Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: **Інтелектуальна система виявлення захворювання листя рослин**

Виконав: студент VI курсу, групи КН-62м
спеціальності 122 – "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)
Підставський В. Р.
(прізвище та ініціали)

Керівник Пірко І. Б.
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Львів – 2024 р.

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Борецька І. Б.
"____" _____2024 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Підставський Віталій Романович
(прізвище, ім'я, по батькові)

1. Тема роботи **Інтелектуальна система виявлення захворювання листя рослин**

керівник роботи Пірко І. Б., канд. фіз.-мат. наук, доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 13.02. 2023 року № С-49

2. Термін подання студентом роботи 05. 01. 2024 р.

3. Вихідні дані до роботи:

- вивчити предметну область, проаналізувати існуючі фактори та методи моделювання, а також відповідні програмні продукти;
- розглянути і використати алгоритми, які лежать в основі математичної моделі впливу негативних факторів на ріст та розвиток рослин;
- спроектувати інформаційну систему з допомогою мови програмування Python та відповідних бібліотек для побудови інтерфейсу та візуалізації результатів.
- протестувати роботу інформаційної системи.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Розділ 1. Стан проблемної області

Розділ 2. Інформаційне забезпечення

Розділ 3. Математичне забезпечення

Розділ 4. Програмне забезпечення

Розділ 5. Стартап проекту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Додаток А.

Додаток Б.

6. Дата видачі завдання 15 лютого 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних даних та інших джерел згідно досліджуваної теми	15.02-31.03.2023	виконано
2	Аналіз досліджуваної теми та вибір відповідних варіантів її розробки	01.04-30.04. 2023	виконано
3	Постановка задачі та її формалізація	01.05-31.05. 2023	виконано
4	Вибір та обґрунтування методів і засобів проведення дослідження	01.06-30.06. 2023	виконано
5	Розроблення концептуальної схеми реалізації завдання	01.07-31.07. 2023	виконано
6	Програмна реалізація завдання	01.08-30.09. 2023	виконано
7	Тестування програмного продукту та отриманих результатів	01.10-30.10. 2023	виконано
8	Розробка пояснювальної записки магістерської роботи	01.11-30.11. 2023	виконано
9	Корегування пояснювальної записки згідно вимог, розроблення презентації	01.12-04.01. 2024	виконано

Студент _____ Підставський В. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пірко І. Б.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 73 сторінки пояснювальної записки, 10 рисунків, 2 таблиці, 10 джерел, 3 додатки.

В дипломній роботі розроблено та реалізовано інформаційну систему для розпізнавання основних видів хвороб листя рослин по їх зображеннях. Розглянуто існуючі методи та запропоновано метод цифрової обробки зображень для діагностики хвороб рослин за цифровими зображеннями листя у видимому спектрі, порівняння отриманих еталонних описів математичних очікувань, а також подальшого розпізнавання вибірки.

Ключові слова: класифікація захворювань листя рослин, обробка зображень, бінаризація ключових ознак, Keras, computer vision.

ABSTRACT

Diploma paper contains 73 pages of explanatory note, 10 figures, 2 tables, 10 sources, 3 appendix.

In the complex course project, an information system was developed and implemented for recognizing the main types of diseases of potato leaves on their images. The existing methods are considered and a method of digital image processing is proposed for the diagnosis of plant diseases based on digital images of leaves in the visible spectrum, comparison of the obtained reference descriptions of mathematical expectations by the maximum of the membership function, as well as further recognition of the sample.

Keywords: classification of plant leaf diseases, image processing, binarization of key features, Keras, computer vision.

ТЕХНІЧНЕ ЗАВДАННЯ

В дипломній роботі потрібно вирішити такі завдання.

Розробити інформаційну систему для розпізнавання основних видів хвороб листя рослин по їх зображеннях:

- провести огляд літератури по даній тематиці;
- проаналізувати існуючі системи прогнозування захворювання рослин;
- розробити математичну модель системи та алгоритм її функціонування для діагностики захворювань рослин;
- реалізувати програмну модель інформаційної системи;
- представити результати роботи даної інформаційної системи.

ПЕРЕЛІК СКОРОЧЕНЬ І СПЕЦІАЛЬНИХ ТЕРМІНІВ

- ANN – Artificial Neural Network (штучна нейронна мережа);
- CNN – Convolutional Neural Network (згорткова нейронна мережа);
- CNTK – Cognitive Toolkit;
- EBA (error backpropagation algorithm) – алгоритм зворотнього розповсюдження помилки;
- GPU – Graphics Processing Unit;
- RL (reinforcement learning) – навчання з підкріпленням;
- SL (supervised learning) – навчання з вчителем;
- UL (unsupervised learning) – навчання без вчителя;
- ML – машинне навчання;
- БД – база даних;
- ПЗ – програмне забезпечення;
- ШНМ – штучна нейронна мережа.

ЗМІСТ

ВСТУП

Актуальність дипломної роботи

Існує велика кількість захворювань, які впливають на урожайність рослин, що призводить до економічних та екологічних втрат. Хвороби рослин призводять до втрати урожаю сільськогосподарських культур кожний рік. В цьому контексті діагностика захворювань рослин, яка дає точний і своєчасний результат, має першочергове значення. В даній роботі проведено дослідження по виявленню захворювань листя помідорів та їх класифікації.

Загальним підходом виявлення патологій у рослин є використання дистанційного методу виявлення, який досліджує фотозображення, які одержані за допомогою безпілотних літальних апаратів. Вони оснащуються камерами, висока чіткість зображення яких дозволяє виявляти проблемні ділянки полів. Більшість захворювань листя рослин породжує якийсь їх прояв у видимому спектрі. У переважній більшості випадків діагноз або перше припущення про захворювання здійснюється людьми візуально. Важливо враховувати, що сільськогосподарські культури займають великі площі, роблячи моніторинг за ними складним завданням. Проблема може бути вирішена за рахунок використання цифрових фотозображень, розпізнавання образів та автоматичними інструментами класифікації, один з яких і був розроблений в даній роботі.

Предметом дослідження є проектування інтелектуальної системи для розпізнавання захворювань листя рослин.

Об’єктом дослідження є набір зображень листя помідорів.

Мета роботи – розроблення інтелектуальної системи для можливості розпізнавання основних видів хвороб листя по їх зображеннях.

Завдання до дипломної роботи:

- проаналізувати існуючі системи прогнозування захворювання рослин;
- розробити математичну модель системи та алгоритм її функціонування для діагностики захворювань рослин;
- реалізувати програмну модель інформаційної системи;
- представити результати роботи даної системи.

Наукова новизна одержаних результатів

Наукова новизна даної роботи визначається рядом аспектів, таких як використання передових технологій та методів для створення практичного застосування. Застосування популярних технологій, таких як TensorFlow для глибокого навчання та Streamlit для створення веб-інтерфейсу інтелектуальної системи визначає новизну в контексті навчання та використання інструментів для розв’язання практичних завдань. Наукова новизна полягає в популяризації та візуалізації алгоритмів машинного навчання для класифікації зображень. Застосування машинного навчання для класифікації стану рослин може мати наукове застосування у сільському господарстві, де автоматизацію та штучний інтелект використати для моніторингу та управління врожаєм.

Створення інтерактивного веб-додатку для демонстрації моделі класифікації дозволяє користувачам взаємодіяти з моделлю класифікації в реальному часі, може вважатися новаторською в контексті навчання та практичного застосування. Використання бібліотеки Streamlit може відзначатися як новаторський підхід, оскільки вона спрощує процес створення веб-додатків для демонстрації результатів аналізу даних чи моделей машинного навчання. Наукова новизна пов’язана також у способі застосування вже відомих технологій для розв’язання практичних задач.

Практичне значення одержаних результатів

Практичне значення отриманих результатів у цій роботі може бути визначено в контексті аграрного сектору та використання технологій машинного навчання для підтримки управління вирощуванням помідорів та контролю за їхнім станом. Використання моделі класифікації для розпізнавання захворювань помідорів або їхнього стану може бути важливим інструментом для фермерів. Це дозволить рано виявляти можливі проблеми та вживати вчасних заходів для їх усунення. Аналіз фотографій помідорів може служити для визначення ефективності вирощування, розпізнавання факторів, що впливають на їх врожайність, та оптимізацію процесів у сільському господарстві. Веб-інтерфейс, який дозволяє взаємодіяти з моделлю класифікації, може бути важливим засобом для навчання та розуміння аспектів вирощування помідорів. Фахівці та фермери можуть використовувати це для покращення своїх навичок та прийняття кращих рішень. Автоматизована система, яка використовує модель класифікації, може спростити процеси моніторингу та аналізу стану помідорів, забезпечуючи більш ефективний та швидкий спосіб обробки великої кількості даних.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Методи навчання глибоких нейронних мереж

Глибокі нейронні мережі стали одним із найпопулярніших методів машинного навчання. Однозначного визначення, що таке глибока нейронна мережа, немає. Під глибокою нейронною мережею розуміють таку мережу, яка містить більше одного прихованого шару. Штучні нейронні мережі, задані таким чином, здатні наблизити будь-яку неперервну функцію з будь-якою точністю. Проте немає конструктивного підходу, який дозволив би гарантовано створювати нейронні мережі із заздалегідь заданими властивостями.

В даний час для навчання нейронних мереж, у тому числі глибоких, використовується алгоритм зворотнього розповсюдження помилки (error backpropagation algorithm), що ґрунтується на методі градієнтного спуску. Алгоритм зворотнього поширення помилки використовує навчання з учителем, для нього потрібна навчальна множина з заздалегідь відомими правильними відповідями. Вводиться міра помилки, яка визначає, наскільки вихідні значення мережі відрізняються від правильних відповідей.

Потім міра помилки мінімізується за допомогою методу градієнтного спуску шляхом зміни значень ваг у мережі. Щоб оцінити, наскільки сильно кожна вага впливає на вихідне значення, розраховуються похідні помилки по вагах. Потім проводиться зміна ваги на невеликі значення з урахуванням градієнта. Так повторюється доти, доки помилка на виході не скоротиться до допустимих значень. Початкові значення вагів нейронів у мережі задаються випадковим чином.

У глибокій нейронній мережі з декількома прихованими шарами проводиться розрахунок помилки, що передається від одного шару до іншого. На першому етапі розраховується значення помилки на виході нейронної мережі, котрі знають правильні відповіді. Потім розраховується

помилка на вході у вихідний шар мережі, яка використовуватиметься як помилка на виході прихованого шару. У такий спосіб розрахунок триває доти, коли буде відома помилка на вхідному шарі. Саме тому алгоритм має назву зворотн поширення помилки.

Нейронна мережа, яка представлена на рис. 1.1, називається повнозв'язною. У такій мережі кожен нейрон наступного шару пов'язаний із усіма нейронами попереднього шару. Однак це не єдиний варіант з'єднання нейронів у мережу.

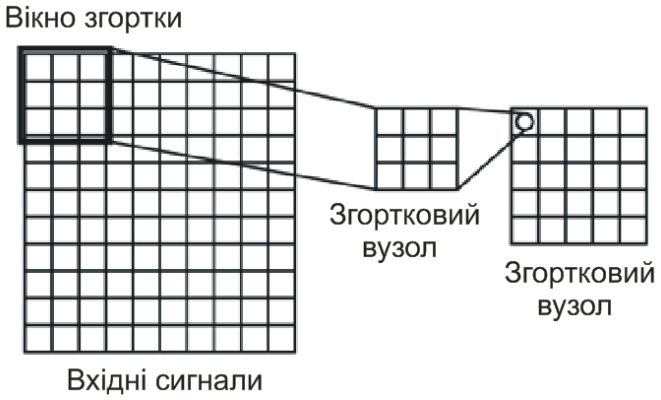


Рис. 1.1. Схема згорткового шару нейронної мережі.

У згорткових шарах вікно згорткового вузла (convolutional unit) із заданим набором ваг (ядро згортки) переміщається по двовимірному масиву вхідних даних, наприклад, пікселях зображення (рис. 1.1). Значення елементів, які співпадають, в даних та ядрі згортки перемножуються, отримані результати додаються і надходять у нейрон наступного шару. Всі згорткові вузли використовують однакові ядра згортки, тому для опису згорткової мережі потрібно небагато параметрів. Як правило, у згорткових шарах використовується не одне, а кілька ядер згортки.

Виходи згорткових шарів підключаються до входів шарів підвибірки. Причому до одного нейрона в шарі підвибірки підключаються кілька нейронів згорткового шару. Нейрони у шарі підвибірки спрацьовують у разі активності хоча б одного із вхідних сигналів. На цьому шарі важлива наявність самого сигналу, а не його конкретні координати, тому шари підвибірки менш чутливі до незначних зрушень та змін масштабу

зображення. Навчання згорткових шарів проводиться за допомогою локальних алгоритмів навчання без вчителя, або задаються ваги заздалегідь.

1.2. Програмні системи навчання глибоких нейронних мереж

В даний час створено велику кількість програмних систем для навчання глибоких нейронних мереж. Серед найбільш популярних можна виділити Caff, Theano, TensorFlow, Torch, CNTK. Система Theano створена в університеті Монреалю (Канада). Theano розроблена на Python, але забезпечує високу продуктивність за рахунок того, що програма на Python автоматично перетворюється на програму C++, яка компілюється і потім виконується. TensorFlow створена компанією Google у 2015 році та включає системи ефективної роботи з тензорами та потокової обробки даних на графі.

Автори Torch вважали за краще використовувати Lua замість Python через простоту інтеграції C і Lua. Компанія Microsoft розробила систему CNTK (Cognitive Toolkit) у 2016 році. Заслуговують на увагу також і нові бібліотеки глибокого навчання, які створені недавно, але набирають популярності. Системи PaddlePaddle та MXNet спеціально розроблені для навчання глибоких нейронних мереж на розподілених кластерах. Бібліотека Neon розроблялася компанією Nervana. MXNet та Neon показують хороші результати на тестах продуктивності.

Широке поширення практичного застосування нейронних мереж є можливим завдяки наявності великої кількості готових рішень для навчання глибоких нейронних мереж, у тому числі з можливістю використання сучасних багатоядерних процесорів, прискорювачів обчислень, а також кластерів з розподіленою пам’яттю.

1.3. Методи аналізу та класифікації зображень

Аналіз та класифікація зображень – завдання, широко затребуване у багатьох областях. Нерідко виникає потреба у обробці та дослідженні таких зображень, які не містять об’єктів чітко визначеної форми. Такі зображення містять випадково розташовані фігури різної форми, орієнтації та яскравості. Конкретне застосування аналізу зображень реалізовано у системі виділення особливостей з метою діагностики хвороб рослин.

