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))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)

(рівень вищої освіти)

на тему: **Розроблення моделі діагностики хвороб картоплі на основі згорткових нейромереж**

Виконав: студент 2 курсу, групи КНС-21
спеціальності
122 – "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Улян Т.І.
(прізвище та ініціали)

Керівник Процах Н.П.
(прізвище та ініціали)

Рецензент Яковлєв Д.І.
(прізвище та ініціали)

Львів – 2025 р.

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти перший (бакалаврський)

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

ЗАТВЕРДЖУЮ
Завідувач кафедри КН

Борецька І.Б.
"10" червня 2025 року

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

Улян Тарас Іванович
(прізвище, ім'я, по батькові)

1. Тема роботи **Розроблення моделі діагностики хвороб картоплі на основі згорткових нейромереж**

керівник роботи Процах Н.П, доктор. фіз.-мат. наук, професор.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 15.11. 2024 р. № С-882

2. Термін подання студентом роботи 10.06. 2025 р.

3. Вихідні дані до роботи:

- вивчити предметну область, проаналізувати існуючі моделі класифікації захворювань рослин;
- розглянути і використати алгоритми, які лежать в основі математичної моделі класифікації хвороб картоплі;
- спроектувати інформаційну систему з допомогою мови програмування Python та відповідних бібліотек машинного навчання та візуалізації результатів.
- представити результати роботи інформаційної системи.

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

Розділ 1. Стан проблемної області

Розділ 2. Інформаційне та математичне забезпечення

Розділ 3. Програмне та технічне забезпечення

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

додаток А, додаток Б

6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних даних	18.11-30.12.2024	виконано
2	Розділ 1. Стан проблемної області	01.01-15.02.2025	виконано
3	Розділ 2. Інформаційне та математичне забезпечення	16.02-30.03.2025	виконано
4	Розділ 3. Програмне та технічне забезпечення	01.04-15.05.2025	виконано
5	Оформлення дипломної роботи	16.05-30.05.2025	виконано
6	Підготовка до захисту дипломної роботи, оформлення презентації	01.06-10.06. 2025	виконано

Студент _____ (підпис) Улян Т.І. (прізвище та ініціали)

Керівник роботи _____ (підпис) Процах Н.П. (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 68 сторінок пояснювальної записки, 15 рисунків, 2 таблиці, 12 джерел, 2 додатки.

У дипломній роботі проведено дослідження сучасних методів виявлення захворювань картоплі за допомогою алгоритмів глибокого навчання. Розроблено програмне забезпечення для інформаційної системи, що використовує згорткові нейронні мережі для аналізу зображень листя картоплі. У результаті реалізовано математичну модель, яка дозволяє точно класифікувати захворювання та оцінювати ймовірність їх наявності. Використано мову програмування Python та бібліотеки, такі як TensorFlow та Keras для обробки даних та навчання моделі. Проведено тестування створеної системи, що підтвердило її ефективність та точність у розпізнаванні захворювань.

Ключові слова: *нейронні мережі, класифікація та прогнозування захворювань рослин, Python, Google Colab, Pandas, TensorFlow, Keras*.

ABSTRACT

Diploma paper contains 68 pages of explanatory note, 15 figures, 2 tables, 12 sources, 2 appendix.

The thesis researched modern methods of detecting potato diseases using deep learning algorithms. Software was developed for an information system that uses convolutional neural networks to analyze potato leaf images. As a result, a mathematical model was implemented that allows for accurate classification of diseases and assessment of the probability of their presence. The Python programming language and libraries such as TensorFlow and Keras were used for data processing and model training. The created system was tested, which confirmed its effectiveness and accuracy in disease recognition.

Keywords: *neural networks, classification and prediction of plant diseases, Python, Google Colab, Pandas, TensorFlow, Keras*.

ТЕХНІЧНЕ ЗАВДАННЯ

В дипломній роботі потрібно розробити інформаційну систему діагностики хвороб картоплі на основі згорткових нейромереж, для цього потрібно вирішити такі завдання:

1. Провести огляд сучасних технологій та методів класифікації захворювань рослин.

2. Розробити математичну модель для аналізу та класифікації хвороб картоплі.

3. Визначити ключові характеристики та параметри, що впливають на точність класифікації хвороб.

4. Реалізувати програмну модель інформаційної системи на основі мови програмування Python та бібліотек глибокого навчання, таких як TensorFlow та Keras.

5. Провести тестування розробленої системи для оцінки її ефективності та точності прогнозування захворювань.

6. Реалізувати графічний інтерфейс інформаційної системи для візуалізації результатів роботи та інтерпретації даних та результатів класифікації.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Accuracy – відсоток правильних передбачень моделі на навчальних даних;

CNN (Convolutional Neural Networks) – згорткова нейронна мережа;

Dense – шар, в якому кожен нейрон підключений до кожного нейрона попереднього шару;

DL (Deep learning) – глибоке навчання;

Flatten – шар у нейронних мережах використовується для перетворення багатовимірного масиву у одномірний масив;

Fully Connected Layers – повнозв'язні шари;

KNN (K-Nearest Neighbors) – метод k-найближчих сусідів;

ML (Machine learning) – машинне навчання;

Loss – міра того, як погано модель передбачає значення, чим менше значення втрат, тим краща модель;

PIL (Python Imaging Library) – модуль для роботи з зображеннями;

Potato_Early_blight – ранній фітофтороз;

Potato_Late_blight – пізній фітофтороз;

Potato_healthy – здорове листя картоплі;

ReLU (Rectified Linear Unit) – функція активації, яка застосовується до виходу кожного нейрона;

Softmax – функція активації, яка перетворює лінійні виходи нейронів в ймовірності;

Sparse Categorical Crossentropy – функція втрат, яка вимірює різницю між справжніми мітками та передбаченими ймовірностями;

SVM (Support Vector Machine) – метод опорних векторів;

ШІ – штучний інтелект.

ВСТУП

Актуальність дипломної роботи

Актуальність дипломної роботи зумовлена збільшенням попиту на картоплю як один з основних продуктів харчування у світі. Хвороби картоплі становлять серйозну загрозу для сільськогосподарського виробництва, оскільки можуть призводити до значних втрат урожаю. Забезпечення високої урожайності та якості продукції є пріоритетом для аграріїв, що підкреслює необхідність своєчасної діагностики. Використання традиційних методів виявлення захворювань обмежене, що негативно впливає на продуктивність. Розробка алгоритмів глибокого навчання, зокрема згорткових нейронних мереж, відкриває можливості для автоматизації процесів класифікації. Це дозволяє не лише підвищити точність діагностики, але й зменшити витрати часу. Інформаційні технології стають невід'ємною частиною агросектору, що робить цю роботу особливо актуальною. Ця работа має потенціал для практичного впровадження в аграрну галузь.

Предмет дослідження – аналіз та моделювання даних, пов'язаних з ідентифікацією та класифікацією захворювань картоплі за допомогою алгоритмів глибококо навчання.

Об’єкт дослідження – захворювання картоплі, які впливають на якість та врожайність рослин.

Мета роботи – розроблення інформаційної системи для виявлення та класифікації захворювань картоплі на основі алгоритмів глибокого навчання.

Завдання:

1. Провести огляд сучасних технологій та методів класифікації захворювань рослин.

2. Розробити математичну модель для аналізу та класифікації хвороб картоплі.

3. Визначити ключові характеристики та параметри, що впливають на точність класифікації хвороб.

4. Реалізувати програмну модель інформаційної системи на основі мови програмування Python та бібліотек глибокого навчання, таких як TensorFlow та Keras.

5. Провести тестування розробленої системи для оцінки її ефективності та точності прогнозування захворювань.

6. Реалізувати графічний інтерфейс інформаційно системи для візуалізації результатів роботи системи та інтерпретації даних та результатів класифікації.

Практичне значення одержаних результатів

Практичне значення отриманих результатів роботи полягає в можливості своєчасного виявлення захворювань картоплі, що може суттєво підвищити врожайність. Розроблена інформаційна система дозволяє фермерам зменшити витрати часу та ресурсів на моніторинг стану рослин. Використання згорткових нейронних мереж забезпечує високу точність класифікації, що сприяє прийняттю обґрунтованих рішень щодо лікування та профілактики. Результати дослідження можуть бути впроваджені в аграрному секторі для автоматизації процесів виявлення захворювань. Крім того, інформаційна система може слугувати основою для розвитку подібних технологій в інших сферах сільського господарства. Отримані дані сприяють підвищенню загальної ефективності управління посівами картоплі та зменшують ризики економічних втрат у виробництві.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Класифікація захворювань картоплі методами глибокого навчання

Картопля - найважливіша овочева культура у світі. Її широке розмаїття та високий рівень споживання роблять її гарним бізнес-варіантом для багатьох виробників. Картоплю, четвертий за важливістю основний продукт харчування у світі, вирощують приблизно в 130 країнах і 95 країнах, що розвиваються. Світове виробництво картоплі, в тому числі в країнах, що розвиваються, неухильно зростає з минулого року. Однак, за оцінками, понад 32% картоплі щороку втрачається через хвороби та шкідників [1].

Вирощування картоплі домінує в сільському господарстві ще 125 країн. Але навіть ці культури вразливі до інфекцій та хвороб, які поділяються на дві основні категорії: (i) рання фітофтороз і (ii) фітофтороз. Насправді, картопля має вищу загальну поживну цінність, ніж багато інших фруктів і овочів, що робить її доступним джерелом їжі в усьому світі [3].

Фітофтороз вражає листя, стебла і бульби. Уражені хворобою листки здуваються і засихають. Висихаючи, листя стає коричневим або чорним. Вирішенням проблеми є висока вологість, низькі температури та підтримання листя у вологому стані. Первинний опік також є специфічним захворюванням, яке виникає на листках на будь-якій стадії росту і викликає характерні плями та опіки на листках. Первинний опік спочатку проявляється у вигляді невеликих чорних уражень на старих листках. Ураження стебла також схожі на ураження листя і можуть охоплювати рослину, якщо вони виникають біля краю ґрунту. Для вирішення цієї проблеми необхідне тепле, вологе і вологе повітря [4].

Швидке і точне виявлення хвороб картоплі має вирішальне значення для мінімізації впливу хвороб на рослини. Ручний моніторинг фермерами є складним і непрактичним, оскільки вимагає багато часу і глибоких знань. Виявлення малорухомих видів хвороб рослин призводить до неконтрольованого поширення хвороб рослин. Крім того, фермери часто описують хвороби рослин у приблизний спосіб і роблять припущення, що робить результати ідентифікації помилковими,

оскільки симптоми на листках мають схожість, яку важко визначити на перший погляд [5]. Не звертаючись за порадою до експертів у галузі хвороб рослин, фермери використовують результати особистої ідентифікації як орієнтир для запобігання поширенню хвороб на посіви. Як наслідок, профілактичні заходи, що вживаються фермерами, є неефективними і навіть завдають шкоди посівам через неадекватну інформацію та неправильну інтерпретацію інтенсивності розвитку хвороби, надмірне та недостатнє обприскування [5]. Ця проблема стала натхненням для пропонованої роботи, яка має на меті полегшити фермерам швидку і точну ідентифікацію та класифікацію хвороб рослин картоплі.

Представлено підхід на основі згорткових нейронних мереж для ідентифікації та класифікації двох поширених інфекцій картоплі. Фермери можуть легко виявляти хвороби картоплі, використовуючи запропонований метод з невеликими обчислювальними зусиллями.

Глибоке навчання (ГН) характеризується складною ієрархічною структурою, яка з'єднує кілька внутрішніх шарів для функцій зондування та репрезентативного навчання. У реальному світі воно використовується для репрезентативного навчання для вилучення важливої інформації зі спостережуваних даних. У штучних процесах для вилучення ознак використовується метод спроб і помилок.

Як показано на рисунку 1.1.1, запропонована методологія складається з чотирьох основних етапів: збір даних, попередня обробка даних, доповнення даних та класифікація зображень [2].

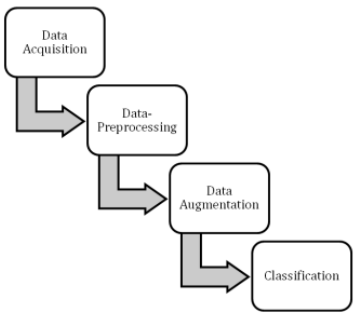

Рисунок 1.1 – Запропонована методологія визначення захворювання картоплі

Збір даних. Зображення різної роздільної здатності та розміру були отримані з різних джерел, в тому числі зібрані авторами на картопляній фермі в Патуахалі. Також була використана база даних зображень з відкритим доступом Kaggle. Автори

зібрали 7848 зображень безпосередньо з полів і 2152 зображення з Kaggle. Загалом для завершення дослідження було використано 10 000 зображень. Всі зображення були розподілені на три класи. Раннє вицвітання, пізнє вицвітання та здорові.

Фітофтороз - це тип захворювання рослин, що викликається бактерією Alternaria solani. Невеликі чорні плями переростають у великі, від коричневих до чорних, округлої або овальної форми ураження, які іноді обмежуються жилками листя, але також можуть бути виявлені і на сочевиці. Потім чорний грибок з'являється на нижньому боці листків. В'янення бульб може бути викликане раннім фітофторозом. Хвороба починає поширюватися, коли температура перевищує 26° С. Захворювання зазвичай виникає, коли активність картоплі знижується через високі температури, сухість і нестачу добрив.

Рисунок 1.2 – Рання стадія хвороби листя картоплі

Фітофтороз - це хвороба рослин, яку спричиняє бактерія Phytophthora infestans. Найчастіше вона виникає в роки з низькими температурами та рясними опадами і завдає значної шкоди посівам картоплі.

Рисунок 1.3 – Фітофтороз листя картоплі

Здорові листки виглядають свіжими та здоровими. Попередня обробка для видалення ділянок за межами області інтересу може зменшити рівень шуму на зображенні. Зображення з надто високим рівнем шуму не використовуються. Для

набору даних вхідні зображення потрібно масштабувати до 256x256 пікселів після того, як вони були зібрані з різних джерел і в різних розмірах.

Доповнення даних - це метод, який використовується для модифікації даних без спотворення їхнього початкового значення. У цьому дослідженні слід використовувати доповнення даних. Параметризація використовується в цьому дослідженні шляхом автоматичного застосування простих геометричних перетворень, таких як переклад, обертання, масштабування, панорамування, а також вертикальне і горизонтальне перенесення [6].

