МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____Юрій КОНДРАТЕНКО

«____»_____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНТЕЛЕКТУАЛЬНА СИСТЕМА ДІАГНОСТУВАННЯ ХВОРОБ РОСЛИН З ВИКОРИСТАННЯМ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувачка _____ Єлизавета РУДА

«___»_____ 2025 р.

Керівник ст. викладач _____ Іван БУРЛАЧЕНКО

«___»_____ 2025 р.

Миколаїв – 2025

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Юрій КОНДРАТЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Руда Єлизавета Василівна
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Інтелектуальна система діагностування хвороб рослин з використанням методів штучного інтелекту».

Керівник роботи: Бурлаченко Іван Сергійович, ст. викладач. кафедри комп'ютерної інженерії.

Затверджена наказом ЧНУ ім. Петра Могили від «18» листопада 2024 р. № 321.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: інтелектуальна система діагностики хвороб рослин; набір даних, що містить фото захворювань рослин; моделі машинного навчання та алгоритми штучного інтелекту.

4. Перелік питань, що підлягають розробці: аналіз сучасного стану задачі автоматичної діагностики хвороб рослин за зображенням; огляд існуючих методів

машинного та глибокого навчання, що застосовуються у задачах класифікації, зокрема у сфері аграрних технологій; навчання моделей на підготовленому наборі даних, оцінка їх точності та ефективності; розробка інтерактивного вебінтерфейсу; тестування та оцінка ефективності розробленої системи.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Іван БУРЛАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувачка _____ Єлизавета РУДА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «18» грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Інтелектуальна система діагностування хвороб рослин з використанням методів штучного інтелекту

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	18.12.2024	20.12.2024	виконано
2	Аналіз предметної області та постановка задачі	21.12.2024	30.01.2025	виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та аналогічних систем, щодо діагностування хвороб рослин	31.01.2025	01.03.2025	виконано
4	Огляд існуючих архітектур штучних нейронних мереж для класифікації хвороб рослин	02.03.2025	01.04.2025	виконано
5	Реалізація системи діагностики хвороб рослин з аналізом отриманих результатів	02.04.2025	15.05.2025	виконано
6	Перший попередній захист КР на засіданні комісії кафедри	16.05.2025	16.05.2025	виконано
7	Корегування роботи за результатами попереднього захисту	29.05.2025	29.05.2025	виконано
8	Другий попередній захист КР на засіданні комісії кафедри	30.05.2025	30.05.2025	виконано
9	Доробка та остаточне оформлення КР	31.05.2025	05.06.2025	виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	12.06.2025	12.06.2025	

Керівник роботи _____ Іван БУРЛАЧЕНКО
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувачка _____ Єлизавета РУДА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2025 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 401з ЧНУ ім. Петра Могили

Рудої Єлизавети Василівни

на тему: **«ІНТЕЛЕКТУАЛЬНА СИСТЕМА ДІАГНОСТИКИ ХВОРОБ РОСЛИН З ВИКОРИСТАННЯМ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ»**

Актуальність роботи полягає у потребі у своєчасного виявлення захворювань сільськогосподарських культур з метою зменшення економічних втрат та підвищення врожайності.

Об'єктом роботи є процес автоматизованої діагностики хвороб рослин на основі зображень.

Предметом роботи є методи глибокого навчання та архітектури згорткових нейронних мереж, які застосовуються для класифікації зображень уражених рослин.

Метою роботи є підвищення якості класифікації хвороб рослин за зображеннями шляхом використання методів глибокого навчання, а також створення вебінтерфейсу інтелектуальної системи для забезпечення зручногодоступу до функціоналу.

У межах роботи здійснено аналіз предметної області та актуальних підходів до задачі класифікації, розглянуто архітектури нейронних мереж ResNet, EfficientNet, MobileNet та RegNet, проведено навчання моделей на основі набору даних PlantVillage. З метою практичного використання розроблено вебзастосунок, що дозволяє проводити класифікацію захворювань за фото.

Структура роботи охоплює чотири розділи. Перший розділ містить огляд предметної області, існуючих рішень та постановку задачі. У другому розділі проаналізовано методи і технології, використані в розробці. У третьому представлено процес навчання моделей і аналіз результатів. У четвертому розглянуто програмну реалізацію застосунку. Загальний обсяг роботи – 81

сторінка. Кваліфікаційна робота містить 28 рисунків, 1 таблицю і 32 джерела посилання.

Ключові слова: глибоке навчання, класифікація зображень, хвороби рослин, нейронні мережі, PlantVillage, штучний інтелект.

ABSTRACT

to the qualification work by the student of the group 401з of Petro Mohyla Black Sea National University

Ruda Yelyzaveta

“INTELLIGENT SYSTEM FOR PLANT DISEASE DIAGNOSIS USING ARTIFICIAL INTELLIGENCE METHODS”

The relevance of this qualification work lies in the need for timely detection of agricultural crop diseases in order to reduce economic losses and increase yield.

The object of the qualification work is the process of automated plant disease diagnosis based on images.

The subject of the qualification work is deep learning methods and convolutional neural network architectures used for classifying images of affected plants.

The aim of the qualification work is to improve the quality of plant disease classification from images by using deep learning methods, as well as to create a web interface for an intelligent system to provide convenient access to its functionality.

Within the scope of the qualification work, the subject domain and current approaches to image classification tasks were analyzed. Neural network architectures such as ResNet, EfficientNet, MobileNet, and RegNet were examined. Model training was carried out using the PlantVillage dataset. For practical application, a web-based application was developed to perform disease classification from images.

The structure of the bachelor qualification work includes four chapters. The first chapter presents a review of the subject domain, existing solutions, and the problem statement. The second chapter analyzes the methods and technologies used in the development. The third chapter presents the model training process and analysis of the results. The fourth chapter discusses the software implementation of the application.

The total length of the bachelor qualification work is 81 pages. It contains 28 figures, 1 table, and 32 references.

Keywords: deep learning, image classification, plant diseases, neural networks, PlantVillage, artificial intelligence.

ЗМІСТ

ВСТУП

Сільське господарство є важливою складовою економіки, що забезпечує продовольчу безпеку людства. З поступовими змінами клімату воно дедалі частіше стикається з різноманітними проблемами та викликами. На сам перед вони несуть за собою зниження врожайності, поширення хвороб, збільшення частоти посух, повеней та інших екстремальних погодних явищ.

За даними Продовольчої та сільськогосподарської організації організації (англ. Food and Agriculture Organization, FAO) щорічно близько 20-40% світового врожаю втрачається через хвороби та шкідників, що призводить до значних фінансових втрат і загострення продовольчої кризи у слаборозвинених країнах світу[1].

Традиційні методи діагностики потребують експертного втручання, що робить їх ресурсозатратними. У ситуаціях коли, пріоритетом є швидке отримання результату існує ризик зменшення точності через недостатній аналіз.

У сучасному світі з’явилася можливість оптимізації багатьох процесів у агросфері за допомогою методів штучного інтелекту. Зокрема, алгоритми машинного зору дозволяють класифікувати хвороби рослин на основі зображень ураженого листя. Агрономія стає перспективним напрямком розвитку ШІ. Провідні компанії займаються впровадженням рішень що направлені на оптимізацію роботи у цій галузі. Серед них є IBM та PEAT GmbH, які займаються розробкою застосунків для визначення хвороб рослин, але зосереджені на вузькому колі задач.

Зважаючи на недоліки традиційних та сучасних методів діагностики хвороб рослин створення інформаційної системи для вирішення даної проблеми є актуальною темою.

Таким чином, зважаючи на недоліки традиційних та сучасних інструментів, розробка інтелектуальної системи для класифікації хвороб рослин із використанням методів машинного зору є актуальним напрямком дослідження.

Об’єктом кваліфікаційної роботи є процес діагностики хвороб рослин на основі зображень листя.

Предметом кваліфікаційної роботи є методи та засоби застосування штучних нейронних мереж для автоматизованого виявлення ознак хвороб рослин.

Метою кваліфікаційної роботи є розробка програмного забезпечення, що дозволяє виявляти та класифікувати хвороби рослин на основі зображень листя за допомогою методів штучного інтелекту, з метою підвищення точності, швидкості та доступності діагностики в агрономії.

1 АНАЛІЗ ПРЕДМЕТНОЇ СФЕРИ ТА ПОСТАНОВКА ЗАДАЧІ В РОЗРОБЦІ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ДІАГНОСТИКИ ХВОРОБ РОСЛИН

1.1 Опис предметної сфери

Сільське господарство є однією з ключових галузей економіки, що відіграє важливу роль у забезпеченні продовольчої та економічної стабільності. Агросфера переживає зміни спричинені як природніми так і антропогенними факторами. На врожайність сільськогосподарських культур впливає низка чинників, серед яких головними є: зміни клімату, дефіцит водних ресурсів, деградація ґрунтів, хвороби та шкідники.

По-перше, зміни клімату призводять до підвищення середньорічних температур, повеней, посух та інших екстремальних погодних явищ, знижуючи врожайність.

По-друге, вирощування багатьох культур залежить від наявності достатньої кількості води. Аграрії з різних регіонів світу все частіше стикаються з дефіцитом водних ресурсів. Вони спричинені як змінами клімату, зменшенням рівня ґрунтових вод так і використанням застарілих зрошувальних систем.

По-третє, ще одним важливим фактором, що впливає на врожайність є виснаження ґрунтів. Надмірна експлуатація, неправильне використання добрив та пестицидів призводять до зниження родючості. Деградацію ґрунтів також можуть спричиняти ерозія та засолення, ускладнюючи або унеможливлюючи ведення рослинництва.

Глобалізація також вплинула на поширення хвороб і шкідників. Підвищення температури і вологості призводять до міграції шкідників у нові регіони, де вони можуть спричиняти значні збитки. Дані умови є також сприятливими для розвитку різноманітних патогенів. За даними Продовольчої та сільськогосподарської організації (англ. Food and Agriculture Organization, FAO) щорічно близько 20-40% світового врожаю втрачається через хвороби та шкідників, що призводить до

значних фінансових втрат і загострення продовольчої кризи у слаборозвинених країнах світу[1].

Традиційні методи діагностики хвороб рослин потребують експертного втручання, що робить їх ресурсозатратними. Візуальний огляд рослин часто не є точним методом бо значною мірою залежить від досвіду фахівця. Також треба враховувати вплив подібності симптомів різних хвороб на початковій стадії розвитку. Нижче на рисунку 1.1 зображено уражене листя рослин, інфіковане Борошнистою росою та Хибною борошнистою росою, що демонструє подібність симптомів на початкових стадіях розвитку хвороб.

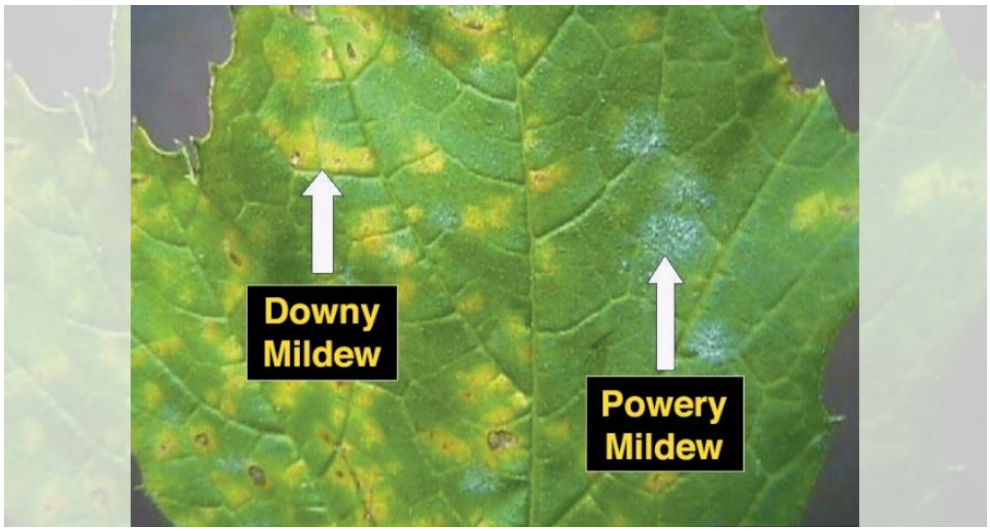

Рисунок 1.1 – Порівняння ураженого листя Борошнистою росою та Хибною борошнистою росою [2]

Лабораторні дослідження є дороговартісним методом діагностики, що може бути недоступним для невеликих господарств та тих, що розміщені в слаборозвинених регіонах з відсутньою відповідною інфраструктурою та фахівцями.

З огляду на вищезазначені недоліки традиційних методів діагностики, використання інформаційних технологій, зокрема засобів штучного інтелекту є перспективним напрямком розвитку галузі сільського господарства.

1.2 Огляд та аналіз наявних аналогів і публікацій

Впродовж останніх десятиліть активно проводяться дослідження спрямовані на поліпшення процесу ідентифікації хвороб рослин. Одним з найперспективніших напрямків вважається застосування методів штучного інтелекту, зокрема машинного зору та глибокого навчання, що дозволяють класифікувати хвороби на основі фото ураженого листя.

У дослідженні Explainable vision transformer enabled convolutional neural network for plant disease identification: PlantXViT [3] проведено огляд сучасних підходів до виявлення хвороб рослин із використанням методів глибокого навчання. Під час роботи над ним розроблено модель на основі згорткових нейронних мереж PlantXViT. ЇЇ було протестовано на п'яти публічно доступних наборах даних, включаючи зображення яблук, кукурудзи та рису. Вона показала високу точність розпізнавання хвороб: понад 93.55% для яблук, 92.59% для кукурудзи та 98.33% для рису, навіть за складних умов фону. PlantXViT демонструє потенціал для практичного застосування в аграрному секторі, забезпечуючи ефективне виявлення хвороб рослин.

Ще одним прикладом є Watson Decision Platform for Agriculture [4] розроблена компанією IBM. Система поєднує в собі застосування методів штучного інтелекту, аналітики великих даних, метеорології та IoT-технологій.
Вона аналізує дані метеорологічної служби, що дозволяє фермерам отримувати інформацію про погоді умови в межах кожного поля. На основі супутникових зображень вона надає детальні дані про наявність захворювань, рівень вологості та стану ґрунтів. Однак, платформа орієнтована на великі фермерські господарства, які мають достатній рівень цифровізації. Крім того у віддалених та слаборозвинених регіонах користування нею може ускладнюватися ще й через відсутність стабільного інтернет з’єднання.

Серед рішень, які можна використовувати в умовах обмежених обчислювальних ресурсів є Plantix [5]. Це мобільний застосунок розроблений

компанією PEAT GmbH, що використовує алгоритми глибокого навчання для визначення захворювань рослин на основі зображень листя.

Застосунок надає персоналізовані поради по догляду на основі вибору 8 агрокультур зі списку (див. рис. 1.2).

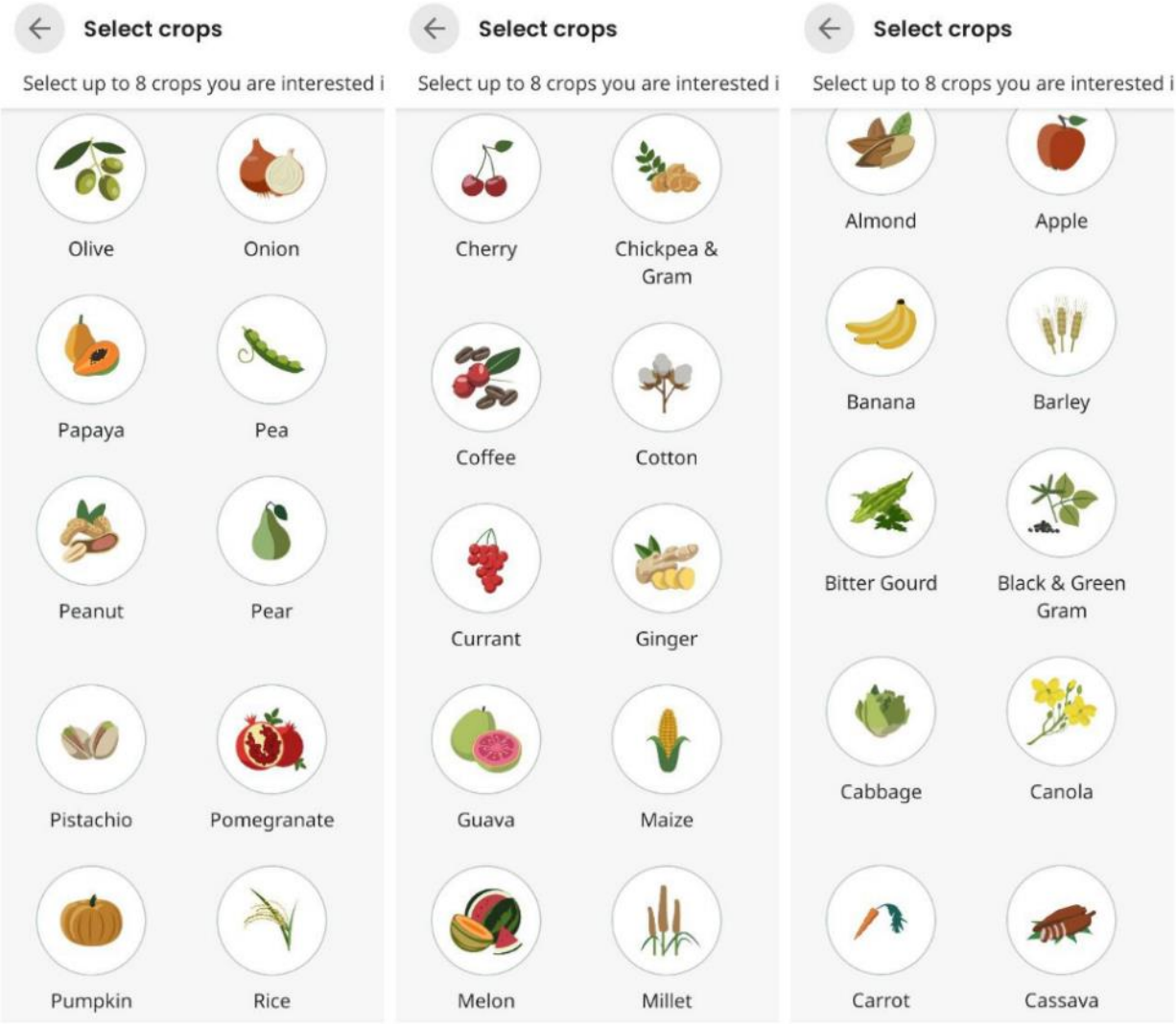

Рисунок 1.2 – Перелік агрокультур

Процес діагностики передбачає фотографування ураженої рослини. Після чого на основі фото проводиться класифікація хвороби та надаються рекомендації щодо лікування та захисту. Крім того на головному екрані розміщено інформацію про погоду, калькулятор розрахунку добрив, довідник хвороб та шкідників (див. рис. 1.3).