Розглядаються статистичні та текстурні ознаки, що допомагають визначити тип захворювання рослини. Різні захворювання характеризуються різними проявами, які видно на поверхні листя рослин. Знімки таких типових ознак аналізуються, накопичується інформація про колір та характер різних плям, уражень та деформацій. Отримані дані використовують для діагностики захворювання рослин.

Для визначення меж контрастних об’єктів можна використовувати один із таких методів: комбінаторний метод або метод порогового градієнту, метод виділення контуру шляхом застосування оператора Лапласа та фільтра Гауса, метод, що використовує оператор Собеля. Проводиться класифікація точок контуру та виділення характерних точок.

Виділення опорних точок використовуватиметься для виділення областей зображення, у яких видно ознаки захворювання. Розроблено алгоритм знаходження хвороби за зразком за допомогою аналізу гістограм та текстурних ознак.

Найчастіше, щоб визначити конкретне захворювання рослини, доводиться вивчати довідники, класифікатори, переглядати велику кількість фотографій уражених рослин, на що йде багато часу. Автоматизація процесу розпізнавання захворювання була метою, з якою було розроблено систему.

Розроблено додаток, за допомогою якого відбувається навчання системи та накопичення відомостей про хвороби, а також здійснюється процес розпізнавання захворювання за знімком. Система проводить

обробку зображень, а користувачеві надаються засоби для зручного виділення областей інтересу на зображенні.

Для виділення особливостей зображення використовуються алгоритми виділення контурів та опорних точок, що дозволяють користувачеві виділити найбільш інформативні області та отримати достовірну гістограму, що точно характеризує те чи інше захворювання. Виділення контурів використовується як допоміжний інструмент.

Не варто забувати про інші особливості об’єктів, такі як форма і характер їх розташування. Для обліку цих параметрів використовуються методи аналізу текстур, що дозволяють враховувати більшість значимих характеристик зображень.

1.4. Текстурний аналіз зображень

При аналізі зображень важливою їх характеристикою служить текстура, яка є у всіх зображеннях, починаючи з зображень, які отримані за допомогою літаків та супутникових пристроїв і закінчуючи мікроскопічними зображеннями в біомедичних дослідженнях.

Один із аспектів текстури пов’язаний із просторовим розподілом та просторовою взаємозалежністю значень яскравості локальної області зображення. Статистики просторової взаємозалежності значень яскравості обчислюються за матрицями переходів значень яскравості між найближчими сусідніми точками.

Матриця суміжності рівнів яскравості є оцінкою густини розподілу ймовірностей другого порядку, які отримані за зображенням у припущенні, що густина ймовірності залежить лише від розташування двох пікселів. Цілком очевидно, що такі матриці містять інформацію, що характеризує текстуру. По цій матриці обчислюється близько двадцяти ознак, такі як ступінь однорідності, максимальна ймовірність, контраст та інші.

Обчислюючи ознаки різних зображень, можна отримати багатовимірний вектор ознак текстур. Такі ознаки чітко пов’язані з

візуальними особливостями текстури та використовуються для пошуку подібних зображень.

ВИСНОВКИ ДО РОЗДІЛУ 1

В цьому розділі дипломної роботи розглянуто методи навчання глибоких нейронних мереж. Показано, що на даний час для їх навчання використовується алгоритм зворотнього розповсюдження помилки. У глибокій нейронній мережі з декількома прихованими шарами проводиться розрахунок помилки, що передається від одного шару до іншого. Представлено структуру повнозв'язної нейронної мережі. Розглянуто програмні системи навчання глибоких нейронних мереж. Показано, що серед найбільш популярних є TensorFlow, Torch, CNTK. Широкого поширення для практичного застосування нейронних мереж стало можливим завдяки наявності великої кількості готових рішень для навчання глибоких нейронних мереж. Приведено методи аналізу та класифікації зображень. Розглянено статистичні та текстурні ознаки для визначення різних типів захворювань рослини, що характеризуються різними проявами, які видно на поверхні листя рослин. Отримані дані використовують для діагностики захворювання рослин. Розроблено алгоритм знаходження хвороби за зразком за допомогою аналізу гістограм та текстурних ознак. Розглянуто методи аналізу текстур, що дозволяють враховувати більшість значимих характеристик зображень. Технології раннього діагностування захворювань рослин використовують нейронні мережі для їх виявлення та їх діагностики на ранньому етапі розвитку.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Розпізнавання та класифікація зображень рослин із застосуванням технологій штучного інтелекту

Задача аналізу зображень є однією з класичних задач галузі практичного застосування штучних нейронних мереж. В даний час знаходять широке застосування та активно розробляються автоматичні системи дистанційного контролю стану рослин, що дозволяють своєчасно визначити наявність захворювань рослин, отримувати відомості про біомасу рослин.

Розпізнавання стану рослин передбачає вирішення задачі класифікації зображень, як класифікатор пропонується використати штучну нейронну мережу (ШНМ) як модель, яка може навчатися, робота якої практично не вимагає втручання користувача.

При задачі оцінки стану рослин використовується багатокласова класифікація, тобто зображення буде відноситися до одного з обраних класів найпоширеніших захворювань рослинних культур:

- bacterial spot (бактеріальна плямистість);
- fungle spot (грибкова плямистість);
- healthy (здорове);
- late blight (фітофтороз);
- leaf mold (листова пліснява);
- leaf scorch (опік листя);
- powdery mildew (борошниста роса).
- yellow curl virus(вірус жовтої кучерявості).

На виході класифікатора формується вектор ймовірностей відношення вхідного зображення кожного класу. Клас із найвищим значенням ймовірності буде результатом розв’язку задачі класифікації. Представлені етапи створення та розгортання для загального користування моделі ШНМ

згорткової архітектури, що вирішує завдання багатокласової класифікації стану рослин за кольоровими зображеннями їх листя.

2.2. Розпізнавання зображень на основі згорткових нейронних мереж

В даний час алгоритми Deep learning широко використовуються для вирішення багатьох практичних завдань. Нейронні мережі, які навчаються за допомогою алгоритмів Deep learning, не лише показують більш високу точність порівняно з альтернативними методами навчання мереж, але й у ряді завдань виявляють розуміння семантики інформації, що їм подається.

Теорія Deep learning доповнює звичайні методи машинного навчання спеціальними алгоритмами для аналізу вхідної інформації на декількох рівнях представлення. Нейронна мережа, яка була навчена за допомогою алгоритмів Deep learning, витрачає менше вхідної інформації для навчання, але в той же час вона здатна аналізувати інформацію з набагато вищою точністю.

Згорткові нейронні мережі в основному застосовуються для роботи із зображеннями. Основна причина цього полягає у наявності спеціальних шарів, які допомагають отримати найкращі результати порівняно з іншими топологіями мереж. Структура згорткової нейронної мережі представена на рис. 2.1.

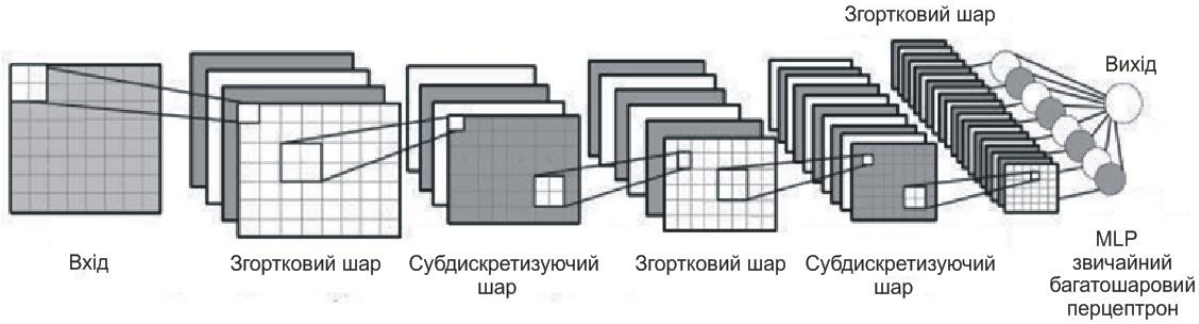


Рис. 2.1. Структура згорткової нейронної мережі.

При побудові нейронної мережі для розпізнавання зображень може бути використаний один із трьох підходів:

- повне навчання нейронної мережі;
- використання ознак вже навченої мережі;
- налаштування останніх шарів навченої мережі.

Повне навчання нейронної мережі

Одним із підходів навчання згорткових нейронних мереж є навчання з нуля. У цьому випадку слід пам’ятати, що навчання на малому об’ємі даних може призвести до проблеми перенавчання. Один із варіантів вирішення проблеми перенавчання – це збільшення кількості даних для навчання.

Метод повного навчання мережі реалізується рідко, оскільки проблематично знайти набір даних достатнього об’єму навчання мережі. Замість цього зазвичай прийнято брати вже навчену нейронну мережу і використовувати її для вирішення поставленого завдання.

Використання ознак вже навченої мережі

Другий підхід полягає в тому, щоб використовувати нейронну мережу, яка була попередньо навчена на великому наборі даних, наприклад ImageNet (1,2 мільйона зображень). Така мережа вже має ознаки, які корисні для більшості проблем комп’ютерного зору.

Основна ідея цього підходу полягає в наступному: використовувати згорткову нейронну мережу, яка попередньо була навчена на наборі даних, потім усунути останній повнозв’язний шар. Після чого запустити один раз отриману модель на наявному навчальному наборі для тестування даних, записуючи при цьому ознаки в масив. Ознаками є ті, які витягуються з малорозмірного прихованого шару, розташованого вже ближче до останніх прихованих шарів нейронної мережі. На завершення необхідно навчити повнозв’язний шар з урахуванням цих збережених ознак.

Налаштування останніх шарів навченої мережі

Ідея третього підходу полягає в налаштуванні останніх шарів, наприклад, згорткових та повнозв’язаних. Працюючи вже з навченою нейронною мережею, необхідно лише перевчити її на новому наборі даних, використовуючи при цьому невеликі оновлення ваги. Іншими варіантами в рамках цього підходу є підстроювання всіх шарів навченої мережі або використання попередніх шарів та налаштування шарів, що знаходяться ближче до виходу мережі.

2.3. Методика сегментації листя рослин та фону

Захворювання сільськогосподарських культур є великою проблемою – для деяких культур хвороби можуть знизити врожай. Для своєчасної протидії необхідні інтелектуальні системи, що дозволяють автоматично виявляти захворювання рослин на ранній стадії шляхом моніторингу їхнього стану з космосу або використовуючи БПЛА. В останні роки стало можливим створення систем, які базуються на нейронних мережах, що вирішують це завдання.

Опис набору даних

Для навчання високоточних класифікаторів необхідно мати набір даних зображень як хворих, так і здорових рослин (рис. 2.2). Для цього в даній роботі використано відкритий датасет PlantVillage Dataset, з якого було взято лише зображення листя помідорів. Класифікація повинна бути багатокласовою, оскільки має бути визначено, чи не тільки здорова рослина, а ще й до якого типу захворювань відноситься хвороба, що вразила її:

- healthy;
- bacterial_spot;
- late_blight;
- septoria_leaf_spot;
- target_spot;

- tomato_yellow_leaf_curl_virus.

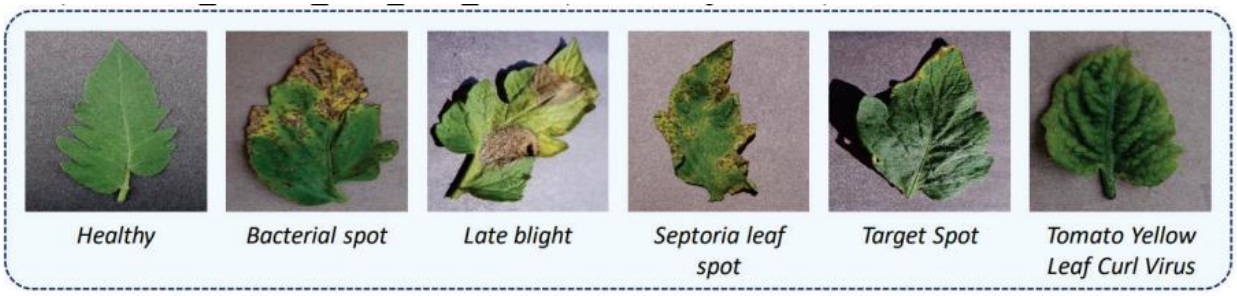


Рис. 2.2. Приклади типових представників класів хвороб помідорів.

Основне завдання – побудувати класифікатор захворювань рослин за зображеннями листя. Природним вибором побудови такого класифікатора є методи традиційного машинного навчання (ML). Вирішення цього завдання включатиме такі етапи:

- попередня обробка даних (автоматичне відділення листка від фону);
- отримання текстурних та статистичних ознак;
- навчання досліджуваного набору класифікаторів на різних наборах ознак;
- оцінка якості роботи навчених класифікаторів;
- аналіз отриманих результатів.

Спочатку задано RGB-зображення листя рослини (помідорів) розміром 256×256 пікселів. Крім самого листка на зображенні знаходиться фон (наприклад, ґрунт). Потрібно отримати зображення листка на чорному фоні. Для цього проводять автоматичну розмітку, цей метод грунтується на побудові ознак фону та листка. Сегментація має багатокроковий характер. Спочатку потрібно позбутися всіх кольорів крім зеленого. Для цього почергово позбавляються білих, чорних та синіх областей, після цього застосовують поріг Оцу для різниці між індексами надлишкового зеленого та червоного кольорів (рис. 2.3).

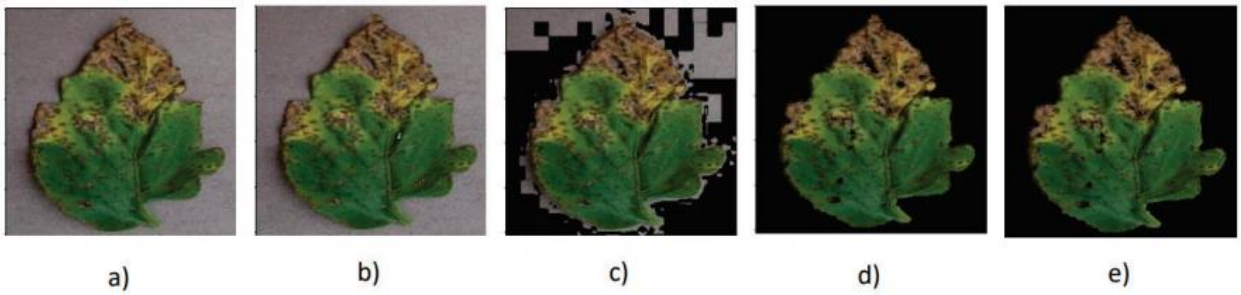


Рис. 2.3. Результати кроків: a) після видалення білих областей, b) після видалення чорних областей; c) після видалення синіх областей, d) після застосування методу Оцу; e) після видалення текстурного фону.

