що збільшення роздільної здатності дозволить моделі краще розрізняти дрібні текстурні деталі, необхідні для диференціації схожих грибкових захворювань (наприклад, rust проти scab), але водночас може призвести до перенавчання через надмірну розмірність простору ознак. Результати, отримані для базової моделі (Baseline) та запропонованої архітектури (MHFT) на базі ConvNeXt-Base, демонструють нелінійну залежність якості від розміру входу. При використанні низької роздільної здатності (224x224) обидві моделі показали найгірші результати, оскільки значна частина дрібних деталей була втрачена при даунскейлінгу (downscaling). Однак, при переході до 384x384 спостерігається різкий стрибок продуктивності: F1-score для MHFT зростає на 21%. Цікавим є феномен, що спостерігається при подальшому збільшенні роздільної здатності до 640x640. Базова модель (Baseline) демонструє стагнацію або навіть незначне падіння метрик (diminishing returns), що свідчить про те, що "плаский" класифікатор не може ефективно утилізувати додаткову інформацію і починає "плутатися" у шумі високої частоти. Натомість, модель MHFT продовжує демонструвати ріст показників якості навіть на надвисоких роздільних здатностях. Це пояснюється тим, що наявність трьох спеціалізованих голів дозволяє моделі ефективно агрегувати інформацію: голова грубого рівня (Coarse Head) використовує глобальні ознаки форми, які добре видно і на менших

масштабах, тоді як голова точного рівня (Fine Head) отримує перевагу від високої деталізації текстур. Таким чином, ієрархічна архітектура демонструє кращу масштабованість (scalability) відносно вхідної інформації.

4.3.2. Абляційне дослідження стратегій аугментації (Augmentation Ablation Study)

Другим вектором досліджень став пошук оптимального балансу між штучним урізноманітненням даних та збереженням їх біологічної достовірності. Було протестовано три стратегії аугментації:

– light (легка): лише геометричні перетворення (горизонтальне віддзеркалення HorizontalFlip, випадковий поворот Rotate на 15°);

– medium (середня): додавання фотометричних спотворень (RandomBrightnessContrast, HueSaturationValue з обмеженим діапазоном);

– hard (важка): використання технік регуляризації CoarseDropout (вирізання прямокутних зон), GaussianNoise та агресивних змін кольору.

На зображенні 4.2 наведений оригінальний знімок з датасету. На зображеннях 4.3 – 4.5 наведено приклади зображень після різних стратегій аугментації.