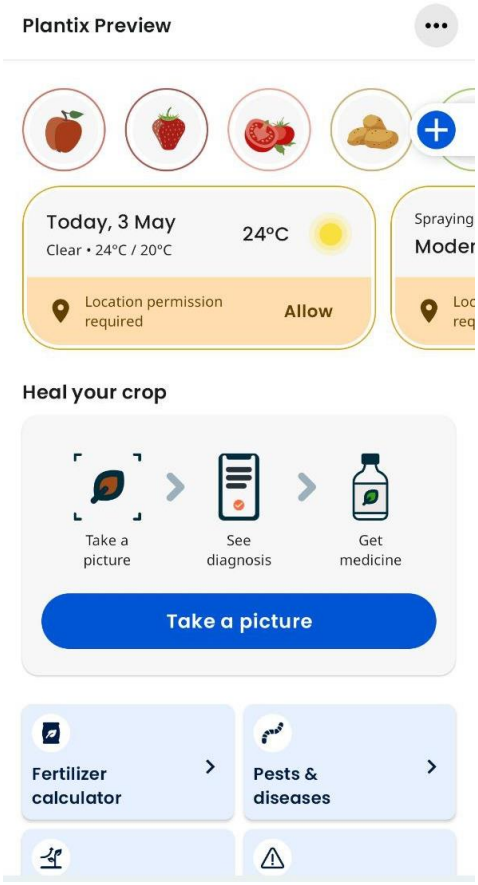

Рисунок 1.3 – Основний функціонал застосунку

Його основною цільовою аудиторією є дрібні фермери в регіонах з обмеженим або відсутнім доступом до відповідної інфраструктури. Попри всі переваги застосунок має обмежену точність при роботі з неякісними зображеннями та у випадку комбінованих уражень посівів.

Проведений огляд існуючих рішень в області автоматизації діагностики хвороб рослин показує активний розвиток технологій штучного інтелекту та машинного зору в агрономії. Втім кожне з проаналізованих рішень має свої недоліки – від обмеженої точності та вузької сфери застосування до складності впровадження в умовах слаборозвинених регіонів.

Таким чином створення інтелектуальності системи для діагностики хвороб рослин, яка враховує сучасні технологічні досягнення і потреби користувачів, є актуальним напрямом і перспективною задачею для реалізації в межах кваліфікаційної роботи.

1.3 Постановка задачі

На підставі обраної теми кваліфікаційної роботи було сформульовано низку завдань, які необхідно реалізувати в межах дослідження для досягнення основної мети – розробки інтелектуальної системи виявлення хвороб рослин за зображенням пошкодженого листя. Проблема виявлення захворювань сільськогосподарських культур є надзвичайно актуальною в умовах зростаючого тиску глобальних викликів, таких як зміни клімату, зростання населення, деградація ґрунтів та обмеження водних ресурсів.

В умовах сучасного аграрного виробництва своєчасне виявлення та локалізація уражень рослин стає критично важливою умовою збереження врожаїв, зниження втрат і забезпечення продовольчої безпеки. Традиційні методи діагностики хвороб, які передбачають участь фахівця-агронома, потребують глибоких знань, досвіду та чимало часу. Такий підхід не завжди є можливим або ефективним, особливо в регіонах з обмеженим доступом до фахівців, лабораторного обладнання або в умовах масового виробництва, коли обстеження значних площ є складним і повільним процесом.

Особливу складність становить той факт, що на ранніх етапах розвитку багато хвороб мають схожі симптоми: пожовтіння листя, плями, скручування або в’янення – усе це може бути спричинене як грибковими інфекціями, так і дефіцитом елементів живлення чи пошкодженнями від шкідників. Саме тому велике значення має точність діагностики, яка безпосередньо впливає на вибір методів боротьби та агротехнічних рішень.

У зв’язку з цим постає потреба у створенні автоматизованої інтелектуальної системи, яка б дозволила здійснювати попередню діагностику на основі фото зображень, знятих у польових умовах. Така система повинна бути зручною у використанні, не вимагати спеціальної підготовки користувача, бути адаптивною до різних умов зйомки (освітлення, якість камери, відстань до об’єкта) та

забезпечувати високу точність розпізнавання на основі нейронних мереж і алгоритмів глибокого навчання.

Формулювання вимог до майбутньої інформаційної системи: точність, швидкодія, зручність інтерфейсу, кросплатформність.

Розробка прототипу системи: реалізація базового функціоналу, інтеграція модуля розпізнавання, механізмів збереження історії діагностики.

Тестування системи на прикладах реальних зображень рослин, уражених захворюваннями, визначення точності, кількості помилок класифікації, розробка рекомендацій щодо покращення точності моделі. Оцінка перспектив подальшого вдосконалення системи та можливості її впровадження в аграрні підприємства.

Таким чином, розробка даної інформаційної системи є відповіддю на запит сучасного аграрного сектору щодо доступних, швидких і точних засобів діагностики, які дозволять підвищити врожайність, знизити витрати на пестициди та обробки, а також запобігти поширенню хвороб на ранніх етапах. Використання методів штучного інтелекту у цій сфері відкриває нові можливості для автоматизації процесів та інтеграції з розумним фермерством, що є основою агротехнологій майбутнього.

Висновки розділу 1

У першому розділі було проведено аналіз предметної області, яка охоплює актуальні проблеми сільського господарства, пов’язані з виявленням хвороб рослин. Було встановлено, що на сучасному етапі агросфера стикається з численними викликами: змінами клімату, дефіцитом водних ресурсів, деградацією ґрунтів, поширенням шкідників та патогенів. Ці чинники значно знижують врожайність сільськогосподарських культур і призводять до економічних збитків.

Здійснений огляд традиційних методів діагностики (візуальні огляди, лабораторні дослідження) виявив їх основні недоліки – суб’єктивність, висока вартість, потреба у кваліфікованих спеціалістах і складність застосування в польових умовах. У зв’язку з цим актуальним стає використання інноваційних

підходів до автоматизованого виявлення хвороб рослин, зокрема методів машинного зору та штучного інтелекту.

Було проаналізовано сучасні технологічні рішення, зокрема PlantXViT, Watson Decision Platform for Agriculture та мобільний застосунок Plantix. Кожне з них демонструє потенціал для підвищення точності діагностики, проте водночас має певні обмеження – складність впровадження, потреба у стабільному інтернет з’єднанні, вузькість сфери застосування або зниження ефективності при обробці неякісних зображень.

Поставлена задача у межах цієї кваліфікаційної роботи передбачає розробку універсальної інформаційної системи, здатної здійснювати діагностику захворювань рослин за зображеннями листя з використанням алгоритмів глибокого навчання. Така система повинна бути зручною, наочною, доступною для широкого кола користувачів, включаючи фермерів без технічної підготовки, і забезпечувати високу точність діагностики навіть в умовах обмежених ресурсів.

Таким чином, результати першого розділу підтверджують наукову й практичну доцільність розробки інформаційної системи діагностики хвороб рослин.

2 МОДЕЛІ, МЕТОДИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КЛАСИФІКАЦІЇ ХВОРОБ РОСЛИН

2.1 Огляд методів для вирішення задачі класифікації хвороб рослин

Задача класифікації – це процес розподілу об’єктів на визначені категорії на основі їхніх ознак чи характеристик. У контексті штучного інтелекту це означає створення моделі, яка навчиться прогнозувати, до якого класу належить новий, раніше невідомий об’єкт. Класифікація широко застосовується у різних сферах – від розпізнавання образів і текстів до медичної діагностики та аграрних технологій. В контексті цієї роботи задача класифікації полягає у визначенні хвороби рослини за її зображенням – чи є вона здоровою, чи вражена певною хворобою, що дозволяє автоматизувати процес діагностики і підвищити ефективність агротехнічних заходів.

Варто також зазначити, що в методах штучного інтелекту існують різні базові підходи, які можна застосовувати окремо або комбінувати для розв’язання конкретних задач. Ці підходи відрізняються за рівнем обчислювальної складності, швидкістю навчання, точністю моделей та здатністю працювати з різними типами даних. У задачах машинного навчання класифікація є однією з найбільш поширених і важливих проблем.

Класичні методи машинного навчання – це набір алгоритмів, які базуються на статистичних і математичних моделях для розпізнавання закономірностей у даних та віднесення нових прикладів до певних класів. На відміну від сучасних глибоких нейронних мереж, ці методи зазвичай мають меншу обчислювальну складність та потребують менше ресурсів для навчання, що робить їх особливо корисними в задачах із обмеженими обчислювальними потужностями або невеликими наборами даних.

KNN (k-найближчих сусідів) [7] – це алгоритм для класифікації, який базується на ідеї, що подібні об’єкти мають бути близькими один до одного у просторі ознак. Логіка роботи цього методу полягає в тому, що для визначення

класу нового об’єкта алгоритм знаходить у навчальній вибірці певну кількість найближчих точок за відстанню в просторі ознак. Потім він аналізує, до яких класів належать ці сусіди, і на основі більшості визначає клас для нового об’єкта. Якщо у об’єкта найбільше схожих сусідів певного класу, найімовірніше, він теж належить до цього класу. Це робить алгоритм гнучким та простим у застосуванні, адже він не вимагає побудови складної моделі або тренування – вся логіка полягає у пошуку найближчих сусідів.

Алгоритм k-найближчих сусідів (KNN) є одним із базових і широко застосовуваних методів у машинному навчанні для задач класифікації. Принцип роботи алгоритму базується на припущенні, що об’єкти, які знаходяться близько один до одного у просторі ознак, ймовірно належать до одного класу. Для класифікації нового об’єкту визначається множина k найближчих сусідів серед навчальних даних. Далі аналізується розподіл класів у цій множині, і новому об’єкту присвоюється клас, який переважає серед найближчих сусідів. Важливими параметрами, що впливають на ефективність алгоритму, є кількість сусідів k, вибір метрики відстані та масштабування ознак для уникнення домінування одних змінних над іншими. Метод KNN не потребує етапу навчання в класичному розумінні, що робить його простим у реалізації, проте він є обчислювально затратним при роботі з великими обсягами даних через необхідність порівняння з усіма навчальними зразками.

Логістична регресія [7] – це статистичний метод, який використовується для моделювання ймовірності належності об'єкта до певного класу на основі його вхідних ознак. У задачах класифікації вона дозволяє прогнозувати імовірність приналежності об’єкта до певної категорії. Для багатокласової класифікації логістична регресія узагальнюється шляхом використання функції softmax, яка дозволяє оцінити ймовірність належності об’єкта до кожного з K класів:

$$softmax(z)_i = \frac{exp(z_i)}{\sum_{k \in K} exp(z_k)} \quad (2.1)$$

де z – вхідний вектор активаційної функції softmax;

z_i – i-й елемент вхідного вектора;

$exp(z_i)$ – стандартна експоненціальна функція, застосована до z_i;

$\sum_{k \in K} exp(z_k)$ – нормувальний член, який гарантує, що сума значень вихідного вектора дорівнює 1 для i-го класу.

Для багатокласової класифікації, після обчислення значень, які називають логітами і відображають міри належності до кожного класу, застосовується функція softmax, яка перетворює ці значення у ймовірності приналежності об’єкта до відповідних класів.

Наївний Байєсівський класифікатор (Naive Bayes) [9] – це ефективний метод машинного навчання, який застосовується для задач класифікації. Його робота ґрунтується на теоремі Байєса, яка дозволяє обчислювати ймовірність того, що об'єкт належить до певного класу, виходячи з його ознак.

$$P(C|X) = \frac{P(X|C) \times P(C)}{P(X)} \tag{2.2}$$

де $P(C|X)$ – апостеріорна ймовірність, ймовірність класу C за заданих вхідних даних X;

$P(X|C)$ – правдоподібність, ймовірність спостереження вхідного сигналу X, враховуючи, що він належить до класу C;

$P(C)$ – априорна ймовірність класу C, що відображає його поширеність у наборі даних;

$P(X)$ – гранична ймовірність X, яка служить нормуючою константою.

Основною ідеєю методу є припущення про те, що всі ознаки є незалежними одна від одної, тобто вплив кожної ознаки на кінцеве рішення розглядається окремо. Хоча в реальних задачах така незалежність трапляється рідко, це спрощення дозволяє значно зменшити складність обчислень.

Під час навчання модель запам’ятовує, наскільки часто певні значення ознак з’являються в кожному класі. А під час класифікації нового прикладу вона використовує цю інформацію, щоб оцінити, до якого класу найбільш ймовірно належить об’єкт. У результаті алгоритм визначає клас, для якого загальна ймовірність (з урахуванням усіх ознак) є найбільшою.

На практиці наївний Naive Bayes часто використовується для аналізу текстів, зокрема для фільтрації спаму або визначення тематики повідомлень, однак також може ефективно застосовуватись і в інших сферах, таких як медицина чи аграрні дослідження. Його основні переваги – простота реалізації, швидкість роботи та стабільність на невеликих або обмежених вибірках.

Хоча класичні методи машинного навчання демонструють базову ефективність у задачах класифікації, включаючи розпізнавання стану рослин за зображенням, вони мають низку суттєвих обмежень. Зокрема, ці алгоритми зазвичай потребують ручного виділення ознак і мають обмежену здатність до узагальнення при високій складності або варіативності даних. Крім того, класичні моделі погано справляються з вхідними даними великої розмірності, такими як зображення, де просторові зв’язки між пікселями відіграють ключову роль.

Ці обмеження стають критичними при роботі з реальними візуальними даними, де важливо автоматично виявляти та аналізувати приховані патерни, враховувати контекст та складні зв’язки між елементами зображення. Саме тому у подібних задачах використовуються методи глибокого навчання, які дозволяють будувати моделі з багатьма шарами і здатні самостійно виділяти ознаки з необроблених даних.

Багатошаровий персептрон (MLP) [10] є одним із типів штучних нейронних мереж, який використовується для задач класифікації. Його архітектура складається з послідовності шарів: вхідного, одного або кількох прихованих та вихідного шару. Кожен шар містить певну кількість нейронів, з’єднаних із нейронами сусідніх шарів (див. рис. 2.1). Основна ідея MLP полягає у послідовному

перетворенні вхідних даних через ці шари з використанням навчених ваг та активаційних функцій.

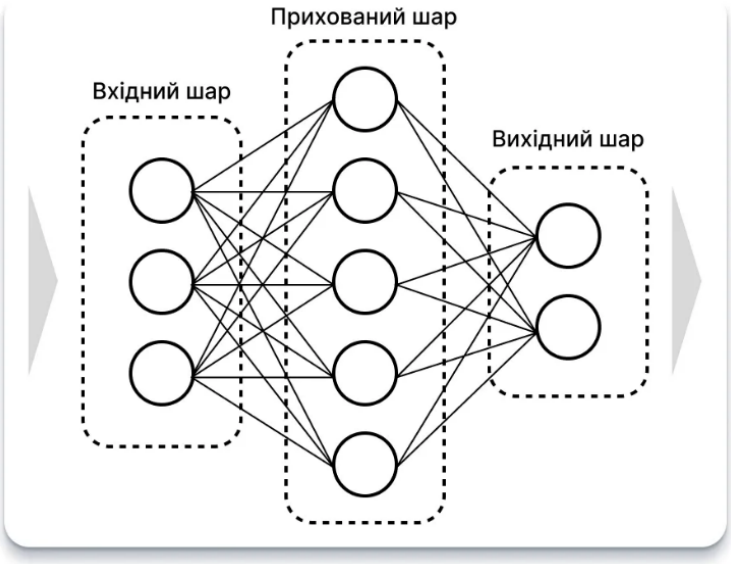

Рисунок 2.1 – Схематичне зображення багатошарового персептрона [11]

На відміну від класичних моделей машинного навчання, MLP не потребує ручного визначення ознак – він може навчитися виявляти важливі зв’язки у даних автоматично. Кожен нейрон обчислює зважену суму вхідних сигналів, до якої додається зсув (bias), після чого застосовується нелінійна активаційна функція, така як ReLU (Rectified Linear Unit) чи сигмоїда. Це дозволяє мережі моделювати складні нелінійні залежності. У процесі навчання MLP використовує метод зворотного поширення помилки (backpropagation) у поєднанні з градієнтним спуском для мінімізації функції втрат, що відображає відхилення прогнозу від очікуваного значення. Мережа поступово коригує ваги зв’язків між нейронами, щоб поліпшити точність передбачень.