Однак ураження листя може бути значним і мати інший колір, який відрізняється від здорової частини листка, тоді зображення буде ділитися не на дві великі рівномірно яскраві області (фон і листок), а на три (фактичний фон, здорова частина листка, хвора частина листка). Тому використовувати лише різницю між індексами недостатньо. Також має бути врахована насиченість зображення (saturation), попередньо перевівши зображення у колірному просторі HSV. Значення насиченості ділять на дві групи методом Оцу, об’єднують маски, які отримані на основі різниці індексів (надлишкового зеленого та червоного) та насиченості. Далі видаляють текстурний фон (рис. 2.4).

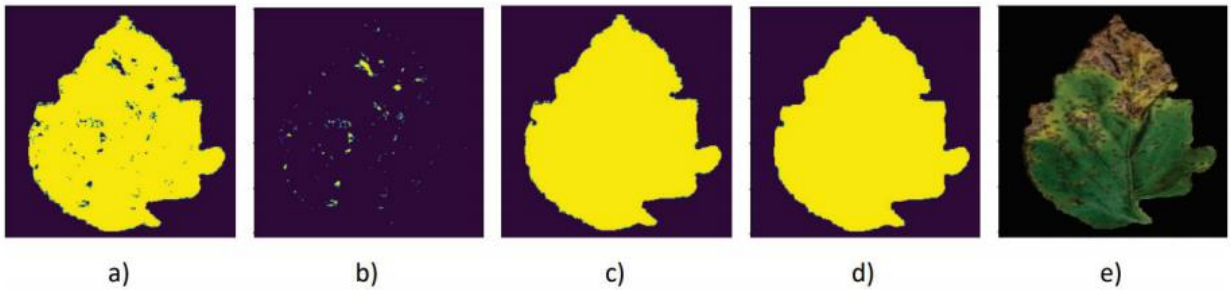


Рис. 2.4. Зображення: a) маска листка з дефектами; b) маска дефектів; c) маска листка після об’єднання масок; d) маска листка після згладжування операціями морфології; e) результат сегментації листка.

В результаті видалення текстурного фону буде отримано маску листка з дефектами (дірками) всередині області листка, які необхідно заповнити (рис. 2.4a). Для цього будують бінарне зображення маски дефектів

(0 – фон та лист, 1 – дефект). Після цього бінарне зображення маски листка без дефектів отримано поєднанням масок (рис. 2.4c). Згладжування зайвої зашумленості контурів реалізовано морфологічними операціями. Фіналом автоматичної сегментації є застосування отриманої маски до зображення (рис. 2.4e).

Вихідні класи незбалансовані. Для забезпечення балансу для кожного з класів взято автоматично сегментовані зображення На рис. 2.5 представлено сегментовані зображення для класів захворювання листя рослин.

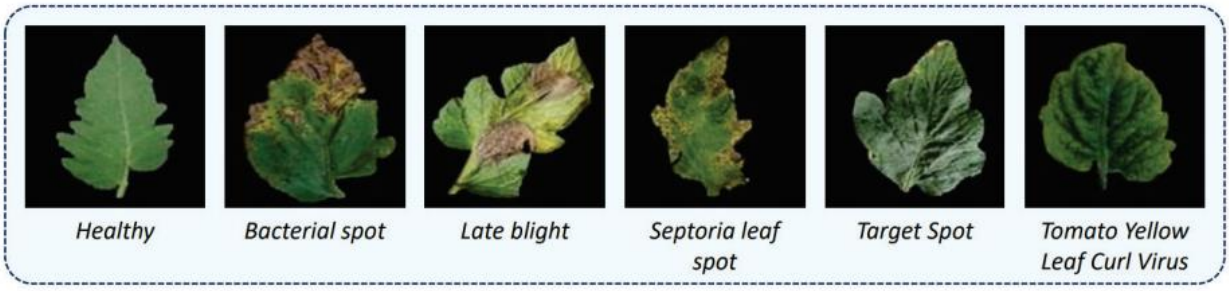


Рис. 2.5. Приклади сегментованих зображень для класів захворювань листя рослин.

ВИСНОВКИ ДО РОЗДІЛУ 2

Розглянуто питання про розпізнавання та класифікацію зображень листя рослин із застосуванням технологій штучного інтелекту. Розпізнавання стану рослин передбачає вирішення задачі класифікації зображень, як класифікатор пропонується використати штучну нейронну мережу як модель, яка може навчатися. При задачі оцінки стану рослин використовується багатокласова класифікація, тобто зображення буде відноситися до одного з обраних класів найпоширеніших захворювань рослинних культур. На виході класифікатора формується вектор ймовірностей відношення вхідного зображення кожного класу. Клас із найвищим значенням ймовірності буде результатом розв’язку задачі класифікації.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Математичне моделювання захворювання листя рослин

Більшість захворювань рослин породжують зміни виду листя у видимому спектрі. Багато з цих проблем можуть бути вирішені за рахунок цифрової обробки отриманих за допомогою безпілотних літальних апаратів зображень листя рослин RGB камерою та інструментами класифікації. Для вирішення завдання виділення особливостей на зображеннях з метою їхньої класифікації застосовуються різні методи формування набору ознак, що дозволяють однозначно ідентифікувати зображення, тобто відносити їх до певного класу.

Найчастіше для вирішення завдання виділення особливостей на зображеннях листя рослин з метою класифікації виду захворювання рослин використовуються методи нечіткої логіки та нейронні мережі, а діагностика проводиться як безпосередньо за кольоровими RGB або HSV зображеннями листя, так і за їх текстурними описами.

Найбільшого застосування при розв’язанні задач розпізнавання хвороб рослин за зображеннями листя знайшли ознаки текстури, що використовують матриці суміжності (матриці GLCM для напівтонових зображень та CCM – для кольорових зображень), це ознаки, що ґрунтуються на вимірюваннях просторових частот, ознаки, що використовують статистичні характеристики зображень (середнє, однорідність, контраст, коефіцієнт кореляції, ентропію, диференціальну дисперсію), ознаки, що ґрунтуються на описі структурних елементів.

На основі RGB-нормалізованого зображення листка рослини можуть бути отримані шість GLCM матриць для компонентів R, G, B, RG, RB, GB, для кожної з GLCM нормалізованих матриць обчислені характеристики текстури Contrast, Correlation, Energy, Homogeneity:

1) contrast:

$$CN = \frac{1}{(G-1)^2}\left|u-v\right|^2 p(u,v) \tag{3.1}$$

2) correlation:

$$CR = \frac{1}{2}\sum_{u=0}^{G-1}\sum_{v=0}^{G-1}\frac{(u-\mu_u)(u-\mu_v)}{\sigma_u^2\sigma_v^2}p(u,v)+1 \tag{3.2}$$

3) energy:

$$EN = \sum_{u=0}^{G-1}\sum_{v=0}^{G-1} p(u,v)^2 \tag{3.3}$$

4) homogeneity:

$$HM = \sum_{u=0}^{G-1}\sum_{v=0}^{G-1}\frac{p(u,v)}{1+\left|u-v\right|}, \tag{3.4}$$

де u, v – координати матриці суміжності, G – кількість рівнів сірого, μ_u, μ_v, σ_u і σ_v – середні значення та стандартні відхилення u-го рядка та v-го стовпця матриці співпадіння. Наведені вище визначення гарантують, що всі функції знаходяться в діапазоні [0, 1].

Проте однозначно визначити вид захворювання рослин щодо належності значення параметрів Contrast, Correlation, Energy, Homogeneity до інтервалів еталонних описів неможливо, так як кількість видів захворювань у сільськогосподарських культур є досить великою, а функції розподілу та інтервали для параметрів Contrast, Correlation, Energy, Homogeneity зображень листя з різними видами захворювань навіть для здорових та хворих рослин значною мірою перекриваються (рис. 3.1).

Це визначає необхідність використання нечіткої логіки, при використанні якої результатом аналізу конкретного зображення буде один або кілька результатів, для кожного з яких буде визначено можливість його реалізації. Однозначний висновок щодо належності зображення до одного з класів (можливих видів захворювання) може бути зроблений по максимальній ймовірності.

Для формування результатів розпізнавання застосовано апарат нечіткої логіки. Особливістю запропонованого рішення цього завдання полягає в тому, що воно передбачає обчислення функцій належності для кожного з шести GLCM-матриць R, G, B, RG, RB, GB зображень, граничну бінаризацію результатів за функціями належності та прийняття остаточного рішення про належність зображення до одного із N класів.

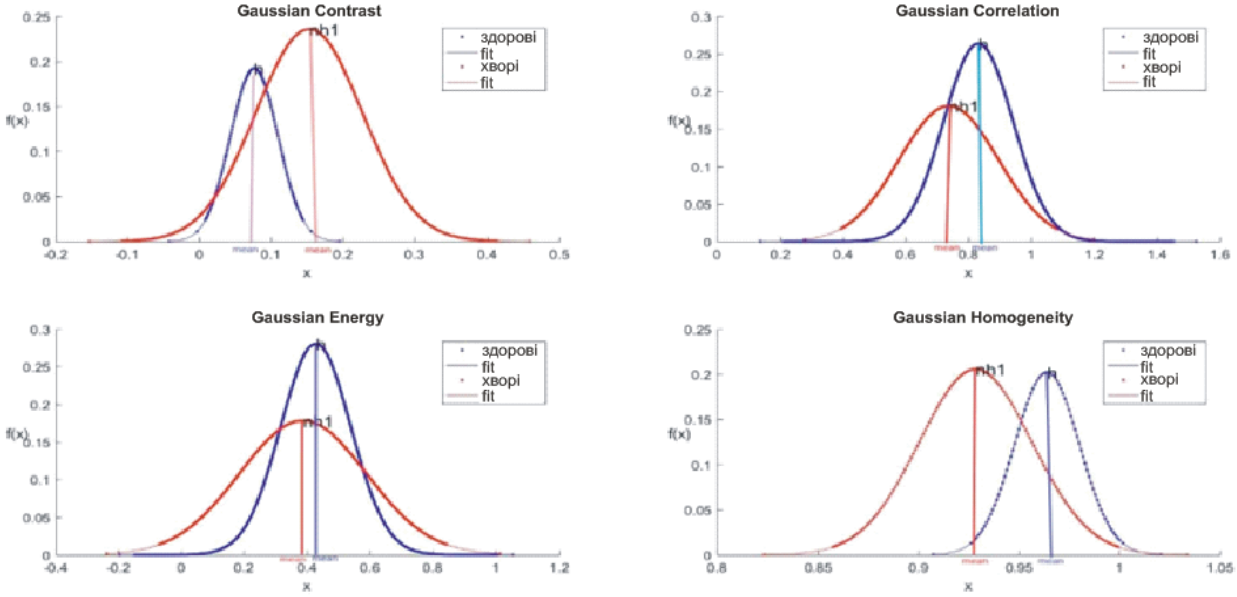


Рис. 3.1. Функції розподілу параметрів текстур для компонентів нормалізованих зображень листя здорових та хворих рослин.

Однією з функцій попередньої обробки є нормалізація зображень листя, що включає приведення зображень листя до стандартизованого масштабу та орієнтації, виділення інформативної частини листка.

Запропонована методика діагностики виду захворювань рослин із зображенням листя включає два етапи. На першому етапі проводиться обчислення параметрів Contrast, Correlation, Energy, Homogeneity для всіх шести підматриць GLCM і порівняння цих параметрів з еталонними описами, які представлені у вигляді функцій належності.

На рис. 3.2 наведено запропоновану структуру системи діагностики захворювань рослин за зображеннями листя.

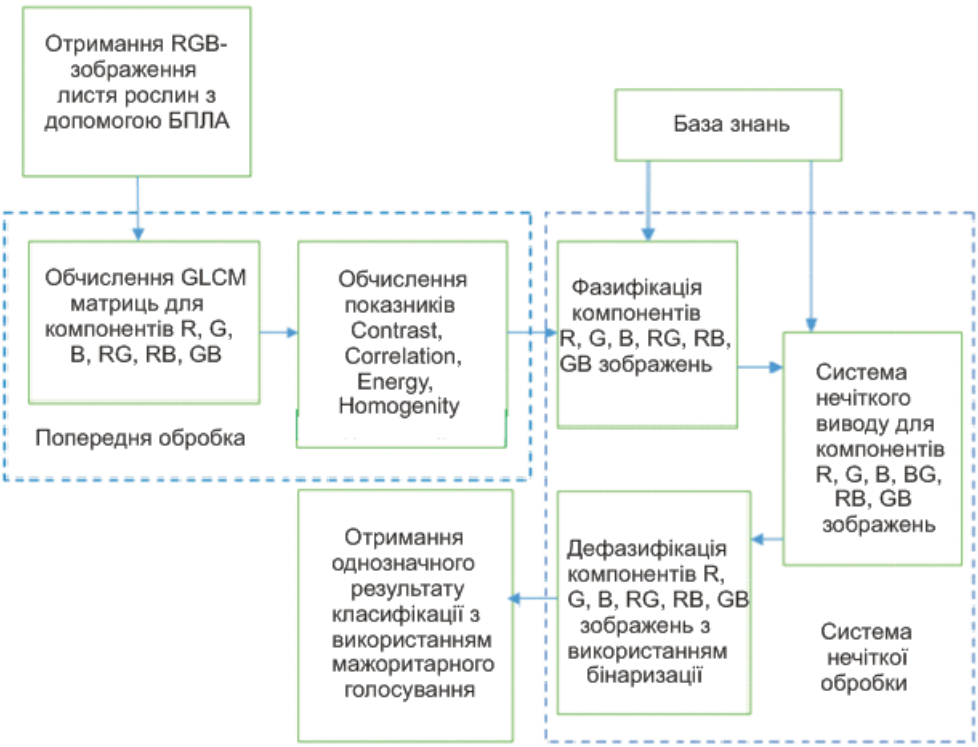


Рис. 3.2. Структура системи діагностики захворювань рослин по зображеннях їх листя.

Прийняття рішення про вид захворювання проводиться по максимуму функції належності. У випадку, як видно з рис. 3.3, максимум відповідає третій хворобі. Однак до максимуму функції належності можуть бути близькі кілька значень, тому приймати рішення про вид захворювання потрібно шляхом бінаризації по порогу P, взятого, наприклад, рівним 0,95 від максимального значення функції належності, що дорівнює 1. У цьому випадку рішення на цьому етапі може виявитися неоднозначним, але дозволить не втратити правильний результат.