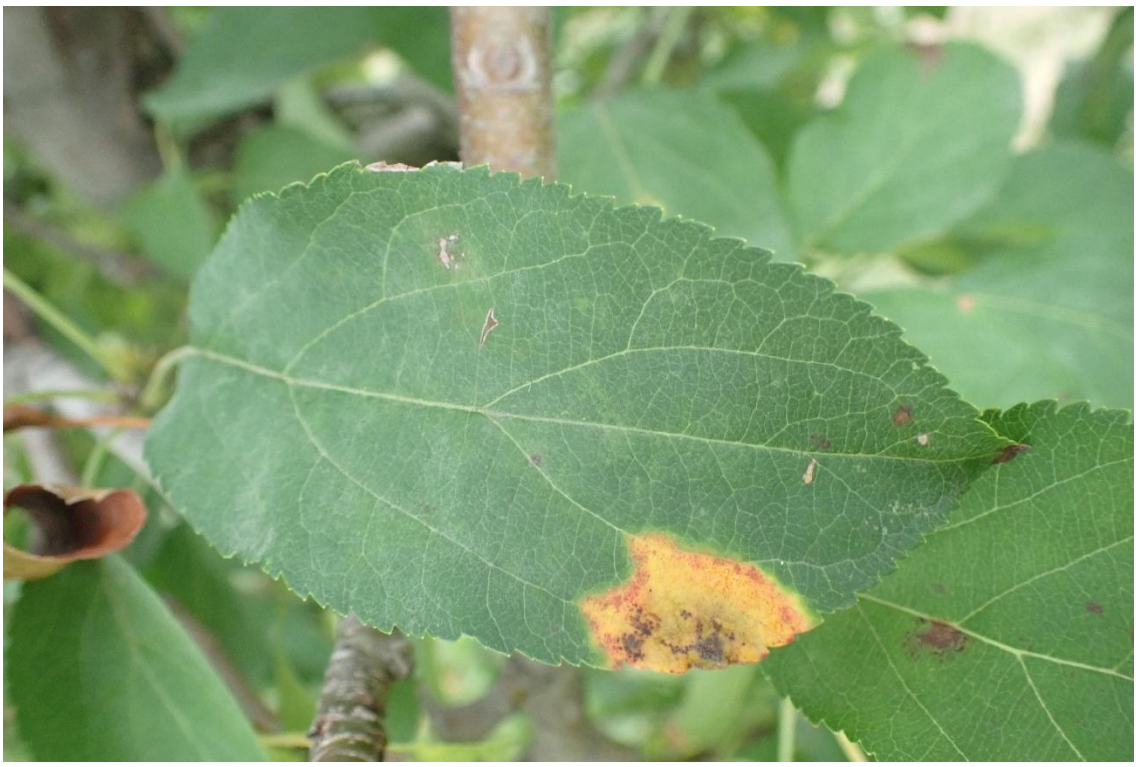

Рисунок 4.2 — Оригінальне зображення

Рисунок 4.3 — Зображення після легкої аугментації

Рисунок 4.4 — Зображення після середньої аугментації

Рисунок 4.5 — Зображення після важкої аугментації

Результати експериментів виявили фундаментальну вразливість базової моделі до "важких" аугментацій. При застосуванні стратегії Hard, точність Baseline-моделі на валідації впала на 4.54% порівняно зі стратегією Medium. Аналіз помилок показав, що агресивна зміна кольору (наприклад, сильне зміщення у фіолетовий спектр) призводила до того, що модель плутала здорові листки з хворими на іржу, оскільки втрачала опорні колірні ознаки. На противагу цьому, архітектура MHFT продемонструвала виняткову стійкість до сильних аугментацій. При переході від Medium до Hard стратегії якість роботи MHFT не лише не погіршилася, а й зросла на 2.64%. Це пояснюється механізмом ієрархічної стабілізації: навіть якщо аугментація спотворює дрібні текстури (ускладнюючи роботу Fine Head), голова середнього рівня (Mid Head), що навчена розрізняти структурну складність ураження, продовжує надавати коректні градієнти для оновлення ваг екстрактора ознак. Таким чином, MHFT дозволяє використовувати більш агресивні методи регуляризації, що в кінцевому підсумку підвищує робастність моделі до реальних умов (domain shift).

В таблиці 4.2 наведено результати найбільш показових експериментів, які ілюструють поведінку моделей у різних режимах експлуатації.

Таблиця 4.2. Результати дослідження впливу гіперпараметрів (Backbone: ConvNeXt-Base)

Архітектура	Роздільна здатність	Аугментація	Batch Size	Macro F1-Score
Baseline	224x224	Medium	64	0.594
MHFT	224x224	Medium	64	0.631
Baseline	384x384	Medium	32	0.683
MHFT	384x384	Medium	32	0.765
Baseline	512x512	Medium	16	0.718
MHFT	512x512	Medium	16	0.794

Baseline	384x384	Hard	32	0.652
MHFT	384x384	Hard	32	0.773
Baseline	512x512	Hard	16	0.702
MHFT	512x512	Hard	16	0.815

Аналіз Таблиці 4.2 дозволяє стверджувати, що архітектура MHFT перевершує Baseline у всіх протестованих конфігураціях. Найбільш значущим є те, що MHFT дозволяє ефективно використовувати зображення високої роздільної здатності та агресивні методи аугментації, перетворюючи додаткову складність даних на приріст точності, тоді як базова модель досягає межі своїх можливостей значно раніше. Це підтверджує тезу про те, що "вузьким місцем" у даній задачі є не кількість даних чи потужність backbone, а саме спосіб формулювання задачі навчання, який у нашому випадку успішно вирішується через ієрархічну декомпозицію.

4.4 Програмна реалізація та користувацький інтерфейс системи

Логічним завершенням даної роботи є інтеграція розробленої математичної моделі у закінчений програмний продукт. Хоча основний фокус дослідження був зосереджений на покращенні алгоритмічних метрик якості, для практичного застосування результатів у сфері прецизійного землеробства необхідно створити ознайомчий прототип інструменту для взаємодії з системою. У цьому підрозділі описано прототип програмного рішення інтегрований з розроблюваною математичною моделлю, який адаптовано для відображення специфічних ієрархічних прогнозів моделі MHFT.