MLP є прикладом повно зв’язної нейронної мережі, де кожен нейрон одного шару з’єднаний із кожним нейроном наступного. Це робить модель досить універсальною, але й ресурсозатратною при великій кількості входів, наприклад при обробці зображень. Тому, хоча MLP може бути застосований до задач

класифікації зображень, його ефективність обмежується, зокрема, тим, що модель не враховує просторову структуру даних.

У контексті задачі виявлення хвороб рослин за зображеннями, MLP може бути використаний як базовий підхід до класифікації. Проте його здатність розпізнавати складні візуальні патерни є обмеженою порівняно з більш спеціалізованими архітектурами.

Рекурентні нейронні мережі (RNN) [12] є типом штучних нейронних мереж, призначених для роботи з послідовними даними. На відміну від звичайних багатошарових персептронів, які обробляють кожен вхід незалежно, RNN здатні враховувати контекст попередніх входів завдяки внутрішньому механізму пам’яті. Це досягається тим, що на кожному кроці обчислень мережа аналізує поточний вхід та використовує інформацію про попередній стан, тобто результати обробки попередніх елементів послідовності.

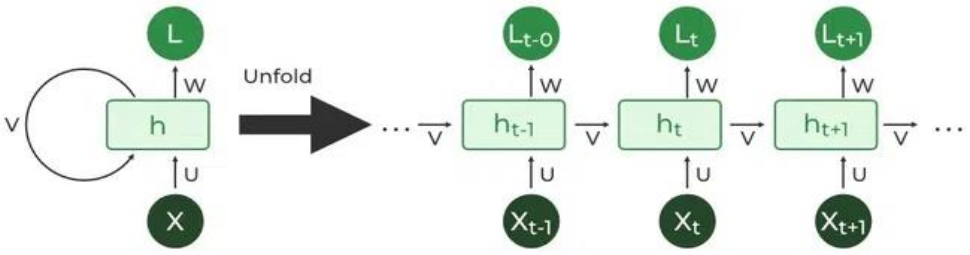

Рисунок 2.2 – Схематичне зображення RNN [13]

Робота RNN полягає у тому, що на кожному кроці обробки вхідного сигналу обчислюється новий прихований стан на основі поточного входу та прихованого стану з попереднього кроку. Цей прихований стан оновлюється за допомогою вагових коефіцієнтів і нелінійної активації, зазвичай функції tanh чи ReLU. Завдяки цьому мережа поступово накопичує інформацію в процесі обробки послідовності.

У контексті задачі класифікації стану рослин RNN можуть бути застосовані тоді, коли доступні послідовні спостереження за рослиною у часі – наприклад, серії зображень, що демонструють розвиток захворювання. У таких випадках мережа

може навчитися розпізнавати патерни, характерні для різних етапів хвороби, і таким чином точніше визначати її наявність або тип.

Згорткові нейронні мережі (CNN) [14] є однією з найефективніших архітектур глибокого навчання, особливо в задачах, пов'язаних з аналізом візуальної інформації. Їх принципова відмінність від класичних повнозв'язних нейронних мереж полягає у здатності автоматично виділяти просторові залежності та локальні особливості зображень за допомогою операції згортки. Це дозволяє зменшити кількість параметрів моделі, зберігаючи при цьому здатність до виявлення складних структурних ознак, характерних для певних класів об'єктів.

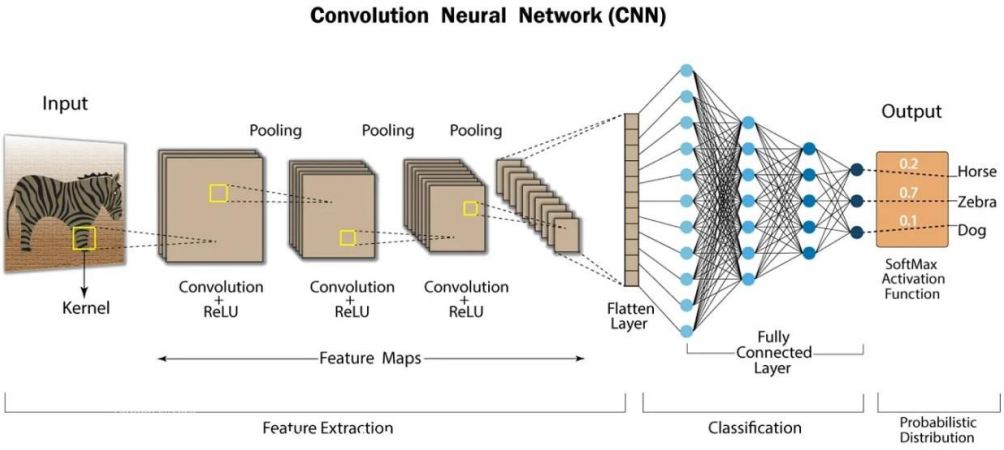

Рисунок 2.3 – Схематичне зображення CNN [15]

Після згорткових шарів зазвичай застосовуються нелінійні функції активації, такі як ReLU, що дозволяють моделі апроксимувати складні нелінійні залежності. Для зменшення розмірності та узагальнення ознак використовуються шари підвибірки (pooling), зокрема max pooling або average pooling. В заключній частині мережі додаються один або кілька повнозв'язних шарів, які агрегують виявлені ознаки та здійснюють класифікацію.

Завдяки використанню локальних зв'язків та поділу ваг, CNN демонструють високу ефективність у задачах класифікації зображень, особливо в умовах складних вхідних даних з високим рівнем варіативності. У контексті розпізнавання

хвороб рослин за зображенням, CNN здатна навчитися виявляти специфічні візуальні симптоми захворювань (плями, зміни кольору, пошкодження листкової пластинки) та класифікувати об’єкти з високою точністю.

Таким чином, згорткові нейронні мережі забезпечують ефективний підхід до автоматизованої діагностики на основі візуального аналізу, що є особливо цінним у сільському господарстві, де важливо вчасно і точно виявляти ознаки патологій у рослин для прийняття рішень щодо подальших агротехнічних заходів.

2.2 Архітектури моделей глибокого навчання

ResNet [16] – це архітектура глибоких нейронних мереж, розроблена для подолання проблем, що виникають під час навчання дуже глибоких моделей. Традиційні глибокі нейронні мережі часто стикаються з проблемою деградації точності, коли збільшення кількості шарів призводить не до покращення, а до погіршення результатів. ResNet вирішує цю проблему завдяки впровадженню залишкових (residual) блоків, які включають прямі пропуски сигналу (skip connections) між шарами.

Архітектурно ResNet складається із послідовності таких залишкових блоків. Кожен блок містить кілька послідовних шарів (див. рис. 2.4), згорткові шари з нормалізацією та функцією активації. Важливою особливістю є те, що на виході блока відбувається додавання вхідного сигналу без змін (skip connection) до результату обробки. Це дозволяє зберігати інформацію і передавати її далі, що значно полегшує навчання глибоких мереж і знижує ризик затухання градієнтів.

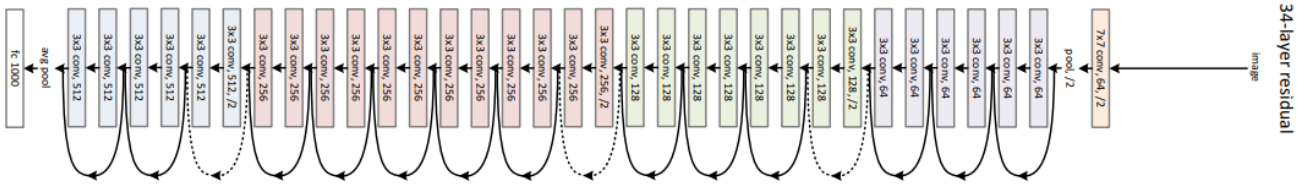

Рисунок 2.4 – Архітектура ResNet [17]

Завдяки такій архітектурі ResNet може ефективно навчатись навіть при дуже великій глибині (до сотень шарів), що забезпечує більш точне вилучення ознак зі складних зображень. Ця особливість робить ResNet одним із провідних методів у задачах комп'ютерного зору, зокрема для класифікації зображень, розпізнавання об'єктів, а також медичної діагностики та аграрних застосунків, таких як автоматична ідентифікація хвороб рослин за фотографіями.

Таким чином, ResNet є потужним інструментом у галузі глибокого навчання, що забезпечує високу продуктивність та стабільність навчання завдяки інноваційному використанню залишкових зв'язків. Що робить її надзвичайно корисною для створення інтелектуальних систем, які потребують точного аналізу та класифікації зображень у різних практичних сферах.

EfficientNet [18] – це сучасна архітектура згорткових нейронних мереж, запропонована компанією Google, яка спрямована на досягнення високої точності при значно меншій кількості параметрів і обчислень порівняно з традиційними моделями. Головною особливістю EfficientNet є систематичне масштабування всіх параметрів мережі: глибини, ширини і роздільної здатності вхідних зображень, що дозволяє ефективно збалансувати продуктивність і ресурси.

На відміну від простого збільшення розміру моделі у будь-якому одному напрямку (наприклад, лише збільшення кількості шарів або ширини каналів), EfficientNet використовує метод «компаундного масштабування» (compound scaling). Цей підхід одночасно збільшує глибину (кількість шарів), ширину (число каналів у кожному шарі) і роздільну здатність вхідних зображень за певними коефіцієнтами, що підбираються таким чином, щоб максимізувати точність при фіксованих ресурсах обчислень.

Базовою архітектурою EfficientNet є покращена версія MobileNetV2, яка включає інвертовані залишкові блоки (inverted residual blocks) із глибинними згортками (depthwise convolutions). Завдяки цьому EfficientNet зберігає компактність і ефективність обчислень, проте при масштабуванні демонструє значне покращення якості розпізнавання.

EfficientNet містить моделі різної складності – від EfficientNet-B0 до EfficientNet-B7, кожна з яких оптимально масштабована відповідно до доступних обчислювальних ресурсів та цілей задачі.

Завдяки унікальному підходу до масштабування та архітектурним інноваціям EfficientNet стала однією з найефективніших моделей у задачах класифікації зображень, сегментації та інших задачах комп'ютерного зору. Вона широко застосовується в наукових дослідженнях і промислових рішеннях, де важливі і висока точність, і оптимальне використання ресурсів.

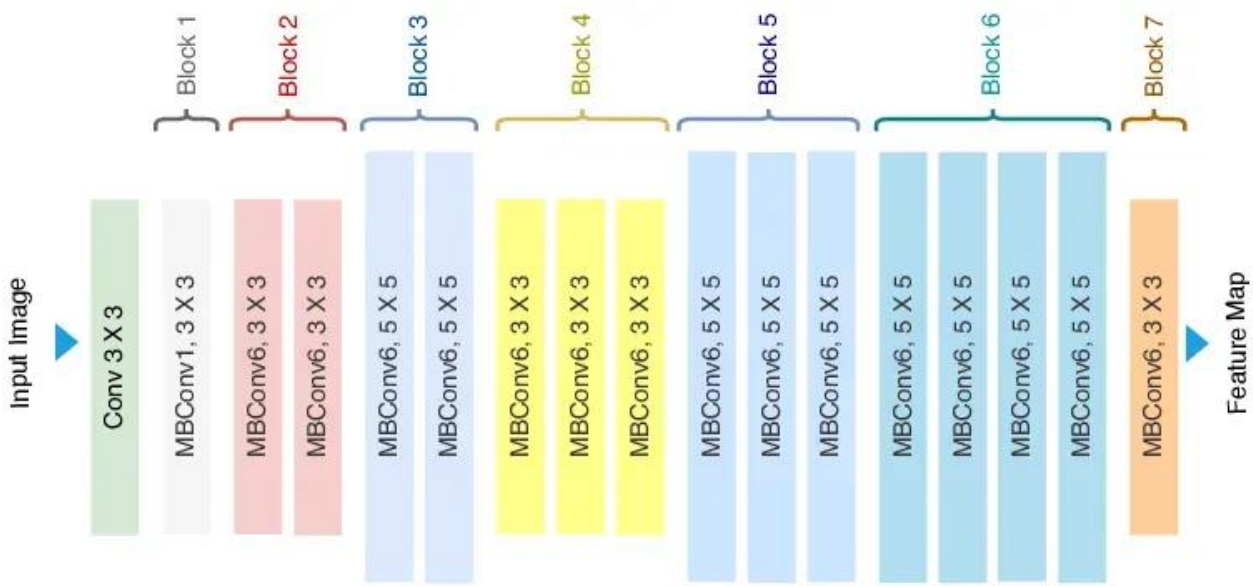

Рисунок 2.5 – Архітектура EfficientNet [19]

Модель починається з базового варіанту EfficientNet-B0, який містить послідовність таких блоків, що поетапно зменшують просторові розміри ознак і збільшують їх глибину. Наступні варіанти EfficientNet B1-B7 реалізують масштабування за допомогою компаундної формули, що одночасно розширює архітектуру по всіх трьох осях.

Завдяки своїй архітектурі EfficientNet демонструє високу продуктивність у задачах класифікації зображень при значно меншому споживанні ресурсів порівняно з класичними глибокими мережами. Це робить її особливо цінною для

застосувань, де обчислювальна ефективність та точність мають критичне значення, наприклад, у системах автоматизованої діагностики хвороб рослин за зображеннями.

RegNet [22] – це згорткова нейронна мережа, побудована на основі параметризованого шаблону, що дозволяє формувати архітектури з передбачуваними обчислювальними характеристиками та масштабованістю. Мережі цього типу розроблені на основі аналітичної функції, яка визначає ширину, глибину та кількість каналів у шарах, що забезпечує більшу регулярність у порівнянні з архітектурами, сформованими вручну або методом перебору.

Архітектурно RegNet складається з послідовних блоків із залишковими з'єднаннями (residual connections), які підтримують стабільне передавання градієнтів у глибоких мережах. Кожен блок включає згорткові шари, нормалізацію, активацію ReLU, а також групові згортки, що зменшують обчислювальні витрати без втрати якості ознак. У варіаціях типу RegNet-Y, до яких належить RegNet-Y-400MF, додатково використовуються SE-блоки (Squeeze-and-Excitation), які дозволяють мережі адаптивно підсилювати найбільш інформативні канали ознак.

Модель RegNet-Y-400MF є однією з найбільш компактних у серії RegNet і орієнтована на баланс між точністю та обчислювальною ефективністю. Вона дозволяє отримувати високоякісні результати у задачах класифікації зображень при значно меншому споживанні ресурсів, ніж класичні глибокі архітектури.

MobileNet [23] – це архітектура згорткової нейронної мережі, спеціально розроблена для ефективного використання в умовах обмежених обчислювальних ресурсів, зокрема на мобільних пристроях і вбудованих системах. Основна ідея цієї моделі полягає у зменшенні обчислювальної складності при збереженні достатнього рівня точності, що досягається за рахунок зміни стандартного підходу до згорток.

Замість згорткових шарів, MobileNet використовує глибинні згортки (depthwise separable convolutions). Така операція поділяє згортку на два етапи: спочатку виконується окрема згортка для кожного каналу зображення (depthwise

convolution), а потім проводиться згортка 1×1, яка поєднує отримані результати між каналами (pointwise convolution). Цей підхід суттєво зменшує кількість параметрів і обчислень, порівняно зі звичайними згортками, незначно впливаючи на точність.

Архітектура MobileNet є гнучкою завдяки двом гіперпараметрам: коефіцієнту ширини (width multiplier) та коефіцієнту роздільності (resolution multiplier). Перший контролює кількість каналів у шарах, а другий – розмір вхідних зображень. Це дозволяє адаптувати модель до конкретних обмежень апаратного забезпечення або до вимог щодо швидкості обробки.

Завдяки своїй компактності та ефективності, MobileNet стала популярним вибором для задач комп'ютерного зору на мобільних пристроях, таких як розпізнавання об'єктів, класифікація зображень, детекція хвороб рослин тощо. Її архітектура забезпечує прийнятний компроміс між швидкістю роботи й точністю моделі, що особливо важливо для інтерактивних або автономних застосунків.

2.3 Технології розробки системи

PyTorch [24] – це програмна платформа з відкритим кодом, призначена для реалізації моделей глибокого навчання та машинного навчання. Представлена у 2016 році дослідницьким підрозділом Facebook AI Research, швидко здобула визнання завдяки поєднанню високої гнучкості, ефективності та інтеграції з мовою Python.