Рисунок 1.4 – Здорове листя картоплі

Класифікація зображень Машинне навчання (ML), також відоме як глибоке навчання (DL), глибоке нейронне навчання або глибокі нейронні мережі, є компонентом штучного інтелекту (ШІ). Глибоке навчання включає більше рівнів, ніж передбачає термін глибоке в машинному навчанні. Методи глибокого навчання підняли планку в різних сферах, таких як виявлення об'єктів, розпізнавання мови, класифікація об'єктів і класифікація зображень [7].

Згорткові нейронні мережі - один з найпоширеніших класів глибокого навчання. Згорткові нейронні мережі використовуються в різних дослідженнях для виявлення хвороб рослин на основі стану листя. Один або кілька згорткових шарів, організованих у групи відповідно до їхніх функцій, утворюють згорткову нейронну мережу в цілому. За шаром субдискретизації зазвичай слідує один або декілька зв'язаних шарів, що є звичайним явищем у нейронних мережах. Набір ознак в обмеженому просторі попереднього шару є входом кожного шару ознак.

1.2. Алгоритми машинного навчання для визначення захворювань рослин

Картопля є одним з основних продуктів харчування. Хвороби картоплі є основною причиною зниження якості та врожайності картоплі. Відомо, що багато хвороб картоплі можна виявити за зовнішнім виглядом листя. У цьому контексті можлива ідентифікація хвороб картоплі на основі алгоритмів машинного навчання, особливо згорткових нейронних мереж, які можуть автоматизувати процес розпізнавання типів хвороб картоплі. Відомо, що картопля уражається низкою хвороб до та після збирання врожаю [8]. Хвороби під час вегетації картоплі значно знижують якість та врожайність.

Однією з проблем, пов'язаних з втратами врожаю, є пізнє виявлення або неправильна діагностика хвороб картоплі. Наразі хвороби картоплі виявляються фермерами вручну, що забирає багато часу. Як наслідок, профілактика захворювань є неефективною, а тип хвороби картоплі часто ідентифікується неправильно, що призводить до зниження врожайності. Методи, засновані на алгоритмах машинного навчання для виявлення хвороб картоплі за зовнішнім виглядом листя, показали свою ефективність у ряді досліджень [2].

Інспекція на основі машинного навчання зазвичай складається з трьох етапів: попередня обробка зображення, виділення ознак і класифікація. Як методи класифікації зазвичай використовують машини опорних векторів (SVM), k-найближчих сусідів (k-NN) та лінійну регресію [3]. До загальних проблем цих методів належать нездатність обробляти складні фонові зображення і нездатність виявляти кілька об'єктів на зображенні. Алгоритми на основі згорткових нейронних мереж (CNN) підходять для вирішення цієї проблеми [4].

В останні роки якість класифікації зображень стрімко покращилася завдяки збільшенню швидкості обчислень і широкому використанню різних алгоритмів, таких як ImageNet, Large-Scale, Visual Recognition Challenge, VGG, GoogLeNet і ResNet [5]; YOLO, RCNN [6] та інші методи глибокого навчання були продемонстровані в задачах виявлення об'єктів. Архітектура CNN складається з двох основних етапів: вилучення ознак і класифікації. Представлення вилучених ознак передається компоненту класифікації архітектури, після чого модель робить

висновки про ймовірність належності дефекту до певного класу. Ваги та упередження моделі оптимізуються шляхом навчання нейронної мережі за допомогою алгоритму зворотного поширення помилки.

Підхід машинного навчання використовується для вирішення проблеми виявлення хворих листків картоплі на зображеннях. Система знань, розроблена для розв'язання цієї задачі, базується на згортковій нейронній мережі, яка працює з відкритим набором даних про хворі листки картоплі. Набір даних містить три класи зображень: здорові листки Healthy, рання хвороба листя Early_Blight та пізня хвороба листя Late_Blight, по 809, 152 та 1000 зображень для кожного класу. Зображення листків (рис. 1.5) мають колірний простір RGB і представлені у форматі jpeg з розміром 256 x 256 пікселів. Початкова вибірка зображень була розділена на два класи, тестовий і навчальний, з розподілом вибірки 1:9.

Рисунок 1.5 – Вихідні зображення з датасету для розпізнавання хвороб листя картоплі

Програмний код інформаційної системи розроблено в Google Colab на мові програмування Python, а основною бібліотекою машинного навчання є TensorFlow та її модуль Keras. Модель згорткової нейронної мережі - Sequential з декількома згортковими шарами для обробки зображень, архітектуру якої візуалізовано на рисунку 1.6.

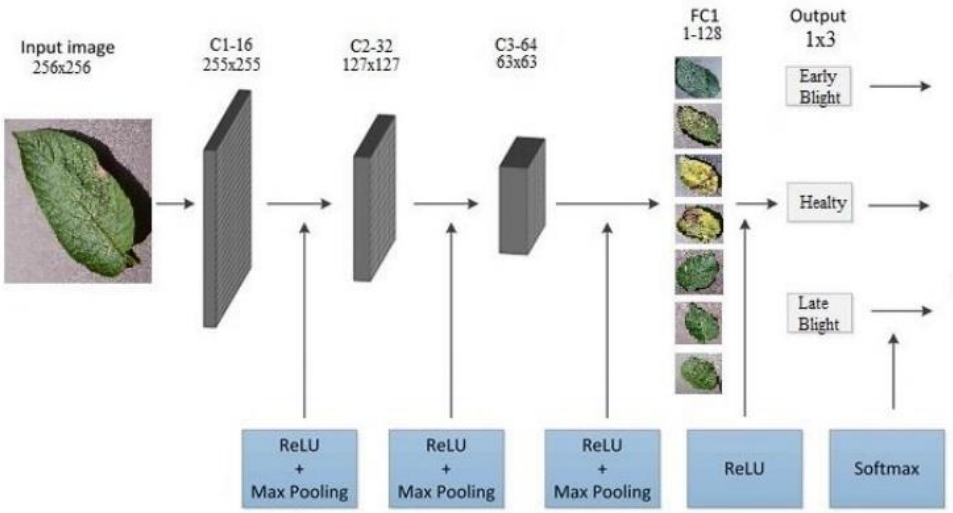

Рисунок 1.6 – Архітектура CNN для розпізнавання хвороб листя картоплі

Основними методами навчання нейронної мережі є алгоритм зворотного поширення помилки та використання матриці згортки для виявлення закономірностей на зображенні. Помилки розпізнавання розраховувалися за методом середньоквадратичного відхилення.

Алгоритми та моделі CNN. Згорткова нейронна мережа - це алгоритм глибокого навчання, який може приймати вхідне зображення, присвоювати важливість різним напрямкам або об'єктам на зображенні та відрізняти їх один від одного. ШНМ можуть успішно вловлювати просторові та часові залежності на зображенні, застосовуючи відповідні фільтри. Архітектура згорткової нейронної мережі подібна до моделі з'єднання нейронів у людському мозку. Окремі нейрони реагують лише на стимули в обмеженій зоровій області, яка називається рецептивним полем. Кластери цих ділянок перекриваються і покривають все поле зору. Ці шари мають наступні призначення.

Шар згортки - фільтр проходить через зображення, скануючи кілька пікселів за раз і створюючи карту об'єктів, яка передбачає клас, до якого належить кожен об'єкт.

Шар об'єднання зменшує кількість інформації для кожної ознаки, отриманої в шарі згортки, і зберігає найважливішу інформацію.

Повністю підключений вхідний шар приймає вихід попереднього шару, коригує його і перетворює в єдиний вектор, який може бути використаний як вхідний для наступного етапу.

Перший повністю підключений шар отримує вхідні дані від аналізу об'єкта і застосовує ваги для прогнозування правильної мітки.

Повністю підключений вихідний шар забезпечує максимальний потенціал кожної етикетки.

Щільний шар - це шар з глибокими зв'язками, тобто кожен нейрон у щільному шарі отримує вхідні дані від усіх нейронів у попередньому шарі.

Dropout (випадання). Метод відсіву працює шляхом випадкового зменшення кількості взаємопов'язаних нейронів у нейронній мережі. На кожному кроці навчання кожен нейрон має шанс бути видаленим або випасти з відповідного входу з'єднаних нейронів. Ця методика мінімізує можливість перенавчання, оскільки кожен нейрон у шарі стає самодостатнім, коли він дізнається свої вагові значення.

Максимальний шар об'єднання: процес об'єднання, який обчислює максимальну корекцію карти об'єктів і використовує її для створення карти об'єктів зі зменшеною вибіркою, що використовується після шару згортки.

Шар згортки також має функцію активації нейронів, яка визначає важливість отриманих параметрів для зміни ваг нейронної мережі за допомогою алгоритму зворотного поширення помилки.

Гіперпараметри - це змінні, які визначають структуру мережі, наприклад, кількість прихованих шарів, а також змінні, які визначають спосіб навчання мережі (швидкість навчання). Гіперпараметри задаються перед початком навчання мережі.

Алгоритм навчання згорткової нейронної мережі для виявлення хворих листків на картоплі показано на рисунку 1.7. Процес навчання починається з імпорту бібліотеки для машинного навчання, яка працює з послідовностями, архівами, зображеннями та хмарними сховищами. Після імпорту зображення розподіляються на кілька класів і поміщаються в тимчасове сховище для навчання. Наступним кроком є визначення архітектури мережі, яка буде розглянута більш детально пізніше. Основний процес навчання займає близько 30 хвилин [9]. Потім моделі та їхні ваги зберігаються в хмарному сховищі, щоб процес навчання не потрібно було повторювати в майбутньому.

Рисунок 1.7 – Алгоритм навчання згорткової нейронної мережі

На цьому рисунку зображено блок-схему процесу розроблення нейромережевої моделі для класифікації зображень картоплі. Предметна область стосується створення моделі для діагностики хвороб картоплі за допомогою методів глибокого навчання. Початок - старт процесу роботи над моделлю. Підключення бібліотек - імпортуються необхідні бібліотеки TensorFlow, Keras, NumPy, PIL для обробки зображень та побудови моделі. Робота з зображеннями - завантаження набору даних із зображеннями картоплі, зміна розміру, нормалізація значень пікселів та аугментація. Визначення архітектури - побудова архітектури нейромережі з необхідними шарами згортки, активації, пулінгу. Навчання нейромережі - тренування моделі на навчальних даних із заданою кількістю епох. Перевірка умови (Epochs > 10) - цикл навчання продовжується, поки не буде досягнуто 10 або більше епох. Збереження моделі - після досягнення необхідної кількості епох модель зберігається у файл model.h5. Кінець - завершення процесу навчання моделі.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Використання згорткових нейронних мереж для розпізнавання об'єктів на зображеннях

Нейронні мережі, технологія глибокого навчання, за останні роки досягли значного прогресу в таких сферах, як комп'ютерний зір, обробка природної мови, обробка та розпізнавання зображень. Нейронні мережі використовуються для вирішення складних завдань, які вимагають аналітичних розрахунків, подібних до тих, які виконує людський мозок.

Поширеними завданнями, на вирішення яких залучаються нейронні мережі, є:

- класифікація – поділ даних за значимими ознаками;
- прогнозування – прогноз наступного кроку;
- розпізнавання – аналіз зображення з подальшою класифікацією [4, 5].

Python є найбільш поширеною мовою програмування нейронних мереж; бібліотека Numpy може бути використана для розробки простих нейронних мереж для вирішення завдань прогнозування; бібліотека Keras може бути використана для програмування мереж прямого поширення і вирішення завдань розпізнавання образів, а також для вирішення завдань розпізнавання зображень. Нейронні мережі, що працюють з зображеннями, потребують підключення окремого модуля TensorFlow [6].

TensorFlow - це бібліотека машинного навчання від Google. Використовується для побудови та навчання нейронних мереж для вирішення задач виявлення закономірностей та класифікації. Бібліотека базується на програмуванні потоків даних для оптимізації математичних обчислень; обчислення в TensorFlow виконуються за допомогою графа потоків даних, де вузли представляють процеси, а ребра - потоки даних між вузлами.

Також потрібне середовище розробки. Для нейромережевих експериментів зазвичай використовують Jupyter Notebook - інструмент для інтерактивної розробки та презентації програмних проектів, своєрідний блокнот. Якщо ви не можете

встановити цей інструмент, Google пропонує схожий інструмент під назвою Colaboratory.

Google Colaboratory - це сучасний хмарний сервіс, спрямований на спрощення досліджень машинного навчання та глибокого навчання; всі обчислення в Colaboratory виконуються на віддалених серверах; встановлюється Colaboratory, TensorFlow та необхідні бібліотеки Python. Пакети, яких не вистачає, можна встановити за допомогою команди pip.

При роботі з зображеннями нейромережа повинна визначити, що зображено на зображенні, і привласнити мітку класу (класифікувати розпізнаний об'єкт).

Згорткові нейронні мережі використовуються для роботи з зображеннями. На відміну від перцептронів, які розглядають все зображення відразу, згорткові нейронні мережі сканують зображення частинами. Згорткові нейронні мережі працюють на основі фільтрів, які розпізнають певні особливості зображення [7]. Фільтр - це набір ядер, матриця чисел, які називаються вагами, які навчаються і в результаті створюють карту ознак ядра-зображення. Фільтр переміщується по зображенню, щоб визначити, чи присутній потрібний об'єкт (ядро) у сканованій області (рис. 2.1). Для отримання відповіді виконується операція згортки, яка є сумою добутків елементів фільтра і матриці вхідного сигналу пікселів зображення.

