University of Hertfordshire

Department of Physics, Astronomy and Mathematics MSc Data Science

Evaluating the effectiveness of Deep Learning Models in Detecting Leaf Diseases: A Comprehensive Study

By

SAQIB SADDIQUE

Student (ID 22019524)

Supervisor: Dr. Man Lai Tang

Word Count : 10000

Submitted: Date: 29-8-2024

MSC FINAL PROJECT DECLARATION

This proposal statement is written as part of the qualification for the Master of Data Science degree programme of the University of Hertfordshire (UH). I hereby certify that all the content of this work is my creation, unless specific sources are mentioned in this report. With regards to permission to disseminate this report on the web, through module sites, I still permit it, on condition that the source is acknowledged.

DEDICATION

I dedicate this research to all my family: my colleagues; and friends; individuals who have supported me throughout the time I used to complete this effort especially when I felt discouraged while going through the process.

ACKNOWLEDGMENT

I would like to precise my deep gratitude to my thesis supervisor, Dr. Man Lai Tang, for his leadership and motivation through the course of my thesis. His passion and tolerance have allowed me to mark significant assistances to my research. He has been my main source of support, both academically and emotionally. This work would not have been probable without the motivation and inspiration of my family.

ABSTRACT

Initial discovery of plant diseases is vital for preventing adverse effects on plant growth and overall yield. While traditional machine learning models have remained used for plant disease discovery and classification, deep learning models, particularly Convolutional Neural Networks (CNNs), have shown significant possible in improving accuracy. CNNs have been widely utilized in various agricultural applications, including plant species identification, yield organization, weed detection, soil and water management, fruit counting, pest detection, and nutrient status evaluation. An automated disease detection system leveraging CNNs can enable farmers to diagnose plant diseases swiftly and accurately, thus supporting timely intervention. In this study, we explored the application of several CNN architectures—AlexNet, VGG19, MobileNetV2, and InceptionV3—for classifying plant leaf diseases using a recreated PlantVillage dataset comprising 87,000 RGB images across 38 categories of crop leaves. Each model was trained and evaluated on separate test sets to assess their performance. AlexNet achieved a test accuracy of 77.95%, demonstrating solid performance but with limitations in generalizing across all categories. VGG19 performed strongly with a test accuracy of 90.77%, although it struggled with certain disease categories, resulting in a varied macro average f1-score of 0.43. MobileNetV2 reached a test accuracy of 89.01%, excelling in some classes but facing challenges in others, reflected in a macro average f1-score of 0.45. InceptionV3 demonstrated high performance with a test accuracy of 90.73%, but like the other models, it exhibited inconsistent results across different categories, with a macro average f1-score of 0.46. The findings highlight the challenges of achieving consistent classification accuracy across a diverse range of plant diseases and underscore the need for further model refinement and experimentation. Automating plant leaf disease detection through CNNs not only facilitates faster crop diagnosis but also contributes to the broader goal of precision agriculture.

Keywords: Convolutional Neural Network, Plant Disease, Deep Learning

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. CHAPTER 1: INTRODUCTION

1.1. Introduction

Plant diseases can significantly affect agriculture, leading to substantial yield reductions and financial damages for farmers. Additionally, approximately diseases can degrade crop value, building them less attractive to customers. Consequently, it is crucial to detect, classify, and know crop infections at an early stage to improve both the excellence and quantity of the yield and prevent crop harm (Gebbers and Adamchuk, 2010; Rajasekaran and Anandamurugan, 2019). However, classifying plant diseases grounded on leaf symptoms with the bare eye is challenging for farmers, and seeking expert consultation can be costly, especially in developing countries. Common symptoms such as dieback, decline, wilting, leaf blight, leaf spots, fruit rots, and root rots can indicate the presence of plant diseases (Al Bashish et al., 2010). Detecting diseases by examining these visible indications requires significant information and experience, making it necessary for a trained or experienced individual to examine the plant.Fig1.1 show different healthy unhealthy leaf. Moreover, visual biases and illusions can result in incorrect diagnoses, rendering this approach impractical for large agricultural areas (Thangaraj et al., 2022). It is applied for finding out solutions of initial discovery of plant diseases at a pace as swift as the appearance of disease on the leaves through the application of machine learning and deep learning methods. This broadsheet delivers state-of-art (SOTA) analysis of deep learning mockups for plant disease detection founded on experimental results (Bock et al., 2010).