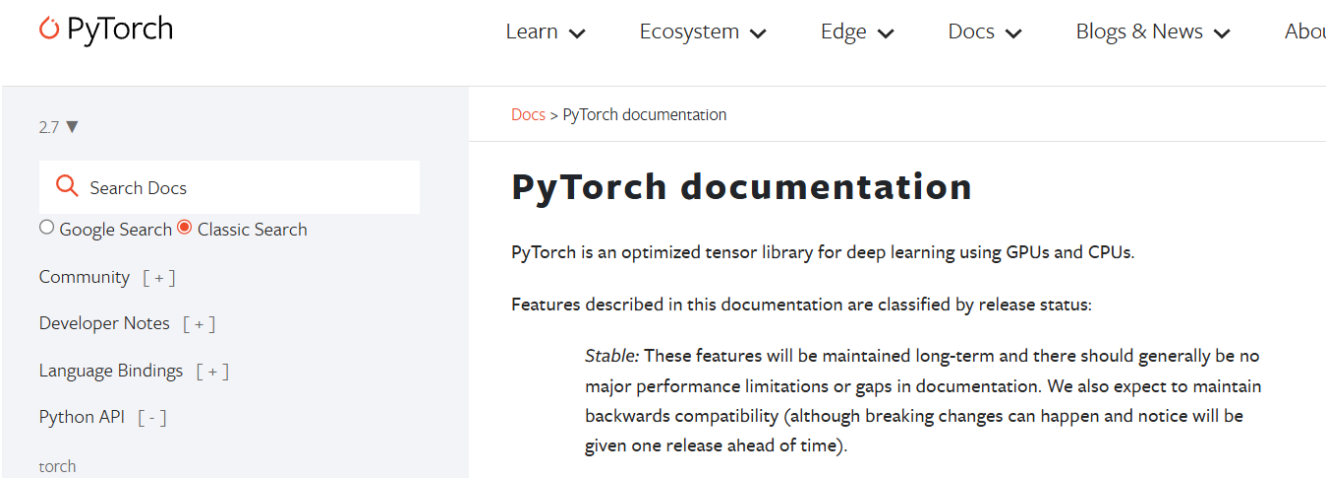

Рисунок 2.7 – Документація PyTorch

На відміну від багатьох інших фреймворків, PyTorch використовує динамічну модель обчислень, яка дозволяє будувати і модифікувати архітектуру нейронної мережі під час виконання, а не в процесі попередньої компіляції. Це важливо у наукових дослідженнях, де експериментальні моделі часто змінюються на етапі проєктування.

Фреймворк підтримує автоматичне диференціювання, що забезпечує ефективне обчислення градієнтів у процесі зворотного поширення помилки. PyTorch має модульну структуру, що дозволяє користувачам комбінувати різні компоненти мережі, оптимізатори, функції втрат і інструменти обробки даних у єдину експериментальну систему.

Завдяки глибокій інтеграції з CUDA, PyTorch дозволяє легко реалізовувати навчання на графічних процесорах (GPU), що суттєво прискорює обчислення при роботі з великими наборами даних. Бібліотека також підтримує розподілені обчислення, що робить її придатною для побудови масштабованих моделей у кластерних середовищах.

Середовище PyTorch активно розвивається завдяки внеску відкритої спільноти науковців і інженерів. Платформа широко використовується в академічних дослідженнях, а також в індустріальних застосуваннях – зокрема в задачах комп’ютерного зору, обробки природної мови, біоінформатики та рекомендаційних систем.

Загалом, PyTorch є не лише інструментом для реалізації моделей, а й повноцінною платформою для підтримки всього життєвого циклу машинного навчання – від початкового прототипування до розгортання у продуктивному середовищі.

OpenCV [25] – широко використовувана бібліотека з відкритим вихідним кодом, призначена для задач комп’ютерного зору, машинного навчання та обробки зображень. Вона була розроблена компанією Intel у 1999 році з метою сприяння

просуванню досліджень у галузі комп'ютерного зору та забезпечення доступного інструменту для використання у промислових та академічних проектах.

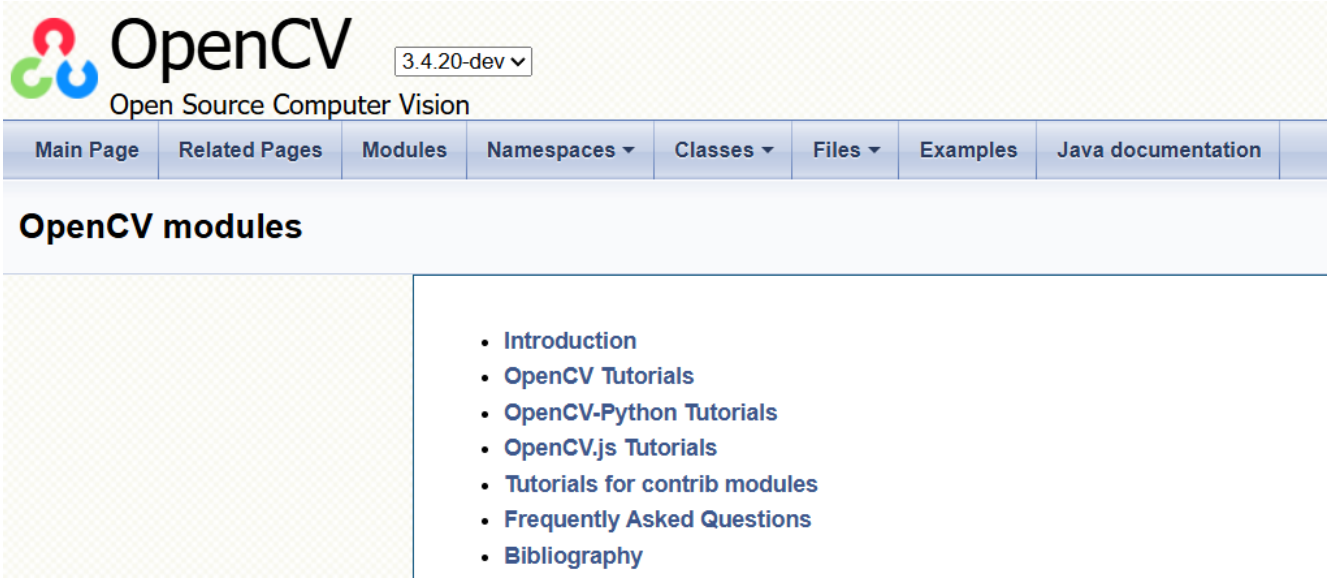

Рисунок 2.8 – Документація OpenCV

Бібліотека реалізована на мові програмування C++ з прив'язками до Python, Java та інших мов, що забезпечує її гнучке застосування в різноманітних системах. OpenCV включає понад 2500 оптимізованих алгоритмів, які охоплюють широкий спектр задач, таких як виявлення та розпізнавання об'єктів, відстеження руху, аналіз сцен, побудова глибини, морфологічна обробка зображень, сегментація, виявлення контурів, трансформації зображень, калібрування камери та багато іншого.

OpenCV часто використовується у поєднанні з іншими бібліотеками, такими як TensorFlow або PyTorch, для попередньої обробки зображень перед передачею їх у моделі глибокого навчання. У контексті аграрних застосунків бібліотека може слугувати інструментом для автоматичної обробки фотоматеріалів, виділення контурів листя, підрахунку кількості об'єктів, виявлення пошкоджених ділянок, а також створення масок для навчання сегментаційних моделей.

Таким чином, OpenCV є невід'ємною складовою у реалізації сучасних комп'ютерних систем зору та становить важливу платформу для розробки

програмного забезпечення, зокрема в задачах діагностики хвороб рослин за зображенням.

Висновки розділу 2

У цьому розділі були розглянуті основні інформаційні технології, які планується використати для розв'язання поставленої задачі створення інтелектуальної системи діагностики хвороб рослин. Мова програмування Python, завдяки своїй універсальності та наявності потужних бібліотек, є основою для реалізації алгоритмів машинного навчання. Бібліотека PyTorch дозволяє створювати та навчати моделі нейронних мереж, що будуть використовуватися для виявлення хвороб. Для обробки зображень, які будуть використовуватися для навчання моделі, планується застосування бібліотеки OpenCV.

3 НАВЧАННЯ НЕЙРОМЕРЕЖ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних та структури системи

Для вирішення задачі діагностики хвороб рослин використано набір даних PlantVillage [26], що містить близько 54 тисяч зображень листя рослин з візуальними ознаками різних хвороб. Загалом набір даних містить 38 класів зображень здорових та хворих рослин (див. рис. 3.1).

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        15.04.2025     12:09                Apple___Apple_scab
d-----        15.04.2025     12:09                Apple___Black_rot
d-----        15.04.2025     12:09                Apple___Cedar_apple_rust
d-----        15.04.2025     12:09                Apple___healthy
d-----        15.04.2025     12:09                Blueberry___healthy
d-----        15.04.2025     12:09                Cherry_(including_sour)___healthy
d-----        15.04.2025     12:09                Cherry_(including_sour)___Powdery_mildew
d-----        15.04.2025     12:09                Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot
d-----        15.04.2025     12:09                Corn_(maize)___Common_rust_
d-----        15.04.2025     12:09                Corn_(maize)___healthy
d-----        15.04.2025     12:09                Corn_(maize)___Northern_Leaf_Blight
d-----        15.04.2025     12:09                Grape___Black_rot
d-----        15.04.2025     12:09                Grape___Esca_(Black_Measles)
d-----        15.04.2025     12:09                Grape___healthy
d-----        15.04.2025     12:09                Grape___Leaf_blight_(Isariopsis_Leaf_Spot)
d-----        15.04.2025     12:09                Orange___Haunglongbing_(Citrus_greening)
d-----        15.04.2025     12:09                Peach___Bacterial_spot
d-----        15.04.2025     12:09                Peach___healthy
d-----        15.04.2025     12:09                Pepper,_bell___Bacterial_spot
d-----        15.04.2025     12:09                Pepper,_bell___healthy
d-----        15.04.2025     12:09                Potato___Early_blight
d-----        15.04.2025     12:09                Potato___healthy
d-----        15.04.2025     12:09                Potato___Late_blight
d-----        15.04.2025     12:09                Raspberry___healthy
d-----        15.04.2025     12:09                Soybean___healthy
d-----        15.04.2025     12:09                Squash___Powdery_mildew
d-----        15.04.2025     12:09                Strawberry___healthy
d-----        15.04.2025     12:09                Strawberry___Leaf_scorch
d-----        15.04.2025     12:09                Tomato___Bacterial_spot
d-----        15.04.2025     12:09                Tomato___Early_blight
d-----        15.04.2025     12:09                Tomato___healthy
d-----        15.04.2025     12:09                Tomato___Late_blight
d-----        15.04.2025     12:09                Tomato___Leaf_Mold
d-----        15.04.2025     12:09                Tomato___Septoria_leaf_spot
d-----        15.04.2025     12:09                Tomato___Spider_mites Two-spotted_spider_mite
d-----        15.04.2025     12:09                Tomato___Target_Spot
d-----        15.04.2025     12:09                Tomato___Tomato_mosaic_virus
d-----        15.04.2025     12:09                Tomato___Tomato_Yellow_Leaf_Curl_Virus
```

Рисунок 3.1 – Структура набору даних

Кожне зображення має розмір 256×256 пікселів та зберігається у форматі .jpg. Набір зображень було попередньо поділено на три підмножини: навчальну (70 % зображень), валідаційну (15 %), тестову (15 %). Поділ виконано із створенням окремих директорій для кожної підмножини, що відповідає формату, прийнятому в бібліотеці torchvision.datasets.ImageFolder. Усі три підмножини було об'єднано у єдину директорію plant_dataset_split, після чого вона була архівована у форматі .zip та завантажена до Google Drive. Такий підхід дає змогу уникнути дублювання процесу підготовки даних при кожному запуску та дозволяє зручно розгортати набір даних у середовищі Colab без втрати структури директорій.

Архітектура системи включає кілька послідовних етапів. На першому етапі виконується підготовка даних, яка передбачає попередню обробку зображень, зокрема нормалізацію, масштабування до єдиного розміру та застосування аугментації для підвищення узагальнювальної здатності моделі.

Другий етап – вибір та навчання моделі. Для класифікації зображень застосовуються CNN, які навчаються з нуля, тобто без використання попередньо натренованих ваг. У якості функції втрат обрано крос-ентропію, що є стандартним підходом для задач багатокласової класифікації.

На етапі оцінювання та валідації проводиться аналіз точності моделі на валідаційній і тестовій вибірках. Для оцінки ефективності навчання будуються графіки функції втрат і точності у процесі тренування, що дозволяє виявити можливе перенавчання або недонавчання моделі.

Завершальний модуль системи відповідає за передбачення класу для нових зображень. Після класифікації результати зберігаються, і користувачу виводиться назва класу, до якого належить вхідне зображення.

3.2 Навчання моделей

Навчання глибоких згорткових нейронних мереж здійснювалося у хмарному середовищі Google Colaboratory із використанням середовища виконання GPU A100 (NVIDIA A100-SXM4-40GB). Обране середовище забезпечило високу

обчислювальну потужність, що є критично важливим для ефективного опрацювання великої кількості зображень та багатошарових моделей без істотного сповільнення навчального процесу.

Для організації ефективної роботи з набором даних та результатами моделювання було обрано використання хмарного сховища Google Drive, яке дозволяє зберігати великі обсяги даних і забезпечує безперервний доступ до них з будь-якого середовища, зокрема з Google Colab.

Інтеграція Google Drive із середовищем виконання Colab здійснюється шляхом монтування диска до файлової системи за допомогою вбудованої функції drive.mount() з пакету google.colab. Команда ініціалізує процес автентифікації, під час якого користувач підтверджує дозвіл на доступ до свого хмарного сховища. Після підтвердження Google Drive монтується як зовнішній диск, що стає доступним у файловій системі середовища виконання за шляхом /content/drive.

```
from google.colab import drive
drive.mount('/content/drive')
```

Після чого здійснюється імпорт необхідних бібліотек, які використовуються на всіх подальших етапах – від попередньої обробки зображень до побудови, навчання моделей глибокого навчання та візуалізації результатів.

```
import os, zipfile              # File and archive handling
import time, copy               # Timing and copying objects (e.g.,
saving best model)
import csv, json                   # Logging and saving results in
CSV/JSON formats

import torch
import torch.nn as nn           # Building neural network layers
import torch.optim as optim     # Optimizers like SGD, Adam
from torch.utils.data import DataLoader  # Efficient data loading

# Torchvision for computer vision
from torchvision import datasets, transforms, models
```

```
# datasets  - prebuilt image datasets
# transforms - image preprocessing (resize, crop, normalize, etc.)
# models - pretrained models like ResNet, Regnet, etc.

# Visualization
import matplotlib.pyplot as plt   # Plotting training metrics and
images

# Image processing
from PIL import Image             # Load and process images
```

Після підключення Google Drive, із нього зчитується попередньо підготовлений архів з розділеним набором даних (plant_dataset_split.zip). Далі цей архів розпаковується в робочу директорію середовища виконання. Завдяки цьому користувач отримує миттєвий доступ до всієї структури даних без повторного поділу чи обробки, що істотно скорочує час підготовчого етапу.

```
With
zipfile.ZipFile('/content/drive/MyDrive/plant_dataset_split.zip',
'r') as zip_ref:
    zip_ref.extractall('/content/')
```

Створено спеціальну директорію для збереження всіх результатів моделювання.

```
results_dir = '/content/drive/MyDrive/plant_results'
os.makedirs(results_dir, exist_ok=True)
```

Ця директорія у хмарному сховищі Google Drive дозволяє зберігати лог-файли, зображення графіків та зважені моделі, що забезпечує можливість подальшого аналізу та повторного використання результатів навіть після завершення сесії Colab.

Для підвищення узагальнення моделей та зменшення перенавчання було застосовано методи аугментації зображень у тренувальному наборі. Застосовано такі перетворення:

```
train_transform = transforms.Compose([
```

```
    transforms.RandomResizedCrop(224, scale=(0.8, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    transforms.RandomRotation(45),
    transforms.ColorJitter(brightness=0.4,            contrast=0.4,
saturation=0.4, hue=0.3),
    transforms.RandomAffine(degrees=20,   translate=(0.15,   0.15),
scale=(0.8, 1.2), shear=10),
    transforms.RandomGrayscale(p=0.1),
    transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0)),
    transforms.ToTensor(),
    transforms.Normalize(imagenet_mean, imagenet_std)
])
```

Ці трансформації імітують реальні варіації у вхідних зображеннях (освітлення, орієнтація, масштаб), що дозволяє моделі краще адаптуватися до нових прикладів. Для валідаційного та тестового наборів використовувалися стандартні перетворення без аугментації, аби забезпечити об’єктивну оцінку якості моделей.