Нижче наведено частини програмного коду нейронної мережі, реалізованої в Google Collab. Перша частина - це посилання на бібліотеки.

```
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
import matplotlib.pyplot as plt
from google.colab import files
```

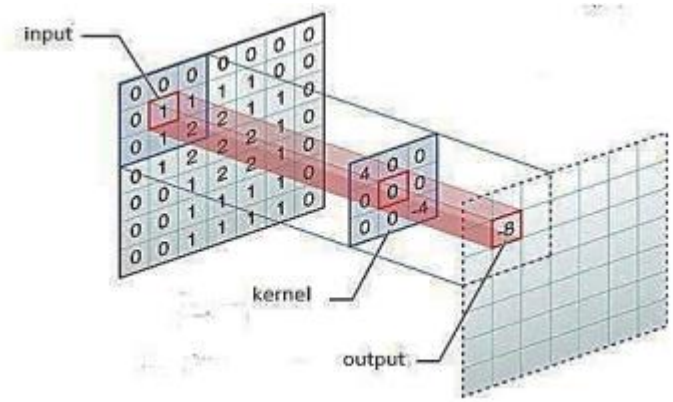

Рисунок 2.1 – Операція згортки

Оскільки нейронна мережа працює із зображеннями, необхідно завантажити набір даних для її навчання.

```
train, _ = tfds.load('cats_vs_dogs', split=['train[:100%]'], with_info=True,
as_supervised=True)
```

Зображення для навчання нейронної мережі повинні мати однакову роздільну здатність, тому вкажіть її у змінній SIZE (наприклад, 224 x 224). Далі нормалізуйте зображення. Переведіть в діапазон [0, 1], розділивши число кожного пікселя на 255.

Потім збирається архітектура нейронної мережі. Викликається метод для оголошення шарів по порядку: шар згортки, шар об'єднання та шар підсумовування. Шар об'єднання - це нелінійне стиснення карти ознак, яке стискає групу пікселів в один піксель за допомогою нелінійного перетворення. Шар повного злиття виконує нелінійне перетворення отриманих ознак і виконує класифікацію. Метод Dropout(0.2) використовується для вирішення проблеми перенавчання. Де 0.2 - це частка нейронів, які випадковим чином випадають з процесу навчання.

Нейронна мережа, зібрана із готових функцій:

```
[ ] model = tf.keras.Sequential([
                base_layers,
                GlobalAveragePooling2D(),
                Dropout(0.2),
                Dense(1)
])
model.compile(optimizer='adam',
loss=tf.keras.losses.BinaryCrossen-
tropy(from_logits=True), metrics=['accura-
cy'])
```

Наступним кроком є навчання нейронної мережі. Метод навчання визначає набір даних і кількість періодів навчання. Після навчання нейронна мережа видає

числовий результат, значення якого використовується для класифікації. Таким чином, життєвий цикл нейронної мережі [8] включає наступні етапи:

- підготовка даних (підключення бібліотек, датасетів, нормалізація даних);
- створення нейронної мережі (вибір архітектури, розміру вхідного вектора, параметрів навчання);
- навчання нейронної мережі на датасетах;
- експлуатація (вирішення завдань, для яких створювалася нейронна мережа).

Нейронні мережі, що навчаються шляхом зворотного поширення помилок, зберігають вагові коефіцієнти, що відповідають парам вхід-вихід. Завдання прогнозування, яке вирішує нейронна мережа, полягає в отриманні очікуваного значення на вихідному шарі, коли надаються дані, що відповідають вхідному шару. Навчена нейронна мережа повинна визначити статистичну відповідність між вхідними та вихідними даними. Після підключення бібліотеки Numpy для обробки послідовностей оголошується функція активації, наприклад, сигмоїдна функція.

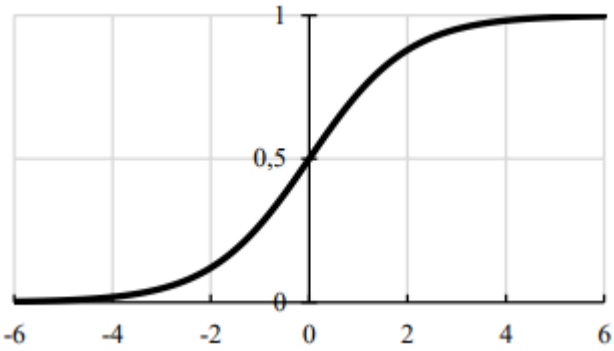

Рисунок 2.2 – Сигмоїдальна функція активації нейронів

Крім сигмоїдної функції, нейромережа використовує такі функції активації: лінійну, нелінійну, ступінчасту, гіперболічний тангенс та інші. Для задачі прогнозування обрано сигмоїдну функцію.

Послідовність вхідних даних X ініціалізується у вигляді матриці. Кожен рядок - це навчальна вибірка, а стовпці - нейрони вхідного шару. Далі йде матриця вихідних даних Y.

Наступним кроком є випадкова ініціалізація матриці ваг нейронної мережі. Після того, як мережа навчена, ваги фіксуються. Навчання нейронної мережі складається з декількох ітерацій, що включають операції:

- визначення першого шару із вхідними даними;
- крок передбачення;
- розрахунок помилок.

2.2. Класифікація зображень з допомогою нейронної мережі

Архітектура двошарової мережі представлена на рис. 2.3.

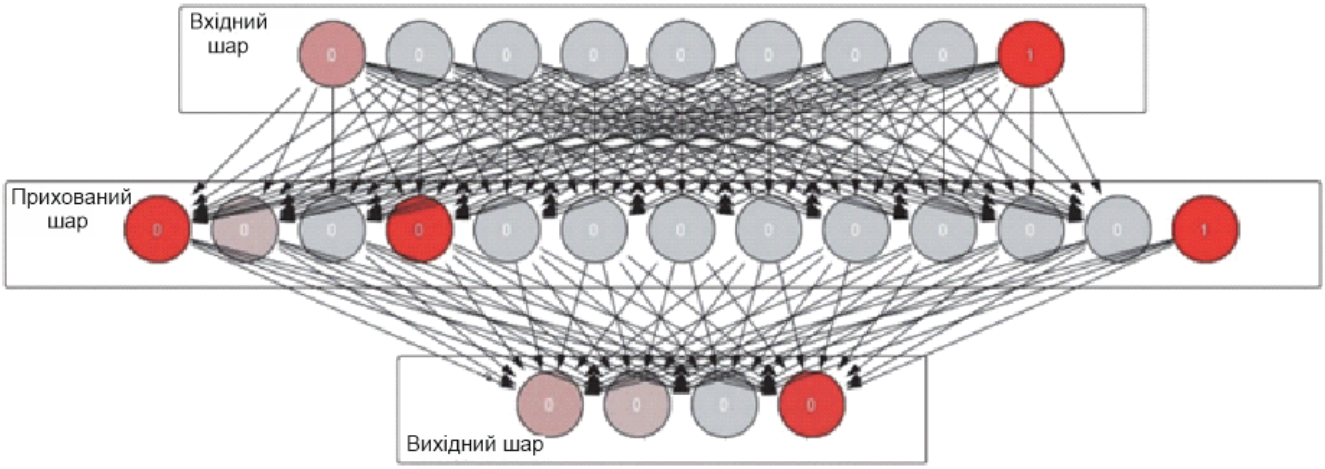

Рисунок 2.3 – Архітектура багатошарового персептрона

Вхідний шар - це не шар нейронів, а вхідний вектор реальних характеристик пісні. Нейрони у прихованому та вихідному шарах використовують сигмоїдну функцію активації.

Багатошаровий персептрон є однією з найпоширеніших і широко використовуваних нейромережевих моделей [10]. Однією з основних переваг багатошарових персептронів є їх здатність вирішувати алгоритмічно складні задачі та задачі, алгоритмічне рішення яких невідоме, але з яких можна скласти набори прикладів з відомими рішеннями. Під час навчання багатошаровий персептрон, завдяки своїй внутрішній структурі, виявляє закономірності у зв'язках між вхідними та вихідними зображеннями і таким чином узагальнює досвід, отриманий на навчальній вибірці. Ця здатність до узагальнення є основою привабливості багатошарового персептрона. У цій статті мережа навчається з використанням алгоритму найшвидшого спуску та зворотного поширення помилки.

Вхідний шар багатошарового персептрона являє собою вхідний вектор у вигляді $x=[x_0, x_1,..., x_N]^T$. З вектором x пов'язані два вихідні вектори мережі: вектор

фактичних вихідних сигналів y =[y_0, y_1,..., y_M]^T і вектор очікуваних вихідних сигналів d=[d_0, d_1, …, d_M]^T.

Позначимо вектор ваг прихованого шару як $w_{ij}^{(1)}$, а вектор ваг вихідного шару як $w_{si}^{(2)}$, де j =0,1,..., N; i = 0,1,....,K; s = 1,2,....,M , N - розмір вхідного вектора, K - число нейронів у прихованому шарі, M - число нейронів у вихідному шарі. Якщо позначити функцію активації нейронів як *f*, то s-ий нейрон вихідного шару виробляє вихідний сигнал, який визначається такою формулою:

$$y_s = f\left(\sum_{i=0}^{K} w_{si}^{(2)} v_i\right) = f\left(\sum_{i=0}^{K} w_{si}^{(2)} f\left(\sum_{j=0}^{N} w_{ij}^{(1)} x_j\right)\right). \quad (2.1)$$

Метою навчання є підбір таких значень ваг, щоб при заданому вхідному векторі x значення вихідного сигналу ys збігалося з очікуваним значенням ds з необхідною точністю [11]. Під час навчання ставиться завдання мінімізувати цільову функцію, яка для P навчальних вибірок визначається наступним чином.

$$E(w) = \frac{1}{2} \sum_{t=1}^{P} \sum_{s=1}^{M} \left(y_s^{(t)} - d_s^{(t)}\right)^2. \quad (2.2)$$

Ваги можна уточнювати кожного разу, коли подається навчальна вибірка. У цій статті багатошаровий персептрон навчається за допомогою алгоритму найшвидшого спуску та методу зворотного поширення. В алгоритмі найшвидшого спуску формула покращення ваг має наступний вигляд:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \cdot \frac{\partial E(t)}{\partial w_{ij}(t)}, \quad (2.3)$$

де h – коефіцієнт навчання, 0<h<1.

Для навчання багатошарової мережі за допомогою градієнтного методу необхідно визначити вектор градієнта відносно ваг усіх шарів мережі. Ця задача має явний розв'язок лише для ваг першого шару. Для інших шарів використовується метод зворотного поширення. Навчання мережі методом зворотного поширення здійснюється в декілька етапів. На першому етапі, коли подається вхідний вектор x, обчислюється вихідний сигнал vi нейронів прихованого шару, а потім отримується значення ys нейронів вихідного шару. Отримавши значення вихідного сигналу ys,

можна обчислити фактичне значення цільової функції помилки E(w). На другому кроці значення цієї функції мінімізується [12].

Розглянемо основне рівняння мережі з одним прихованим шаром, показане на рисунку 2.4. Використовуючи загальні позначення, цільова функція нейронів вихідного шару визначається наступним чином:

$$E = \frac{1}{2}\sum_{s=1}^{M}\left[f\left(\sum_{i=o}^{K} w_{si}^{(2)} v_i\right) - d_s \right]^2 = \\ = \frac{1}{2}\sum_{s=1}^{M}\left[f\left(\sum_{i=0}^{K} w_{si}^{(2)} f\left(\sum_{j=0}^{N} w_{ij}^{(1)} x_i\right)\right) - d_s \right]^2, \quad (2.4)$$

У формулі (2.5) і під змінною u розумітимемо вихідні сигнали суматорів нейронів прихованого або вихідного шару, представлених формулами (2.6) і (2.7).

$$\frac{\partial E}{\partial w_{si}^{(2)}} = (y_s - d_s)\cdot\frac{df(u_s^{(2)})}{du_s^{(2)}}\cdot v_i. \quad (2.5)$$

$$u_i^{(1)} = \sum_{j=0}^{N} w_j^{(1)} x_j, \quad (2.6)$$

$$u_s^{(2)} = \sum_{i=0}^{K} w_{si}^{(2)} v_i. \quad (2.7)$$

Компоненти градієнта щодо нейронів прихованого шару описуються складнішою залежністю:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \sum_{s=1}^{M}(y_s - d_s)\cdot\frac{dy_s}{dv_i}\cdot\frac{dv_i}{dw_{ij}^{(1)}}. \quad (2.8)$$

В іншому вигляді ця залежність може бути виражена формулою:

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \sum_{s=1}^{M}(y_s - d_s)\cdot\frac{df(u_s^{(2)})}{du_s^{(2)}}\cdot w_{si}^{(2)}\frac{df(u_i^{(1)})}{du_i^{(1)}}x_j. \quad (2.9)$$

2.3. Математична модель захворювань картоплі

Згорткові нейронні мережі (CNN) є одним з основних інструментів, що використовуються в обробці зображень, наприклад, для класифікації картопляного листя та виявлення хвороб. Ці мережі базуються на двох основних математичних операціях: згортки та об'єднання [10].

Згортка. Операції згортки є одним з основних елементів ШНМ, що дозволяє виявляти локальні закономірності на зображеннях. До кожної частини вхідного зображення застосовується фільтр. Математично це виглядає як застосування фільтра до вхідного зображення:

$$(I * K)(x, y) = \sum_{m=-a}^{a} \sum_{n=-b}^{b} I(x + m, y + n)K(m, n) \tag{2.10}$$

де I(x,y) - піксель вхідного зображення на позиції (x,y), K(m, n) - значення фільтра на позиції (m, n), a, b - розміри фільтра.

Об'єднання Операції об'єднання використовуються для зменшення розміру зображення після застосування згортки. Це гарантує збереження важливих особливостей і зменшує обчислювальні витрати. Найпоширенішим типом об'єднання є максимальне об'єднання, яке вибирає максимальне значення з кожної області [11]:

$$P(x, y) = \max\{I(i, j)\}, \text{ де } i, j \in \text{область пікселів}. \tag{2.11}$$

Алгоритм класифікації за допомогою CNN полягає в кількох основних етапах.

Підготовка даних. Для обробки вхідного зображення виконується попередня обробка, така як масштабування (нормалізація), зсув значень пікселів до діапазону [0,1][0,1] та зміна розміру до однакового розміру (256x256 пікселів).

Проходження через шари згортки. Кожен шар згортки застосовує до зображення кілька фільтрів для виявлення локальних особливостей. Потім застосовується об'єднання, щоб зменшити розмір і зберегти важливі особливості.

Повністю з'єднані шари. Після проходження через шари згортки та об'єднання вектори ознак переносяться на кілька повністю зв'язаних шарів. Ці шари

використовуються для класифікації зображення в одну з декількох категорій (здорові листки або різні хворі листки) [12].

Класифікація та оптимізація. Вихідні дані моделі використовують функції активації для вибору класів. Наприклад, Softmax для багатокласової класифікації:

$$P(y_i|x) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.12}$$

z_i - вихід моделі для класу i, $P(y_i|\ x)$ - ймовірність належності до класу i, x - вхідне зображення.