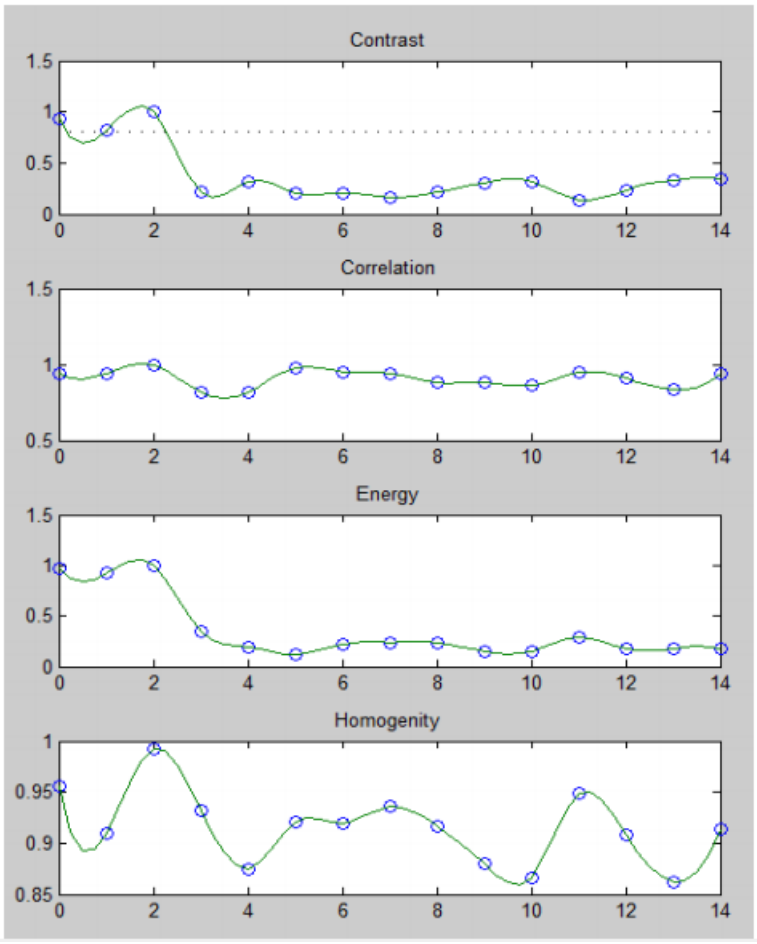


Рис. 3.3. Вид функцій належності для колірної компоненти нормалізованого зображення листя помідорів до еталонних описів 15 різних хвороб.

3.2. Опис алгоритму роботи інформаційної системи

Вхідними даними для алгоритму діагностики є еталонні описи нормалізованих зображень листя при всіх захворюваннях: математичні очікування показників Contrast, Correlation, Energy, Homogeneity для підматриць GLCM: R, G, B, RG, RB, GB. Послідовність дій.

1. Обчислення функцій належності MF(i, k, j) всіх компонентів та всіх хвороб: (i=1..N, k=1..4, j=1..6):

$$MF(i, k, j) = 1\text{-}abs(U_{ref}(i, k, j) – U(i, k)), \quad (3.5)$$

де $U_{ref}(i, k, j)$ – масив еталонних описів; U(i, k) – масив виміряних значень параметрів.

2. Первинна бінаризація показників по порогу P:

$$BMF(i, j, k) = 1, \text{ якщо } MF(i, j, k) \geq P, \text{ та } 0, \text{ якщо } MF(i, j, k) < P. \quad (3.6)$$

3. Сумування бінаризованих показників та вторинна бінаризація:

$$B2MF(i, j) = 1, \text{ якщо } SBMF(k, j) = \max SBMF(k, j) \text{ інакше } 0. \quad (3.7)$$

ВИСНОВКИ ДО РОЗДІЛУ 3

Наведено відомості про засади створення алгоритму для математичної моделі та різні шляхи, щоби її можна було використати в подальшому. На її основі була розроблена інтелектуальна інформаційна система для виявлення захворювань рослин. Проведені в подальшому дослідження (розділ 4) підтверджують правильність та точність цієї моделі, а також демонструють можливості для розроблення системи для діагностики захворювань рослин.

Розглянуто методи математичного моделювання для виявлення захворювань рослин за зображенням їх листя. Захворювання рослин породжують зміни виду листя у видимому спектрі. Для вирішення цього питання використовують методи цифрової обробки зображень та інструменти класифікації. Для класифікації захворювань рослин застосовуються різні методи формування набору ознак, що дозволяють ідентифікувати зображення, відносити їх до певного класу. Виділення особливостей на зображеннях листя рослин використовують методи нечіткої логіки та нейронні мережі, діагностика проводиться за кольоровими зображеннями листя та їх текстурними описами.

При розв’язанні задач розпізнавання хвороб рослин за зображеннями їх листя найбільшого застосування знайшли ознаки текстури, що використовують матриці суміжності - ознаки, що ґрунтуються на вимірюваннях просторових частот, статистичні характеристики зображень, ознаки, що ґрунтуються на описі структурних елементів.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1. Проектування інтелектуальної системи виявлення захворювання листя рослин

Захворювання рослин можна виявити неозброєним оком, що вимагає багато часу, зусиль та робочої сили. Дана робота призначена для того, щоб полегшити цей трудоємний процес. У цій роботі використовують глибоке навчання для класифікації хвороб помідорів за зображенням їх листя, а саме буде використана модель CNN. Набір даних містить зображення листя трьох категорій: здорове листя, листя, уражене Yellow_Leaf_Curl_Virus, і листя, уражене бактеріальною плямистістю. Набір даних завантажується з Kaggle.

Імпортують необхідні бібліотеки Python для роботи:

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np
```

Імпортують бібліотеку TensorFlow, яка є інструментом для створення та навчання нейронних мереж та інших моделей машинного навчання. Далі імпортують модулі models та layers з Keras бібліотеки TensorFlow. Keras - це високорівневий інтерфейс для визначення та тренування нейронних мереж. Models використовується для створення моделей, а layers - для визначення шарів нейронних мереж. Бібліотека Matplotlib буде використана для візуалізації даних та графіків. NumPy - ця бібліотека використовується для роботи з масивами та матрицями.

```
IMAGE_SIZE=256
BATCH_SIZE=32
```

Визначають дві константи: IMAGE_SIZE та BATCH_SIZE. Перша з них встановлює розмір зображень. Вона буде використовуватися при підготовці даних для нейронної мережі, де всі зображення будуть змінені або обрізані так, щоб мати розмір 256x256 пікселів. Цей розмір обирається

згідно з потребами конкретної задачі або моделі. Константа BATCH_SIZE визначає кількість прикладів даних, які використовуються за один раз під час навчання моделі. У нейронних мережах навчання по частинах використовується для ефективнішого оновлення параметрів моделі. Навчання відбуватиметься частково, модель буде оновлюватися з кожним міні-пакетом даних, що полегшуватиме оптимізацію.

```
images_dataset=tf.keras.preprocessing.image_dataset_from_directory(
 'dataset',
 shuffle=True,
 image_size=(IMAGE_SIZE,IMAGE_SIZE),
 batch_size=BATCH_SIZE,
 )
```

Створюють датасет зображень з каталогу, готовий до використання для тренування моделі машинного навчання. Отриманий датасет містить зображення, їх класи та інші необхідні атрибути. Функція image_dataset_from_directory з бібліотеки tf.keras.preprocessing буде використовуватися для створення датасету зображень з каталогу. Перший її параметр dataset - шлях до каталогу, в якому розташовані зображення. Дані розташовані в підкаталогах, кожен підкаталог представляє собою один клас. Параметр shuffle=True вказує, щоб дані були перемішані у випадковому порядку. Це використовують при навчанні моделі, щоб уникнути порядку даних, який може вплинути на навчання. image_size=(IMAGE_SIZE, IMAGE_SIZE) – це розмір зображень, який передається до датасету. Використовуються значення, які були визначені раніше в константі IMAGE_SIZE. batch_size=BATCH_SIZE - розмір партії (mini-batch), який буде використовуватися під час навчання моделі.

```
Found 3000 files belonging to 3 classes
```

Це повідомлення свідчить про те, що функція image_dataset_from_directory знайшла 3000 файлів у каталозі, які належать до 3 класів. Це важлива інформація для тренування моделі, оскільки кількість файлів та класів використовується для налаштування параметрів

моделі та визначення вихідного шару. У машинному навчанні важливо мати потрібну кількість даних для ефективного навчання моделі. Далі можна продовжити роботу з обробки та навчання моделі за допомогою цього датасету. Потрібно буде налаштувати параметри моделі, такі як шари, функції активації, оптимізатор, функцію втрат та інші, залежно від конкретної задачі та даних.

```
class_names=images_dataset.class_names
class_names
```

Використано атрибут class_names датасету images_dataset, який містить імена класів (категорій) для зображень у датасеті. Це використовується для подальшого використання цих імен для відображення результатів або аналізу класифікації. Отримавши class_names, можна вивести їх, щоб переглянути, які класи були визначені під час формування датасету:

```
class_names = images_dataset.class_names
print("Class Names:", class_names)
```

Це виведе імена класів. В подальшому можна використовувати ці імена для подальших аналізів чи для виведення результатів прогнозу моделі.

```
['Tomato___Bacterial_spot',
 'Tomato___Tomato_Yellow_Leaf_Curl_Virus',
 'Tomato___healthy']
```

Отримано список з трьох імен класів:

- Tomato___Bacterial_spot;
- Tomato___Tomato_Yellow_Leaf_Curl_Virus;
- Tomato___healthy.

Ці імена вказують на три класи чи категорії, які будуть використовуватися для класифікації помідорів за станом їхнього листового покрову або за наявністю захворювань.

Якщо використати наявний датасет для тренування моделі класифікації, то зображення можуть будуть позначені як Tomato_Bacterial_spot, Tomato_Tomato_Yellow_Leaf_Curl_Virus та Tomato_healthy в залежності від їхнього стану. Дані класи будуть важливими для подальшого аналізу результатів та для інтерпретації виведених моделлю прогнозів.

```
len(images_dataset)
```

Дана функція виведе кількість партій у датасеті, а не кількість зображень. Якщо треба знайти кількість зображень у всьому датасеті, потрібно скористатися функцією tf.data.experimental.cardinality(images_dataset):

```
dataset_length = tf.data.experimental.cardinality(images_dataset).numpy()
print("Кількість зображень у датасеті:", dataset_length)
94
```

Це означає, що у даному датасеті є 94 партії, а не зображення. Кожна партія містить певну кількість зображень відповідно до значення BATCH_SIZE. Якщо треба знати загальну кількість зображень у всьому датасеті, потрібно використати функцію tf.data.experimental.cardinality(images_dataset):

```
dataset_length = tf.data.experimental.cardinality(images_dataset).numpy()
print("Кількість зображень у датасеті:", dataset_length * BATCH_SIZE)
```

Так можна отримати загальну кількість зображень у датасеті, яка буде дорівнювати кількості партій, яку треба помножити на розмір партії (BATCH_SIZE).

Далі проводять вивчення набору даних:

```
for image_batch, label_batch in images_dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

Використовують метод take(1) для того, щоб взяти лише одну партію з датасету та вивести інформацію про цю партію. Це ітерація через один батч з датасету. images_dataset.take(1) бере лише одну партію.

BATCH_SIZE - розмір партії, IMAGE_SIZE - розмір зображень, num_channels - кількість каналів. З допомогою методу print(label_batch.numpy()) виводять мітки класів для кожного зображення у вигляді масиву NumPy. Мітки класів вказують на категорію, до якої належить кожне зображення.

```
(32, 256, 256, 3)
[1 1 1 1 1 1 2 0 0 1 1 2 2 1 2 2 0 0 2 0 0 0 0 1 2 2 0 2 0 1 1 1]
```

Дана інформація вказує на те, що отриманий батч має такі характеристики:

- розмір батчу: 32;
- розмір зображень: 256x256 пікселів;
- кількість каналів (channels): 3 (це, ймовірно, RGB кольори);

Під цими характеристиками виведено масив міток класів для кожного зображення у батчі:

```
[1 1 1 1 1 1 2 0 0 1 1 2 2 1 2 2 0 0 2 0 0 0 0 1 2 2 0 2 0 1 1 1]
```

Даний масив вказує на клас (категорію) для кожного відповідного зображення у батчі. Перше зображення має мітку класу 1, друге - також 1 і так далі. Це є необхідною інформацією для подальшого аналізу та перевірки правильності міток у датасеті.

Далі виводять перше зображення з першого батчу датасету у вигляді масиву NumPy:

```
for image_batch, label_batch in images_dataset.take(1):
    print(image_batch[0].numpy())
```

Проводять ітерацію через один батч з датасету за допомогою методу take(1). Далі виводять перше зображення з батчу у вигляді масиву NumPy. Зображення може бути представлене у вигляді масиву пікселів з висотою, шириною та кількістю колірних каналів. Як результат роботи можна побачити вигляд першого зображення у датасеті. Якщо потрібно вивести

зображення, використовуючи бібліотеку Matplotlib, це можна зробити таким чином:

```
import matplotlib.pyplot as plt
plt.imshow(image_batch[0].numpy().astype("uint8"))
plt.show()
```

```
[[[ 97.  96.  91.]
  [ 97.  96.  91.]
  [ 98.  97.  92.]
  ...
  [ 76.  76.  78.]
  [ 79.  79.  81.]
  [ 82.  82.  84.]]

 [[ 97.  96.  91.]
  [ 97.  96.  91.]
  [ 97.  96.  91.]
  ...
  [ 79.  79.  81.]
  [ 80.  80.  82.]
  [ 82.  82.  84.]]

 [[ 97.  96.  91.]
  [ 97.  96.  91.]
  [ 97.  96.  91.]
  ...
  [ 81.  81.  83.]
  [ 80.  80.  82.]
  [ 80.  80.  82.]]

 ...

 [[150. 155. 148.]
  [150. 155. 148.]
  [149. 154. 147.]
  ...
  [125. 129. 128.]
  [116. 120. 119.]
  [112. 116. 115.]]

 [[150. 155. 148.]
  [150. 155. 148.]
  [149. 154. 147.]
```

```
  ...
  [139. 143. 142.]
  [133. 137. 136.]
  [131. 135. 134.]]

 [[150. 155. 148.]
  [150. 155. 148.]
  [149. 154. 147.]
  ...
  [139. 143. 142.]
  [139. 143. 142.]
  [140. 144. 143.]]]
```

Як результат отримано конкретне значення пікселів для першого зображення з датасету. Це представлення у вигляді масиву NumPy, де кожен елемент масиву представляє собою значення пікселя для кожного каналу кольору. Отримали зображення розміром 256x256 та воно має три канали кольору RGB. Кожен піксель містить три значення для червоного, зеленого та синього каналів. Наприклад, перший піксель має значення [97, 96, 91] для каналів RGB. Це числа відповідають інтенсивності кольорів у певному порядку. Аналогічно це справедливо для всіх інших пікселів у зображенні.