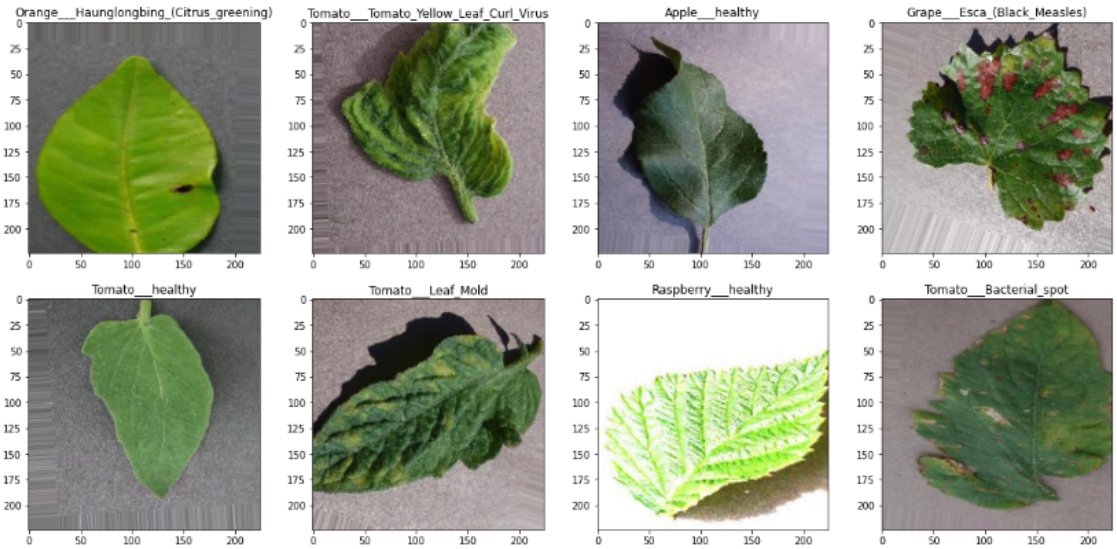


Fig. 1.1. Healthy and unhealthy leaves.

Researchers have suggested technological solutions to address the inadequacies of traditional approaches. However, choosing the most appropriate model for a given dataset,

parameters, hardware arrangement, and experimental circumstances has develop progressively stimulating owing to the wide variety of mechanism knowledge and deep learning models proposed in the literature for predicting plant diseases. Examples of these models include artificial neural networks (ANN) (K.-Y. Huang, 2007), provision vector machines (SVM)(T. Huang et al., 2018), Bayes classifiers (Bauer et al., 2011), Accidental Woodland Li et al., 2016, and the K-Nearest Neighbor algorithm (KNN) (Tan et al., 2016). A thorough review of models is necessary to properly handle data preprocessing, forecast, and organization of plant diseases.

1.2. Machine Learning Overview

Machine learning is a branch of artificial intelligence (AI) that concentrates on building systems and procedures that can study from data and mark forecasts or choices based on that learning. The fundamental concept of machine learning involves creating algorithms that can autonomously discover designs and relations in data without the need for explicit programming.

1.3. Deep Learning Overview

Deep learning is a field of machine learning that emphasizes the usage of neural networks with manifold layers (hence the name "deep") to study from vast quantities of data. It has brought significant advancements to numerous fields, particularly those dealing with complex data such as images, speech, and text.

1.4. Problem Statement

The agriculture sector is pivotal to the Pakistan economy, employing closely 50% of the country's staff. India grips the difference of being the world's main builder of pulses, rice, wheat, spices, and spice products. The financial well-being of farmers pivots on the excellence and yield of their produce, which, in turn, relies deeply on plant health. Detecting diseases in plants is crucial as they can significantly impact growth and ecological balance for farmers. Automatic disease discovery techniques are increasingly advantageous for identifying plant diseases at initial stages. Symptoms manifest distinctly in various plant parts, particularly leaves. Manual inspection of leaf images for disease detection is labor-intensive. Therefore, developing computational methods for automatic disease detection and classification using leaf images is imperative for agricultural advancement.