Оптимізація. Для навчання моделей використовуються алгоритми оптимізації, такі як Adam. Алгоритм Адама поєднує в собі переваги адаптивного навчання та імпульсу і дає більш стабільні результати, ніж інші методи.

Функція втрат Функція втрат. Для задач класифікації використовуються розріджені категоріальні функції втрат крос-спентропії, які вимірюють різницю між істинною міткою та передбачуваною ймовірністю:

$$L = -\sum_i y_i \log(p_i) \tag{2.13}$$

де y_i - реальна мітка класу, p_i - передбачена ймовірність для класу i.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розроблення інформаційної системи діагностики хвороб картоплі

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Підготовка середовища для роботи з нейронними мережами на основі TensorFlow та Keras TensorFlow - одна з найпопулярніших бібліотек для побудови, навчання та розгортання моделей глибокого навчання. Використовується для побудови моделей згорткових нейронних мереж (CNN) для діагностики хвороб картоплі. Модель - для створення послідовної нейромережевої моделі, Шари - для додавання шарів нейронної мережі.

Matplotlib використовується для побудови графіків і візуалізації результатів навчання моделі, показуючи вибіркові зображення втрат, точності або різних захворювань картоплі.

```
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
```

Визначає основні параметри обробки зображень, що використовуються в моделі діагностики хвороб картоплі: IMAGE_SIZE - розмір зображення, яке буде вхідним для нейронної мережі. Зображення масштабується до 256x256 пікселів. Цей розмір є звичайним для задач комп'ютерного зору. Він досить великий для розпізнавання хвороб.BATCH_SIZE - розмір партії, тобто кількість зображень, оброблених за одну ітерацію під час навчання моделі.32 - стандартне значення, яке балансує між ефективністю навчання та продуктивністю.CHANNELS - кількість каналів на зображенні. Значення 3 означає, що зображення є RGB. Це необхідно, оскільки зміна кольору листя картоплі може бути ознакою важливого захворювання. Ці параметри використовуються для підготовки набору даних і побудови архітектури моделі згорткової нейронної мережі.

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
```

```
    "Dataset", shuffle=True, image_size  = (IMAGE_SIZE, IMAGE_SIZE),
batch_size = BATCH_SIZE)
```

tf.keras.preprocessing.image_dataset_from_directory() - Функція для створення набору даних зображень з вказаного каталогу. Дані зчитуються з теки Dataset: Dataset - шлях до папки, що містить набір даних зображення. Структура папки наступна:

Dataset/

Хвороба1/

img1.jpg

img2.jpg

Хвороба2/

img1.jpg

img2.jpg

tf.keras.preprocessing.image_dataset_from_directory() - Функція для створення набору даних зображень з вказаного каталогу. Дані завантажуються з теки Dataset: Dataset - шлях до папки, що містить набір даних зображення. Структура папки наступна:

```
Found 2152 files belonging to 3 classes.
```

TensorFlow знайшов 2152 зображення, що належать до трьох класів. Це означає, що в папці Dataset є три підпапки, кожна з яких відповідає певній хворобі або стану картоплі: 2152 файли - загальна кількість зображень, 3 класи - модель класифікує зображення за трьома категоріями. Структура папок:

Dataset/

Healthy/ - 1-й клас: здорове листя;

Early_Blight/ - 2-й клас: ранній фітофтороз;

Late_Blight/ - 3-й клас: пізній фітофтороз.

```
classname = dataset.class_names
classname
```

dataset.class_names - атрибут набору даних, що містить список імен класів, визначених відповідно до структури папок каталогу набору даних. classname =

dataset.class_names - зберігає цей список у змінній classname. classname виводить список класів у консоль.

```
['Potato_Early_blight', 'Potato_Late_blight', 'Potato_healthy']
```

Модель виявила три класи: Potato_Early_blight - ранній фітофтороз, Potato_Late_blight - пізній фітофтороз, Potato_healthy - здорове листя картоплі.

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

Перевірте структуру пакету з набору даних, щоб переконатися, що дані зчитуються правильно. Пройдіться по набору даних один раз, щоб отримати перший фрагмент зображення і відповідну йому мітку. Витягніть фрагмент з набору даних і виведіть розміри фрагмента зображення. Результат:

```
(32, 256, 256, 3)
```

32 - розмір пакету (BATCH_SIZE), 256x256 - розмір зображення (IMAGE_SIZE), 3 - кількість каналів RGB. Масив міток зображень у цифровому форматі. Результат:

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

0 → Potato___Early_blight

1 → Potato___Late_blight

2 → Potato___healthy

Мітки мають бути у діапазоні 0, 1, 2 відповідно до знайдених класів. Візуалізуємо деякі зображення з набору даних.

```
plt.figure(figsize=(10,10))
for image_batch, label_batch in 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(classname[labels_batch[i]])
        plt.axis("off")
```

12 зображень виводяться у три рядки по чотири зображення у кожному. На виході виводиться 12 зображень з набору даних у форматі 3x4, кожне з яких супроводжується підписом, що вказує на клас зображення.

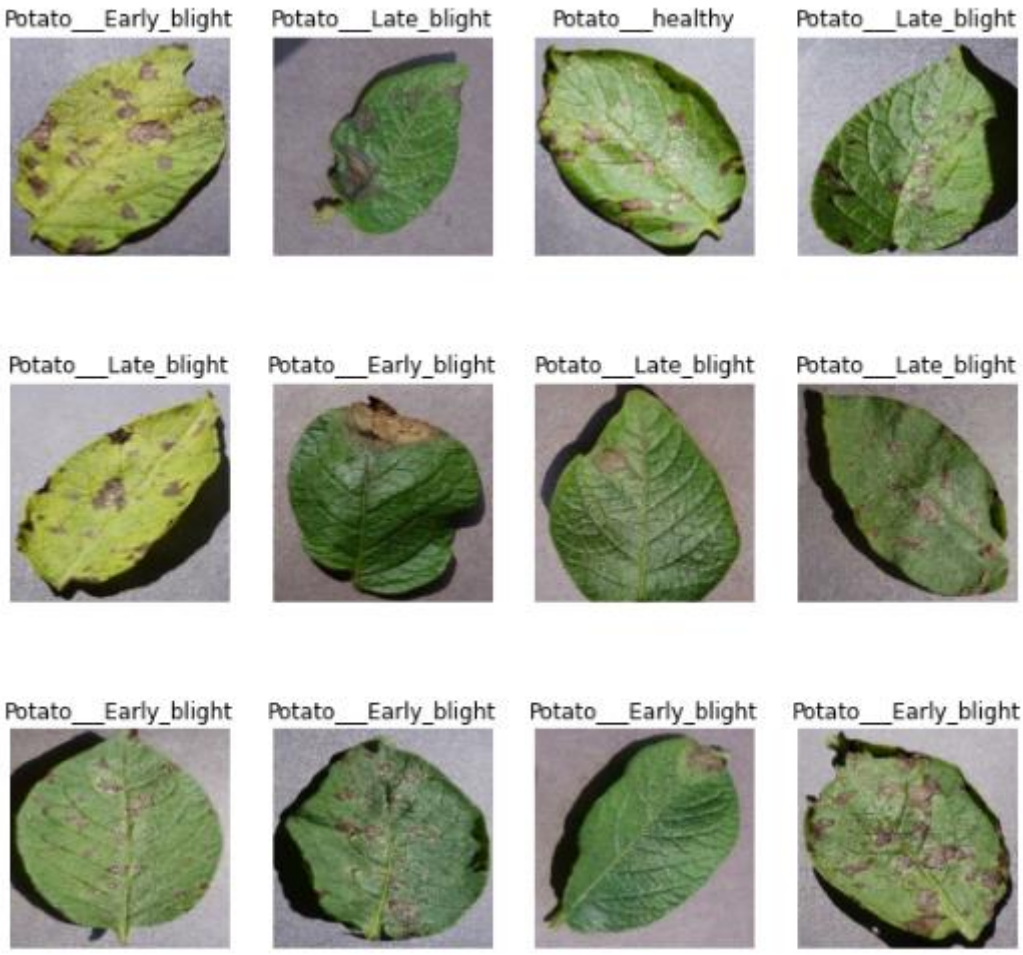

Рисунок 3.1 – Візуалізація зображень з набору даних

Навчання, валідація та валідація даних

```
len(dataset)
```

Функція len(dataset) повертає кількість пакетів у наборі даних. Це кількість ітерацій, які можуть пройти через набір даних, якщо розмір кожного пакету дорівнює розміру, заданому параметром batch_size.

```
68
```

Датасет містить 68 пакетів.

```
train_size = 0.8
len(dataset)*train_size
```

Обчислює кількість зображень для навчання моделі на основі заданого розміру навчального набору. len(dataset) - кількість агрегованих пакетів у наборі даних. train_size = 0.8 означає, що 80% всіх зображень буде використано для навчання, а 20% буде залишено для тестування. len(dataset) * train_size обчислює кількість фрагментів, що відповідають 80% від загальної кількості.

```
54.400000000000006
```

Кількість пакетів у навчальній вибірці обчислюється як 54.4, але фактичний розмір пакетів має бути цілим числом. Округлимо значення, щоб отримати ціле число, яке відповідає кількості пакетів.

```
train_size = int(54.4)
```

Отримали 54 як кількість пакетів для тренувальної вибірки.

```
train_ds = dataset.take(54)
len(train_ds)
6.800000000000001
```

Пройдіться по набору даних і підрахуйте кількість пакетів у train_ds. sum(1 for _ in train_ds) - пройдіться по train_ds і присвойте кожному елементу єдине значення. Результат - кількість пакетів у train_ds.

```
val_size = 0.1
len(dataset) * val_size
```

Якщо розмір верифікаційної вибірки становить 10% від загальної кількості пакетів у наборі даних, обчисліть кількість пакетів, які потрібно виділити для верифікаційної вибірки. Результатом буде десяткове число 6.7. Це значення можна округлити до найближчого цілого числа. Якщо розмір набору даних становить 67 пакетів, перевірочна вибірка складатиме 6 пакетів.

```
train_split=0.8
val_split=0.1
test_split=0.1
```

Значення train_split, val_split і test_split визначають, як дані розподіляються між навчальною, валідаційною та тестовою вибірками:

- train_split = 0.8 – 80 % даних йде на тренування;
- val_split = 0.1 – 10 % даних йде на валідацію;
- test_split = 0.1 – 10 % даних йде на тестування.

```
def get_dataset_paratitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert train_split + val_split + test_split == 1
    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 розділяє набір даних ds на три частини: навчальну, перевірочну та тестову.

```
train_ds, val_ds, test_ds = get_dataset_paratitions_tf(dataset)
```

Отримали три датасети:

- train_ds - для тренування моделі;
- val_ds - для оцінки моделі під час тренування;
- test_ds - використовується для фінальної оцінки моделі після тренування.

Зміна розміру та нормалізація даних

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

Створює двоетапну послідовність обробки зображення. Цей шар змінює розмір вхідного зображення як IMAGE_SIZE x IMAGE_SIZE. Це необхідно для того, щоб всі зображення були однакового розміру, оскільки нейронні мережі працюють з фіксованими розмірами вхідних даних.IMAGE_SIZE - параметр, який вказує новий розмір зображення.Якщо IMAGE_SIZE = 256, всі зображення будуть мати розмір 256x256 пікселів. Змінено розмір до 256x256 пікселів. Це необхідно, якщо вхідні зображення мають різні розміри, а їх потрібно привести до одного стандартного формату.

Наступний шар масштабує значення пікселів з діапазону [0, 255] до діапазону [0, 1]. Для цього всі пікселі діляться на 255; пікселі в [0] залишаються 0. Масштабування до [0, 1] допомагає нейромережі ефективніше обробляти дані, оскільки зменшує ймовірність перевантаження моделі великими значеннями пікселів.

Оскільки всі зображення масштабуються до однакового розміру, нейронна мережа може обробляти зображення без додаткових труднощів. Масштабування зображень до інтервалу [0, 1] дозволяє уникнути перевантаження під час навчання і забезпечує більш швидке та стабільне навчання.

При застосуванні цього шару до набору даних кожне зображення спочатку змінюється до потрібного розміру, а потім масштабується до значень пікселів у діапазоні [0, 1]. Таким чином зображення готуються до подальшого навчання.

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

layers.experimental.preprocessing.RandomFlip(«horizontal_and_vertical») - цей шар випадковим чином перевертає зображення по горизонталі та вертикалі. horizontal_and_vertical є вертикальним, тобто зображення можна перевертати як по горизонталі, так і по вертикалі, що дозволяє моделі навчатися на різних варіантах зображення. Це корисно для збільшення різноманітності набору даних і гарантує, що модель не прив'язана до певної орієнтації об'єкта.

layers.experimental.preprocessing.RandomRotation(0.2) - випадковим чином повертає зображення в діапазоні [-0.2, 0.2]. Це корисно для задач, де об'єкти на зображенні можна повертати.

Оскільки доповнення змінює зображення, модель навчається бути стійкою до змін форми об'єкта. Збільшення може значно збільшити розмір навчального набору даних без необхідності додаткового збору даних. Випадкові перетворення, такі як обертання та перевертання, покращують продуктивність моделі в реальних умовах, де зображення можуть бути різноспрямованими або поверненими.

Коли цей шар застосовується до набору даних, кожне зображення випадковим чином модифікується перед тим, як потрапити в модель. Це дозволяє моделі бачити різні версії кожного зображення під час навчання, що робить її більш стійкою до змін.

```
train_ds = train_ds.map(
lambda x, y : (data_augmenation(x, training=True), y)).prefetch(buffer_size =
tf.data.AUTOTUNE)
```

train_ds.map() - функція, яка застосовує операцію до кожного елементу набору даних. lambda x, y: (data_augmentation(x, training=True), y) - функція доповнення data_augmentation(x, training=True) застосовується до кожного зображення x та його мітки y; параметр training=True вказує на те, що доповнення застосовується лише

під час навчання; y - мітка (клас) зображення, яка не змінюється під час доповнення. Кожне зображення в навчальному наборі даних доповнюється за допомогою data_augmentation і змінюється випадковим чином.