Далі проводиться візуалізація першого зображення у цьому пакеті:

```
for image_batch, label_batch in images_dataset.take(1):
    plt.imshow(image_batch[0].numpy().astype('uint8'))
    plt.title(class_names[label_batch[0]])
    plt.axis('off')
```

Далі потрібно візуалізувати конкретне зображення з датасету та відобразити його клас. Зображення буде відображатися без осей, у заголовку потрібно буде вказати, до якого класу відноситься дане зображення. Для цього використовують бібліотеку Matplotlib для візуалізації першого зображення з датасету. Виводять перше зображення з батчу у вигляді графічного зображення за допомогою функції imshow. Додають заголовок до графіку, який відображає назву класу для першого зображення. Використовуючи метод label_batch[0], отримують мітку класу

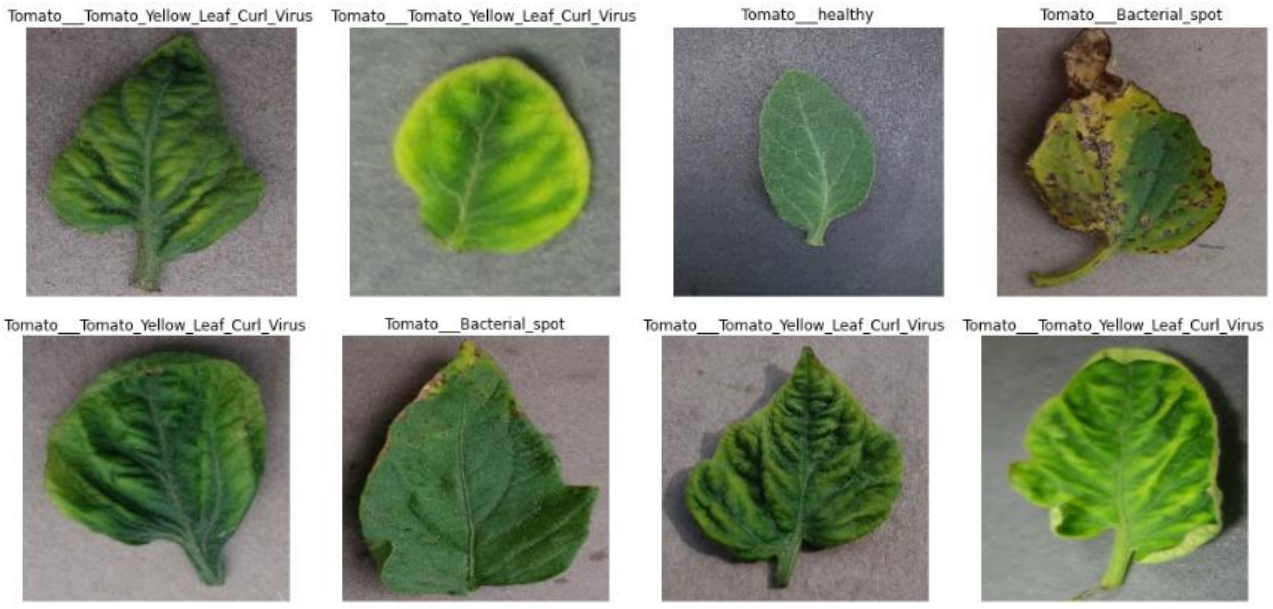
для першого зображення. Вимикають відображення координатних осей на графіку за допомогою методу plt.axis('off').

Рис. 4.1. Зображення здорового листка помідора.

Далі відображають 12 зображень з датасету разом з їхніми класами в одній великій фігурі для кращого огляду.

```
plt.figure(figsize=(18,18))
for image_batch, label_batch in images_dataset.take(1):
    for i in range (12):
        ax=plt.subplot(3,4, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
```

Результат роботи цього фрагмента коду:



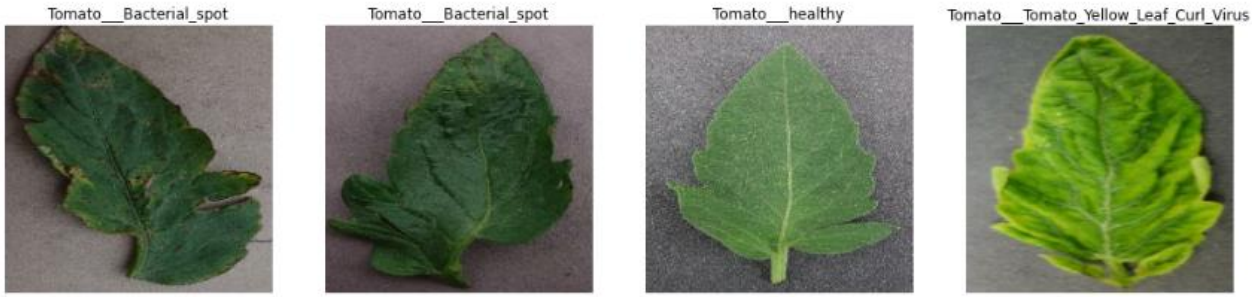


Рис. 4.2. Зображення здорових та хворих листків помідорів з датасету разом з їхніми класами.

```
len(images_dataset)
```

Якщо треба дізнатися кількість партій (батчів) у датасеті, використовують функцію len. Отримають кількість партій у датасеті. Так як датасет буде використано для навчання, кількість партій визначається розміром датасету та розміром партії BATCH_SIZE, який був визначений при його створенні.

```
94
```

В цьому датасеті є 94 партії. Кожна партія містить певну кількість зображень згідно з вказаним розміром партії BATCH_SIZE.

```
def get_dataset_partitions_tf(ds,train_split=0.8, val_split=0.1,
test_split=0.1,shuffle=True, shuffle_size=10000):
    ds_size=len(ds)
    if shuffle:
        ds=ds.shuffle(shuffle_size,seed=12)
        train_size= int(train_split* ds_size)
    val_size=int(val_split* ds_size)
    train_ds=ds.take(train_size)
    val_ds=ds.skip(train_size).take(val_size)
    test_ds=ds.skip(train_size).skip(val_size)
    return train_ds, val_ds, test_ds
```

Функція get_dataset_partitions_tf призначена для розбиття датасету на тренувальний, валідаційний та тестовий набори. Перемішують датасет, якщо вказано:

```
    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)
```

Проводять розрахунок розмірів тренувального, валідаційного та тестового наборів даних:

```
train_size = int(train_split * ds_size)
val_size = int(val_split * ds_size)
```

Виділення тренувального набору:

```
train_ds = ds.take(train_size)
```

Виділення валідаційного набору:

```
val_ds = ds.skip(train_size).take(val_size)
```

Виділення тестового набору

```
test_ds = ds.skip(train_size).skip(val_size)
return train_ds, val_ds, test_ds
```

Розглянемо параметри даної функції:

- ds: вхідний датасет, який потрібно розбити;
- train_split, val_split, test_split: відсоток даних, які призначаються для тренування, валідації та тестування;
- shuffle: якщо цей параметр рівний True, то датасет буде перемішаний перед розбиттям.
- shuffle_size: розмір зразків для перемішування датасету, застосовується тільки при shuffle=True.

```
train_ds, val_ds, test_ds=get_dataset_partitions_tf(images_dataset)
```

Датасет був розбитий на тренувальний, валідаційний та тестовий набори даних. Отримано змінні train_ds, val_ds і test_ds, які представляють собою ці розбиті набори. Після цього можна використовувати ці набори для тренування, валідації та тестування моделі машинного навчання. Якщо є модель з назвою model:

```
model.fit(train_ds, epochs=10, validation_data=val_ds)
```

Може ще знадобитися налаштувати параметри тренування залежно від конкретного випадку.

```
print(len(train_ds),len(val_ds),len(test_ds))
```

Використано функцію len для визначення кількості партій у кожному з розбитих датасетів. Кількість партій визначається розміром партії та розміром кожного з розбитих наборів даних. Якщо треба отримати кількість зображень, а не кількість партій, слід використати метод tf.data.experimental.cardinality:

```
len_train = tf.data.experimental.cardinality(train_ds).numpy() * BATCH_SIZE
len_val = tf.data.experimental.cardinality(val_ds).numpy() * BATCH_SIZE
len_test = tf.data.experimental.cardinality(test_ds).numpy() * BATCH_SIZE
print("Кількість зображень у тренувальному наборі:", len_train)
print("Кількість зображень у валідаційному наборі:", len_val)
print("Кількість зображень у тестовому наборі:", len_test)
```

BATCH_SIZE - розмір партії, який був використаний при створенні датасету. Отримано такі значення:

```
75 9 10
```

- кількість зображень у тренувальному наборі: 75;
- кількість зображень у валідаційному наборі: 9;
- кількість зображень у тестовому наборі: 10.

Ці значення представляють собою кількість зображень у кожному з розбитих наборів даних, враховуючи розмір партії, який був вказаний при формуванні датасету. Можна використовувати ці набори для тренування, валідації та тестування моделі.

```
train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experi
mental.AUTOTUNE)
val_ds=val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experiment
al.AUTOTUNE)
test_ds=test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experime
ntal.AUTOTUNE)
```

Використовують методи cache(), shuffle(), та prefetch() для оптимізації використання датасетів при тренуванні моделі. Зберігають дані в кеші після їхнього першого читання з файлу або джерела. Це дозволяє уникнути повторного читання даних під час тренування. Перемішують дані у випадковому порядку. Розмір буфера 1000 вказує, скільки зразків буде

зберігатися в буфері перемішування. Автоматично вибирають оптимальний розмір буфера для попереднього завантаження даних. Це дозволяє моделі завантажувати дані в пам’ять, тоді коли вона зайнята обчисленнями. Те саме застосовується і до змінних val_ds та test_ds. Оптимізації, такі як кешування та попереднє завантаження даних, можуть покращити продуктивність тренування моделі.

```
resize_and_rescale=tf.keras.Sequential([
    layers.experimental.preprocessing.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])
```

Використовують метод бібліотеки TensorFlow tf.keras.Sequential для створення послідовності операцій перед обробкою зображень. Проводять операцію зміни розміру зображення. Вона змінює розмір зображення на вказаний розмір IMAGE_SIZE. Зображення буде адаптоване до нового розміру. Далі проводять операцію масштабування, яка використовується для нормалізації значень пікселів у діапазоні від 0 до 1.0. Розділяючи значення пікселів на 255, отримують числа у діапазоні від 0 до 1.0, що допомагає полегшити тренування моделі. Ця послідовність може бути використана як частина моделі для попередньої обробки вхідних зображень. Можна використати цю послідовність перед подачею зображень на вхід нейронної мережі.

```
data_augmentation=tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])
```

Наступні операції можна використовувати в якості частини моделі, яка відповідає за попередню обробку та аугментацію даних. Такі операції особливо корисні для збільшення ріноманітності тренувальних даних та поліпшення загального навчання моделі. Визначають ще один tf.keras.Sequential для створення послідовності операцій, які відповідають за аугментацію даних під час тренування. Наступна операція випадковим чином виконує обертання зображення по горизонталі і вертикалі. Це може

допомогти моделі навчитися розпізнавати об’єкти під різними кутами та орієнтаціями. Слідуюча операція випадковим чином обертає зображення на випадковий кут, вказаний в радіанах. Обертання обмежене значенням 0.2 радіан (11.5 градусів). Це також може допомогти моделі бути більш стійкою до різних положень об’єктів.

4.2. Розроблення інтерфейсу інтелектуальної системи

Інтерфейс інформаційної системи розроблявся з допомогою бібліотеки Streamlit. Також використано бібліотеки для роботи з нейромережами (TensorFlow), операцій з зображеннями (PIL), роботи з масивами (NumPy). Клас Image з бібліотеки Python Imaging Library (PIL) використовується для роботи з зображеннями. Бібліотека NumPy надає можливості для роботи з масивами та матрицями в Python. Модуль random дозволяє генерувати випадкові числа та виконувати інші операції, пов'язані з випадковістю. Цей набір імпортів забезпечує доступ до різних функціональностей, які можуть бути використані в подальшому проектуванні інтелектуальної системи.

```
st.set_page_config(
    page_title = 'Прогноз хвороби листя помідорів',
    page_icon = ":tomato:",
    initial_sidebar_state = 'auto'
)
```

Тут використано функцію set_page_config з бібліотеки Streamlit для налаштування конфігурації сторінки веб-застосунку. page_title - визначає заголовок сторінки веб-застосунку. У даному випадку, заголовок встановлюється як Прогноз хвороби листя помідорів. page_icon - визначає піктограму сторінки tomato. initial_sidebar_state - цей параметр визначає початковий стан бічної панелі. Auto означає автоматичне визначення стану бічної панелі залежно від розміру вікна броузера та інших факторів. Ці параметри служать для задання зовнішнього вигляду веб-застосунку та його поведінки при завантаженні.

```
def prepare(file):
```

```
    img_array=file/255
    return img_array.reshape(-1,128,128,3)
```

Функція prepare приймає один параметр file, який представляє собою зображення та виконує певні операції підготовки для подальшого використання у моделі. img_array = file / 255 - ділять значення пікселів зображення на 255. Це ділення використовується для нормалізації значень пікселів у діапазоні [0, 1]. return img_array.reshape(-1, 128, 128, 3) - використано метод reshape, щоб змінити форму масиву img_array. Зображення часто представляються як тривимірні масиви, де перша вісь представляє розмір зразка (кількість зображень), а інші три відповідають розмірам зображення та каналам кольорів (128x128 з трьома каналами для RGB). -1 вказує на автоматичний розмір, що дозволяє коректно змінювати форму масиву, якщо вхідне зображення має різний розмір.

```
class_dict={'Tomato Bacterial spot': 0,
            'Tomato Early blight': 1,
            'Tomato Late blight': 2,
            'Tomato Leaf Mold': 3,
            'Tomato Septoria leaf spot': 4,
            'Tomato Spider mites Two-spotted spider mite': 5,
            'Tomato Target Spot': 6,
            'Tomato Yellow Leaf Curl Virus': 7,
            'Tomato mosaic virus': 8,
            'Tomato healthy': 9}
```