```
val_test_transform = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(imagenet_mean, imagenet_std)
])
```

Для організації ефективної подачі зображень у модель було створено три датасети відповідно до підкаталогів train, val і test, а також генератори батчів:

```
image_datasets = {
    'train':  datasets.ImageFolder(os.path.join(data_dir,  'train'),
transform=train_transform),
    'val':    datasets.ImageFolder(os.path.join(data_dir,    'val'),
transform=val_test_transform),
```

```
    'test':  datasets.ImageFolder(os.path.join(data_dir,  'test'),
transform=val_test_transform)
}
dataloaders  =  {x:  DataLoader(image_datasets[x],  batch_size=32,
shuffle=True, num_workers=2) for x in ['train', 'val', 'test']}
```

Це забезпечує автоматичне зчитування зображень та відповідних міток, а також багатопотокову обробку та змішування батчів для тренування, що покращує загальну продуктивність навчання.

Функція train_model(), яка реалізує повний цикл навчання моделі. Вона приймає як вхідні дані модель, функцію втрат (CrossEntropyLoss), оптимізатор (Adam) та кількість епох:

```
def  train_model(model,  criterion,  optimizer,  num_epochs=50,
model_name="model"):
    since = time.time()
    train_losses, val_losses = [], []
    train_accs, val_accs = [], []

    log_file              =              os.path.join(results_dir,
f"{model_name}_training_log.txt")
    csv_file              =              os.path.join(results_dir,
f"{model_name}_training_stats.csv")

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    with  open(log_file,  'w')  as  logf,  open(csv_file,  'w',
newline='') as csvf:
        csv_writer = csv.writer(csvf)
        csv_writer.writerow(['Epoch',  'Train  Loss',  'Train  Acc',
'Val Loss', 'Val Acc'])

        for epoch in range(num_epochs):
            print(f"\nEpoch {epoch+1}/{num_epochs}")
```

```
            print('-' * 10)
            logf.write(f"\nEpoch        {epoch+1}/{num_epochs}\n{'-
'*10}\n")

            for phase in ['train', 'val']:
                model.train() if phase == 'train' else model.eval()
                running_loss = 0.0
                running_corrects = 0

                for inputs, labels in dataloaders[phase]:
                    inputs,     labels     =     inputs.to(device),
labels.to(device)
                    optimizer.zero_grad()

                    with torch.set_grad_enabled(phase == 'train'):
                        outputs = model(inputs)
                        _, preds = torch.max(outputs, 1)
                        loss = criterion(outputs, labels)
                        if phase == 'train':
                            loss.backward()
                            optimizer.step()

                    running_loss += loss.item() * inputs.size(0)
                    running_corrects    +=    torch.sum(preds    ==
labels.data)

                epoch_loss = running_loss / dataset_sizes[phase]
                epoch_acc     =     running_corrects.double()     /
dataset_sizes[phase]

                if phase == 'train':
                    train_losses.append(epoch_loss)
                    train_accs.append(epoch_acc.item())
                else:
```

```
                    val_losses.append(epoch_loss)
                    val_accs.append(epoch_acc.item())
                    if epoch_acc > best_acc:
                        best_acc = epoch_acc
                        best_model_wts                                  =
copy.deepcopy(model.state_dict())

                print(f"{phase}    Loss:    {epoch_loss:.4f}    Acc:
{epoch_acc:.4f}")
                logf.write(f"{phase}  Loss:  {epoch_loss:.4f}  Acc:
{epoch_acc:.4f}\n")

            csv_writer.writerow([epoch+1,         train_losses[-1],
train_accs[-1], val_losses[-1], val_accs[-1]])
            csvf.flush()

    time_elapsed = time.time() - since
    print(f"\nTraining  complete  in  {time_elapsed  //  60:.0f}m
{time_elapsed % 60:.0f}s. Best val Acc: {best_acc:.4f}")
    logf.write(f"\nTraining complete in {time_elapsed // 60:.0f}m
{time_elapsed % 60:.0f}s. Best val Acc: {best_acc:.4f}\n")

    model.load_state_dict(best_model_wts)
    return model, train_losses, val_losses, train_accs, val_accs
```

У процесі навчання моделі реалізовано окремий цикл для кожної з фаз – тренування та валідації, що дозволяє точно контролювати хід оптимізації та відстежувати зміну якості моделі на незалежних даних. Для кожної фази обчислюються основні метрики, зокрема функція втрат та точність класифікації. Після кожної епохи здійснюється перевірка досягнутої точності на валідаційному наборі, і в разі покращення результату зберігаються відповідні ваги моделі як найкращі. Усі метрики, включно з втратою та точністю для обох фаз, логуються та зберігаються у форматах .txt і .csv, що забезпечує можливість подальшого аналізу та відтворення експериментів.

Було реалізовано навчання кількох архітектур CNN, зокрема ResNet-18, EfficientNet-B0, MobileNetV2 та RegNet-Y-400MF. Всі ці моделі завантажувалися з бібліотеки torchvision.models без попередньо навчених ваг (weights=None), тобто навчання здійснювалося на підготовленому наборі зображень рослин.

```
from torchvision.models import regnet_y_400mf

model_names = {
    'resnet18': models.resnet18,
    'efficientnet_b0': models.efficientnet_b0,
    'mobilenet_v2': models.mobilenet_v2,
    'regnet_y_400mf': regnet_y_400mf
}

comparison_stats = {}

for name, constructor in model_names.items():
    print(f"\n--- Training model: {name} ---")

    if name == 'resnet18':
        model = constructor(weights=None)
        model.fc          =          nn.Linear(model.fc.in_features,
len(class_names))

    elif name == 'efficientnet_b0':
        model = constructor(weights=None)
        model.classifier[1]                                          =
nn.Linear(model.classifier[1].in_features, len(class_names))

    elif name == 'mobilenet_v2':
        model = constructor(weights=None)
        model.classifier[1]                                          =
nn.Linear(model.classifier[1].in_features, len(class_names))
```

```
    elif name == 'regnet_y_400mf':
        model = constructor(weights=None)
        model.fc          =          nn.Linear(model.fc.in_features,
len(class_names))  # RegNet використовує .fc як фінальний шар

    model = model.to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    trained_model, train_losses, val_losses, train_accs, val_accs =
train_model(
        model, criterion, optimizer, num_epochs=50, model_name=name
    )

    torch.save(trained_model.state_dict(),
os.path.join(results_dir, f"{name}.pth"))
    with                              open(os.path.join(results_dir,
f"{name}_class_to_idx.json"), "w") as f:
        json.dump(image_datasets['train'].class_to_idx, f)

    save_plot({'train_loss': train_losses, 'val_loss': val_losses},
'loss', 'Loss', name)
    save_plot({'train_acc':   train_accs,   'val_acc':   val_accs},
'accuracy', 'Accuracy', name)

    print(f"\n--- Testing model: {name} ---")
    test_loss, test_acc = test_model(trained_model)

    comparison_stats[name] = {
        'train_loss': train_losses,
        'val_loss': val_losses,
        'train_acc': train_accs,
        'val_acc': val_accs,
        'test_loss': test_loss,
```

```
        'test_acc': test_acc
    }
```

Кожна з обраних моделей була адаптована до конкретного завдання багатокласової класифікації шляхом модифікації фінального шару відповідно до кількості класів у наборі даних. Зокрема, у моделі ResNet-18 було замінено останній повнозв’язний шар fc, у EfficientNet-B0 та MobileNetV2 – елемент classifier[1], а у RegNet-Y-400MF – також фінальний шар fc, який відповідає за класифікацію. Така адаптація дозволила кожній з моделей здійснювати передбачення саме для 38 класів, що відповідають категоріям у наборі зображень рослин.

Навчання кожної моделі відбувалося впродовж 50 епох із паралельним веденням детального журналу в лог-файл .txt та збереженням статистики в .csv-формат. Після завершення навчання зберігалися найкращі ваги моделі у форматі .pth, а також відображалися й зберігалися графіки динаміки втрати та точності на етапах тренування і валідації. Це дозволяло не лише порівняти якість моделей, а й візуально проаналізувати стабільність їх збіжності.

Після навчання кожна модель додатково тестувалася на окремому тестовому наборі для об'єктивного вимірювання узагальнювальної здатності. Отримані фінальні метрики тестування (втрата та точність) заносилися до підсумкової структури comparison_stats, яка дозволяла сформувати графіки порівняння між усіма моделями за ключовими характеристиками. Завдяки цьому етапу стало можливим визначити найбільш ефективну архітектуру для розв'язання поставленої задачі класифікації зображень рослин.

На завершальному етапі було реалізовано порівняльну аналітику ефективності всіх натренованих моделей на основі зібраних статистичних метрик. Для кожної архітектури – ResNet-18, EfficientNet-B0, MobileNetV2 та RegNet-Y-400MF – були збережені та проаналізовані значення функції втрат і точності на всіх етапах навчання: тренувальному, валідаційному та тестовому.

```
for metric in ['train_loss', 'val_loss', 'train_acc', 'val_acc']:
```

```
    values = {name: stats[metric] for name, stats in
comparison_stats.items()}
    ylabel = 'Loss' if 'loss' in metric else 'Accuracy'
    save_plot(values, f'comparison_{metric}', ylabel, 'all_models')

print("\nAll training, evaluation, and saving completed!")
```

Зібрані дані, які накопичувалися у словнику comparison_stats, містили повну динаміку зміни метрик за кожну епоху, що дозволило провести детальний аналіз процесу навчання. На основі цих даних будувалися узагальнені графіки порівняння:

— графік тренувальної втрати для всіх моделей;

— графік валідаційної втрати;

— графік тренувальної точності;

— графік валідаційної точності.

Кожен із графіків формувався за допомогою функції save_plot(), яка зберігала візуалізації у форматі .png до директорії на Google Drive. Це надало можливість не лише чисельно, а й візуально оцінити переваги та недоліки кожної моделі.

3.3 Аналіз отриманих результатів

Для оцінки ефективності моделей було проведено повний цикл навчання протягом 50 епох з використанням аугментаційного підходу для тренувальної вибірки та стандартизованої передобробки для валідаційної та тестової. Було протестовано чотири моделі глибокого навчання: ResNet18, EfficientNet-B0, MobileNetV2 та RegNet-Y-400MF, які навчалися без попередньо натренованих ваг.

Модель ResNet18 продемонструвала стале зменшення втрат на тренувальній та валідаційній вибірках. Починаючи з точності 29.3% на першій епосі, вона досягла валідованої точності 98.04% на 45-й епосі. Найнижча валідаційна втрата становила 0.0595, що свідчить про здатність моделі до узагальнення. Суттєве покращення точності спостерігалося після 10-ї епохи, що може свідчити про поступове засвоєння важливих ознак.

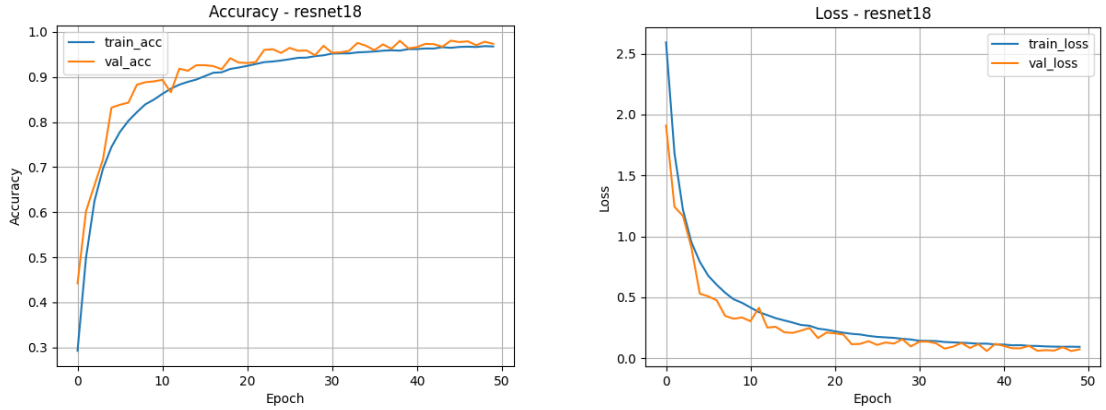

Рисунок 3.2 – Результати навчання ResNet18

Модель EfficientNet-B0 також показала високі результати, досягнувши кращої валідаційної точності 98.27% на 49-й епосі. Стартова точність на валідації становила 45.9%, що є вищим, ніж у ResNet18, і може бути пов’язано з більшою початковою ємністю мережі. Загалом, зниження втрат відбувалося стабільно, а навчання не виявляло ознак перенавчання навіть на останніх епохах.

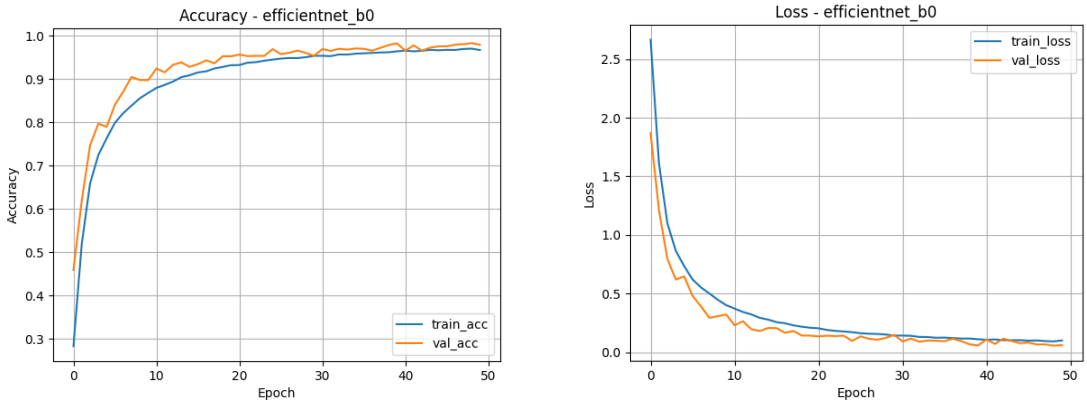

Рисунок 3.3 – Результати навчання EfficientNet-B0

MobileNetV2, як полегшена архітектура, забезпечила валідаційну точність до 97.79%. Незважаючи на меншу кількість параметрів у порівнянні з ResNet18 та EfficientNet, модель змогла досягти майже аналогічного рівня точності. Навчання проходило ефективно, з чіткою тенденцією зменшення втрат, особливо після 15-ї епохи. Найнижча валідаційна втрата становила 0.0643.

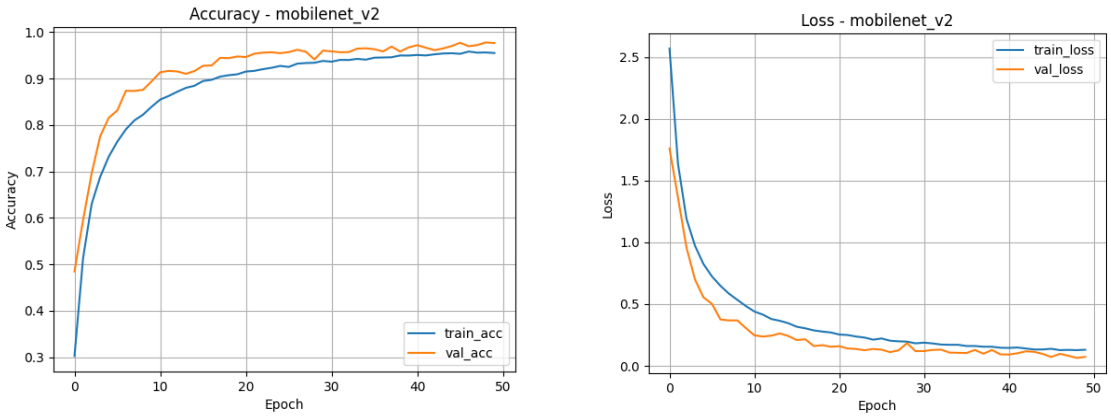

Рисунок 3.4 – Результати навчання MobileNetV2

Остання з протестованих моделей RegNet-Y-400MF – показала найвищу валідаційну точність 98.46% вже на 50-й епосі. Валідаційна втрата опустилась до 0.0476. Початкові втрати були дещо вищими, ніж у інших моделей, однак швидке зменшення з 3-ї по 15-у епоху вказує на ефективне засвоєння ключових патернів у даних.

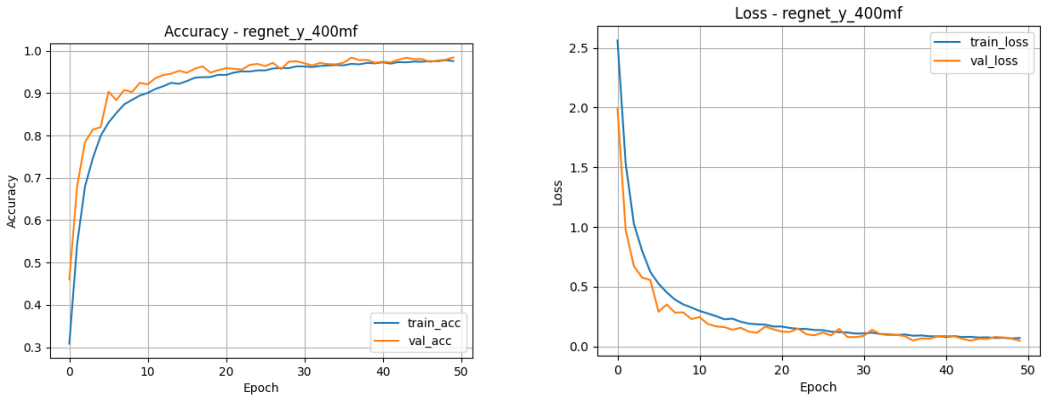

Рисунок 3.5 – Результати навчання RegNet-Y-400MF

У результаті проведеного навчання та оцінювання моделей глибокого навчання було отримано високі показники точності на валідаційній вибірці для кожної з них. Найкращі результати показали RegNet-Y-400MF (98.46%) та EfficientNet-B0 (98.27%), що свідчить про їхню високу ефективність у

поставленому завданні. Модель MobileNetV2, хоча й трохи поступалася за точністю (97.79%), має перевагу в менших обчислювальних витратах і швидкості навчання. ResNet18 забезпечила максимальну точність 98.04% і продемонструвала стабільне навчання протягом усіх епох.

Потенційними напрямками подальшого покращення результатів є більш гнучке налаштування гіперпараметрів, зокрема швидкості навчання та параметрів регуляризації, а також використання стратифікованої крос-валідації для більш точного контролю якості моделі. Крім того, перспективним є збільшення обсягу навчального набору або розширення варіативності в техніках аугментації зображень.

Таким чином, результати навчання підтверджують доцільність використання розглянутих архітектур для автоматизованого розпізнавання хвороб рослин на основі зображень.

Висновки до розділу 3

У третьому розділі було проведено аналіз вхідних даних, описано структуру системи та реалізовано процес навчання чотирьох моделей глибокого навчання. Отримано високі результати точності на валідаційних даних, що підтверджує ефективність обраного підходу. Було також проаналізовано сильні та слабкі сторони моделей, і визначено можливі шляхи подальшого покращення якості класифікації.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ДІАГНОСТИКИ ХВОРОБ РОСЛИН

4.1 Технології розробки системи

Python [28] – це інтерпретована мова програмування високого рівня загального призначення, створена Гвідо ван Россумом у 1991 році. Розробка мови розпочалась у 1989 році в дослідницькому центрі Centrum Wiskunde & Informatica (CWI, Нідерланди). З 2001 року її розвитком займається організація Python Software Foundation. Основною ідеєю при створенні Python було забезпечення максимальної простоти та читабельності коду. Синтаксис мови наближений до природної мови, що робить її зручною для початківців, при цьому вона є достатньо потужною для складних проєктів. Однією з її переваг є модульність, що дозволяє розбивати великі програмні компоненти на частини, які можна повторно використовувати.

Швидкому розвитку мова завдячує численній спільноті, яка активно займається розробкою бібліотек та фреймворків. Це дозволяє використовувати Python у сфері наукових обчислень, візуалізації даних, веброзробки, створення автоматизованих систем та розробки інтелектуальних застосунків.

Рисунок 4.1 – Бібліотеки Python призначені для машинного навчання [29]

PyCharm [30] – це інтегроване середовище розробки для мови програмування Python. Працює під операційними системами Windows, Mac OS і Linux. Це середовище надає програмістам повний спектр інструментів для написання, редагування, налагодження та тестування програмного забезпечення. Основною перевагою PyCharm є підтримка інтелектуального автодоповнення, статичного аналізу коду, швидкого рефакторингу та інтеграції з системами контролю версій. PyCharm підтримує віртуальні середовища, інтеграцію з Docker, а також містить зручний редактор коду з підсвічуванням синтаксису, систему налагодження, модульне тестування та зручну навігацію між файлами проєкту. Середовище активно використовується для розробки як у сфері наукових досліджень, так і у вебпрограмуванні та автоматизації завдань.

Окрім цього, PyCharm підтримує роботу з базами даних завдяки вбудованому інструменту Database Tools, що дозволяє напряму підключатися до SQL-серверів, виконувати запити та переглядати структуру таблиць. Важливою перевагою є також підтримка фреймворків, таких як Django, Flask, Pyramid, що робить середовище зручним для веброзробників. PyCharm забезпечує інтеграцію з Jupyter Notebook, що є надзвичайно корисним для аналізу даних та машинного навчання. У редакції Professional передбачено також підтримку JavaScript, HTML, CSS та інших вебтехнологій. Вбудована система профілювання дає змогу оптимізувати продуктивність програм. Завдяки автоматичному аналізу залежностей, PyCharm допомагає уникнути помилок конфігурації та забезпечує високу якість програмного коду.

Flask [31] – це мікрофреймворк, призначений для створення вебзастосунків мовою програмування Python.

Рисунок 4.2 – Документація фреймворку Flask

Ключовою ідеєю, яка лежить в основі Flask, є принцип «мікроядра» – фреймворк забезпечує лише базовий функціонал (обробка HTTP-запитів, маршрутизація, рендеринг шаблонів, робота з сесіями), залишаючи розробнику свободу у виборі додаткових інструментів (ORM, засобів автентифікації, обробки форм, тощо). Такий підхід особливо корисний для реалізації високоспеціалізованих інформаційних систем, зокрема, систем аналізу зображень на основі нейромереж, де необхідна тісна інтеграція з модулями штучного інтелекту.

Для задач, пов’язаних із діагностикою хвороб рослин, Flask використовується як інтерфейсна оболонка, яка надає можливість доступу до моделі машинного навчання через вебзастосунок або REST API.

Завдяки своїй архітектурі Flask чудово підходить для розробки серверної частини ІС, яка взаємодіє з фронтендом або іншими зовнішніми модулями. Він може використовуватися у зв’язці з базами даних (PostgreSQL, SQLite, MongoDB), а також іншими Python-бібліотеками для обробки вхідних даних (OpenCV, NumPy, PIL).

Вебінтерфейс є важливою складовою частиною інтелектуальної системи діагностики хвороб рослин, оскільки він забезпечує зручний доступ користувачів до системи через браузер. Для розробки вебінтерфейсу використовуються три основні технології: HTML, CSS та JavaScript. Ці технології дозволяють створити інтерактивний, функціональний та привабливий інтерфейс, що взаємодіє з серверною частиною системи та забезпечує користувачам можливість

завантажувати зображення рослин, отримувати результати діагностики та здійснювати налаштування. У таблиці 4.1 представлено опис основних технологій, що використовуються для розробки вебінтерфейсу інтелектуальної системи діагностики хвороб рослин.

Таблиця 4.1 – Технології розробки вебінтерфейсу

Технологія	Опис	Функції та застосування
JavaScript	мова програмування для додавання динамічних елементів на веб-сторінки;	обробка подій; взаємодія з сервером; динамічне оновлення інформації на сторінці;
HTML	мова розмітки веб-сторінок. Визначає структуру контенту на сторінці;	структурує елементи; організовує логічний порядок елементів інтерфейсу;
CSS	мова стилізації веб-сторінок. Визначає, як елементи на сторінці виглядатимуть;	оформлення шрифтів, кольорів, розмірів, відступів, розташування елементів; адаптивний дизайн для різних екранів;

4.2 Розробка вебзастосунку для діагностики хвороб рослин

Ключовим компонентом системи є натренована глибока згорткова нейронна мережа EfficientNet-B0. Її завантаження, ініціалізація, попередня обробка вхідного зображення та отримання результатів передбачення реалізовані в класі PlantDiseaseModel.

На початку здійснюється імпорт необхідних бібліотек: torch для обчислень із тензорами та глибоким навчанням, torchvision.models для виклику архітектури моделі, PIL для обробки зображень та json для завантаження назв класів:

```
import torch
from torchvision import transforms, models
from PIL import Image
import json
```

Метод __init__ відповідає за повну ініціалізацію моделі для подальшого використання. Визначається обчислювальний пристрій, створюється архітектура EfficientNet-B0 з 38 вихідними класами, а також завантажуються попередньо збережені ваги моделі:

```
self.device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
self.model = models.efficientnet_b0(weights=None, num_classes=38)
self.model.load_state_dict(torch.load(model_path,
map_location=self.device))
self.model.to(self.device)
self.model.eval()
```

Перемикання моделі у режим eval() є важливим для коректної роботи під час передбачення, оскільки вимикає операції, притаманні режиму навчання, зокрема Dropout та оновлення статистик BatchNorm.

Крім завантаження моделі, конструктор також відповідає за зчитування списків назв класів. З англомовного файлу JSON завантажується назва класу, а з українського словника – відповідні локалізовані назви:

```
with open(class_names_path, 'r') as f:
    self.class_names = json.load(f)

if class_names_uk_path:
    with open(class_names_uk_path, 'r', encoding='utf-8') as f:
        self.class_names_uk = json.load(f)
else:
```

```
    self.class_names_uk = {}
```

Для забезпечення уніфікованого подання зображення до моделі створюється об’єкт self.transform, який формує стандартну послідовність попередньої обробки зображення відповідно до параметрів ImageNet:

```
self.transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])
```

Цей ланцюг трансформацій змінює розмір зображення, центровано обрізає його до розміру 224×224, перетворює у тензор та виконує нормалізацію значень пікселів.

Основна логіка передбачення реалізована в методі predict_top3. У ньому оброблене зображення передається на обчислювальний пристрій, де виконується прямий прохід через модель. Результати проходять через softmax-функцію для перетворення логітів на ймовірності:

```
image = self.transform(image).unsqueeze(0).to(self.device)
with torch.no_grad():
    outputs = self.model(image)
    probs = torch.nn.functional.softmax(outputs, dim=1)[0]
```

Метод torch.topk обирає три найвірогідніші класи разом із рівнями впевненості. Для кожного класу здійснюється пошук української назви, після чого результати структуруються у вигляді списку словників:

```
top3_probs, top3_idxs = torch.topk(probs, 3)
results = []
for prob, idx in zip(top3_probs, top3_idxs):
    class_key = self.class_names[idx.item()]
    class_uk          =          self.class_names_uk.get(class_key,
class_key.replace('_', ' '))
```

```
    results.append({
        "class_name": class_uk,
        "confidence": round(prob.item() * 100, 2)
    })
```

Таким чином, модель EfficientNet-B0, яка була попередньо навчена на наборі даних із 38 класів хвороб рослин, ефективно інтегрується у вебінтерфейс системи. Такий підхід демонструє гнучкість архітектури застосунку та готовність до масштабування, у тому числі з використанням інших моделей.

Після створення класу PlantDiseaseModel модель інтегрується у вебінтерфейс за допомогою фреймворку Flask. Відповідна логіка розміщена у файлі app.py, який містить основний серверний код вебзастосунку. Модель ініціалізується один раз при старті сервера:

```
model = PlantDiseaseModel(
    model_path=app.config['MODEL_PATH'],
    class_names_path=app.config['CLASS_NAMES_PATH'],
    class_names_uk_path=app.config['CLASS_NAMES_UK_PATH'])
```

Зображення передається користувачем через HTML-форму на головній сторінці сайту. Після натискання кнопки «Діагностувати», дані надсилаються методом POST, обробка відбувається у функції index():

```
if request.method == "POST":
    file = request.files.get("image")
```

Якщо зображення успішно передано, перевіряється його розширення за допомогою утилітарної функції allowed_file(), після чого зображення передається на попередню обробку:

```
image = preprocess_image(file)
```

Функція preprocess_image, реалізована у mage_utils.py, відкриває зображення як об’єкт PIL.Image та повертає його в RGB-форматі. Після цього викликається метод моделі predict_top3, який повертає список із трьох найвірогідніших класів разом із відсотками впевненості:

```
predictions = model.predict_top3(image)
```

Отримані результати зберігаються в Flask-сесії разом із зображенням, закодованим у формат base64. Це дозволяє реалізувати історію запитів прямо в клієнтському інтерфейсі та забезпечує можливість багатосторінкового відображення результатів:

```
session["history"].append({
    "image": img_str,
    "predictions": predictions
})
```

Сформовані дані передаються у шаблон index.html, де відображаються результати діагностики у зручному форматі. Крім назви хвороби, виводиться впевненість моделі у відсотках, що полегшує сприйняття користувачем:

```
<li>{{ item.class_name }} — {{ item.confidence }}%</li>
```

Таким чином, модель EfficientNet-B0, завдяки поетапній реалізації в модулі predictor.py та зручній обгортці у класі PlantDiseaseModel, успішно інтегрується у вебінтерфейс.

Інтерфейс користувача реалізовано за допомогою шаблона index.html, створеного у рамках шаблонізатора Jinja2, який інтегрований у фреймворк Flask. Головне завдання цього інтерфейсу – забезпечити інтуїтивно зрозумілий процес взаємодії з користувачем: завантаження зображення, отримання результату та перегляд історії запитів.

На головній сторінці відображається форма завантаження зображення (див. рис. 4.3).

```
<form method="POST" enctype="multipart/form-data" id="uploadForm">
    <input type="file" name="image" accept="image/*" required
id="imageInput" />

    <label for="imageInput" class="upload-icon" title="Завантажити
зображення"></label>
```

```
    <button type="submit">Визначити хворобу</button>

    <div class="preview-container" id="previewContainer"
style="display:none;">
        <p>Попередній перегляд:</p>
        <img id="previewImage" src="" alt="Попередній перегляд" />
    </div>
</form>
```

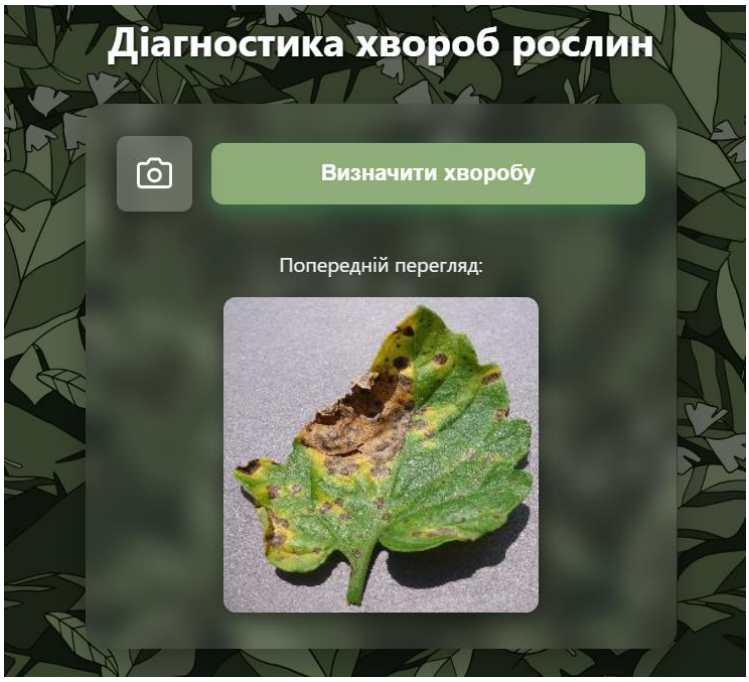

Рисунок 4.3 – Форма діагностики хвороби

Після вибору файлу та натискання кнопки «Визначити хворобу», форма надсилає запит методом POST. Отримане зображення обробляється сервером, модель повертає прогноз, і результати передаються у шаблон (див. рис. 4.4).

```
{% if results %}
    <h2>Результати діагностики:</h2>
    <ul class="results-list">
        {% for item in results %}
            <li>{{ item.class_name }} — {{ item.confidence }}%</li>
        {% endfor %}
```

```
    </ul>
{% endif %}
```

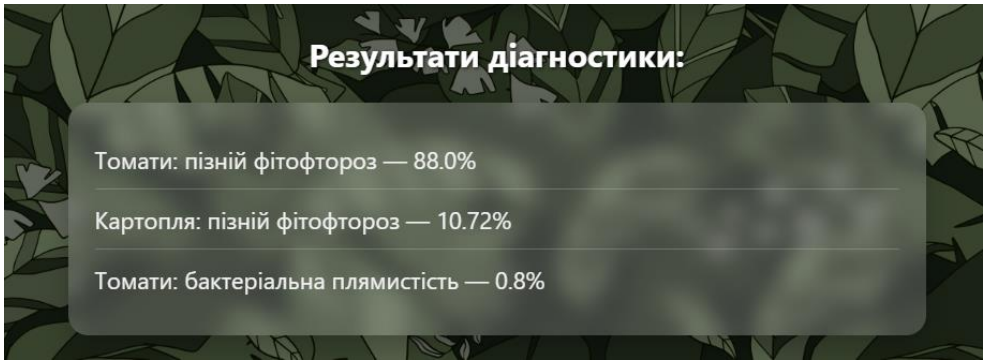

Рисунок 4.4 – Відображення результатів діагностики

Крім поточного запиту, система веде історію звернень до моделі. Вона реалізована через механізм сесій Flask. Кожен запит зберігає як зображення так і список передбачених класів. Це дозволяє користувачу переглядати попередні діагнози (див. рис. 4.5):

```
{% if history %}
    <h2>Історія ваших запитів:</h2>
    {% for entry in history %}
        <div class="history-entry">
            <img  src="{{  entry.image  }}"
alt="Запит" />
            <ul class="history-list">
                {% for item in entry.predictions %}
                    <li>{{ item.class_name }} — {{ item.confidence
}}%</li>
                {% endfor %}
            </ul>
        </div>
    {% endfor %}
{% endif %}
```

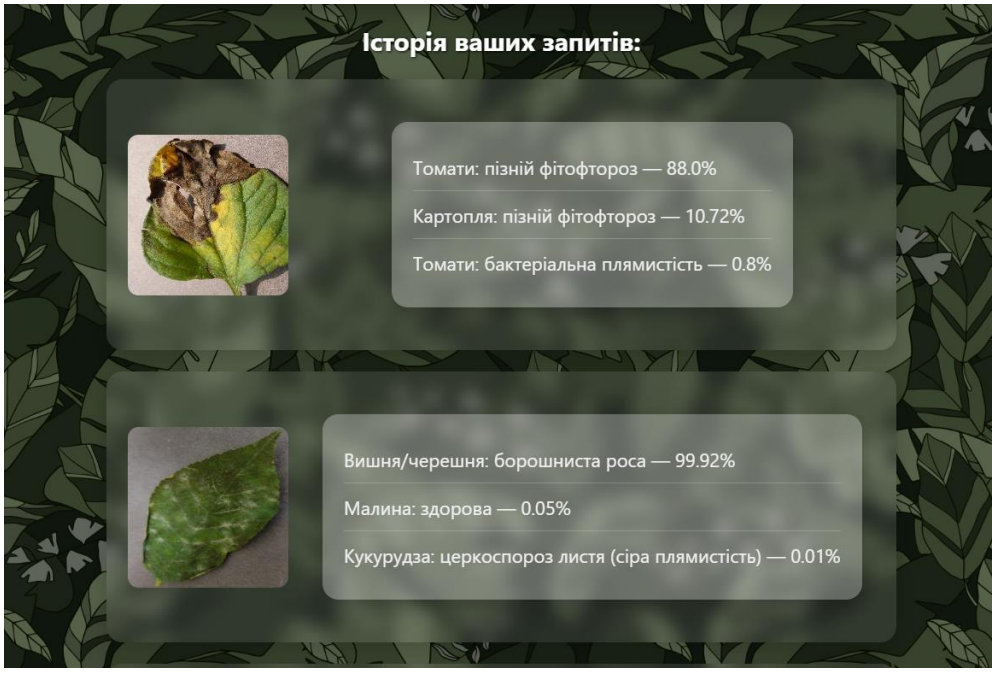

Рисунок 4.5 – Відображення історії запитів

У межах однієї сесії результати зберігаються автоматично у змінній session["history"]. Це реалізовано у коді серверної частини наступним чином:

```
session["history"].append({
    "image": img_str,
    "predictions": predictions
})
session.modified = True
```

Механізм розбиває історію на сторінки (див. рис. 4.6), що дозволяє уникнути перевантаження інтерфейсу при великій кількості запитів. Управління відображенням реалізується через змінні start, end та total_pages, які обчислюються у функції index().

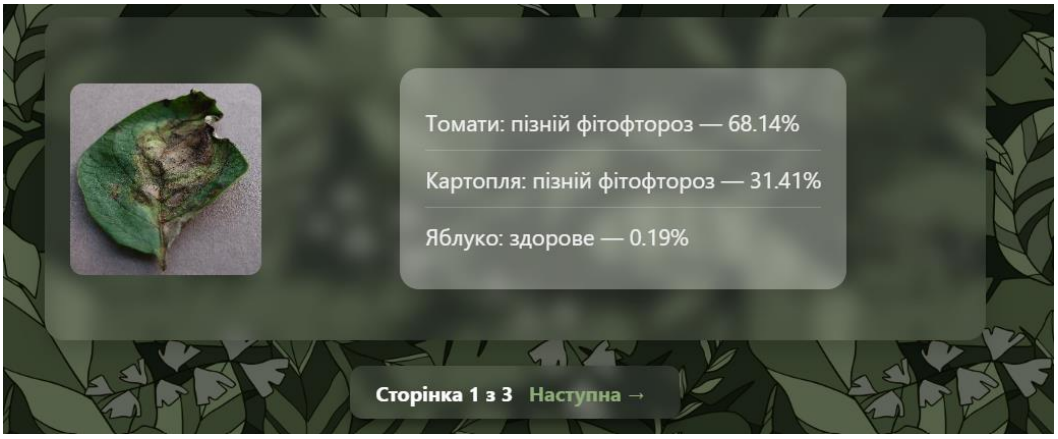

Рисунок 4.6 – Відображення сторінок історії запитів

Також додано обробку типових помилок. Наприклад, якщо користувач намагається завантажити файл, що перевищує допустимий розмір 5 МБ, йому буде виведено відповідне повідомлення:

```
@app.errorhandler(413)
def too_large(e):
    return "Файл завеликий. Максимум — 5MB", 413
```

Розроблений вебінтерфейс забезпечує повноцінну взаємодію користувача з моделлю класифікації хвороб рослин, дозволяє отримувати миттєві результати аналізу та зручно переглядати історію запитів. Застосування сесій дозволяє персоналізувати досвід користування навіть без реєстрації, а базові механізми перевірки забезпечують стабільність роботи при типових помилках користувача. Це робить систему придатною для реального використання в аграрній сфері та освітньому середовищі.

4.3 Тестування

Для перевірки коректності функціонування розробленого застосунку було проведено тестування із використанням зображення листка картоплі, ураженого пізнім фітофторозом – однією з найпоширеніших та економічно небезпечних хвороб пасльонових культур.

Після запуску застосунку користувач отримує доступ до вебінтерфейсу, в якому доступна форма для завантаження зображення. Була обрана фотографія, яка чітко демонструє характерні симптоми хвороби: темні плями з чіткими межами, що поширюються по листовій пластинці (див. рис. 4.7).

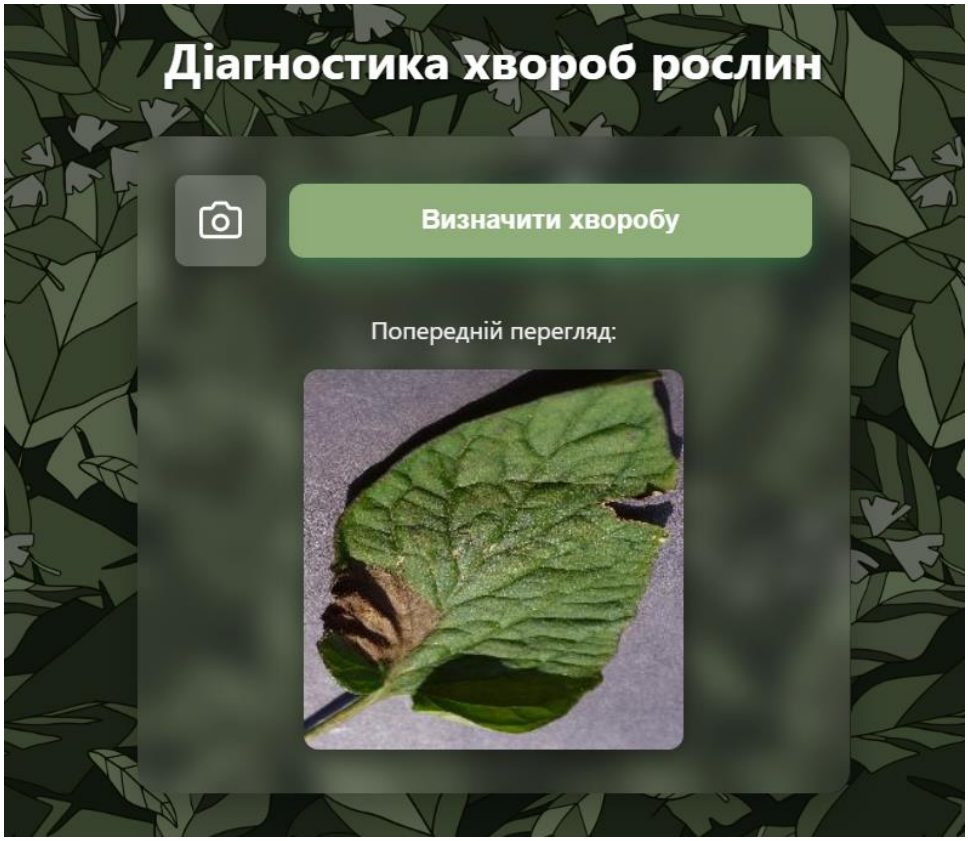

Рисунок 4.7 – Форма діагностики

Зображення завантажується через елемент <input type="file">, після чого активується механізм попереднього перегляду. Це дозволяє користувачеві переконатися у правильності обраного файлу перед надсиланням. Після натискання кнопки «Визначити хворобу» форма надсилає POST-запит на сервер, де вбудована модель EfficientNet-B0, попередньо навчена на наборі з 38 класів рослинних хвороб, виконує класифікацію.

Модель обробляє зображення, нормалізує його згідно з параметрами ImageNet, і передає у нейронну мережу. У відповідь система повертає топ-3 передбачення із відсотковим рівнем впевненості. У випадку тестування зображення

хворого листка картоплі система визначила пізній фітофтороз як перший за ймовірністю варіант, з високим рівнем довіри (див. рис. 4.8).

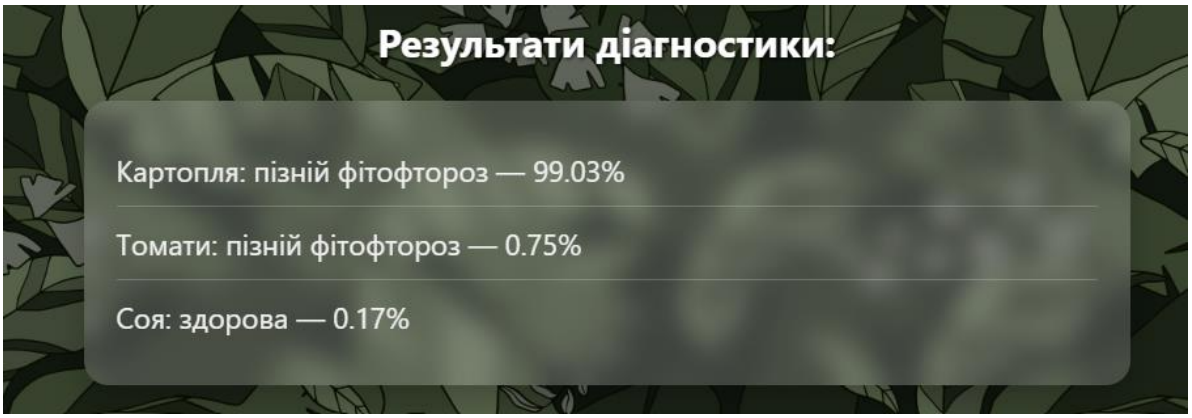

Рисунок 4.8 – Відображення результатів діагностики

Результат відображається в інтерфейсі у вигляді списку, що містить назву хвороби українською мовою та відповідну ймовірність. Крім того, здійснюється автоматичне збереження результатів у сесії користувача, що дозволяє переглянути історію запитів з пагінацією – зображення та пов’язані з ними діагнози зберігаються у форматі base64, що забезпечує стабільне відображення незалежно від джерела файлу (див. рис. 4.9).

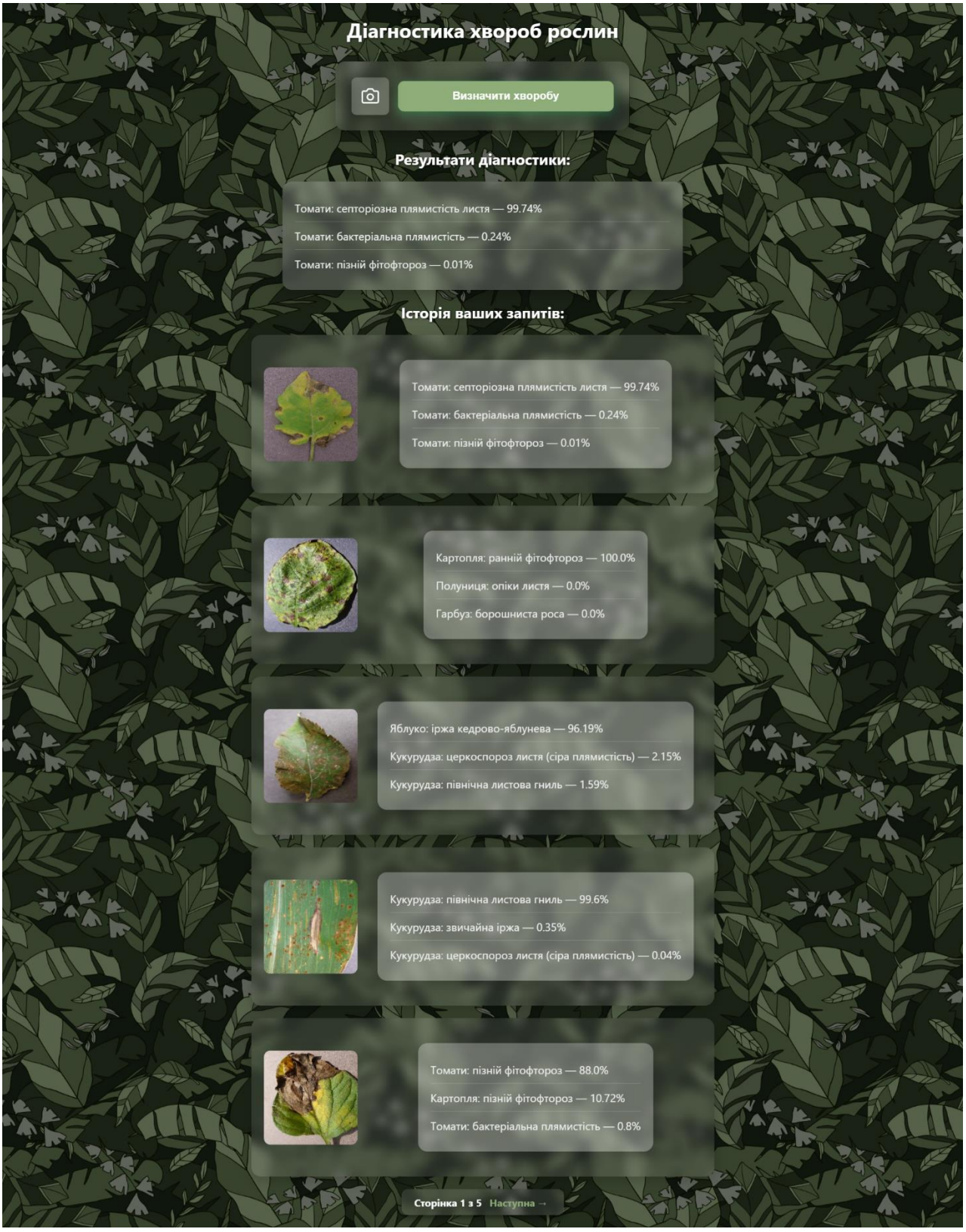

Рисунок 4.9 – Зображення повного застосунку

Таким чином, тестування із зображенням пізнього фітофторозу підтвердило коректність роботи моделі та всього вебзастосунку загалом. Інтелектуальна система продемонструвала здатність точно ідентифікувати хворобу, забезпечуючи користувача швидким і надійним діагностичним висновком. Це свідчить про практичну придатність застосунку до використання у реальних умовах фермерського господарства або аграрного моніторингу.

Висновки до розділу 4

У цьому розділі було реалізовано вебзастосунок для автоматизованої діагностики хвороб рослин на основі глибокої нейронної мережі EfficientNet-B0. Розробку виконано з використанням фреймворку Flask, із врахуванням зручності інтерфейсу та збереження історії запитів. Проведене тестування підтвердило коректність роботи системи та її здатність точно визначати захворювання за зображенням. Система готова до практичного застосування у агросфері.

ВИСНОВКИ

У процесі виконання роботи було реалізовано повноцінну інтелектуальну систему для діагностики хвороб рослин за зображенням листя. Аналіз предметної області засвідчив актуальність поставленої задачі, особливо в контексті сучасного агровиробництва, де автоматизація виявлення хвороб може значно знизити втрати врожаю та витрати на ручну експертизу.

Було проведено детальний огляд існуючих підходів і архітектур нейронних мереж, зокрема таких як ResNet, EfficientNet, MobileNet та RegNet, що дозволило обґрунтовано обрати найефективніші моделі для вирішення задачі багатокласової класифікації. У результаті навчання моделей було доведено доцільність використання моделі EfficientNet-B0, яка показала найкраще співвідношення між точністю, швидкістю та компактністю.

Розроблено програмну реалізацію системи на основі Flask, яка включає завантаження зображення, автоматичну діагностику хвороб та збереження результатів у сесії користувача. Інтерфейс вебзастосунку забезпечує зручну взаємодію та доступність для користувачів без технічної підготовки. Система підтримує українську локалізацію, інтерактивний попередній перегляд зображень, історію запитів та адаптивний дизайн. Результати тестування показали високу точність класифікації та стабільність роботи застосунку. Зокрема, при тестуванні на зображеннях листя з пізнім фітофторозом система змогла правильно визначити захворювання серед топ-3 передбачень з високим рівнем впевненості.

Таким чином, поставлені в роботі задачі були досягнуті. Розроблена система є ефективним інструментом для автоматизованої діагностики рослин у польових умовах. Робота має перспективи для подальшого розвитку, зокрема шляхом розширення набору даних та інтеграції мобільного застосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Pest and Pesticide Management. URL: https://www.fao.org/pest-and-pesticide-management/about/understanding-the-context/en/ (дата звернення: 15.04.2025).

2. Plant Parasites and Agents of Decomposition. URL: https://nigerianscholars.com/tutorials/protists/plant-parasites-and-agents-of-decomposition/ (дата звернення: 15.04.2025).

3. Explainable vision transformer enabled convolutional neural network for plant disease identification: PlantXViT. URL: https://arxiv.org/abs/2207.07919 (дата звернення: 15.04.2025).

4. Інноваційна платформа Watsons від IBM: планування урожаю, боротьба зі шкідниками та прогноз цін. URL: https://landlord.ua/news/innovatsiina-platforma-watsons-vid-ibm-planuvannia-urozhaiu-borotba-zi-shkidnykamy-ta-prohnoz-tsin/ (дата звернення: 17.04.2025).

5. Plantix: Інструмент діагностики культур AI. URL: https://agtecher.com/uk/product-uk/plantix/ (дата звернення: 17.04.2025).

6. В. О. Харченко. Основи машинного навчання: навчальний посібник. Суми: Сумський державний університет, 2023. 264 с. URL: https://files.znu.edu.ua/files/Bibliobooks/Inshi79/0059501.pdf (дата звернення: 17.04.2025).

7. М.В. Лавренюк. Алгоритми машинного навчання. Глибокі нейромережі в задачах механіки суцільних середовищ: навчальний посібник. Київ: КНУ ім.Тараса Шевченка, 2024. 100 с. URL: https://mechmat.knu.ua/wp-content/uploads/2024/05/alhorytmy-mashynnoho-navchannia.-hlyboki-nejromerezhi-v-zadachakh-mekhaniky-sutsilnykh-seredovyshch.pdf (дата звернення: 17.04.2025).

8. K-Nearest Neighbor(KNN) Algorithm. URL: https://www.geeksforgeeks.org/k-nearest-neighbours/ / (дата звернення: 17.04.2025).

9. Наївний алгоритм Байєса в машинному навчанні. URL: https://www.guru99.com/uk/naive-bayes-classifiers.html (дата звернення: 17.04.2025).

10. Багатошаровий персептрон (Multilayer Perceptron, MLP) у машинному навчанні. URL: https://surl.lu/gzcrus (дата звернення: 18.04.2025).

11. Multilayer Perceptron, MLP. URL: https://thetransmitted.com/wp-content/uploads/2023/12/mlp.webp (дата звернення: 18.04.2025).

12. Architecture of LSTM/GRU and RNN. URL: https://medium.com/@abidaubaid229/architecture-of-lstm-gru-and-rnn-7ad0b8124782 (дата звернення: 18.04.2025).

13. Introduction to Recurrent Neural Networks. URL: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ (дата звернення: 18.04.2025).

14. Convolutional neural networks vs Vision transformers: evolving paradigm in computer vision. URL: https://medium.com/@talhawaqas1998/convolutional-neural-networks-vs-vision-transformers-evolving-paradigm-in-computer-vision-744a775195e3 (дата звернення: 18.04.2025).

15. Convolutional Neural Network | Deep Learning. URL: https://developersbreach.com/convolution-neural-network-deep-learning/ (дата звернення: 18.04.2025).

16. Residual Networks (ResNet) - Deep Learning. URL: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/ (дата звернення: 20.04.2025).

17. Residual Networks (ResNet) - Deep Learning. URL: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/ (дата звернення: 20.04.2025).

18. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. URL: https://arxiv.org/abs/1905.11946v5 (дата звернення: 20.04.2025).

19. EfficientNet: Optimizing Deep Learning Efficiency. URL: https://viso.ai/deep-learning/efficientnet/ (дата звернення: 20.04.2025).

20. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. URL: https://arxiv.org/abs/1905.11946v5 (дата звернення: 20.04.2025).

21. Architecture of EfficientNet-B0 with MBConv as Basic building blocks. URL: https://www.researchgate.net/figure/Architecture-of-EfficientNet-B0-with-MBConv-as-Basic-building-blocks_fig4_344410350 (дата звернення: 20.04.2025).

22. RegNet: The Most Flexible Network Architecture For Computer Vision. URL: https://medium.com/data-science/regnet-the-most-flexible-network-architecture-for-computer-vision-2fd757f9c5cd (дата звернення: 20.04.2025).

23. Image Classification With MobileNet. URL: https://builtin.com/machine-learning/mobilenet (дата звернення: 20.04.2025).

24. Learn PyTorch for Deep Learning: Zero to Mastery book. URL: https://www.learnpytorch.io/ (дата звернення: 20.04.2025).

25. OpenCV modules. URL: https://docs.opencv.org/4.x/index.html (дата звернення: 20.04.2025).

26. PlantVillage.URL:https://www.kaggle.com/datasets/mohitsingh1804/plantvillage (дата звернення: 20.04.2025).

27. What is Google Colab and How to Use It for Data Science and Machine Learning. URL: https://medium.com/@aleksej.gudkov/what-is-google-colab-and-how-to-use-it-for-data-science-and-machine-learning-20aaa61afc2e (дата звернення: 20.04.2025).

28. What is Python? Executive Summary. URL: https://www.python.org/doc/essays/blurb/ (дата звернення: 24.04.2025).

29. The Best Python Libraries for Machine Learning and AI: Features & Applications. URL: https://www.scalablepath.com/python/python-libraries-machine-learning (дата звернення: 24.04.2025).

30. PyCharm: all about the most popular Python IDE. URL: https://datascientest.com/en/pycharm-all-about-the-most-popular-python-ide (дата звернення: 24.04.2025).

31. Toast to Deployment: Deploying Deep Learning Models with Flask. URL: https://medium.com/@edwinjaya/toast-to-deployment-deploying-deep-learning-models-with-flask-283d13ab237c (дата звернення: 24.04.2025).

32. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 24.04.2025).

ДОДАТОК А Код програмної реалізації

app.py