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3
```

Визначають параметри для вхідних даних та кількість класів у задачі класифікації захворювання рослин.

BATCH_SIZE - кількість зображень у пакеті. Під час навчання модель обробляє кілька зображень одночасно, а не по одному, щоб прискорити обробку і стабілізувати процес навчання.IMAGE_SIZE - розмір (висота і ширина) кожного зображення. Розмір кожного зображення становить 256x256 пікселів.CHANNELS - кількість колірних каналів у зображенні. Для стандартного RGB-зображення це значення дорівнює 3 (червоний, зелений і синій канали). Розмір кожного зображення з 3-ма каналами (RGB) дорівнює РОЗМІР_Зображення x РОЗМІР_Зображення пікселів; якщо РОЗМІР_Зображення = 256 і КАНАЛИ = 3, кожне зображення має форму 256 x 256 x 3. Пакетне зображення з BATCH_SIZE = 32 має форму (32, 256, 256, 256, 256, 3).

n_classes визначає кількість класів, на які модель має класифікувати зображення. Це три класи, тому що існує три типи хвороб картоплі: ранній фітофтороз, фітофтороз та фітофтороз здорової картоплі. Після побудови моделі параметр input_shape задає форму вхідних даних (формат зображення), а n_classes використовується для визначення кількості вихідних нейронів у кінцевому шарі моделі, що відповідають за класифікацію.

```
model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation = 'relu', input_shape =
input_shape),
    layers.MaxPooling2D(2,2),
```

Почніть будувати згорткову нейромережеву модель для задачі класифікації хвороб картоплі. Цей шар було попередньо визначено для зміни розміру зображення до IMAGE_SIZE x IMAGE_SIZE і масштабування пікселів з діапазону [0, 255] до

діапазону [0, 1]. Цей шар додається на початку мережі для підготовки зображення перед надсиланням його до шару згортки.

Layer; Conv2D - шар згортки, який застосовує процес згортки до вхідного зображення; 32 - кількість фільтрів (ядер); kernel_size=(3, 3) - розмір фільтра (3x3 пікселі); activation='relu' - функція активації ReLU (Rectified Linear Unit - випрямлена лінійна одиниця) дозволяє вивчати більш складні патерни, передаючи тільки позитивні значення. Це одна з найпоширеніших функцій активації в згорткових мережах.

input_shape=input_shape - цей параметр визначає форму вхідних зображень у першому шарі. input_shape вказує, що на вхід мережі буде подаватися зображення розміром BATCH_SIZE x IMAGE_SIZE x IMAGE_SIZE x IMAGE_SIZE x CHANNELS. Вказується як пакет; MaxPooling2D - це шар агрегації (об'єднання), який використовується для зменшення розміру зображення після згортки. (2, 2) означає, що кожне об'єднання застосовується до суцільного блоку 2x2 пікселів. Наприклад, зображення 4x4 зменшується до 2x2 при застосуванні цього шару, і кожен піксель є максимальним з блоку 2x2 пікселів у вихідному зображенні. Прінгінг допомагає зменшити кількість параметрів зі збереженням важливих особливостей зображення, що важливо для покращення узагальнення моделі.

Зображення після попереднього шару resize_and_rescale надсилається до шару згортки Conv2D. Потім шар згортки Conv2D застосовує 32 фільтри 3x3 для створення 32 нових елементів зображення. Потім застосовується шар об'єднання MaxPooling2D Pooling, щоб зменшити просторовий розмір зображення і підкреслити найважливіші риси. Інші шари можна додати пізніше.

Створюють базову згорткову нейронну мережу для класифікації зображень. Згортання та об'єднання використовуються для вилучення ознак. Модель можна розширювати, додаючи шари для покращення продуктивності.

```
layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
layers.MaxPooling2D(2,2),
```

Додайте до моделі ще один шар згортки та шар агрегації (об'єднання). layers.Conv2D - шар згортки, який застосовує згортку до зображення для вилучення ознак. 64 - кількість фільтрів (ядер) у цьому шарі. У порівнянні з попереднім шаром

згортки з 32 фільтрами, тут використовується 64 фільтри для вилучення більш складних ознак. kernel_size=(3, 3) - розмір фільтрів, які будуть переміщатися по зображенню для пошуку закономірностей. В даному випадку фільтр розміром 3x3 пікселі. activation='relu' - функція активації ReLU, яка гарантує, що модель пропускає лише додатні значення та ефективно працює під час навчання. Цей згортковий шар створює 64 нові особливості зображення, шукаючи різні патерни на кожному зображенні.

MaxPooling2D - шар для підсумовування (об'єднання) зображення. (2, 2) - розмір блоку об'єднання. В даному випадку це стосується кожних 2x2 пікселів. Це означає, що кожен блок 2x2 пікселів зменшується до одного пікселя, що містить максимальне значення цього блоку. Об'єднання допомагає зменшити кількість параметрів при збереженні важливих особливостей зображення і зменшує ймовірність надмірної підгонки моделі.

Після того, як зображення проходить через перший шар згортки (32 фільтри), застосовується об'єднання. Зображення надсилається на наступний шар згортки з 64 фільтрами для виділення більш складних ознак. Це дозволяє моделі краще зрозуміти більш абстрактні патерни зображення. Потім застосовується ще один шар об'єднання, щоб зменшити просторовий розмір зображення. Після застосування цього фрагмента коду модель стає складнішою, оскільки додаються шари згортки та об'єднання, що дозволяє мережі виокремлювати більш абстрактні ознаки та покращує її класифікаційні можливості.

```
layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
layers.MaxPooling2D(2,2),
```

Додайте до моделі ще один шар згортки Conv2D і шар агрегації MaxPooling2D. layers.Conv2D - шар згортки, який застосовує процес згортки до зображення і шукає особливості та патерни. 64 - кількість фільтрів у цьому шарі. Іншими словами, для кожного зображення генерується 64 особливості. kernel_size=(3, 3) - розмір кожного фільтра. activation='relu' - функція активації ReLU, допускаються лише додатні значення. Цей шар аналізує вхідне зображення за допомогою 64 фільтрів і дозволяє мережі знаходити більш складні об'єкти, ніж попередній шар.

MaxPooling2D - шар об'єднання, який зменшує просторовий розмір зображення після застосування операцій згортки. (2, 2) - розмір блоку, який об'єднується. Цей шар вибирає максимальне значення з кожного блоку 2x2 пікселів. Це означає, що зображення зменшується вдвічі по кожній осі (ширині та висоті). Якщо вхідне зображення має розмір 32x32 пікселі, кожен блок 2x2 буде замінено на максимальне значення, тому після цього шару зображення матиме розмір 16x16.

Після проходження через перший шар згортки зображення передається на другий шар згортки з 64 фільтрами для вилучення більш складних патернів. Потім застосовується підсумовування MaxPooling2D(2, 2) для зменшення розміру зображення, зберігаючи лише найважливіші ознаки, що використовуються для класифікації. В результаті модель може виділити важливі ознаки із зображення і зменшити його розмір для подальшої класифікації.

```
layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
layers.MaxPooling2D(2,2),
layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
layers.MaxPooling2D(2,2),
layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
layers.MaxPooling2D(2,2),
```

До моделі додано ще два шари згортки і один шар підсумовування з об'єднанням. Тепер модель має три шари згортки і три шари об'єднання. Кожен новий шар згортки дозволяє моделі знаходити більш складні закономірності та особливості на різних рівнях. Після кожного шару згортки накладається шар об'єднання, щоб зменшити розмір зображення і залишити лише важливі елементи. Це зменшує кількість параметрів і дозволяє уникнути надмірної підгонки.

```
layers.Flatten(),
```

Шар нейронної мережі layers.Flatten() використовується для перетворення багатовимірного масиву в одновимірний. Це розширює матрицю до вектора, який необхідний для подальших етапів класифікації. Після проходження зображення через кілька шарів згортки та об'єднання, зображення перетворюється на багатовимірний тензор. Щоб подати ці дані в щільний шар (де кожен нейрон пов'язаний з усіма нейронами попереднього шару), 3D-тензор потрібно перетворити в одновимірний вектор. Flatten() робить саме це. Іншими словами, вона сплющує

матрицю до одного рядка. Іншими словами, вона перетворює всі елементи двовимірної або тривимірної матриці в одномірний масив.

Припустимо, що після трьох шарів згортки і трьох шарів об'єднання вхідне зображення розміром (256, 256, 64) стало матрицею розміром (8, 8, 64); Flatten() перетворює цей тензор розміром (8, 8, 64) у вектор розміром (4096) (8 * 8 * 64 = 4096).

Flatten готує дані для наступного шару Dense, який використовується для класифікації. Без цього шару було б неможливо передати багатовимірний тензор до шару Dense; шар Flatten() є важливим етапом переходу від обробки зображення до фінального етапу класифікації, де нейронна мережа повинна приймати рішення на основі виявлених ознак.

```
    layers.Dense(64, activation='relu'),
```

layers.Dense(64, activation='relu') - шар, де кожен нейрон з'єднаний з кожним нейроном попереднього шару. Використовується для виконання важливих операцій з ознаками, отриманими після шарів згортки та об'єднання, і використовується для класичної класифікації. 64 - кількість нейронів у цьому шарі. activation='relu' - функція активації ReLU, що застосовується до виходу кожного нейрона. ReLU відкидає всі від'ємні значення і пропускає лише додатні, що допомагає збільшити швидкість навчання.

Щільний шар приймає на вхід одновимірний масив ознак, отриманий шаром Flatten(), зважує кожну ознаку і застосовує лінійне перетворення. Після того, як кожен нейрон отримує лінійну комбінацію вхідних значень, він застосовує функцію активації ReLU, щоб визначити, чи активувати цей нейрон. Якщо результат лінійної комбінації від'ємний, нейрон не активується і результат дорівнює нулю. Якщо результат позитивний, нейрон активується і передає позитивне значення.

Цей шар є важливим для побудови остаточного рішення моделі і починає розуміти взаємозв'язки між ознаками, отриманими на попередньому шарі. Після того, як ознаки були перетворені у вектори, настає перший етап класифікації, на якому починається аналіз і класифікація цих ознак.

```
    layers.Dense(n_classes, activation='softmax'),
])
```

Цей шар виконує імовірнісну класифікацію для кожного класу. n_classes - кількість класів, до яких належить зображення (3 класи: здорова картопля, картопля, що швидко гниє, картопля, що повільно гниє).

activation='softmax' - функція активації softmax перетворює вихід цього шару на ймовірність, де сума дорівнює 1. activation='softmax' - функція активації softmax перетворює вихід цього шару на ймовірність, де сума дорівнює 1. Іншими словами, softmax дає ймовірність того, що зображення належить до кожного класу. Тому цей шар використовується для задач класифікації.

Функція softmax перетворює лінійні виходи нейронів у ймовірності. Виходи softmax завжди знаходяться в діапазоні від 0 до 1 і можуть бути інтерпретовані як ймовірності, оскільки їх сума дорівнює 1. Шар dense(n_classes) виробляє вихід для кожного класу. Оскільки softmax використовується як функція активації, кожен вихід є ймовірністю того, що зображення належить до одного з класів.

Якщо модель визначає картоплю, ймовірності можуть виглядати так:

- здорове: 0.80;
- раннє загнивання: 0.15;
- пізнє загнивання: 0.05.

Шар Dense(n_classes, activation='softmax') є останнім шаром моделі класифікації і надає ймовірності для кожного класу. На основі результатів цього шару модель може вирішити, до якого класу віднести зображення. Цей шар особливо важливий для багатокласової класифікації, де кожен клас є взаємовиключним, тобто кожне зображення належить лише до одного класу.

```
model.build(input_shape=input_shape)
```

input_shape=input_shape - розмір вхідних даних, які модель повинна обробити. model.build(input_shape) ініціалізує модель, створюючи всі шари та визначаючи параметри відповідно до форми вхідних даних.

```
model.summary()
```

Цей метод використовується TensorFlow для відображення детальної інформації про архітектуру моделі. Він дозволяє побачити, як виглядає модель, які

шари вона містить, скільки параметрів у кожному шарі, а також загальну кількість параметрів, які потрібно навчити TensorFlow відображає наступну інформацію:

- ім'я шару: назва кожного шару моделі (Conv2D, Dense, Flatten);
- тип шару: тип шару, який використовується (згортковий шар Conv2D або повнозв'язний шар Dense);
- вхідна форма: розмір вхідних даних для кожного шару;
- вихідна форма: розмір вихідних даних після проходження через кожен шар;
- кількість параметрів: кількість параметрів, які будуть навчатися в кожному шарі;
- загальна кількість параметрів: загальна кількість параметрів усіх шарів моделі.

Метод model.summary() можна використовувати для отримання загальної картини моделі, включаючи кількість параметрів, структуру шарів та їхні властивості. Це корисно для перевірки архітектури моделі перед навчанням. Вивести результати.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0
 )

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0
 2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928
```

```
max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0
 2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0
 2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0
 2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

Це перший шар, що містить геометрію вхідного зображення (32 - розмір стеку, 256x256 - розмір зображення, 3 - кількість кольорових каналів зображення). Параметр під номером 0 - цей шар не містить жодних параметрів.

conv2d (згортка): вихідна форма: (32, 254, 254, 254, 254, 32) - після застосування першого шару згортки розмір зображення зменшується до 254x254, а кількість каналів збільшується до 32. Параметр номер 896 - кількість параметрів у шарі згортки. Кожен фільтр має свої власні параметри, які визначаються під час навчання.

max_pooling2d (MaxPooling2D): вихідна форма: (32, 127, 127, 127, 127, 32) - після застосування максимального об'єднання розмір зображення зменшується вдвічі (з 254x254 до 127x127), але кількість каналів залишається незмінною (32). Параметр #: 0 - шар максимальної суми не має параметрів для навчання.

conv2d_1 (Conv2D): вихідна форма: (32, 125, 125, 64) - після застосування другого шару згортки кількість каналів збільшується до 64. Кількість параметрів: 18496 - кількість параметрів для цього шару згортки. max_pooling2d_1 (MaxPooling2D): вихідна форма: вихідна форма: (32, 62, 62, 64) - після цього шару розмір зображення зменшується вдвічі (з 125x125 до 62x62). Параметр #: 0.

conv2d_2 (Conv2D), conv2d_3 (Conv2D), conv2d_4 (Conv2D), conv2d_5 (Conv2D) - кількість каналів залишається 64, хоча зображення продовжує зменшуватися з кожним новим шаром згортки.

Кожен шар містить параметри, які залежать від розміру та кількості фільтрів. Вирівняти: Output Shape: (32, 256) - перетворює багатовимірне зображення в одномірний вектор для перенесення на шар Dense. Параметр #: 0 - цей шар не має параметрів для вивчення.

dense (Dense): Output Shape: (32, 64) - це повнозв'язний шар з 64 нейронами. Param #: 16448 — кількість параметрів цього шару, що потрібно навчати. dense_1 (Dense): Output Shape: (32, 3) - це фінальний шар з 3 вихідними нейронами для класифікації на 3 класи хвороб картоплі. Param #: 195 - кількість параметрів цього шару.

Загальна кількість параметрів: total params: 183,747 - загальна кількість параметрів, що підлягають навчанню в моделі; Trainable parameters: 183,747 - всі параметри моделі можуть бути навчені під час навчання.

Модель має кілька шарів згортки та максимального підсумовування, які можуть виявляти особливості на різних рівнях зображення. Останні два шари є щільними шарами, які поділяються на три класи. Загальна кількість параметрів, які потрібно навчити 183,747. Цей вивід допомагає зрозуміти, як виглядає модель і скільки параметрів потрібно навчати для класифікації зображень.