Для практичної апробації результатів дослідження та демонстрації роботи розробленої моделі MHFT було створено десктопний програмний додаток. Програмний продукт реалізовано мовою Python з використанням бібліотеки для створення графічного інтерфейсу Tkinter та фреймворку глибокого навчання PyTorch. Додаток дозволяє користувачеві завантажувати

зображення листя, автоматично проводити його попередню обробку та отримувати діагноз у зрозумілій формі українською мовою. Інтерфейс для завантаження фотографії листка продемонстровано на рисунку 4.6

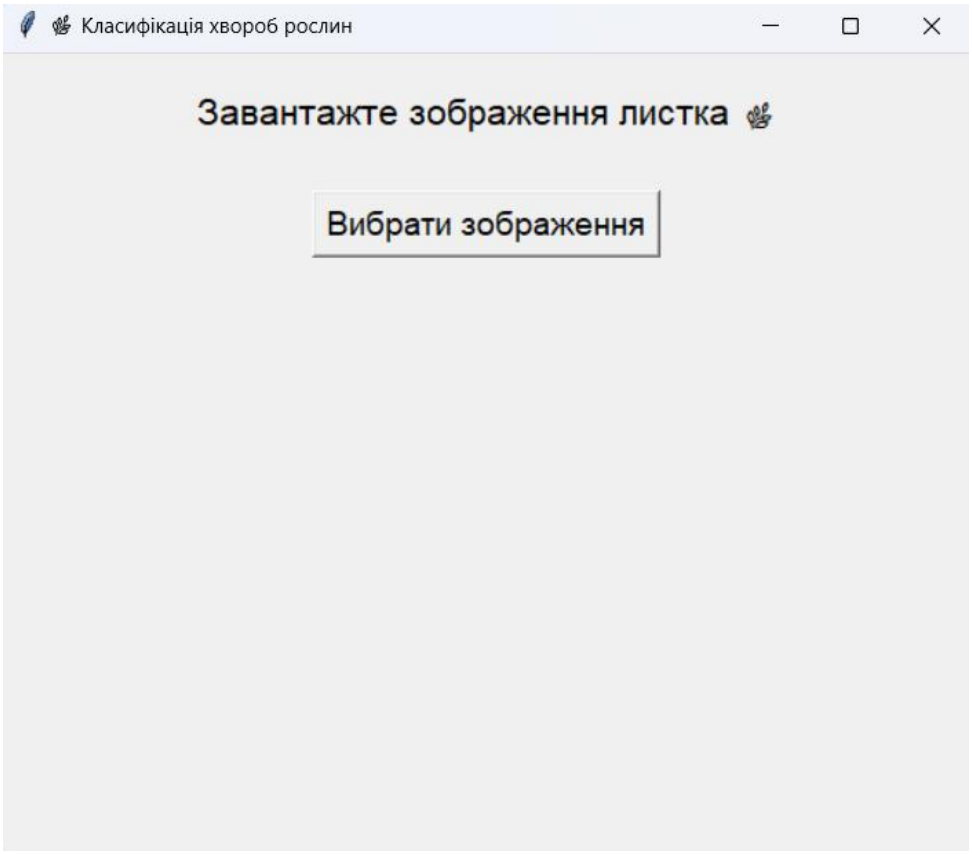

Рисунок 4.6. — Інтерфейс для завантаження зображення для аналізу

Програмна реалізація зосереджена в скрипті, який структурно поділяється на кілька логічних блоків: конфігурація, визначення архітектури нейронної мережі, завантаження ваг, конвеєр обробки даних та реалізація графічного інтерфейсу.

У блоці налаштувань визначено параметри вхідного тензора (розмір зображення 512×512 пікселів) та пристрій для обчислень (CPU або CUDA, залежно від наявності дискретної відеокарти). Важливою особливістю реалізації є модуль локалізації, представлений словником label_to_ua. Він забезпечує мапінг технічних англомовних міток моделі (наприклад, powdery_mildew, frog_eye_leaf_spot) у загальновживані українські назви

хвороб («Борошниста роса», «Плямистість листка»), що робить інтерфейс дружнім до кінцевого користувача.

У коді реалізовано клас MHFT (Multi-Head Fine-Tuning), який описаний в попередньому розділі, він є основою пропонованої архітектури.

Програма реалізує робастний механізм завантаження контрольної точки (checkpoint) з файлу best_mhft_fgvc8.pth. Враховуючи можливі відмінності у найменуванні шарів під час різних етапів навчання, впроваджено алгоритм адаптації словника станів (state_dict). Скрипт автоматично перейменовує ключі (наприклад, замінює застарілі назви head_binary на актуальні bin_head), що забезпечує коректну десеріалізацію моделі без помилок виконання. Також передбачено завантаження індивідуальних порогів чутливості (thresholds) для кожного класу, що дозволяє тонко налаштувати точність роботи системи.

Для підготовки вхідних даних використовується бібліотека Albumentations. Конвеєр трансформацій tfm включає:

- Resize: зміна розміру зображення до 512×512 пікселів;
- Normalize: нормалізація значень пікселів з використанням середніх значень та стандартного відхилення набору ImageNet;
- ToTensorV2: конвертація масиву NumPy у тензор PyTorch.