1.5. Motivation of this Study

Identification of crop diseases and selecting appropriate remedies are fundamental steps in farming that have been experienced by farmers for ages to prevent crop damage and preserve quality. However, international climate changes, lifestyle changes, pollution, and other factors have led to a significant increase in various crop diseases. It has become crucial for farmers to have knowledge of these diseases for effective identification and control. Given the numerous types of infections, it is challenging for farmers to be aware by all of them. Additionally, in large-scale farming, it is not feasible for farmers to monitor vast areas of crops economically or physically. Consequently, many crop infections and poisons go unobserved, disturbing the overall yield and quality.To address these challenges, automatic discovery and arrangement of crop diseases are necessary. Machine Learning and Computer Vision are increasingly being utilized by farmers and researchers worldwide in agriculture for various purposes. Machine Learning-based techniques can provide accurate and real-time detection and classification of crop diseases, enhancing productivity and quality while reducing labor and costs.

1.6. Need of plant leaf disease detection

Plant leaf disease recognition is critical due to its profound impact on farmers and consumers alike. Diseases that distort plants not only diminish farmers' incomes but also lead to supply shortages for consumers. The global economic loss attributed to plant leaf diseases amounts to an estimated 60 billion dollars annually, underscoring the severity of the issue.A key motivation for this project is to simplify disease detection for farmers, who often struggle to identify diseases merely by visual inspection of leaves. The proposed system aims to empower farmers by providing easy-to-use tools for disease identification. Moreover, it will recommend appropriate pesticides to treat identified diseases promptly, thereby mitigating further spread and potential yield losses. Effective disease management is crucial as infections can quickly escalate from a single leaf to affect entire crops.

1.7. Research Question

The main research question of this research is the following:

- Is it possible to identify the leaf diseased with deep learning models?
- How it would help in agriculture sector while identify leaf diseases ?
- Can the research help in country economy while helping in agriculture sector ?

1.8. Aims and Objectives

The key contributions of this research:

- Plant leaf recognition and its reputation in the plant science and agriculture fields.
- A review of current deep learning-based leaf detection methods
- The presentation assessment of deep learning models.

2. CHAPTER II: LITERATURE REVIEW

2.1. LITERATURE REVIEW

A plant affected by disease typically exhibits symptoms such as changes in hue, form, size, or a slowed development cycle. As a result, dissimilar phases of a plant disease manifest various indications that progressively appear as the disease-causing agents begin to impact a strong plant. At certain stages, it can be challenging to distinguish between healthy and diseased plants. A disease compromises the plant's immune system, making it vulnerable to multiple diseases at once, with symptoms of different diseases often appearing similar(Barbedo, 2016). Bangari et al. (Bangari et al., 2022) presented a evaluation of infection discovery using CNN (convolutional neural networks) and, after evaluating several studies, concluded that CNNs significantly enhance disease detection correctness. Abade et al. (Abade et al., 2021) reviewed 121 papers available among 2010 and 2019, identifying Plant Village as the greatest commonly used dataset and Tensor Flow by way of the greatest often employed framework. Additionally, Nagaraju et al. (Nagaraju and Chawla, 2020) discoursed the maximum suitable datasets, pre-processing methods, and deep learning techniques aimed at plant disease discovery, reviewing and analyzing 84 papers on the submission of deep learning in identifying plant diseases. Pre-processing images is often crucial for effectively analyzing them using deep learning methods. Wang et al. (Wang et al., 2013) future a novel technique for classifying insect pests and plant sicknesses based on the latest computer vision and image processing techniques. Their analysis of smartphone images revealed bug pests and illnesses in vegetables, with a new abstraction and organization technique aiding in the identification of leaf diseases. Segmented images were used to determine insect totals and diseased areas through region classification. Their planned method was verified in the arena using mobile smart devices, with bond areas detached by means of exact morphology. In (Geetharamani and Pandian, 2019), a nine-layer CNN model remained introduced aimed at noticing plant diseases, utilizing information increase methods to augment the dataset and assess the model's presentation. They originate that information augmentation could enhance the replica's effectiveness. Ferentinos et al. (Ferentinos, 2018) used CNN architectures to classify 58 different plant diseases, testing the CNN models with real-time images. Their approach successfully detected plant sicknesses using humble imageries of fit or diseased leaves founded on specific convolutional neural network projects. In a recent investigation (Tirkey et al., 2023), several deep learning-based methods were assessed for their effectiveness in actual bug discovery and documentation in soybean crops. The education explored the feasibility and reliability of transmission learning models. The method achieved impressive accuracies of 98.75%, 97%, and 97% using YoloV5, InceptionV3, and CNN, correspondingly. Amongst these, YoloV5 was particularly notable for its ability to operate at 53 fps, creation it highly appropriate for real-time discovery scenarios. A

varied dataset of crop insects, sourced and branded from pictures captured with different plans, was utilized. This method significantly alleviated the burden on producers while delivering superior results.