```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics = ['accuracy'])
```

Функція model.compile() готує модель до навчання. використовується оптимізатор adam. оптимізатор adam є одним з найпоширеніших та

найефективніших оптимізаторів для нейронних мереж. adam є ефективним для навчання моделей на реальних даних, оскільки він адаптує швидкість навчання для кожного параметра і тому є ефективним для навчання моделей на реальних даних.

Модель використовує розріджену категоріальну функцію втрат перехресної ентропії. Вона використовується для задач багатокласової класифікації, коли кожен об'єкт належить до одного з декількох класів. from_logits=False означає, що вихід моделі вже є ймовірністю. розріджена категоріальна крос-ентропія використовується, коли мітки класів виражаються цілим числом, а не в однократному форматі.

Під час навчання модель оцінює точність класифікації. Це дає уявлення про те, скільки правильних прогнозів модель робить для кожної категорії. Функція втрат обчислює різницю між передбаченнями моделі та фактичними мітками. Точність - це міра продуктивності моделі під час навчання. Тепер, коли модель готова до навчання, метод model.fit() можна використовувати для навчання моделі на даних.

```
history = model.fit(
    train_ds,
    epochs = 10,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds
)
```

Метод підгонки використовується для навчання моделі на навчальних даних. train_ds - це навчальний набір даних, на якому буде навчатися модель. Імпортується підготовлений набір train_ds, що містить зображення та відповідні їм мітки.

epochs = 10 - кількість епох у всьому навчальному наборі. У цьому випадку модель тренується 10 разів на всіх зображеннях у train_ds. batch_size = BATCH_SIZE - Розмір партії визначає, скільки зображень обробляється одночасно на кожному кроці навчання. validation_data = val_ds - Набір валідаційних даних, який використовується для оцінки моделі після кожної епохи навчання. Модель оцінюється на точність на цьому наборі протягом кожної епохи.

3.2. Результати роботи інформаційної системи

Модель навчається на даних train_ds для 10 епох з використанням 32 вимірних груп. Після кожної епохи модель оцінюється на валідаційному наборі даних val_ds. Результати навчання зберігаються в історії змінних, яка містить інформацію про втрати та точність в кожну епоху і може бути використана для подальшого аналізу.

```
Epoch 1/10
54/54 [==============================] - 123s 2s/step - loss: 0.9104 - accuracy: 0.4821 - val_loss: 0.7924 - val_accuracy: 0.5312
Epoch 2/10
54/54 [==============================] - 97s 2s/step - loss: 0.7168 - accuracy: 0.6476 - val_loss: 0.6183 - val_accuracy: 0.6979
Epoch 3/10
54/54 [==============================] - 96s 2s/step - loss: 0.5402 - accuracy: 0.7465 - val_loss: 0.4211 - val_accuracy: 0.8021
Epoch 4/10
54/54 [==============================] - 96s 2s/step - loss: 0.4120 - accuracy: 0.8264 - val_loss: 0.5617 - val_accuracy: 0.7396
Epoch 5/10
54/54 [==============================] - 94s 2s/step - loss: 0.4165 - accuracy: 0.8160 - val_loss: 0.4091 - val_accuracy: 0.8229
Epoch 6/10
54/54 [==============================] - 95s 2s/step - loss: 0.3353 - accuracy: 0.8605 - val_loss: 0.2591 - val_accuracy: 0.8854
Epoch 7/10
54/54 [==============================] - 96s 2s/step - loss: 0.2622 - accuracy: 0.9057 - val_loss: 0.1855 - val_accuracy: 0.9271
Epoch 8/10
54/54 [==============================] - 97s 2s/step - loss: 0.2194 - accuracy: 0.9184 - val_loss: 0.3256 - val_accuracy: 0.8646
Epoch 9/10
54/54 [==============================] - 95s 2s/step - loss: 0.1532 - accuracy: 0.9369 - val_loss: 0.2265 - val_accuracy: 0.9219
Epoch 10/10
54/54 [==============================] - 94s 2s/step - loss: 0.1675 - accuracy: 0.9369 - val_loss: 0.1217 - val_accuracy: 0.9479
```

Це результат тренування моделі на 10 епохах.

Ключові метрики. Втрати - це міра того, до якої міри модель не може передбачити значення. Чим менша втрата, тим краща модель. Точність - відсоток правильних прогнозів, зроблених моделлю на навчальних даних. Збільшення цієї

метрики вказує на те, що модель покращується. val_loss (втрати на перевірочному наборі) - втрати на перевірочному наборі даних, використовується для оцінки загальної якості моделі. val_accuracy (точність на перевірочному наборі) - точність моделі допомагає оцінити, наскільки добре модель працює з новими, невідомими даними.

У перші епохи (1-3) точність навчальної вибірки поступово зростає (з 48% до 74%), а втрати зменшуються (з 0,91 до 0,54). Точність валідації також зростає, що свідчить про хороше загальне навчання.

Середина тренування (4-7). Точність моделі на навчальній множині досягає 85% у шостій епосі, а точність валідації продовжує зростати (з 74% до 92%). Втрати на валідаційній множині зменшуються, що є добрим знаком.

Заключний етап (8-10). Точність навчальної вибірки стабільно висока (близько 94%). Точність валідації досягала 94-95% в епохах 9 і 10. Втрата верифікаційних даних залишається низькою (0.12).

Модель постійно покращувала свою точність на навчальних та валідаційних даних. Це свідчить про те, що модель добре навчається і може успішно класифікувати хвороби картоплі. Точність валідаційної вибірки близька до точності навчальної вибірки.

```
scores = model.evaluate(test_ds)
```

Метод оцінки моделі викликається для оцінки продуктивності моделі на тестовому наборі даних. Метод повертає список значень: перше - втрати, друге - точність на тестовому наборі.

```
8/8 [==============================] - 7s 510ms/step - loss: 0.2563 - accuracy: 0.9062
```

Результат виводу команди model.evaluate(test_ds) показує, що модель досягла таких результатів на тестовому наборі:

- Test Loss (втрати на тесті): 0.2563;
- Test Accuracy (точність на тесті): 90.62 %

Це означає, що точність моделі перевищує 90% на тестових даних і показує, що вона може класифікувати нові зображення, яких немає в навчальному наборі. Той факт, що показник втрат є досить низьким, також свідчить про хорошу

продуктивність моделі. Модель показує стабільну роботу як на навчальній, так і на тестовій вибірках.

```
scores
 [0.2563192844390869, 0.90625]
```

Втрати на тестовому наборі становлять 0,2563, а точність - 90,62%. Ці значення підтверджують, що модель добре працює, класифікуючи зображення тестового набору з високою точністю.

```
history.history.keys()
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Це вказує, які метрики зберігаються в об'єкті історії під час навчання моделі:

- loss - втрати на тренувальному наборі;
- accuracy - точність на тренувальному наборі;
- val_loss - втрати на валідаційному наборі;
- val_accuracy - точність на валідаційному наборі.

Ці метрики використовуються для моніторингу продуктивності моделі в кожну епоху під час навчання. Ці значення можна використовувати для побудови графіків та оцінки динаміки навчання моделі.

```
type(history.history['loss'])
list
```

Значення втрат зберігаються у списку. Кожен запис у цьому списку відповідає значенню втрат для відповідної епохи тренування.

```
len(history.history['loss'])
10
```

Для кожних десяти епох навчання у списку зберігаються відповідні значення втрат. Кількість елементів у списку дорівнює кількості епох, тобто для кожної епохи зберігається 10 значень.

```
acc = history.history['accuracy']
val_acc  = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
val_acc
 [0.53125,
 0.6979166865348816,
 0.8020833134651184,
 0.7395833134651184,
```

```
 0.8229166865348816,
 0.8854166865348816,
 0.9270833134651184,
 0.8645833134651184,
 0.921875,
 0.9479166865348816]
```

Кожна епоха показує значення точності у валідаційному наборі val_acc. Ці значення показують, як змінювалася точність моделі у валідаційному наборі під час навчання. На початку навчання точність валідаційного набору була низькою - близько 53%. З кожною епохою точність зростала, досягнувши понад 94% на 10-й епосі. Це свідчить про те, що модель добре навчилася і значно покращила свою здатність правильно класифікувати зображення у валідаційному наборі.

Візуалізація втрат та точності

```
plt.figure(figsize=(15,5))
plt.subplot(1, 2, 1)
plt.plot(range(10), acc, label = "Training Accuracy")
plt.plot(range(10), val_acc, label = "Validation Accuracy")
plt.legend(loc = 'lower right')
plt.title('Training & Validation Accuarcy')
plt.subplot(1, 2, 2)
plt.plot(range(10), loss, label = "Training Loss")
plt.plot(range(10), val_loss, label = "Validation Loss")
plt.legend(loc = 'upper right')
plt.title('Training & Validation Loss')
```

Було створено два графіки. Перший графік показує, як змінюється точність моделі протягом 10 епох у наборах даних Training Accuracy і Validation Accuracy; другий графік показує зміну втрат у наборах даних Training Loss і Validation Loss. Він показує, як змінюється точність моделі протягом 10 епох. Цей графік дає візуальне уявлення про те, як навчається модель і наскільки добре вона працює на валідаційному наборі порівняно з навчальним.

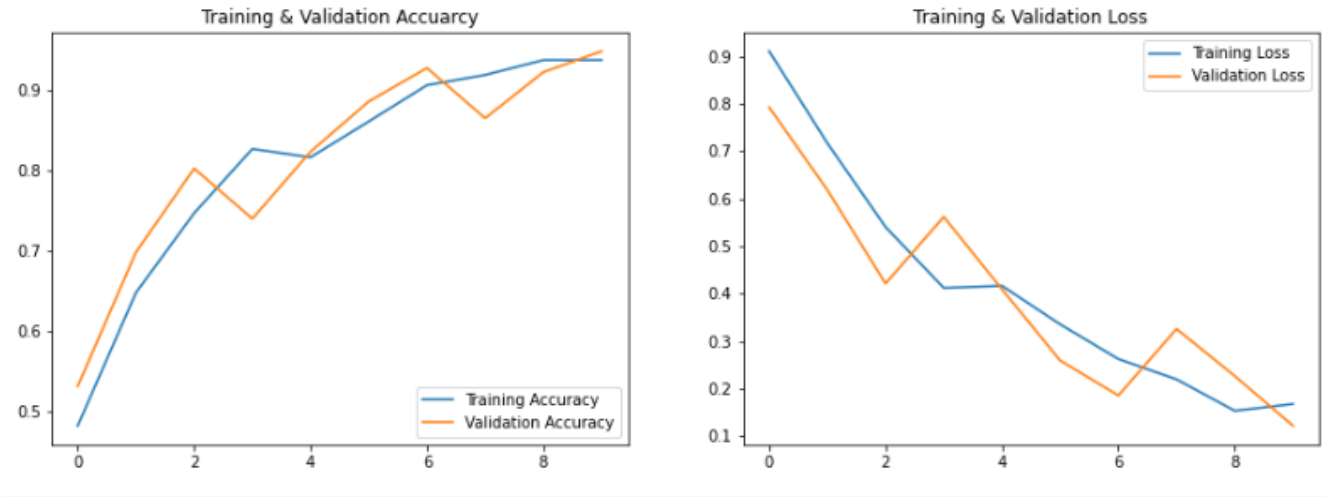

Рисунок 3.2 – Динаміка точності та функції втрат під час навчання моделі

Нижче наведено два графіки, що відображають процес навчання згорткової нейронної мережі (CNN) для класифікації зображень картопляного листя на три класи: Картопля__Ранній_опік, Картопля__Пізній_опік та Картопля__Здорова. ліворуч Графік точності навчання та валідації показує точність моделі в діапазоні від 0,5 до 1,0. Горизонтальна лінія показує кількість епох навчання від 0 до 9.

Синя лінія - точність навчання - показує точність навчальної вибірки. Вона зростає з кожною епохою і досягає приблизно 95%. Помаранчева лінія показує точність у прикладі Validation Accuracy. Спостерігається невелике зниження в епохах 3 і 7, але простежується схожа тенденція до зростання. Показник точності моделі постійно зростає, що свідчить про ефективність процесу навчання. Незначні коливання точності валідації можуть свідчити про ранній етап перенавчання.

На графіку втрат при навчанні та валідації праворуч показано функцію втрат зі значеннями втрат в діапазоні від 0,9 до 0,1. Горизонтальна лінія показує кількість епох навчання від 0 до 9. Синя лінія показує функцію втрат Training Loss, що безперервно зменшується від 0,9 до 0,15 для навчальної вибірки. Помаранчева лінія показує функцію втрат для вибірки Verification Loss. Спостерігається невелике зростання в Епохах 4 і 7, але в цілому спостерігається тенденція до зменшення. Модель успішно мінімізує втрати в обох прикладах, що підтверджує правильність навчання. Незначні коливання валідаційних втрат можуть свідчити про зміну складності даних або можливі аномалії у валідаційній вибірці.