Графічний інтерфейс користувача (GUI) реалізовано на базі класу PlantClassifierApp. Головне вікно програми має розмір 700×700 пікселів і містить інтуїтивно зрозумілі елементи керування.

Алгоритм взаємодії користувача з системою виглядає наступним чином:

– Завантаження: Користувач натискає кнопку "Вибрати зображення", що викликає стандартне діалогове вікно операційної системи для вибору файлів форматів .jpg, .jpeg або .png;

– Візуалізація: Обране зображення зчитується бібліотекою PIL, автоматично масштабується для коректного відображення у вікні програми

(thumbnail) та виводиться на екран.

Інференс (Прогнозування):

– Зображення проходить через конвеєр трансформацій. Сформований тензор передається на обчислювальний пристрій (Device). Модель виконує прямий прохід (forward pass), генеруючи вихідні логіти. До виходів "точної" голови (fine_head) застосовується сигмоїдальна функція активації (torch.sigmoid), що перетворює їх у ймовірності від 0 до 1;

– Інтерпретація результатів: Отримані ймовірності порівнюються із завантаженими порогами. Класи, ймовірність яких перевищує поріг, вважаються виявленими;

– Вивід результату: Технічні назви виявлених класів перекладаються українською мовою та виводяться у текстове поле "Прогноз" (наприклад, "Прогноз: Парша | Іржа"). У випадку, якщо жоден клас не перевищив поріг, система діагностує "Здоровий листок".