Similarly, (Jasim and Al-Tuwaijari, 2020) introduced a system that employed deep learning to categorize and notice plant leaf diseases, utilizing the Plant Village dataset. They cast-off a CNN to categorize viruses across 15 classes, including 12 disease categories (caused by bacteria, fungi, etc.) and three healthy leaf categories. The system achieved remarkable accuracies of 98.29% during training and 98.029% during testing.

In another research (Upadhyay and Kumar, 2022), an efficient way aimed at classifying and categorizing rice vegetable diseases founded on the scope, form, and hue of cuts was proposed. Using Otsu's global thresholding method for image binarization, the study effectively removed background noise. The fully linked CNN, trained on 4000 imageries of both fit and unhealthy rice leaves, attained an outstanding correctness of 99.7%, outperforming previous methods.

Moreover, (Guerrero-Ibañez and Reyes-Muñoz, 2023) industrialized a CNN-based model to categorize and categorize tomato leaf viruses by means of a public dataset supplemented by locally captured images. To prevent overfitting, generative adversarial networks (GANs) were recycled to make examples similar to the exercise data. The model demonstrated exceptional performance, achieving over 99% correctness in both exercise and test datasets.

A comprehensive study (Ahmed and Yadav, 2023) made use of the PlantVillage dataset to identify a variety of plant diseases, including those produced by bacteria, viruses, mold, and mites, across 12 crop species. The study employed machine learning techniques such as SVMs, grey-level co-occurrence matrices (GLCMs), and CNNs. The planned method attained overall accuracies of 99% for rice, 98% for apples, and varying accuracies for tomatoes (96%, 94%, 95%, and 97%). Evaluation was based on accuracy, recollection, and f-measure metrics for multi-class arrangement problems.

Additionally, (Balaji et al., 2023) demonstrated the application of enhanced CNN techniques for rice disease detection. The education highlighted the achievement of deep neural networks (DNNs) in duplicate arrangement tasks, achieving accuracies of 80%, 85%, 90%, and 95% for TL, CNN + TL, ANN, and ECNN + GA methods, correspondingly. Another research [40] compared machine learning (ML) then deep learning (DL) techniques aimed at plant disease prediction, with random forest (RF) attaining the uppermost correctness (97.12%) among ML techniques, while CNN excelled among DL models with 98.43

In another study, (Gopi and Kondaveeti, 2023) tackled the challenges faced by DL models in classifying rice leaf diseases owing to image backgrounds and gaining conditions. The study evaluated well-known transfer learning models, using both frozen layers and fine-tuning approaches. DenseNet169 achieved a testing correctness of 99.66%, while fine-tuned Xception models performed exceptionally well with a 99.99% testing accuracy.

In recent work (Abd Algani et al., 2023), the authors introduced an innovative deep learning technique named Ant Gathering Optimization with Convolution Neural Network (ACO-CNN) aimed at noticing then ordering plant infections. The ACO method remained recycled to measure the efficiency of illness diagnostics popular plant leaves, while the CNN classifier extracted color, texture, and geometric features from the images. The ACO-CNN classical outdone C-GAN, CNN, and SGD models, achieving a 99.98% accuracy rate. This model also excelled in exactness, recall, and F1-score, surpassing the other models, which attained correctness rates of 99.6% (C-GAN), 99.97% (CNN), and 85% (SGD).

Another study (Dai et al., 2023) proposed a deep learning model named PPLCNet, which incorporated opened difficulty, a multi-level care instrument, and GAP layers. This classical utilized climate data increase to increase taster size and improve feature abstraction. The feature abstraction link used saw-tooth opened difficulty to address inadequate data extraction, while the CBAM kindness instrument improved information representation. The GAP layer helped to prevent overfitting by dipping the difficulty of network limitations. PPLCNet attained a gratitude correctness of 99.702% and an F1-score of 98.442%, with the amount of limits and FLOPs being 15.486M and 5.338G, individually.

In (Pushpa and Aiswarya, 2022), for classification of the tomato leaf diseases, the authors presented a 2-dimensional Convolutional Neural Network (2DCNN) by 2-Max Assembling and densely connected coatings. The perfect obtained an correctness of 96 percent and the same has been represented in the following figure which shows that the organization model of the perfect outperformed additional organization replicas such by way of SVM, VGG16, Beginning V3, and Mobile Net CNN.