Модель CNN була успішно навчена. На графіку видно закономірне зростання точності та зменшення функції втрат. Невеликі відмінності у валідаційній вибірці можуть свідчити про можливе перенавчання або нерівномірний розподіл даних у вибірці. Висока точність моделі показує, що вона є ефективною для діагностики хвороб картоплі.

```
import numpy as np
for image_batch , labels_batch in test_ds.take(1):
    first_img = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()
    print("first image to predict")
    plt.imshow(first_img)
    print('actual label:', classname[first_label])
    batch_prediction = model.predict(image_batch)
    print('predicted label:', classname[np.argmax(batch_prediction[0])])
```

Завантажте перше зображення з тестового набору, виведіть його на екран і зробіть прогноз за допомогою навченої моделі. Перетворіть перше зображення з тестового набору в масив для відображення та отримайте фактичні мітки для цього зображення. Перше зображення відображається на графіку і відображається фактична мітка для цього зображення. Модель робить прогнози для всіх зображень у пачці image_batch і витягує з результатів прогнозовану мітку першого зображення. Порівняйте фактичні та передбачені мітки, щоб перевірити точність моделі для одного зображення з тестового набору.

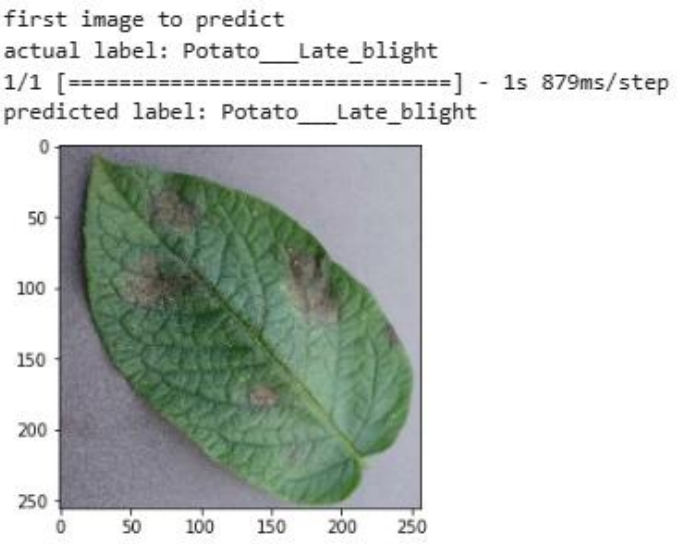

Рисунок 3.3 – Результат прогнозування класу хвороби картоплі для тестового зображення

На цьому рисунку показано результати роботи моделі згорткової нейронної мережі, яка класифікує листя картоплі на зображенні. На листі картоплі є характерні чорні плями, які є типовими ознаками фітофторозу. Власне етикетка: Potato_Late_blight - це істинна мітка класу для цього зображення, що вказує на наявність фітофторозу. Прогнозована мітка: Фітофтороз_картоплі - прогнозований клас, визначений моделлю. Цей результат збігається з фактичним класом, що свідчить про те, що модель працює правильно. Модель успішно розпізнала фітофтороз картоплі, що підтверджує її ефективність у класифікації хвороб листя картоплі.

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(img.numpy())
    img_array = tf.expand_dims(img_array, 0)
    prediction = model.predict(img_array)
    predicted_class = classname[np.argmax(prediction[0])]
    confidence = round(100 * (np.argmax(prediction[0])), 2)
    return predicted_class, confidence
```

Функція прогнозування використовує модель для прогнозування на одному зображенні і повертає передбачений клас і рівень достовірності у відсотках; вона перетворює зображення, тензор TensorFlow, в масив NumPy, придатний для обробки моделлю. Модель робить прогнози на вхідному зображенні. Результатом є ймовірність кожного класу.

Знаходиться індекс класу з найбільшою ймовірністю, і відповідний клас береться з цього індексу зі списку назв класів. Знаходиться ймовірність і множиться на 100, щоб отримати оцінку рівня довіри. Функція повертає передбачену мітку та рівень довіри.

```
for images,labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3 ,i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predicted_class, confidence = predict(model, images[i])
        actual_class = classname[labels[i]]
           plt.title(f"Actual :{actual_class}, \n Predicted: {predicted_class}. \n
Confidence: {confidence}%")
        plt.axis("off")
1/1 [==============================] - 0s 39ms/step
```

```
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 73ms/step
```

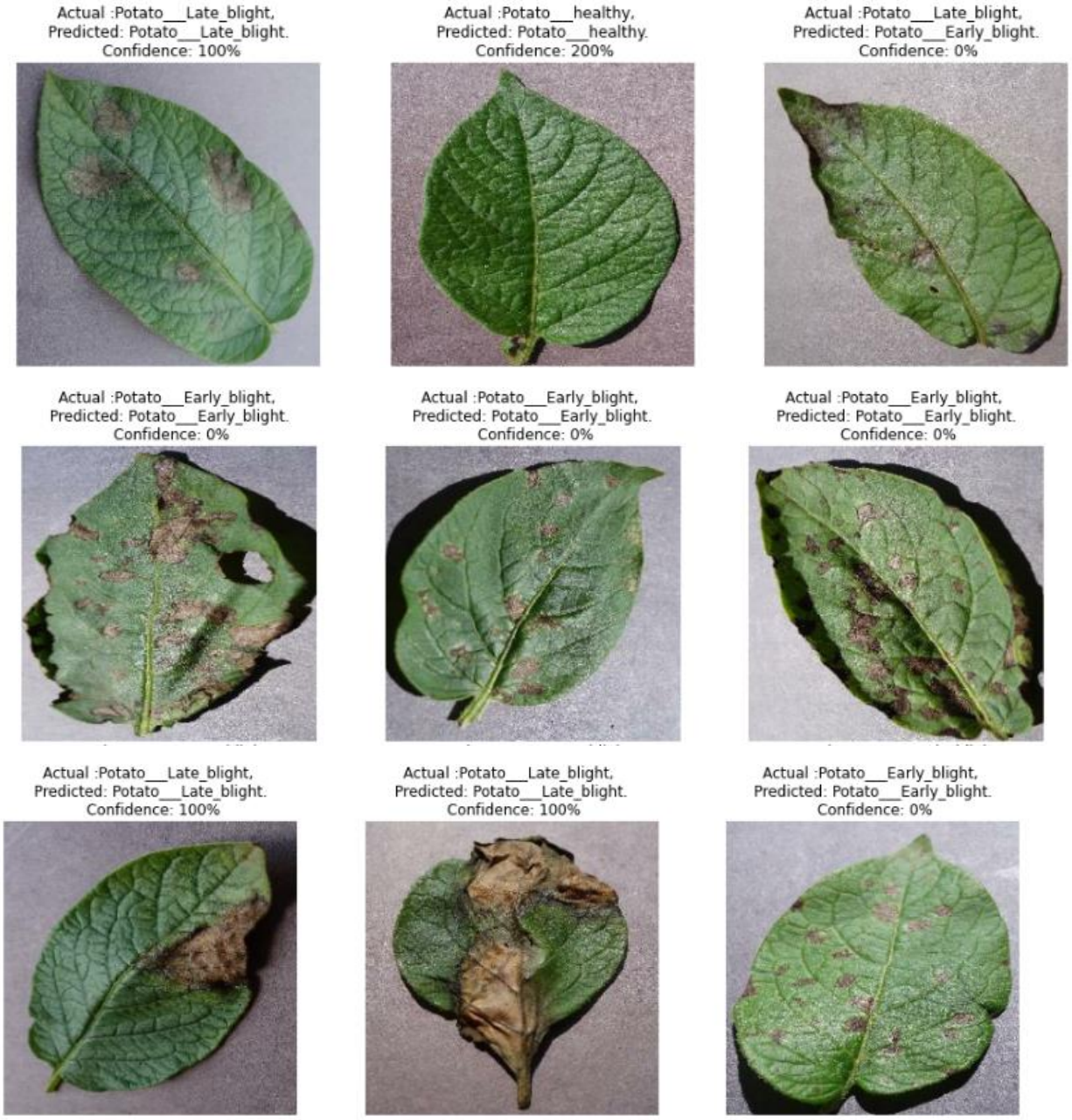

Рисунок 3.4 – Результати класифікації зображень листя картоплі з прогнозом моделі та рівнем впевненості

Для кожного зображення відображається істинний клас, передбачений клас і відсоток достовірності. Результати показують, що модель передбачила правильно для кожного зображення. На рисунку показано результати роботи моделі глибокого навчання для класифікації хвороб листя картоплі. Модель класифікує зображення на три класи:

- Potato_Late_blight (пізній фітофтороз);
- Potato_Early_blight (ранній фітофтороз);
- Potato_healthy (здоровий листок).

Зображення організовані у вигляді сітки. Для кожного листка наведено:

- фактичний клас Actual - реальна мітка класу з набору даних;
- прогнозований клас Predicted - клас, визначений нейромережею;
- confidence - рівень впевненості моделі у своєму прогнозі, виражений у відсотках.

У деяких випадках моделі правильно класифікували хворобу з рівнем достовірності до 100%. Також трапляються помилки класифікації, коли модель плутає листя, уражене раннім фітофторозом, зі здоровим листям або з іншою хворобою. Прогнози з довірчою ймовірністю 200% можуть свідчити про неправильну інтерпретацію результатів. Результати демонструють здатність моделі розпізнавати фітофтороз картоплі, але є також випадки помилок, коли модель може потребувати подальшого вдосконалення або стабілізації набору даних.

3.3. Розроблення графічного інтерфейсу інформаційної системи

```
import streamlit as st
from PIL import Image
import numpy as np
import tensorflow.keras as keras
import matplotlib.pyplot as plt
import tensorflow_hub as hub
```

Імпортуйте бібліотеки, необхідні для створення графічного інтерфейсу системи класифікації хвороб картоплі за допомогою Streamlit - бібліотеки для створення веб-додатків з браузерним інтерфейсом. Використовується для створення кнопок, завантаження зображень та відображення результатів.PIL (Python Imaging Library) -

модуль для роботи з зображеннями.Image дозволяє завантажувати імпортовані в модель зображення.Numpy використовується для роботи з числовими масивами, що використовуються в обробці зображень. Library. tensorflow.keras - бібліотека для завантаження збережених моделей та запуску прогнозів. Використовується для взаємодії з уже навченими моделями. matplotlib - бібліотека для візуалізації зображень, результатів прогнозування тощо. tensorflow_hub - бібліотека для завантаження навчених моделей.

```
hide_streamlit_style = """
            <style>
            footer {visibility: hidden;}
            </style>
            """
st.markdown(hide_streamlit_style, unsafe_allow_html = True)
```

Визначають багаторядковий рядок, що містить правила CSS. За замовчуванням, значення Made with приховує нижню частину сторінки, що містить тег Streamlit.

```
st.title('Прогнозування хвороб листя картоплі')
```

st.title() - функція з бібліотеки Streamlit, яка відображає великий заголовок на сторінці. Прогнози хвороб картоплі - текст, що виводиться як заголовок. Заголовок виводиться у формі на сторінці:

```
def main():
    file_uploaded = st.file_uploader('Оберіть зображення...', type='jpg')
    if file_uploaded is not None:
        image = Image.open(file_uploaded)
        st.write("Завантажене зображення.")
        figure = plt.figure()
        plt.imshow(image)
        plt.axis('off')
        st.pyplot(figure)
        result, confidence = predict_class(image)
        st.write(f'Прогноз: {result}')
        st.write(f'Ймовірність: {confidence}%')
```

main() створює інтерфейс для завантаження зображення, його відображення та прогнозування класу хвороби листя картоплі. st.file_uploader - інтерактивний віджет для завантаження файлів. type='jpg' означає, що можна завантажувати лише файли з розширенням .jpg. Використовуйте бібліотеку PIL для відкриття завантажених

зображень. Для відображення зображення створюється нова форма і відображається зображення.

Функція predict_class() викликається для визначення класу зображення. Вона повертає два значення: результат передбаченого класу та рівень достовірності (відсоток достовірності). Передбачений клас і рівень достовірності виводяться на екран.

```
def predict_class(image):
    with st.spinner('Завантаження моделі...'):
        classifier_model = keras.models.load_model(r'final_model.h5', compile=False)
    shape = (256, 256, 3)
    model = keras.Sequential([hub.KerasLayer(classifier_model, input_shape=shape)])
    test_image = image.resize((256, 256))
    test_image = keras.preprocessing.image.img_to_array(test_image)
    test_image /= 255.0
    test_image = np.expand_dims(test_image, axis=0)
    class_name = ['Potato__Early_blight', 'Potato__Late_blight', 'Potato__healthy']
```

predict_class() - завантажує модель, обробляє зображення та прогнозує клас хвороби листя картоплі. Модель нейронної мережі завантажується з final_model.h5. shape = ((256, 256, 3)) - задає форму вхідного зображення (256x256 пікселів, 3 канали RGB). hub.KerasLayer() - використовує попередньо завантажену модель як шар для нейронної мережі. hub.KerasLayer() - для прогнозування. Sequential() - створює одношарову послідовну модель.

image.resize((256, 256)) - змінює розмір зображення до 256x256 пікселів. img_to_array() - конвертує зображення у формат масиву Numpy. /255.0 - нормалізує пікселі до діапазону [0, 1]. np.expand_dims() - додає розміри до формату стеку (1, 256, 256, 256, 3).

Оголошують список можливих класів:

- Potato__Early_blight - ранній фітофтороз картоплі;
- Potato__Late_blight - пізній фітофтороз картоплі;
- Potato__healthy - здоровий лист картоплі.

Щоб завершити функцію, треба додати передбачення та повернення результату.