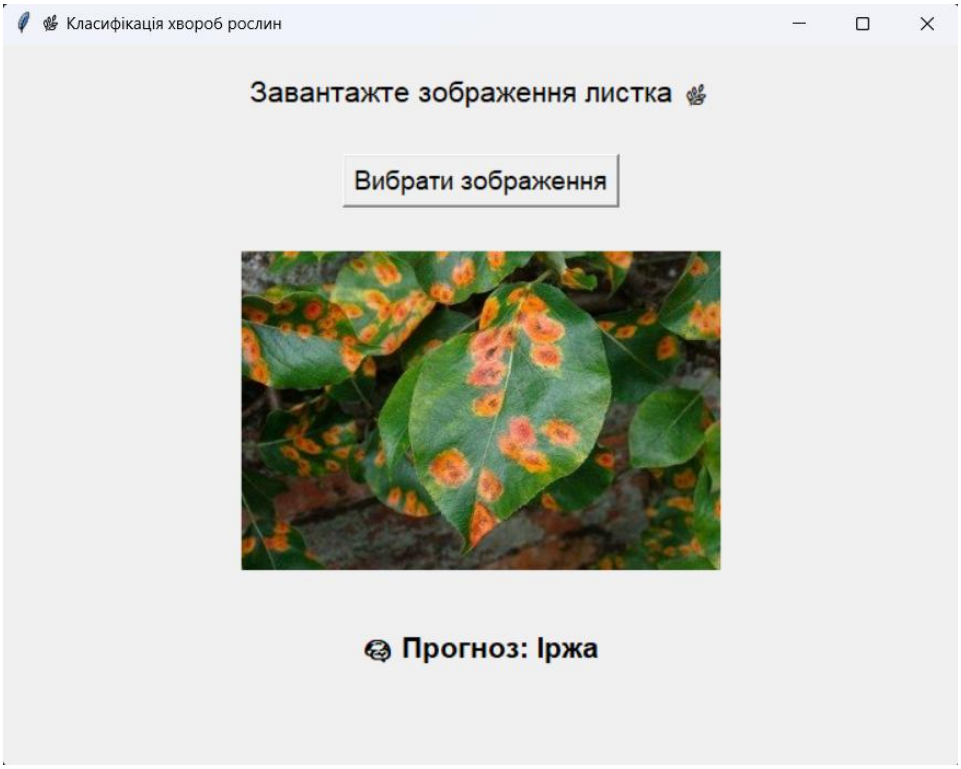

Рисунок 4.7 — Результат класифікації

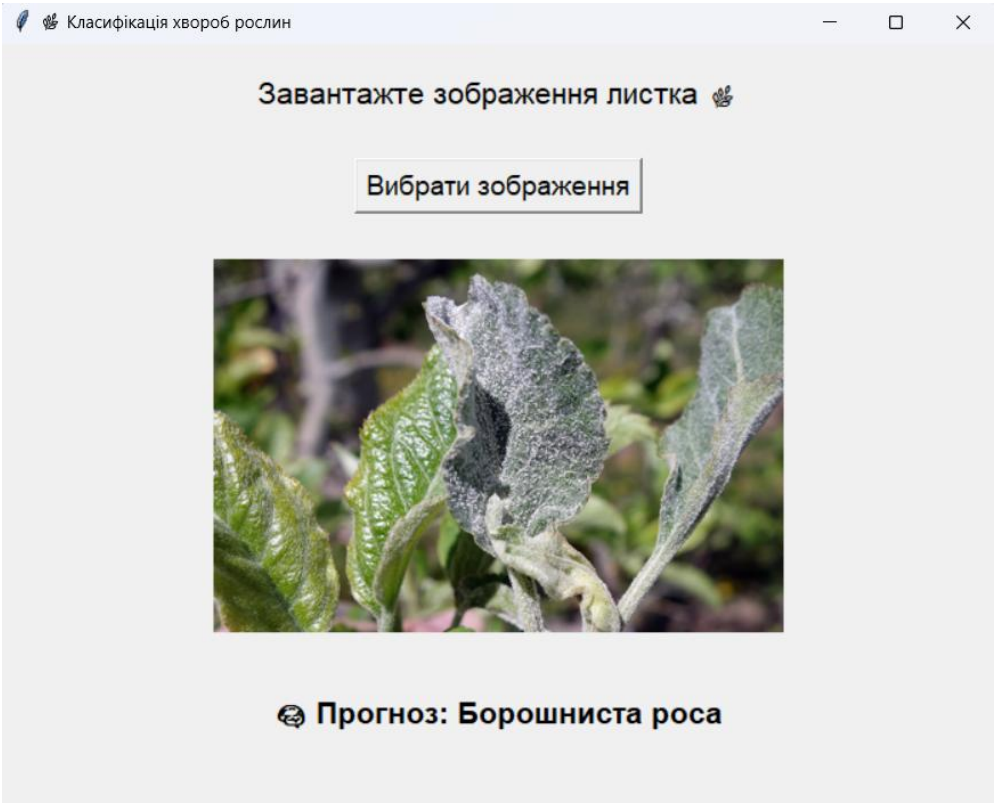

Рисунок 4.8 — Результат класифікації

Рисунок 4.9 — Результат класифікації

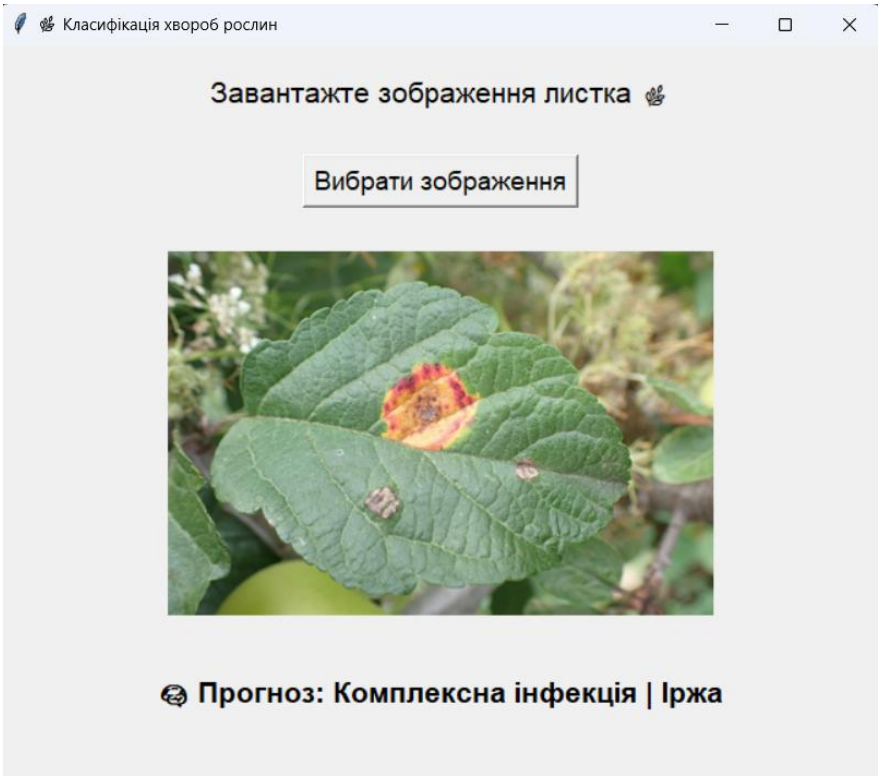

Рисунок 4.10 — Результат класифікації

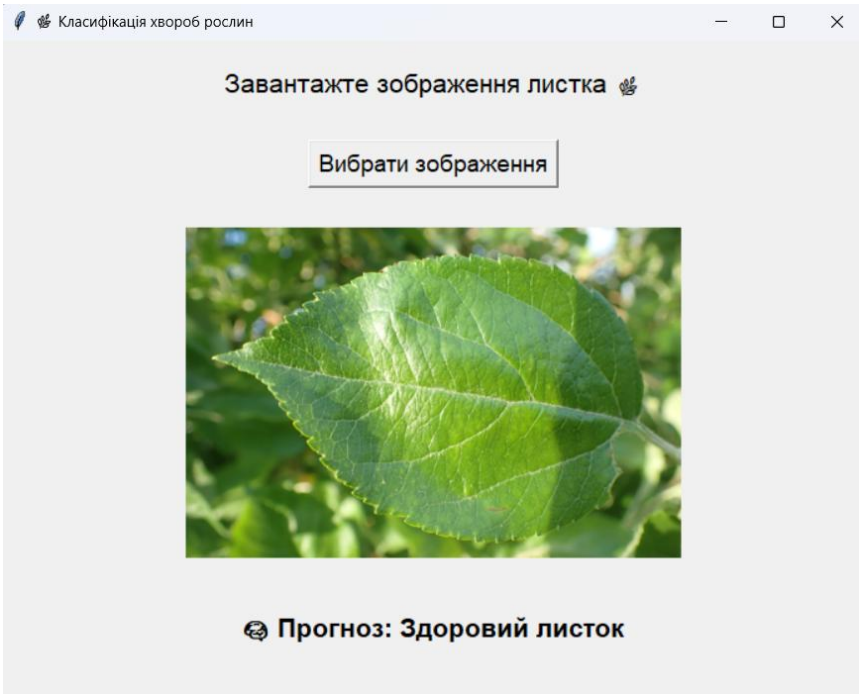

Рисунок 4.11 — Результат класифікації

Цей програмний продукт демонструє повну готовність розробленої технології до впровадження і слугує зручним інструментом для демонстрації можливостей ієрархічної моделі MHFT.

ВИСНОВОК

У даній роботі вирішено актуальну науково-прикладну задачу підвищення ефективності автоматизованої візуальної діагностики патологій рослин шляхом розробки, дослідження та програмної реалізації методу ієрархічної тонкої настройки глибоких нейронних мереж. У межах першого етапу детально проаналізовано предметну область сучасної цифрової рослинної патології, ключові підходи до автоматизованої діагностикиї. Це дозволило чітко сформулювати вимоги до майбутньої системи, окреслити типові сценарії її використання та виявити основні виклики, пов’язані з багатокласною і багато-мітковою природою задачі.

У процесі аналізу предметної області було з’ясовано, що традиційні ручні методи діагностики хвороб рослин є трудомісткими, суб’єктивними і погано масштабуються при збільшенні площ посівів. Натомість використання комп’ютерного зору та глибокого навчання дає змогу автоматизувати процес виявлення та класифікації симптомів захворювань, підвищити швидкість прийняття рішень та зменшити людський фактор. Окрему увагу приділено специфіці задачі багатолейблової класифікації, коли на одному зображенні можуть співіснувати ознаки