Research (Saberi Anari, 2022) employed model engineering (ME) and various SVM models to enhance feature judgement and dispensation speed. The model was skilled on six leaf image sets after the PlantVillage and UCI databases, covering fit and unhealthy leaves of apple, corn, cotton, grape, pepper, and rice. The experimental results indicated an regular correctness of 98.5% for leaf disease credit, demonstrating the model's stability and effectiveness.

In (V. Singh and Misra, 2017), a hereditary algorithm-based image division technique was proposed for the involuntary detection and organization of plant leaf illnesses. The study also reviewed various disease classification techniques, underscoring the importance of the genetic algorithm for disease detection.

The collaborative classifiers (EC) developed in (Saraswathi and FarithaBanu, 2023) remained tested by means of the PlantVillage and Taiwan tomato leaves databases. The top-performing ensemble classifier attained an correctness of 96%, showcasing precision and reliability under various conditions, including shadow, brightness fluctuations, and diverse textures.

A hybrid deep learning approach, joining CNN, convolutional attention module (CBAM), and SVM, was presented in (Altalak et al., 2022) for early discovery and organization of

tomato plant leaf diseases. The model attained an correctness of up to 97.2%, surpassing state-of-the-art approaches, and was lightweight enough to be used on smart devices equipped with digital cameras and processing capabilities.

In another study (Nawaz et al., 2022), the authors evaluated their method on the PlantVillage Kaggle dataset, achieving map and correctness values of 98.10% and 99.97%, correspondingly, with a examination time of 0.23 seconds. This robust method for tomato leaf disease classification proved effective under various image transformations and capture conditions.These educations collectively prove the effectiveness of deep learning models, particularly CNNs and transfer learning techniques, in refining the correctness and reliability of plant disease detection and organization, offering substantial benefits to agricultural productivity.

3. CHAPTER III: RESEARCH METHODS

3.1. Methodology

In this research, we employed a systematic approach to analyze leaf images by first extracting the feature matrix and subsequently applying various deep learning models to the dataset. The presentation of these models was then measured based on exercise and authentication accurateness. Fig. 3.1 illustrates the block diagram of our methodology. The models, trained on a mix of diseased and healthy leaf images, were utilized for prediction tasks. A number of presentation metrics were cast-off to assess the efficiency of the replicas, such as exactness, recollection, F-score, and precision. Following the initial evaluation, we tested four distinct pre-trained convolutional neural networks (CNNs). These models, including AlexNet, VGG19, MobileNetV2, and InceptionV3, were trained on new plant disease datasets to detect leaf diseases.