Визначають словник class_dict, який містить зв’язок між назвами класів (хворобами помідорів) та їх числовими індексами. Це може бути використано при побудові моделі класифікації для присвоєння числового індексу кожному класу. Наприклад, якщо отримано прогноз від моделі у вигляді числа, можна використовувати цей словник для отримання відповідної назви класу. Або, якщо є назва класу, можна використати цей словник для отримання числового індексу. Цей підхід зручний для забезпечення однозначної відповідності між назвами класів і числовими індексами, що часто є необхідним для побудови та оцінки моделей у випадках класифікації.

```
def prediction_cls(prediction):
```

```
    for key, clss in class_dict.items():
        if np.argmax(prediction)==clss:
            return key
```

Функція prediction_cls приймає прогноз з моделі класифікації та використовує словник class_dict для визначення назви класу, який має максимальний індекс у прогнозі. Ця функція використовує метод np.argmax(prediction), щоб отримати індекс максимального значення у масиві прогнозу. Потім вона порівнює цей індекс із числовим індексом класу зі словника class_dict і, якщо знаходить відповідність, повертає назву класу. Ця функція призначена для зручності отримання інтерпретованого класу з прогнозу моделі, який може бути у форматі, де індекси відповідають числовим класам.

```
@st.cache
def load_image(image_file):
    img=Image.open(image_file)
    img=img.resize((128,128))
        return img
```

Функція load_image використовує декоратор @st.cache, який надає можливість кешування результатів функції. Це дозволяє зберігати результати в пам’яті, щоб уникнути повторних завантажень, коли функція викликається з тими ж самими аргументами. Застосування декоратора @st.cache гарантує, що якщо функція вже була викликана з певними аргументами, результат буде взятий з кешу, а не викликано фактично. Це може бути корисно для ефективного управління ресурсами при використанні веб-застосунків на основі Streamlit. Image.open(image_file) - відкривають зображення, яке передається як аргумент image_file за допомогою бібліотеки PIL. img.resize((128, 128)) - змінюють розмір зображення на 128x128 пікселів. return img - повертають змінене зображення.

```
def main():
    with st.sidebar:
        st.header("ПОМІДОРИ")
        st.image('./img2.jpg')
    st.title("Прогноз хвороби листя помідорів")
```

```
    st.subheader("Завантажити зображення листя помідорів, щоб передбачити
їх захворювання.")
    image_file=st.file_uploader("Завантажити
зображення",type=["png","jpg","jpeg"])
```

Це основна функція, яка використовується в веб-застосунку на основі Streamlit для прогнозу хвороб листя помідорів за допомогою завантажених зображень. with st.sidebar - цей блок визначає бічну панель веб-застосунку. В ньому відображається заголовок "ПОМІДОРИ" і зображення ./img2.jpg. st.title("Прогноз хвороби листя помідорів") - виводиться заголовок для головної частини сторінки. st.subheader("Завантажити зображення листя помідорів, щоб передбачити їх захворювання.") - виводиться підзаголовок, який пояснює користувачеві, що робити. image_file = st.file_uploader("Завантажити зображення", type=["png", "jpg", "jpeg"]) - відображається кнопка для завантаження зображення. Обране зображення буде доступне у змінній image_file, яку можна використовувати для подальшого аналізу чи передачі до моделі для прогнозу.

```
    if image_file == None:
        st.warning("Завантажте зображення листя помідорів, щоб передбачити
їх захворювання.")
    else:
        if st.button("Процес"):
            img=load_image(image_file)
            img=tf.keras.preprocessing.image.img_to_array(img)
            model=tf.keras.models.load_model("model_vgg19.h5")
            img=prepare(img)
            with st.sidebar:
                st.image(img,caption="Завантажити зображення")
                x = random.randint(90,98)+ random.randint(0,99)*0.01
                st.subheader("Виявлене захворювання:")
                if (prediction_cls(model.predict(img))) == 'Tomato
healthy':
                    st.success("Рослина здорова")
                    st.write("Точність передбачення %:",x)
                else:
                    st.warning(prediction_cls(model.predict(img)))
                    st.write("Точність прогнозу %:",x)
```

```
            if (prediction_cls(model.predict(img))) == 'Tomato Bacterial
spot':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Early
blight':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Late
blight':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Leaf
Mold':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Septoria
leaf spot':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Spider
mites Two-spotted spider mite':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Target
Spot':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Yellow
Leaf Curl Virus':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato mosaic
virus':
                st.subheader("Засоби захисту :")
```

Цей фрагмент коду відповідає за обробку завантаженого зображення та використання моделі для прогнозу захворювання листя помідорів. Якщо зображення не було завантажено image_file == None, виводиться попередження, щоб користувач завантажив зображення. Якщо користувач натиснув кнопку "Процес" st.button("Процес"), тоді виконується обробка зображення та прогноз захворювання. Зображення спершу завантажується за допомогою функції load_image. Зображення підготовлюється для передачі в модель за допомогою функції prepare і потім передається для прогнозу моделі. Прогноз обробляється функцією prediction_cls, яка визначає клас хвороби. Результати виводяться у бічній панелі та головній

частині сторінки з вказанням прогнозованого захворювання та точності прогнозу.

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

Умовна конструкція if __name__ == "__main__" перевіряє, чи модуль запускається безпосередньо, а не імпортується в інший файл. Це дозволяє створювати модульні функції та класи, які можуть бути використані в інших програмах, а також визначити код, який повинен виконуватися лише тоді, коли цей файл запускається самостійно. Якщо викликати main(), коли файл виконується безпосередньо (а не імпортується в інший файл), функція main() буде викликана, і весь основний код виконається.

4.3. Результати роботи інтелектуальної системи

Додають шар активації ReLU (Rectified Linear Unit) до моделі (додаток Б, файл 2.ipynb). Цей тип активації широко використовується в нейромережах через свою ефективність у вирішенні проблеми виводу градієнту та швидкої збіжності під час тренування. Релізація ReLU шару виглядає наступним чином:

```
model.add(Activation("relu"))
```

Додають шар активації ReLU (Rectified Linear Unit) до моделі. Цей тип активації широко використовується в нейромережах через свою ефективність у вирішенні проблеми виводу градієнту та швидкої збіжності під час тренування.

Релізація ReLU шару виглядає наступним чином. Вихід від попереднього шару подається на вхід до цього ReLU-шару. Для кожного елемента входу активація ReLU визначається як максимум між нулем і значенням елемента. Це призводить до виводу, який є додатнім значенням, якщо вхід більший за нуль, або нулем в іншому випадку. ReLU-активація часто використовується в шарах нейромереж для нелінійності та підвищення ефективності навчання моделі.

Активація – це математичне рівняння, яке визначає вихідні дані нейронної мережі. Перед тим, як передаються дані на інший шар, потрібно вирівняти дані в одному вимірі. Кожний з цих нейронів буде забезпечувати виходи для того шару, що знаходиться поряд з ним.

На наступному етапі роботи:

```
history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
    )
```

Справа знаходяться дані про точність кожної епохи. Після 25 епохи точність складе 97 %.

Результати.

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```

На кінцевому етапі перевіряють, наскільки достовірні отримані дані. На графіку нижче видно, що точність навчання трохи перевищує точність перевірки:

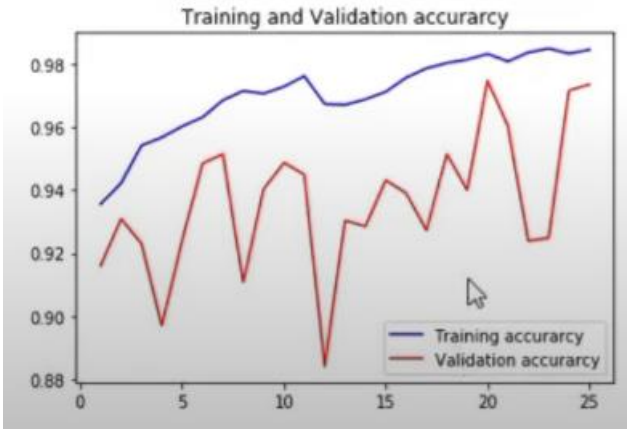


Рис. 4.3. Навчання та перевірка точності.

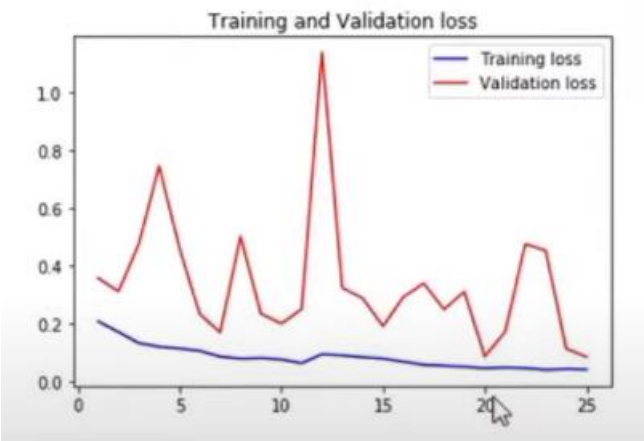


Рис. 4.4. Втрати навчання та перевірки.

Втрати при використанні навчальних даних менші за втрати при використанні тестових даних. З рис. 4.4 видно, що модель краще справляється з даними навчання. Через 25 епох ці дані майже однакові між собою.

Отримано точність 97 %:

```
print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")
```

```
[INFO] Calculating model accuracy
589/589 [==============================] - 36s 61ms/step
Test Accuracy: 97.34012668727816
```

Для зручного управління користувачеві інтелектуальною системою було спроектовано веб-застосунок з допомогою бібліотеки Streamlit (додаток В, файл 3.py) Стартова сторінка системи представлена на рис. 4.5.

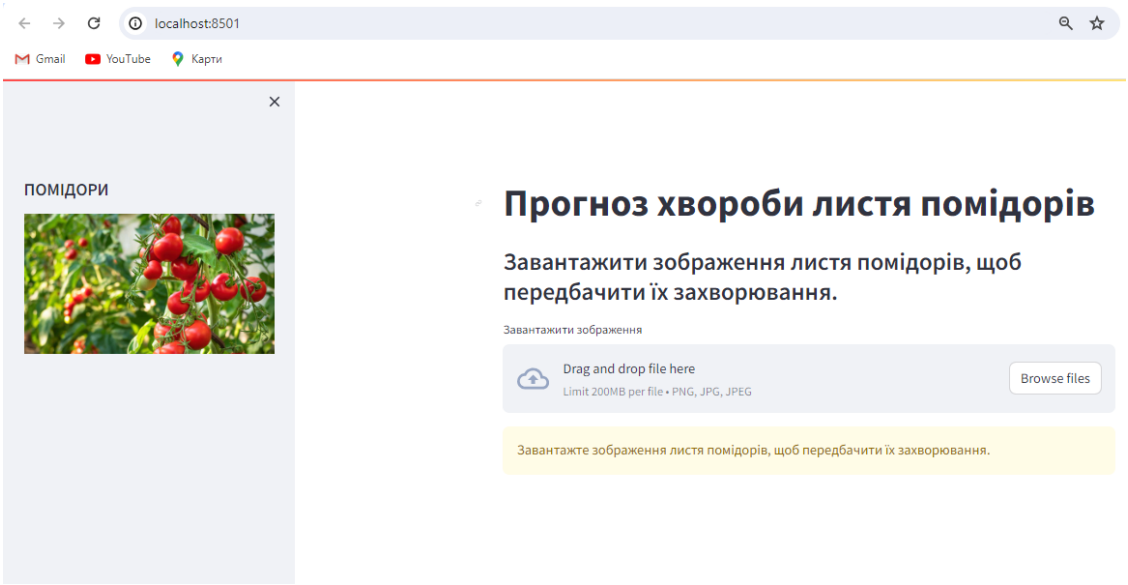


Рис. 4.5. Інтерфейс інтелектуальної системи.

У цьому вікні потрібно натиснути кнопку Browse files, після цього в діалоговому вікні вибрати зображення листка помідора для діагностики рослини, чи вона здорова. В протилежному випадку визначити захворювання за зображенням із зовнішніми пошкодженнями листка.

Рис. 4.6. Рослина здорова.

Рис. 4.6. Рослина має ознаки захворювання.

Рис. 4.6. Рослина має ознаки захворювання.

ВИСНОВКИ ДО РОЗДІЛУ 4

В даному розділі проведено дослідження з метою вирішення задачі діагностики хвороб рослин. Як вхідні дані використано датасет PlantVillage Dataset, що включає зображення здорових рослин і зображення для п’яти видів захворювань листя рослин: Bacterial spot, Late blight, Septoria leafspot, Target Spot, Tomato Yellow Leaf Curl Virus. Розроблено інформаційну систему, що включає такі етапи: автоматичну сегментацію, отримання ознак, класифікацію ML-моделями. Проведено попередню обробку вхідних зображень з метою забезпечення повноти сегментації листків та повноти ознак, що характеризують аномалії листків, які пов’язані з їх захворюваннями.

Розроблена модель може бути використана при проектуванні різних програмних комплексів, які будуть орієнтовані на вирішення задач класифікації об’єктів, які задані у вигляді зображень.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП ПРОЕКТУ

5.1. Основні компоненти та структура проекту інтелектуальної інформаційної системи

Розглянемо розроблення структури і створення інтелектуальної інформаційної системи виявлення захворювання листя рослин (табл. 5.1).

Табл. 5.1. Структура інтелектуальної інформаційної системи

Назва номінації	Python програма
Назва проекту	Інтелектуальна система виявлення захворювання листя рослин
Назва ВНЗ, факультету, спеціальності	НЛТУ, кафедра комп'ютерних наук, 122 «Комп'ютерні науки»
Прізвище, ім'я, по-батькові	Підставський Віталій Романович
Цілі і задачі проекту	Мета проекту – розробка та реалізація інтелектуальної системи для можливості розпізнавання основних видів хвороб листя рослин по їх зображеннях. Задачі проекту: • провести огляд літератури по даній тематиці; • проаналізувати існуючі системи прогнозування захворювання рослин; • розробити математичну модель інтелектуальної системи для діагностики захворювань рослин; • реалізувати програмну модель інтелектуальної інформаційно-довідкової системи;

	• педставити результати роботи цієї інтелектуальної системи.
Короткий зміст проекту	Розроблено та реалізовано інтелектуальну систему для розпізнавання основних видів хвороб листя рослин по їх зображеннях. Розглянуто існуючі методи та запропоновано метод цифрової обробки зображень для діагностики хвороб рослин за цифровими зображеннями листя у видимому спектрі, а також подальшого розпізнавання вибірки зображень. Проведено математичне моделювання запропонованого методу, розроблено та реалізовано класифікатор для визначення виду хвороби листя помідорів.

5.2. Стратегії проекту інтелектуальної системи виявлення захворювань рослин

Розглянуто ідею контролю стану рослин в теплицях за рахунок машинного навчання при масовому вирощуванні помідорів. Для цього розроблена платформа для моніторингу рослин у реальному часі.