кількох станів (наприклад, одночасна наявність кількох типів уражень чи поєднання «здорових» та пошкоджених ділянок). Також розглянуто ієрархічну структуру класів, де високорівневі категорії (умовно «здоровий/хворий», «складний/простий випадок») деталізуються до конкретних типів уражень.

На основі проведеного аналізу обґрунтовано вибір підходу, який поєднує сучасні SOTA-backbone-архітектури глибоких нейронних мереж (зокрема, CNN/Transformer-подібні моделі, попередньо натреновані на великих загальних вибірках зображень) із спеціалізованим дизайном вихідних голів під потреби рослинної патології. Було проаналізовано сильні та слабкі сторони «пласких» моделей, що мають лише одну класифікаційну голову, і показано, що ігнорування ієрархії класів та особливостей розподілу міток

призводить до втрати частини інформації та може обмежувати якість узагальнення.

У другому розділі розроблено і описано базову мультиголову (multi-head) архітектуру моделі, яка виступає вихідною точкою для подальших удосконалень у рамках дипломної роботи. Запропонована архітектура передбачає використання спільного глибокого backbone-модуля для вилучення ознак із зображення, поверх якого побудовано декілька класифікаційних голів, орієнтованих на різні рівні деталізації. Зокрема, одна голова відповідає за більш грубе чи бінарне розділення (наприклад, здоровий/уражений або простий/складний випадок), одна або кілька проміжних голів — за класифікацію в середньорівневих категоріях, а фінальна голова — за тонку багато-міткову класифікацію конкретних типів хвороб. Такий підхід дозволяє моделі одночасно враховувати як загальний стан рослини, так і специфіку конкретних уражень.