```
from flask import Flask, render_template, request, session
from flask_session import Session
from config import Config
from model.predictor import PlantDiseaseModel
from utils.image_utils import allowed_file, preprocess_image
import logging
from datetime import timedelta
import io
import base64

app = Flask(__name__)
app.config.from_object(Config)
app.permanent_session_lifetime = timedelta(hours=1)

 Session(app)

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)



 model = PlantDiseaseModel(
    model_path=app.config['MODEL_PATH'],
    class_names_path=app.config['CLASS_NAMES_PATH'],
    class_names_uk_path=app.config['CLASS_NAMES_UK_PATH']
)

@app.route("/", methods=["GET", "POST"])
def index():
    if "history" not in session:
        session["history"] = []
```

```
    results = None
    error = None

    if request.method == "POST":
        file = request.files.get("image")

        if not file or file.filename == "":
            error = "Файл не обрано."
        elif not allowed_file(file.filename):
            error = "Непідтримуваний формат зображення."
        else:
            try:
                image = preprocess_image(file)

                buffered = io.BytesIO()
                image.save(buffered, format="JPEG")
                img_str                                                    =
base64.b64encode(buffered.getvalue()).decode()

                predictions = model.predict_top3(image)

                session["history"].append({
                    "image": img_str,
                    "predictions": predictions
                })
                session.modified = True

                results = predictions
            except Exception as e:
                logger.error(f"Помилка при обробці зображення: {e}")
                error = "Помилка при обробці зображення."

    page = request.args.get("page", 1, type=int)
    per_page = app.config["PER_PAGE"]
```

```
    history = session.get("history", [])
    total = len(history)
    start = (page - 1) * per_page
    end = start + per_page
    paginated_history = history[::-1][start:end]

    return render_template(
        "index.html",
        results=results,
        error=error,
        history=paginated_history,
        page=page,
        total_pages=(total + per_page - 1) // per_page
    )

@app.errorhandler(413)
def too_large(e):
    return "Файл завеликий. Максимум — 5MB", 413

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

config.py

```
import os

class Config:
    SECRET_KEY = os.getenv("SECRET_KEY", "your_safe_dev_key")
    SESSION_TYPE = "filesystem"
    SESSION_PERMANENT = False
    MODEL_PATH = "models/efficientnet_b0.pth"
    CLASS_NAMES_PATH = "models/class_names.json"
    CLASS_NAMES_UK_PATH = "models/class_names_uk.json"
    MAX_CONTENT_LENGTH = 5 * 1024 * 1024  # 5 MB
    PER_PAGE = 5
```

predictor.py

```
import torch
from torchvision import transforms, models
from PIL import Image
import json

class PlantDiseaseModel:
    def __init__(self, model_path, class_names_path, class_names_uk_path=None):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = models.efficientnet_b0(weights=None, num_classes=38)
        self.model.load_state_dict(torch.load(model_path, map_location=self.device))
        self.model.to(self.device)
        self.model.eval()

        with open(class_names_path, 'r') as f:
            self.class_names = json.load(f)

        if class_names_uk_path:
            with open(class_names_uk_path, 'r', encoding='utf-8') as f:
                self.class_names_uk = json.load(f)
        else:
            self.class_names_uk = {}

        self.transform = transforms.Compose([
            transforms.Resize(256),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406],
                                 [0.229, 0.224, 0.225])
```

```
        ])

    def predict_top3(self, image: Image.Image):
        image = self.transform(image).unsqueeze(0).to(self.device)
        with torch.no_grad():
            outputs = self.model(image)
            probs = torch.nn.functional.softmax(outputs, dim=1)[0]
            top3_probs, top3_idxs = torch.topk(probs, 3)
            results = []
            for prob, idx in zip(top3_probs, top3_idxs):
                class_key = self.class_names[idx.item()]
                class_uk    =    self.class_names_uk.get(class_key,
class_key.replace('_', ' '))
                results.append({
                    "class_name": class_uk,
                    "confidence": round(prob.item() * 100, 2)
                })
            return results
image_utils.py
from PIL import Image
import io

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

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

def preprocess_image(file_storage):
    image                                                          =
Image.open(io.BytesIO(file_storage.read())).convert("RGB")
    file_storage.seek(0)
    return image
index.html
```

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8" />
    <meta  name="viewport"  content="width=device-width,  initial-
scale=1" />
    <title>Діагностика хвороб рослин</title>
    <link     rel="stylesheet"     href="{{     url_for('static',
filename='css/style.css') }}" />
</head>
<body>
    <h1>Діагностика хвороб рослин</h1>

<div class="form-wrapper">
    <form        method="POST"        enctype="multipart/form-data"
id="uploadForm">
        <input  type="file"  name="image"  accept="image/*"  required
id="imageInput" />

        <label         for="imageInput"         class="upload-icon"
title="Завантажити зображення">
            <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24
24" fill="none"
                 stroke="currentColor"   stroke-width="2"   stroke-
linecap="round" stroke-linejoin="round">
                <path d="M23 19a2 2 0 0 1-2 2H3a2 2 0 0 1-2-2V7a2 2
0 0 1 2-2h4l2-3h6l2 3h4a2 2 0 0 1 2 2z"/>
                <circle cx="12" cy="13" r="4"/>
            </svg>
        </label>

        <button type="submit">Визначити хворобу</button>
    </form>
```

```
    <div       class="preview-container"       id="previewContainer"
style="display:none;">
        <p>Попередній перегляд:</p>
        <img id="previewImage" src="" alt="Попередній перегляд" />
    </div>
</div>



<div         class="preview-container"         id="previewContainer"
style="display:none;">
    <p>Попередній перегляд:</p>
    <img id="previewImage" src="" alt="Попередній перегляд" />
</div>
    {% if error %}
        <p class="error-message">{{ error }}</p>
    {% endif %}

    {% if results %}
        <h2>Результати діагностики:</h2>
        <ul class="results-list">
            {% for item in results %}
                <li>{{  item.class_name  }}  —  {{  item.confidence
}}%</li>
            {% endfor %}
        </ul>
    {% endif %}

    {% if history %}
        <h2>Історія ваших запитів:</h2>
        {% for entry in history %}
            <div class="history-entry">
                <img src="{{ entry.image }}"
alt="Запит" />
                <ul class="history-list">
```

```
                    {% for item in entry.predictions %}
                        <li>{{ item.class_name }} — {{
item.confidence }}%</li>
                    {% endfor %}
                </ul>
            </div>
        {% endfor %}

        {% if total_pages > 1 %}
            <div class="pagination">
                {% if page > 1 %}
                    <a href="{{ url_for('index', page=page - 1)
}}">← Попередня</a>
                {% endif %}
                <span class="current-page">Сторінка {{ page }} з {{
total_pages }}</span>
                {% if page < total_pages %}
                    <a href="{{ url_for('index', page=page + 1)
}}">Наступна →</a>
                {% endif %}
            </div>
        {% endif %}
    {% endif %}

    <button id="scrollTopBtn" title="Повернутись нагору">↑</button>

    <script src="{{ url_for('static', filename='js/main.js')
}}"></script>
</body>
</html>
style.css
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    margin: 0;
```

```
    padding: 0;
    color: #f1f1f1;

    background-image: url("../img/bg.jpg");
    background-repeat: repeat;
    background-size: 600px 600px; /
    position: relative;
}

body::before {
    content: "";
    position: fixed;
    top: 0; left: 0;
    width: 100%; height: 100%;
    background: rgba(0, 0, 0, 0.6);
    z-index: -1;
}

h1, h2 {
    text-align: center;
    color: #ffffff;
    margin-bottom: 20px;
    text-shadow: 1px 1px 3px rgba(0, 0, 0, 0.8);
}

form {
    border-radius: 15px;
    display: flex;
    flex-direction: column;
    gap: 15px;
    animation: fadeIn 1s ease-out;
}

input[type="file"] {
```

```
    cursor: pointer;
    color: white;
}

button {
    background-color: #8ba373;
    color: white;
    border: none;
    padding: 14px;
    border-radius: 10px;
    font-size: 1.1rem;
    cursor: pointer;
    transition: all 0.3s ease;
    font-weight: bold;
    box-shadow: 0 5px 15px rgba(46, 204, 113, 0.3);
}

button:hover {
    background-color: #8ba373;
    transform: scale(1.03);
}

.preview-container {
    text-align: center;
}
.preview-container img {
    max-width: 100%;
    max-height: 250px;
    border-radius: 10px;
    box-shadow: 0 6px 15px rgba(0,0,0,0.4);
}

.error-message {
    text-align: center;
```

```
    color: #ff6b6b;
    font-weight: bold;
    margin: 10px 0;
}

ul.results-list, ul.history-list {
    list-style: none;
    padding: 20px;
    max-width: 600px;
    margin: 20px auto;
    background: rgba(255, 255, 255, 0.2);
    backdrop-filter: blur(8px);
    border-radius: 15px;
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.25);
    animation: fadeIn 0.8s ease-out;
}
ul li {
    padding: 10px 0;
    border-bottom: 1px solid rgba(255,255,255,0.2);
    font-size: 1.1rem;
}
ul li:last-child {
    border-bottom: none;
}

.history-entry {
    display: flex;
    align-items: center;
    gap: 20px;
    margin: 20px auto;
    max-width: 700px;
    padding: 20px;
    background: rgba(255,255,255,0.1);
    border-radius: 15px;
```

```
    box-shadow: 0 6px 20px rgba(0,0,0,0.2);
    backdrop-filter: blur(8px);
    animation: fadeInUp 0.7s ease-out;
}
.history-entry img {
    max-width: 150px;
    border-radius: 10px;
}

.pagination {
    background: rgba(255,255,255,0.1);
    backdrop-filter: blur(6px);
    -webkit-backdrop-filter: blur(6px);
    padding: 10px 20px;
    margin: 20px auto;
    text-align: center;
    border-radius: 10px;
    width: fit-content;
    box-shadow: 0 4px 10px rgba(0,0,0,0.5);
    color: #fff;
    font-size: 16px;
}



.pagination a {
    margin: 0 8px;
    text-decoration: none;
    color: #8fad78;
    font-weight: bold;
}
.pagination a:hover {
    color: #8fad78;
}
.pagination .current-page {
```

```
    color: #fff;
    font-weight: bold;
}

#scrollTopBtn {
    position: fixed;
    bottom: 25px;
    right: 25px;
    background: #8fad78;
    border: none;
    color: white;
    width: 45px;
    height: 45px;
    border-radius: 50%;
    font-size: 20px;
    display: none;
    justify-content: center;
    align-items: center;
    box-shadow: 0 5px 15px rgba(0,0,0,0.3);
}
#scrollTopBtn:hover {
    background: #8ba373;
}

@media (max-width: 600px) {
    .history-entry {
        flex-direction: column;
        align-items: center;
    }
    .history-entry img {
        max-width: 100%;
    }
}
```

```
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(10px); }
    to { opacity: 1; transform: translateY(0); }
}
@keyframes fadeInUp {
    from { opacity: 0; transform: translateY(20px); }
    to { opacity: 1; transform: translateY(0); }
}
input[type="file"] {
  display: none;
}

.custom-file-upload {
  display: inline-block;
  background-color: rgba(255, 255, 255, 0.1);
  color: #fff;
  padding: 10px 20px;
  border-radius: 8px;
  cursor: pointer;
  font-weight: bold;
  transition: all 0.3s ease;
  backdrop-filter: blur(4px);
  -webkit-backdrop-filter: blur(4px);
  box-shadow: 0 4px 10px rgba(0,0,0,0.3);
  border: 1px solid rgba(255,255,255,0.2);
}

.custom-file-upload:hover {
  background-color: rgba(255, 255, 255, 0.2);
  transform: scale(1.03);
}

form {
    display: flex;
```

```
    flex-direction: row;
    align-items: center;
    gap: 15px;
    animation: fadeIn 1s ease-out;
}

input[type="file"] {
    display: none;
}

.upload-icon {
    display: inline-flex;
    justify-content: center;
    align-items: center;
    width: 60px;
    height: 60px;
    background-color: rgba(255, 255, 255, 0.12);
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.3s ease;
    box-shadow: 0 6px 20px rgba(0,0,0,0.4);
    backdrop-filter: blur(8px);
    -webkit-backdrop-filter: blur(8px);
}

.upload-icon svg {
    width: 28px;
    height: 28px;
    stroke: white;
}

.upload-icon:hover {
    background-color: rgba(255, 255, 255, 0.2);
    transform: scale(1.05);
```

```
}

button[type="submit"] {
    flex-grow: 1;
    padding: 14px;
    border-radius: 10px;
    font-size: 1.1rem;
    font-weight: bold;
    background-color: #8fad78;
    color: white;
    border: none;
    cursor: pointer;
    box-shadow: 0 5px 15px rgba(46, 204, 113, 0.3);
    transition: all 0.3s ease;
}

button[type="submit"]:hover {
    background-color: #8ba373;
    transform: scale(1.03);
}

.form-wrapper {
    max-width: 420px;
    margin: 30px auto;
    background: rgba(255, 255, 255, 0.1);
    backdrop-filter: blur(10px);
    padding: 25px;
    border-radius: 15px;
    box-shadow: 0 8px 24px rgba(0, 0, 0, 0.3);

    display: flex;
    flex-direction: column;
    gap: 15px;
    animation: fadeIn 1s ease-out;
```

```
}

form#uploadForm {
    display: flex;
    flex-direction: row;
    align-items: center;
    gap: 15px;
    margin: 0;
}

.preview-container {
    text-align: center;
}

.preview-container img {
    max-width: 100%;
    max-height: 250px;
    border-radius: 10px;
    box-shadow: 0 6px 15px rgba(0,0,0,0.4);
}
```

main.js

```
const imageInput = document.getElementById('imageInput');
const previewContainer =
document.getElementById('previewContainer');
const previewImage = document.getElementById('previewImage');

imageInput.addEventListener('change', () => {
    const file = imageInput.files[0];
    if (file) {
        const reader = new FileReader();
        reader.onload = e => {
            previewImage.src = e.target.result;
            previewContainer.style.display = 'block';
        };
```

```
        reader.readAsDataURL(file);
    } else {
        previewImage.src = '';
        previewContainer.style.display = 'none';
    }
});

const scrollTopBtn = document.getElementById("scrollTopBtn");

window.onscroll = function() {
    if (document.body.scrollTop > 200 ||
document.documentElement.scrollTop > 200) {
        scrollTopBtn.style.display = "flex";
    } else {
        scrollTopBtn.style.display = "none";
    }
};

scrollTopBtn.addEventListener("click", () => {
    window.scrollTo({ top: 0, behavior: 'smooth' });
});
```