З цією метою можна застосувати штучний інтелект до завдань сільського господарства. Завдання моніторингу стану рослин є дуже актуальним. Воно вирішується людьми не дуже ефективно, тому що теплиці великі, багато рослин і моніторити потрібно іноді кожен листок.

Якщо якесь захворювання помітити пізно, то вся теплиця може постраждати, і тоді фермери зазнають збитків.

Якщо вирішити за допомогою штучного інтелекту проблему втрати врожаю через зараження ослин, то потім можна масштабувати це рішення на весь сільськогосподарський ринок. Постає питання, як реалізувати цей бізнес-проект.

Це рішення – коли треба поставити в теплицях датчики для збирання точних даних про клімат та камери для візуального контролю рослин. Такі стаціонарні камери набагато дешевші, ніж будь-які роботи, але при цьому інформація, яка з них збирається, досить якісна для точного моніторингу стану рослин.

Основна функція даної інтелектуальної системи – визначення хвороб рослин, але, крім цього, в ній можна вирішити деякі завдання, які постають перед фермерами під час виробництва. Треба замінити весь процес прийняття рішень, який ґрунтується на візуальному моніторингу, та зробити його автоматизованим. Так, іноді фермерам потрібно визначати фазу цвітіння або контролювати спеціальну форму рослин. Все це можна робити за допомогою аналізу даних із камер. Іноді у цьому допомагає клімат: це може впливати на форму чи якісь захворювання рослин. Ще можна інтегрувати візуальні та кліматичні дані для того, щоб передбачати врожай і видавати фермерам рекомендації щодо того, як поводитися, щоб знизити витрати і підвищити прибутковість виробництва.

Багато сортів рослин, які є найбільш поширеними, коштують не дуже дорого, тому економія за рахунок технології виходить несуттєвою для фермерів. Робоча праця також не дуже дорога. Отже, цінність установки технології дуже велика.

Дуже важливо, щоб у команді роботою займалися люди з міжнародним досвідом, якщо цікавить міжнародний ринок. На жаль, людей, здатних забезпечити високий рівень міжнародних контактів, знайти досить складно.

У команді потрібна людина, яка відповідає лише за дані. Коли йдеться про роботу з будь-яким новим проектом, коли говорять про якийсь новий тип рослин чи новий регіон – тому від регіону до регіону все відрізняється, завдання теж відрізняються – ця людина знаходить все, що можна знайти у відкритих даних.

Перш ніж йти до клієнта, має бути прототип працюючого рішення. Після цього ведеться інтенсивний збір даних, вже разом із клієнтом та його агрономами, та розмічають дані. Потрібне спеціальне рішення для розмітки даних, щоб весь процес був якісний, інтенсивний, швидкий. Дуже важливий процес збору завдань від виробника, обробки відповідних даних, побудови моделей, які будуть вирішувати завдання, що стоять перед ними.

На початку роботи необхідно знайти комерційного партнера, з урахуванням якого можна розробляти технології, які будуть потрібні і корисні. Далі потрібно дивитися на ті рішення, які роблять – на яких ринках, у яких регіонах вони також можуть бути корисними.

Якщо говорити про рослини, то потрібно дивитися на те, яка вартість у конкретному регіоні цієї продукції, а також яка вартість там робочої сили. Тому треба бути сфокусованим на європейському ринку – там люди коштують дорого, ефект від застосування технології може бути високим.

Одне із завдань, які стоять перед фермерами – це контроль товарного виду рослин. Потрібно не просто стежити, щоб не було хвороб: рослини повинні ідеально виглядати. Тут ця технологія також ефективна. Якщо говорити про овочі, тобто цілком спеціалізовані виробники, які займаються розробкою рослин для дорогих ресторанів, тому дуже важливо, щоб усі рослини ідеально гарно виглядали, щоб вся продукція була рівною, достиглою, мала привабливий вигляд.

Важливо, щоб усі плоди були щільними, однакової форми. У цьому проекті такі завдання також можна вирішити – відстежується вид рослин, навіть якщо це не пов’язане з хворобами.

В основному для вирощування помідорів використовуються теплиці. Завдання з полями складніше. Основне питання тут – як встановити камери, щоб стежити за рослинами. У полях основний спосіб вирішення цього завдання – дрони. Поки що акцент робиться саме на роботу в теплицях і не орієнтуються на поля.

Фермерам важливо заздалегідь знати, що відбувається зі станом рослин, щоб керувати кліматом, підбирати умови для того, щоб кінцева вартість продукту виявилася максимально високою. Тому одне з найважливіших завдань – розробка технологій, які дозволять робити це в реальному часі, передбачати, що відбувається всередині рослин на основі аналізу даних різних спектрів. Плюс аналіз даних довкілля та інтеграція цих даних між собою для того, щоб передбачати та видавати рекомендації фермерам про те, що їм робити для того, щоб їхній продукт був максимально дорогим.

Стартап переважно живе на інвестиційні кошти. Можна очікувати виходу на прибутковість за кілька місяців, але основне завдання – це інвестиційні гроші.

Перші комерційні історії можуть початися після переорієнтації з українського ринку на західний. Тому потрібно працювати з компаніями, які можуть забезпечити встановлення обладнання, яке необхідне для того, щоб ці технології працювали. Відповідно, це – виробники теплиць, тому що ця технологія може стати частиною теплиці, або компанії, які роблять технології для догляду та контролю за рослинами. Тобто можна дізнатися, як рослини почуваються, вам і фермеру сказати, чи все відбувається правильно, чи якісь процеси потрібно оптимізувати.

Сільське господарство взагалі досить консервативна, нецифровізована галузь. В даній інформаційній системі є напрацювання, результати, метрики, моделі. Нині дуже гарний час для агротеху. Із хардової інфраструктури потрібні кліматичні сенсори. Вони включають стандартний набір параметрів, які потрібно знати в теплиці – температуру,

вологість, CO_2, освітлення. Використовуються бездротові сенсори, які дозволяють у високій якості знімати інформацію з теплиці – можна зробити тривимірні карти того, що відбувається з кліматом.

Рентабельність залежить не так від площі, як від типу рослин і від регіону, в якому це відбувається. Площа не так впливає впливає, хоча це може вплинути на бажання включатись у проект: потрібно вибирати великих виробників, у яких видно великий потенціал щодо масштабування потрібних технологій. Площа конкретного виробництва не така важлива: вона може бути малою, але таких виробництв може бути багато.

Хмарний сервіс для цих технологій – це не найкраще рішення. Тут збирають дуже багато даних, важливо збирати зображення високої якості. Тому чиста хмара неоптимальна.

В даній інформаційній системі не вирішують завдання передбачення стану рослин. Тільки якщо будуть використані візуальні дані разом із кліматом, це завдання іноді вирішується. Але більше це стосується якихось проблем із живленням. Що стосується шкідників чи інфекційних захворювань – це визначають лише тоді, коли це вже сталося.

Датасет збирають з усього, що є в інтернеті з відкритих даних. Наприклад, із будь-яких контестів, які проводилися на kegel та інших ресурсах. Збирають фотографії з різних агрономічних форумів, енциклопедій – потрібно все зібрати, розмітити.

Коли появляться клієнти, через якийсь час треба відпрацювати процедуру роботи з ними. Наприклад, клієнт має потребу детектувати якусь досить рідку хворобу рослин, яка може вбити всю теплицю, і зображень цієї хвороби не так багато. Вони самі, як тільки її бачать, фотографують і надсилають зображення на оброблення та в майбутньому на класифікацію цієї хвороби. Також спеціалістам варто регулярно виїжджати – особливо, якщо працюють з новою рослиною: робити фотографії, навчати персонал, щоб постійно надсилали дані для поповнення датасету та покращення якості інтелектуальної системи.

Результатом кінцевого продукту є інтелектуальна інформаційна система для виявлення захворювання рослин. Для цього потрібно пприїхати в теплицю, встановити камери-датчики, далі клієнт матиме дешборд, на якому видно все, що відбувається з кліматом, зі здоров'ям рослин. Клієнт може відзначати та відстежувати, що він робитиме із цими рослинами. Тобто, продукт – це середовище, в якому працює агроном, і в якому видно весь стан та здоров'я рослин.

5.3. Ефективне залучення ресурсів для запуску проекту виявлення захворювань рослин

Для того щоби запустити стартап проекту інтелектуальної системи виявлення захворювань рослин, можна використати експертизу своїх знайомих, близьких друзів, для того щоб вони долучилися для створення базового прототипу. Те, що називається мінімальним життєздатним продуктом і потім цей продукт уже нестиме клієнтам і пробувати його продавати. В залежності від того, що скаже клієнт, еволюціонувати потроху відповідно якщо проект створить для нас друг можливо безкоштовно, можливо за якісь мінімальні кошти. Потрібно зрозуміти, як ці відносини окреслити на майбутнє і тут є досить багато варіантів. З одного боку можна просто домовитись, що коли будуть перші клієнти, то можна розрахуватися з людиною або ж іноді прототипи можна замовляти і створювати за допомогою фріланс бірж. Для того щоб просто його зробити і почати показувати користувачам, користувачі, які будуть готові платити за цей продукт, потрібно знаходити механізми, як залучити айтішників, які будуть його далі розвивати і розробляти. Скільки потрібно коштів для початку стартапу, все дуже залежить. Якщо є проект, який можна зробити своїми силами, то на це не потрібно більше 100-200 доларів. Це для того, щоб купити домен, заплатити за поштовий сервіс і т.д. Якщо людина інженер чи програміст, можна весь повністю продукт зробити самому на

певних етапах. Можливо потрібно буде залучати зовнішню експертизу, для того щоб зробити візуальний дизайн, додаткове тестування продукту.

Якщо немає коштів, потрібно шукати ідеї. Коли вже можна це продати, можна це розбудовувати і додавати функціонал продукту чи потрібно унікальну ідею для стартапу. Питання унікальності ідеї не гарантує її успіх. Інколи потрібно використати ідею, яка вже існує на ринку, та її чимось покращити. Якщо використовується ідея існуючого продукту, то точно відомо, що для нього є ринок. Відповідно якщо цей ринок росте, то для цього продукту точно знайдеться окрема ділянка. Потрібно інколи доробити якийсь продукт або хотілось би залучити тих перших клієнтів, і складається враження, що якщо отримати кошти від якогось іноземного інвестора, то все зміниться. Насправді ні. Найчастіше по-перше іноземний інвестор не буде інвестувати гроші просто в ідею. Тому що йому потрібно розуміння того, що ти є цією людиною, яка дійсно може виростити цей проект, створити його і запустити його. Якщо незрозуміло, як саме розвивати цей проект, то кошти залучати точно не потрібно. Найчастіше кошти залучаються на етапі росту, коли в тебе є вже чітко сформульована ідея, як отримувати нових користувачів. Якщо у вас сьогодні є 100, якщо ти потратиш 100-200 тисяч доларів на рекламу, то через місяць чи через рік у вас буде 100 тисяч. Єдине хотілось би застерегти: кошти – це означає залучення не тільки зовнішніх ресурсів, але і відповідальність. Відповідно якщо ти залучаєш кошти, ти втрачаєш незалежність.

ВИСНОВКИ ДО РОЗДІЛУ 5

Урожай помідорів є важливим основним продуктом на ринку з високою комерційною цінністю та виробляється у великих кількостях. Хвороби шкодять здоров'ю рослини, що, у свою чергу, впливає на їх ріст. Щоб забезпечити мінімальні втрати вирощуваного врожаю, важливо стежити за його зростанням. Існують численні типи хвороб помідорів, які вражають листя культури із загрозливою швидкістю. У даному проекті використовується невелика зміна моделі згорткової нейронної мережі для виявлення та ідентифікації захворювань на листях помідорів. Основною ідеєю стартапу пропонованої роботи є знайти рішення проблеми виявлення хвороб листя рослин за допомогою підходу з використанням мінімальних обчислювальних ресурсів для досягнення результатів, порівняних із сучасними методами. Для цього стартапу в моделі нейронних мереж використовують автоматичне виділення ознак, щоб допомогти класифікувати вхідне зображення за відповідними класами захворювань. Ця запропонована система досягла середньої точності 98 %, що вказує на доцільність підходу нейронної мережі навіть у незручних умовах.

Можна припустити, що якщо не вжити належного догляду щодо цієї проблеми, це призведе до втрати грошей, часу, якості, кількості. Таким чином, основним мотивом є отримання гарного врожаю та збільшити норми виробітку. Захворювання рослин можна виявити за допомогою обробки зображень. Виявлення захворювання виконує кілька етапів, як-от попередня обробка зображення, виділення ознак, класифікація та прогнозування для виявлення захворювання. Таким чином, створена система розпізнавання захворювань рослин може використовуватися в оцінці високоточного зображення рослин для правильного їх лікування та подальшої профілактики.

ВИСНОВКИ

Питання діагностики стану сільськогосподарських культур та інформаційного забезпечення прийняття рішень щодо їх захисту є одними з основних проблем управління врожаєм. Розроблено модель алгоритмів діагностики захворювань рослин щодо зображення їх листя. Основна ідея запропонованої моделі алгоритмів полягає у розпізнаванні стану культур за їх діагностичними ознаками. При цьому формування діагностичних ознак спирається на обчислення різних статистичних характеристик кожного фрагмента вихідного зображення.

Дослідження проведено з метою вирішення задачі діагностики хвороб рослин. Як вхідні дані використано датасет PlantVillage Dataset, що включає зображення здорових рослин і зображення для п’яти видів захворювань листя рослин: Bacterial spot, Late blight, Septoria leafspot, Target Spot, Tomato Yellow Leaf Curl Virus. Розроблено інформаційну систему, що включає такі етапи: автоматичну сегментацію, отримання ознак, класифікацію ML-моделями. Проведено попередню обробку вхідних зображень з метою забезпечення повноти сегментації листків та повноти ознак, що характеризують аномалії листків, які пов’язані з їх захворюваннями.

Розроблена модель може бути використана при проектуванні різних програмних комплексів, які будуть орієнтовані на вирішення задач класифікації об’єктів, які задані у вигляді зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Тимощук П.В. Штучні нейронні мережі / – Львів: Видавництво Львівської Політехніки, 2011. – 444 стор.

2. Адаменко В.О., Мірських Г.О. Штучні нейронні мережі в задачах реалізації матеріальних об’єктів. Частина 1. Принципи побудови та класифікація / – Вісник НТУУ "КПІ". Серія Радіотехніка. Радіоапаратобудування, 2011. – №47 – С. 176-189.