У межах опису архітектури було обґрунтовано вибір функцій втрат, загальних принципів навчання та оцінювання якості моделі. Зокрема, розглянуто використання бінарної крос-ентропії для багато-міткової класифікації, можливість її модифікацій (наприклад, із вагами класів) та поєднання втрат з різних голів в єдиний оптимізаційний критерій. Окремо звернено увагу на те, що мультиголова схема дозволяє експериментувати з різними стратегіями балансування внеску кожної голови в загальну функцію втрат, що відкриває простір для гнучкого налаштування моделі під конкретні вимоги задачі.

Результатом виконаної роботи є сформований цілісний концептуальний та архітектурний фундамент для подальшого розвитку системи. По-перше, проведений аналіз предметної області дав змогу краще зрозуміти потреби кінцевих користувачів (агрономів, фермерів, сервісів дистанційного моніторингу) та сформулювати реалістичні вимоги до якості моделі. По-друге, базова мультиголова архітектура, описана в роботі, є гнучкою платформою, яку можна розширювати додатковими модулями: виявленням зображень поза

розподілом (OOD-детекцією), спеціальними стратегіями роботи з «важкими» негативними прикладами (hard negative mining), модулями покращення якості зображення (super-resolution для дрібних деталей листя), а також адаптацією до домашніх фотографій та складних умов освітлення (нічні/низькоякісні знімки).

Практичним результатом дослідження стала повноцінна програмна реалізація системи діагностики мовою Python з використанням фреймворку PyTorch та бібліотеки Tkinter. Створено графічний інтерфейс, який візуалізує кінцевий діагноз, а також надає українську локалізацію результатів. Загалом, результати магістерської роботи свідчать про те, що інтеграція апріорних знань про структуру предметної області безпосередньо в архітектуру нейронної мережі є потужним методом підвищення якості класифікації. Розроблена система характеризується високою точністю, логічною прозорістю та стійкістю до умов польової зйомки, що робить її готовою до впровадження у складі систем підтримки прийняття рішень у прецизійному землеробстві.

Таким чином, мета етапу роботи — сформувати ґрунтовний аналіз предметної області та створити базову мультиголову архітектуру моделі класифікації хвороб рослин — була досягнута. Розроблені підходи та отримані напрацювання створюють міцний фундамент для наступних етапів дослідження. Результати цієї роботи мають як наукову, так і практичну цінність, оскільки відкривають шлях до побудови доступних та відтворюваних рішень для ранньої діагностики хвороб рослин на основі сучасних методів глибинного навчання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Дідусь О.П., Ситнікова П.Е. Автоматизований аналіз та класифікація біологічних ознак рослин з застосуванням нейронних мереж. Матеріали XVIII міжнародної науково-практичної конференції «Інформаційні технології і автоматизація – 2025»: Зб. матеріалів форуму. м. Одеса, 2025. С. 934–936.
2. Savary S. et al. The global burden of pathogens and pests on major food crops // Nature Ecology & Evolution. 2019. Vol. 3, no. 3. P. 430–439.
3. Agrios G. N. Plant Pathology. 5th ed. Burlington : Elsevier Academic Press, 2005. 922 p.
4. Ficke A., Cowger C., Bergstrom G. Understanding yield loss and pathogen biology to improve disease management: Septoria nodorum blotch – a case study in wheat // Plant Disease. 2018. Vol. 102, no. 4. P. 696–707.
5. Singh V., Misra A. K. Detection of plant leaf diseases using image segmentation and soft computing techniques // Information Processing in Agriculture. 2017. Vol. 4, no. 1. P. 41–49.
6. Liu J., Wang X. Plant disease detection using deep learning: A review // Plants. 2021. Vol. 10, no. 2. P. 238. DOI: 10.3390/plants10020238.
7. Hassan S. M., Maji A. K., Jasim M. A novel deep learning model for plant disease recognition on mobile devices // Journal of King Saud University – Computer and Information Sciences. 2022. Vol. 34, no. 8. P. 5777–5798.
8. Liu Z. et al. A ConvNet for the 2020s // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). New Orleans, LA, USA, 2022. P. 11976–11986.
9. Liu Z. et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows // Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). Montreal, QC, Canada, 2021. P. 10012–10022.
10. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // Proceedings of the 36th International Conference on Machine Learning (ICML). Long Beach, CA, USA, 2019. P. 6105–6114.

11. Dosovitskiy A. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // International Conference on Learning Representations (ICLR). 2021. URL: https://openreview.net/forum?id=YicbFdNTTy. (дата звернення 12.12.2025)

12. He K. et al. Deep Residual Learning for Image Recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA, 2016. P. 770–778.