```
    prediction = model.predict(test_image)
    confidence = round(100 * (np.max(prediction[0])), 2)
    final_pred = class_name[np.argmax(prediction)]
```

```
return final_pred, confidence
```

Це остання частина функції predict_class(), яка пророкує клас зображення. Підготовлене зображення завантажується в модель. prediction - бере набір ймовірностей для кожного класу: [0.05, 0.90, 0.05]. Це означає, що з ймовірністю 90% модель вважає, що на цьому зображенні фітофтороз.

Знайдіть найбільшу ймовірність серед прогнозованих значень. Переведіть ймовірність у відсотки та округліть до двох знаків після коми. Якщо найбільше значення дорівнює 0,90, рівень довіри становить 90,0 відсотків. Визначте індекс найбільшої ймовірності і використовуйте його для визначення класу, що вас цікавить. Якщо індекс дорівнює 1, отримано клас Potato__Late_blight. Повертається назва передбаченого класу та рівень довіри до моделі.

Нехай модель дала такий вектор передбачення: prediction = [[0.05, 0.90, 0.05]].

np.max() → 0.90 → впевненість = 90.0 %.

np.argmax() → 1 → клас = Potato__Late_blight.

Для запуску інформаційної системи використовують команду: `streamlit run potato.py`.

Після цього графічний інтерфейс системи появиться у веб-броузері:

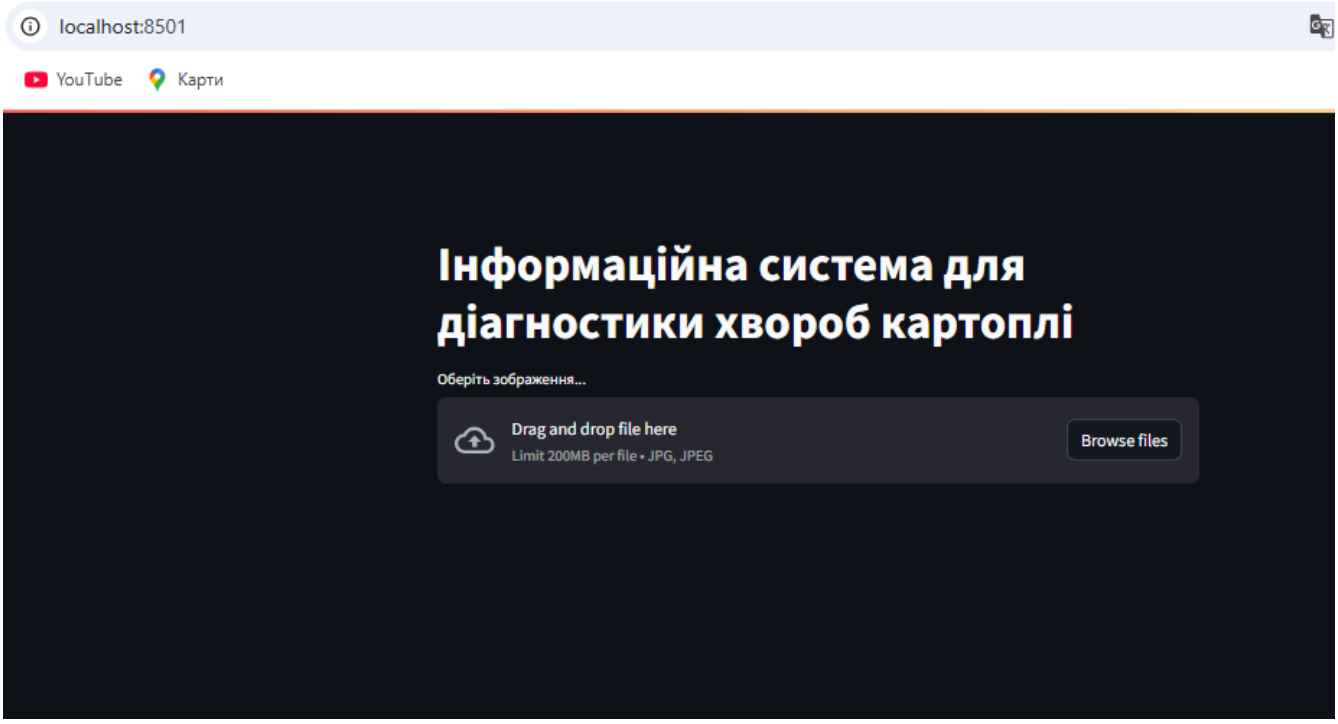

Рисунок 3.5 – Графічний інтерфейс інформаційної системи

Потрібно натиснути на кнопку Browse files та вибрати графічний файл для прогнозування захворювання картоплі.

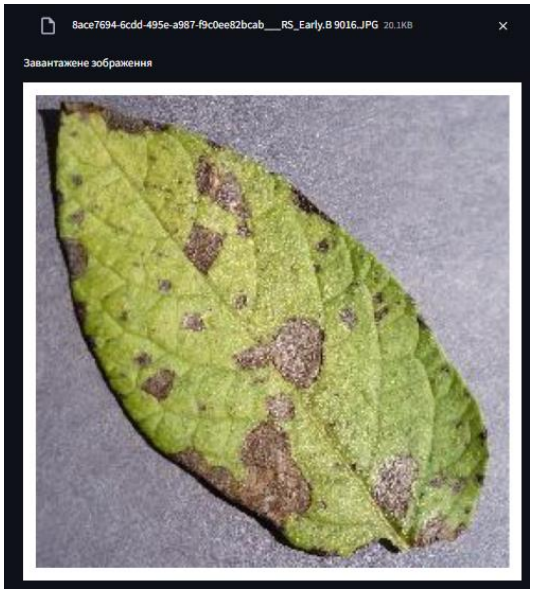

Рисунок 3.6 – Завантажене зображення для діагностики захворювання

Нижче знаходиться результат класифікації захворювання картоплі з відповідними текстовими повідомленнями:

Рисунок 3.7 – Результат класифікації захворювання листя картоплі

Можна вибрати інший графічний файл. Результат прогнозування буде наступним:

Рисунок 3.8 – Результат класифікації захворювання листя картоплі іншого графічного файлу

3.4. Характеристики апаратного та програмного забезпечення

Для проектування інформаційної системи використовувався комп’ютер з наступними характеристиками.

Таблиця 3.2 – Характеристики персонального комп’ютера

Процесор	Intel Pentium	Core i5
Материнська плата	Asus	ZY60-24-Premium
ОП	DDR3	4 GB
Відеокарта	Nvidia GeForce	1024 МГц
Жорсткий диск	Kingstone	300 Гб
Монітор	22″	Samsung

В дипломній роботі використано такі програмні продукти.

Таблиця 3.3 – Програмне забезпечення

Операційна система	Windows 10
Середовище розробки	Python, Tkinter, Google Colab
Растровий графічний редактор	Adobe Photoshop CC
Векторний графічний редактор	CorelDraw X6
Текстовий редактор	Microsoft Word 2016

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено інформаційну систему для діагностики хвороб картоплі на основі згорткових нейронних мереж. Основною метою роботи було створення моделі, здатної точно і швидко класифікувати зображення листя картоплі, виявляючи на них симптоми таких захворювань, як рання і пізня плямистість, а також здорові листки.

Для реалізації проекту був використаний набір даних, що містить зображення картопляних листків, які були класифіковані за трьома категоріями. Спочатку були проведені етапи попередньої обробки даних, такі як масштабування зображень, їх нормалізація та зміна розмірів до стандартних 256x256 пікселів. Це дозволило забезпечити коректне навчання моделі.

Застосовуючи згорткові нейронні мережі, було побудовано глибоку модель для класифікації зображень. Мережа складалася з кількох згорткових і пулінгових шарів, які дозволяли отримати ознаки зображень. У результаті був створений багатокласовий класифікатор, що використовує функцію активації Softmax для визначення ймовірностей для кожного класу.

Для навчання моделі застосовувався алгоритм оптимізації Adam, що дозволив досягти високих результатів. Оцінка точності моделі проводилася за допомогою метрики accuracy та функції втрат Sparse Categorical Crossentropy, що дало змогу контролювати ефективність навчання. Модель продемонструвала високу точність у класифікації тестових зображень, досягнувши результату в 90 % на тестовому наборі.

Для зручності використання моделі була розроблена графічна інформаційна система за допомогою бібліотеки Streamlit, що дозволяє завантажувати зображення листя картоплі і отримувати передбачення про їх стан. Користувачі можуть отримувати результат класифікації з прогнозованим класом захворювання.

Робота довела ефективність використання згорткових нейронних мереж для задач класифікації зображень у сільському господарстві, зокрема для виявлення хвороб на листках картоплі. Високі результати точності показують перспективність застосування даної методики для процесу діагностики захворювань рослин.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Rizqi A., Indra A. Potato leaf disease classification using deep learning approach. 2020 International electronics symposium (IES). 2020, pp. 392-397.

2. Afonso M., Blok P. Blackleg detection in potato plants using convolutional neural networks. International federation of automatic control (IFAC). 2019, pp. 6-11.

3. Qinghua S., Kondo N. Potato quality grading based on depth imaging and convolutional neural network. Hindawi journal of food quality. 2020, pp. 322-331.

4. Moallem P., Razmjooy N., Ashourian M. Computer vision-based potato defect detection using neural networks and support vector machine. International journal of robotics and automation. V.28, № 2. 2013, pp. 137-145.

5. Abeer A., Ibtesam M. Potato classification using deep learning. International journal of academic pedagogical research (IJAPR). V.3, № 12. 2019, pp. 1-8.

6. Simonyan K., Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint. 2014. V.15, pp. 1011-1045.

7. He K., Zhang X., Ren S. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 770-778.

8. Dai J., Li Y., He K. R-FCN: object detection via region-based fully convolutional networks. Neural Inform. Process. Syst. 2016, pp. 379-387.

9. Терейковський І., Бушуєв Д., Терейковська Л. Штучні нейронні мережі: базові положення. Київ: КПІ ім. Ігоря Сікорського. 2022. 123 с.

10. Субботін С. Нейронні мережі: теорія і практика. Житомир: Євенок, 2020. – 184 с.

11. Субботін С., Олійник А. Нейронні мережі: навчальний посібник. Запоріжжя: ЗНТУ, 2014. – 132 с.

12. Руденко О., Бодянський Є. Штучні нейронні мережі. Харків: СМІТ, 2006. – 404 с.

ДОДАТКИ

ДОДАТОК А

potato.ipynb

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "Dataset", shuffle=True, image_size  = (IMAGE_SIZE, IMAGE_SIZE), batch_size =
BATCH_SIZE)
    Found 2152 files belonging to 3 classes.
classname = dataset.class_names
classname
 ['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']


for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())


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

plt.figure(figsize=(10,10))
for image_batch, label_batch in 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(classname[labels_batch[i]])
        plt.axis("off")


len(dataset)
68
```

```
train_size = 0.8
len(dataset)*train_size


train_ds = dataset.take(54)
len(train_ds)


test_ds = dataset.skip(54)
len(test_ds)


val_size = 0.1
len(dataset) * val_size


val_ds = test_ds.take(6)
len(val_ds)


test_ds = test_ds.skip(6)
len(test_ds)


train_split=0.8
val_split=0.1
test_split=0.1

assert train_split + val_split + test_split == 1


def  get_dataset_paratitions_tf(ds,  train_split=0.8,  val_split=0.1,  test_split=0.1,
shuffle=True, shuffle_size=10000):
    assert train_split + val_split + test_split == 1
    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_paratitions_tf(dataset)
```

```
len(train_ds)

len(val_ds)

len(test_ds)


train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size = tf.data.AUTOTUNE)


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


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

train_ds = train_ds.map(
lambda x, y : (data_augmenation(x, training=True), y)).prefetch(buffer_size = 
tf.data.AUTOTUNE)


input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
      layers.Conv2D(32, kernel_size = (3,3), activation = 'relu', input_shape = 
input_shape),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),
```

```
    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Conv2D(64, kernel_size = (3,3), activation = 'relu'),
    layers.MaxPooling2D(2,2),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(n_classes, activation='softmax'),

])

model.build(input_shape=input_shape)

model.summary()

model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics = ['accuracy'])

history = model.fit(
    train_ds,
    epochs = 10,
    batch_size = BATCH_SIZE,
    verbose = 1,
    validation_data = val_ds
)


scores = model.evaluate(test_ds

scores

history.params

history.history.keys()

type(history.history['loss'])
```

```
len(history.history['loss'])

acc = history.history['accuracy']
val_acc  = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

val_acc

plt.figure(figsize=(15,5))
plt.subplot(1, 2, 1)
plt.plot(range(10), acc, label = "Training Accuracy")
plt.plot(range(10), val_acc, label = "Validation Accuracy")
plt.legend(loc = 'lower right')
plt.title('Training & Validation Accuarcy')

plt.subplot(1, 2, 2)
plt.plot(range(10), loss, label = "Training Loss")
plt.plot(range(10), val_loss, label = "Validation Loss")
plt.legend(loc = 'upper right')
plt.title('Training & Validation Loss')

import numpy as np
for image_batch , labels_batch in test_ds.take(1):
    first_img = image_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_img)
    print('actual label:', classname[first_label])

    batch_prediction = model.predict(image_batch)
    print('predicted label:', classname[np.argmax(batch_prediction[0])])

def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(img.numpy())
    img_array = tf.expand_dims(img_array, 0)

    prediction = model.predict(img_array)

    predicted_class = classname[np.argmax(prediction[0])]
    confidence = round(100 * (np.argmax(prediction[0])), 2)
    return predicted_class, confidence
```

```
plt.figure(figsize = (15,15))

for images,labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3 ,i+1)
        plt.imshow(images[i].numpy().astype("uint8"))
        predicted_class, confidence = predict(model, images[i])
        actual_class = classname[labels[i]]

          plt.title(f"Actual  :{actual_class},  \n  Predicted:  {predicted_class}.  \n
Confidence: {confidence}%")

        plt.axis("off")
```

ДОДАТОК Б

potato.py

```
import streamlit as st
from PIL import Image
import numpy as np
import tensorflow.keras as keras
import tensorflow_hub as hub
import matplotlib.pyplot as plt
st.title('Інформаційна система для діагностики хвороб картоплі')

def main():
    file_uploaded = st.file_uploader('Оберіть зображення...', type='jpg')
    if file_uploaded is not None:
        image = Image.open(file_uploaded)
        st.write("Завантажене зображення")
        figure = plt.figure()
        plt.imshow(image)
        plt.axis('off')
        st.pyplot(figure)
        result, confidence = predict_class(image)
        st.write(f'Результат класифікації: {result}')
        st.write(f'Ймовірність: {confidence}%')

def predict_class(image):
    with st.spinner('Завантаження моделі...'):
        classifier_model = keras.models.load_model(r'final_model.h5',
compile=False)
    shape = (256, 256, 3)
    model = keras.Sequential([classifier_model])
    test_image = image.resize((256, 256))
    test_image = np.array(test_image).astype('float32') / 255.0
    test_image = np.expand_dims(test_image, axis=0)
    class_name = ['Potato_Early_blight', 'Potato_Late_blight', 'Potato_healthy']
    prediction = model.predict(test_image)
    confidence = round(100 * np.max(prediction[0]), 2)
    final_pred = class_name[np.argmax(prediction)]
    return final_pred, confidence

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