3. Адаменко В.О., Мірських Г.О. Штучні нейронні мережі в задачах реалізації матеріальних об'єктів. Частина 2. Особливості проектування та застосування / – Вісник НТУУ "КПІ". Серія – Радіотехніка. Радіоапаратобудування, 2012. – №48 – С. 213–221.

4. Лубко Д.В., Шаров С.В. Методи та системи штучного інтелекту: навчальний посібник / – Мелітополь: ФОП Однорог Т.В., 2019. – 264 с.

5. Ситник В.Ф., Краснюк М.Т. Інтелектуальний аналіз даних (дейтамайнінг) / – Київ: КНЕУ, 2007. – 376 с.

6. Копей В.Б. Мова програмування Python для інженерів і науковців. Навчальний посібник / – Івано-Франківськ : ІФНТУНГ, 2019. – 272 с.

7. Субботін С.О. Нейронні мережі: теорія та практика. Навчальний посібник / – Житомир : Вид. О. О. Євенок, 2020. – 184 с.

8. Безрук В.М., Свид І.В., Корсун І.В. Нейронні технології в телекомунікаціях і системах управління. Навчальний посібник / – Харків. : Компанія СМІТ, 2008. – 230 с.

9. Желдак Т.А., Коряшкіна Л.С., Ус С.А. Нечіткі множини в системах управління та прийняття рішень. Навчальний посібник. / – Дніпро: Національний технічний університет "Дніпровська політехніка", 2020. – 387 с.

10. Троцько В.В. Методи штучного інтелекту / – К.: Університет "КРОК", 2020. – 86 с.

ДОДАТКИ

Додаток А

1.ipynb

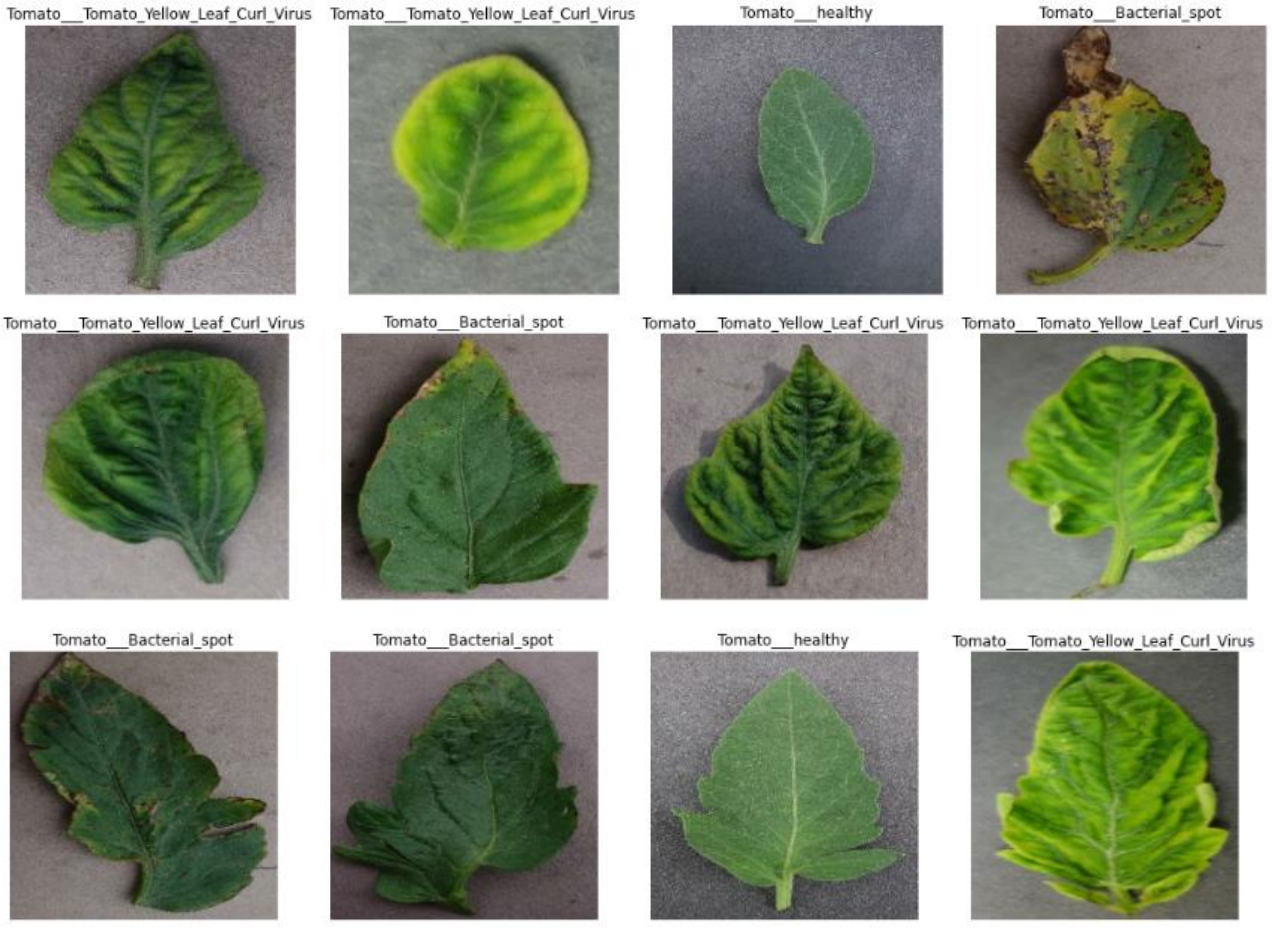
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import numpy as np

IMAGE_SIZE=256
BATCH_SIZE=32

images_dataset=tf.keras.preprocessing.image_dataset_from_directory(
 'dataset',
 shuffle=True,
 image_size=(IMAGE_SIZE,IMAGE_SIZE),
 batch_size=BATCH_SIZE,
 )

class_names=images_dataset.class_names
class_names
print("Class Names:", class_names)

len(images_dataset)

dataset_length = tf.data.experimental.cardinality(images_dataset).numpy()
print("Кількість зображень у датасеті:", dataset_length)

dataset_length = tf.data.experimental.cardinality(images_dataset).numpy()
print("Кількість зображень у датасеті:", dataset_length * BATCH_SIZE)

for image_batch, label_batch in images_dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
(32, 256, 256, 3)
[1 1 1 1 1 1 2 0 0 1 1 2 2 1 2 2 0 0 2 0 0 0 0 1 2 2 0 2 0 1 1 1]

for image_batch, label_batch in images_dataset.take(1):
    print(image_batch[0].numpy())

import matplotlib.pyplot as plt
plt.imshow(image_batch[0].numpy().astype("uint8"))
```

```
plt.show()
```

```
[[[ 97.  96.  91.]
  [ 97.  96.  91.]
  [ 98.  97.  92.]
  ...
  [ 76.  76.  78.]
  [ 79.  79.  81.]
  [ 82.  82.  84.]]

 [[ 97.  96.  91.]
  [ 97.  96.  91.]
  [ 97.  96.  91.]
  ...
  [ 79.  79.  81.]
  [ 80.  80.  82.]
  [ 82.  82.  84.]]

 [[ 97.  96.  91.]
  [ 97.  96.  91.]
  [ 97.  96.  91.]
  ...
  [ 81.  81.  83.]
  [ 80.  80.  82.]
  [ 80.  80.  82.]]

 ...

 [[150. 155. 148.]
  [150. 155. 148.]
  [149. 154. 147.]
  ...
  [125. 129. 128.]
  [116. 120. 119.]
  [112. 116. 115.]]

 [[150. 155. 148.]
  [150. 155. 148.]
  [149. 154. 147.]
  ...
  [139. 143. 142.]
  [133. 137. 136.]
  [131. 135. 134.]]

 [[150. 155. 148.]
  [150. 155. 148.]
  [149. 154. 147.]
  ...
  [139. 143. 142.]
  [139. 143. 142.]
  [140. 144. 143.]]]
```

```
for image_batch, label_batch in images_dataset.take(1):
    plt.imshow(image_batch[0].numpy().astype('uint8'))
    plt.title(class_names[label_batch[0]])
    plt.axis('off')
```


```
plt.figure(figsize=(18,18))
for image_batch, label_batch in images_dataset.take(1):
    for i in range (12):
        ax=plt.subplot(3,4, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis('off')
```




```
len(images_dataset)
94
```

```
def get_dataset_partitions_tf(ds,train_split=0.8, val_split=0.1,
test_split=0.1,shuffle=True, shuffle_size=10000):
    ds_size=len(ds)
    if shuffle:
        ds=ds.shuffle(shuffle_size,seed=12)

    train_size= int(train_split* ds_size)
    val_size=int(val_split* ds_size)

    train_ds=ds.take(train_size)

    val_ds=ds.skip(train_size).take(val_size)

    test_ds=ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds=get_dataset_partitions_tf(images_dataset)

model.fit(train_ds, epochs=10, validation_data=val_ds)

print(len(train_ds),len(val_ds),len(test_ds))

len_train = tf.data.experimental.cardinality(train_ds).numpy() * BATCH_SIZE
len_val = tf.data.experimental.cardinality(val_ds).numpy() * BATCH_SIZE
len_test = tf.data.experimental.cardinality(test_ds).numpy() * BATCH_SIZE
print("Кількість зображень у тренувальному наборі:", len_train)
print("Кількість зображень у валідаційному наборі:", len_val)
print("Кількість зображень у тестовому наборі:", len_test)

75 9 10

train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experi
mental.AUTOTUNE)
val_ds=val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experiment
al.AUTOTUNE)
test_ds=test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.experime
ntal.AUTOTUNE)

resize_and_rescale=tf.keras.Sequential([
        layers.experimental.preprocessing.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])
```

```
data_augmentation=tf.keras.Sequential([

   layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])
```

Додаток Б

2.ipynb

```
import numpy as np
import cv2
from os import listdir
from sklearn.preprocessing import LabelBinarizer
from keras.models import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation, Flatten, Dropout, Dense
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

EPOCHS = 25
INIT_LR = 1e-3
BS = 32
default_image_size = tuple((256, 256))
image_size = 0
directory_root = '../input/plantvillage/'
width=256
height=256
depth=3

directory_root = ' C:\\1\Pictures\plantdisease_dataset '

def convert_image_to_array(image_dir):
    try:
        image = cv2.imread(image_dir)
        if image is not None :
            image = cv2.resize(image, default_image_size)
            return img_to_array(image)
        else :
            return np.array([])
```

```
    except Exception as e:
        print(f"Error : {e}")
             return None

label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer,open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)

print("[INFO] Spliting data to train, test")
x_train, x_test, y_train, y_test = train_test_split(np_image_list,
image_labels, test_size=0.2, random_state = 42)

aug = ImageDataGenerator(
    rotation_range=25, width_shift_range=0.1,
    height_shift_range=0.1, shear_range=0.2,
    zoom_range=0.2,horizontal_flip=True,
    fill_mode="nearest")

model = Sequential()
inputShape = (height, width, depth)
chanDim = -1
if K.image_data_format() == "channels_first":
    inputShape = (depth, height, width)
    chanDim = 1
model.add(Conv2D(32, (3, 3), padding="same",input_shape=inputShape))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(3, 3)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(64, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
model.add(BatchNormalization(axis=chanDim))
model.add(Conv2D(128, (3, 3), padding="same"))
model.add(Activation("relu"))
```

```
model.add(BatchNormalization(axis=chanDim))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1024))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(n_classes))
model.add(Activation("softmax"))
model.add(Activation("relu"))

history = model.fit_generator(
    aug.flow(x_train, y_train, batch_size=BS),
    validation_data=(x_test, y_test),
    steps_per_epoch=len(x_train) // BS,
    epochs=EPOCHS, verbose=1
    )

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()

print("[INFO] Calculating model accuracy")
scores = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {scores[1]*100}")
```

```
[INFO] Calculating model accuracy
589/589 [==============================] - 36s 61ms/step
Test Accuracy: 97.34012668727816
```

Додаток С

3.py

```
import streamlit as st
import tensorflow as tf
from PIL import Image
import numpy as np
import random

st.set_page_config(
    page_title = 'Прогноз хвороби листя помідорів',
    page_icon = ":tomato:",

    initial_sidebar_state = 'auto'
)

def prepare(file):
    img_array=file/255
    return img_array.reshape(-1,128,128,3)

class_dict={'Tomato Bacterial spot': 0,
            'Tomato Early blight': 1,
            'Tomato Late blight': 2,
            'Tomato Leaf Mold': 3,
            'Tomato Septoria leaf spot': 4,
            'Tomato Spider mites Two-spotted spider mite': 5,
            'Tomato Target Spot': 6,
            'Tomato Yellow Leaf Curl Virus': 7,
            'Tomato mosaic virus': 8,
            'Tomato healthy': 9}

def prediction_cls(prediction):
    for key, clss in class_dict.items():
        if np.argmax(prediction)==clss:
            return key

@st.cache
def load_image(image_file):
    img=Image.open(image_file)
    img=img.resize((128,128))
        return img
```

```
 def main():
    with st.sidebar:
        st.header("ПОМІДОРИ")
        st.image('./img2.jpg')
    st.title("Прогноз хвороби листя помідорів")
    st.subheader("Завантажити зображення листя помідорів, щоб передбачити
їх захворювання.")
    image_file=st.file_uploader("Завантажити
зображення",type=["png","jpg","jpeg"])
    if image_file == None:
        st.warning("Завантажте зображення листя помідорів, щоб передбачити
їх захворювання.")
    else:
        if st.button("Процес"):
            img=load_image(image_file)
            img=tf.keras.preprocessing.image.img_to_array(img)
            model=tf.keras.models.load_model("model_vgg19.h5")
            img=prepare(img)
            with st.sidebar:
                st.image(img,caption="Завантажити зображення")
                x = random.randint(90,98)+ random.randint(0,99)*0.01
                st.subheader("Виявлене захворювання:")
                if (prediction_cls(model.predict(img))) == 'Tomato
healthy':
                    st.success("Рослина здорова")
                    st.write("Точність передбачення %:",x)
                else:
                    st.warning(prediction_cls(model.predict(img)))
                    st.write("Точність прогнозу %:",x)
            if (prediction_cls(model.predict(img))) == 'Tomato Bacterial
spot':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Early
blight':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Late
blight':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Leaf
Mold':
                st.subheader("Засоби захисту:")
```

```
            elif (prediction_cls(model.predict(img))) == 'Tomato Septoria
leaf spot':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Spider
mites Two-spotted spider mite':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Target
Spot':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato Yellow
Leaf Curl Virus':
                st.subheader("Засоби захисту:")
            elif (prediction_cls(model.predict(img))) == 'Tomato mosaic
virus':
                st.subheader("Засоби захисту :")

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