13. Wei X.-S. et al. Fine-Grained Image Analysis with Deep Learning: A Survey // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2021. Vol. 44, no. 12. P. 8927–8948.

14. Boulent J., St-Charles P. L., Foucher S. Automatic identification of plant diseases using deep learning: Challenges and opportunities // Precision Agriculture. 2020. Vol. 21. P. 876–900.

15. Buslaev A. et al. Albumentations: Fast and Flexible Image Augmentations // Information. 2020. Vol. 11, no. 2. P. 125. DOI: 10.3390/info11020125.

16. Paszke A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems 32 (NeurIPS). Vancouver, Canada, 2019. P. 8024–8035.

17. Wightman R. PyTorch Image Models. 2019. DOI: 10.5281/zenodo.4414861. URL: https://github.com/rwightman/pytorch-image-models. (дата звернення 12.12.2025)

18. Selvaraju R. R. et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization // Proceedings of the IEEE International Conference on Computer Vision (ICCV). Venice, Italy, 2017. P. 618–626.

19. Ruder S. An Overview of Multi-Task Learning in Deep Neural Networks // arXiv preprint arXiv:1706.05098. 2017.

20. Crawshaw M. Multi-Task Learning with Deep Neural Networks: A Survey // arXiv preprint arXiv:2009.09796. 2020.

21. Shorten C., Khoshgoftaar T. M. A survey on image data augmentation

for deep learning // Journal of Big Data. 2019. Vol. 6. P. 60.

22. Cubuk E. D., Zoph B., Shlens J. RandAugment: Practical automated data augmentation with a reduced search space // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops. Seattle, WA, USA, 2020. P. 702–703.

23. Harris C. R. et al. Array programming with NumPy // Nature. 2020. Vol. 585. P. 357–362.

24. McKinney W. Data Structures for Statistical Computing in Python // Proceedings of the 9th Python in Science Conference. Austin, TX, USA, 2010. P. 51–56.

25. Vaswani A. et al. Attention Is All You Need // Advances in Neural Information Processing Systems (NeurIPS). Long Beach, CA, USA, 2017. P. 5998–6008.

26. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization // International Conference on Learning Representations (ICLR). San Diego, CA, USA, 2015.

27. Loshchilov I., Hutter F. Decoupled Weight Decay Regularization // International Conference on Learning Representations (ICLR). New Orleans, LA, USA, 2019.

28. Saleem M. H., Potgieter J., Arif K. M. Plant Disease Detection and Classification by Deep Learning // Plants. 2019. Vol. 8, no. 11. P. 468.

29. Wang D., Wang J., Li W. Deep learning for plant identification and disease detection: A review // Frontiers in Plant Science. 2022. Vol. 13. P. 1022718.

30. Touvron H. et al. Training data-efficient image transformers & distillation through attention // International Conference on Machine Learning (ICML). 2021. P. 10347–10357.

31. Sandler M. et al. MobileNetV2: Inverted Residuals and Linear Bottlenecks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City, UT, USA, 2018. P. 4510–4520.

32. Chen T. et al. A Simple Framework for Contrastive Learning of Visual

Representations // International Conference on Machine Learning (ICML). 2020. P. 1597–1607.

33. Hu J., Shen L., Sun G. Squeeze-and-Excitation Networks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City, UT, USA, 2018. P. 7132–7141.

34. Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. 2000. Vol. 25. P. 120–125.

35. Falcon W., The PyTorch Lightning Team. PyTorch Lightning. 2019. URL: https://github.com/Lightning-AI/lightning. (дата звернення 12.12.2025)

36. Abadi M. et al. TensorFlow: A system for large-scale machine learning // 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). Savannah, GA, USA, 2016. P. 265–283.

37. Zhang Y. et al. Fine-grained visual classification via progressive multi-granularity training of jigsaw patches // European Conference on Computer Vision (ECCV). Glasgow, UK, 2020. P. 153–168.

38. Kamilaris A., Prenafeta-Boldú F. X. Deep learning in agriculture: A survey // Computers and Electronics in Agriculture. 2018. Vol. 147. P. 70–90.

39. Li Z. et al. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects // IEEE Transactions on Neural Networks and Learning Systems. 2021. Vol. 33, no. 12. P. 6999–7019.

40. Szegedy C. et al. Rethinking the Inception Architecture for Computer Vision // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, USA, 2016. P. 2818–2826.

41. Van Rossum G., Drake F. L. Python 3 Reference Manual. Scotts Valley, CA : CreateSpace, 2009. 242 p.

42. Hunter J. D. Matplotlib: A 2D graphics environment // Computing in Science & Engineering. 2007. Vol. 9, no. 3. P. 90–95.