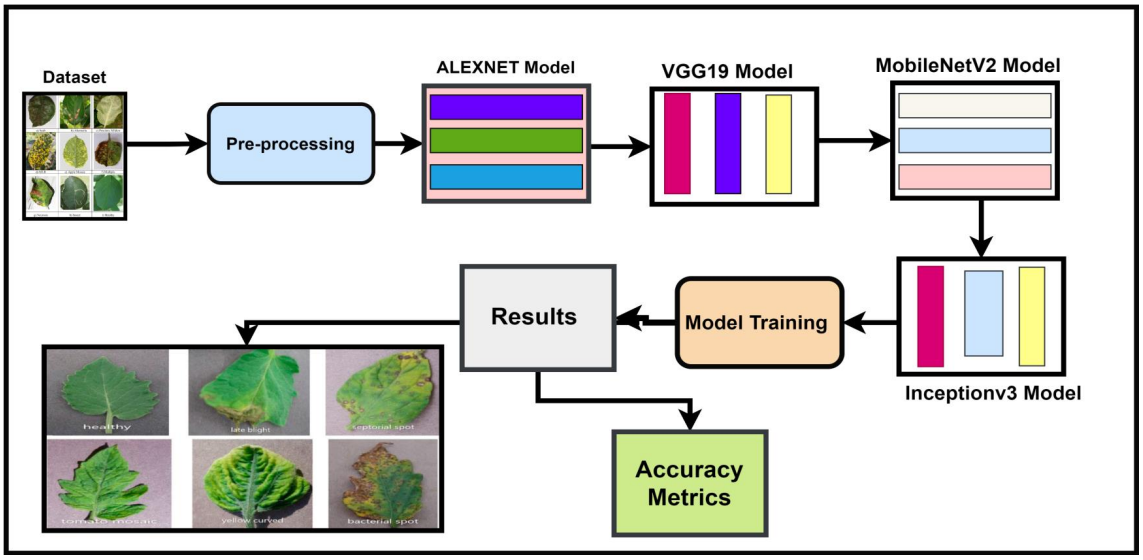


Fig. 3.1. Show Research Framework

3.1.1. Dataset

The dataset was reconstructed by means of the PlantVillage dataset combined with off augmentation techniques. It includes approximately 87,000 RGB pictures, categorized into normal then abnormal crop greeneries across 38 separate lessons. The dataset is divided hooked on exercise then authentication subsets in an 80/20 relation, maintaining the handbook construction. Additionally, 33 new test images were generated in a separate directory for prediction purposes (Mohanty et al., 2016).

3.1.2. Softmax Activation Function

The softmax start purpose is commonly employed in neural networks for addressing multi-class classification problems. It processes a path of actual numbers and outputs a likelihood delivery across the various classes. This purpose smears an exponential transformation to respectively contribution value and regularizes the results so that they sum to 1, thus forming a likelihood delivery. This enables the system to predict the probability of individually class, representing the likelihood that a particular contribution fits to each class. In our experiment, we utilized the softmax start function.

3.1.3. Data Regularization

To mitigate overfitting in neural networks, regularization techniques such as dropout are employed. Throughout exercise, failure erratically "drops out" or circles to zero a portion of the network's neurons. This process compels the network to depend on the remaining neurons, reducing the risk of overfitting by any single neuron. Dropout is commonly applied to fully connected layers, with typical rates ranging between 0.2 and 0.5. In our models, a failure rate of 0.2 is applied.

3.1.4. Dropout in Neural Networks

The concept of dropout in neural networks can be represented mathematically as follows:

3.1.5. 1. Dropout During Training

Let's say you have a fully connected layer with n neurons, where each neuron produces an output y_i for $i = 1, 2, \ldots, n$.

- During training, a dropout mask m_i is applied to each neuron, where m_i is a Bernoulli random variable with probability p (the dropout rate). The probability distribution of m_i is:

$$m_i = \begin{cases} 0 & \text{with probability } p \\ 1 & \text{with probability } (1 - p) \end{cases}$$

- The output of each neuron after applying dropout is:

$$\hat{y}_i = m_i \cdot y_i$$

  Here, $\hat{y}_i$ is the output of the i-th neuron after dropout, which is either 0 (if the neuron is "dropped out") or y_i (if the neuron is kept).

3.1.6. 2. Scaling During Inference

During inference (i.e., when the network is making predictions), dropout is not applied. Instead, to maintain the expected output of the neurons, the outputs are scaled by the probability $1-p$. This scaling ensures that the expected sum of the neuron outputs remains consistent between training and inference.

$$\hat{y}_i = (1 - p) \cdot y_i$$

3.1.7. Early stopping

Initial discontinuing is a regularization technique secondhand to stop overfitting in mechanism learning models, mainly in neural networks. It works by monitoring the replica's presentation on a authentication set throughout exercise. Training is halted when the presentation stops improving, indicating that further training would likely lead to overfitting. Mathematically, early stopping involves tracking the validation loss (or accuracy) and comparing it across epochs. When the loss no longer decreases (or accuracy stops improving) for a sure amount of epochs, known as the "tolerance" parameter, exercise is stopped. This helps in finding the optimal point where the model has learned enough to generalize well but not so much that it starts memorizing the training data.

Mathematical Formulation of Early Stopping

1. **Validation Loss/Accuracy Tracking**:
   - Let $L_{\text{val}}(t)$ represent the validation loss (or $A_{\text{val}}(t)$ for accuracy) at epoch t.
2. **Improvement Calculation**:
   - Calculate the improvement in validation loss as:
   $$\Delta L_{\text{val}}(t) = L_{\text{val}}(t-1) - L_{\text{val}}(t)$$
   - Or for accuracy:
   $$\Delta A_{\text{val}}(t) = A_{\text{val}}(t) - A_{\text{val}}(t-1)$$
3. **Patience Parameter**:
   - Define a "patience" parameter p, which is the number of epochs to wait for an improvement in validation loss or accuracy.
4. **Stopping Criterion**:

- If the validation loss (or accuracy) does not improve by a certain threshold ϵ for p consecutive epochs, training is stopped. This is mathematically expressed as:

  If $\Delta L_{\text{val}}(t) \leq \epsilon$ (or $\Delta A_{\text{val}}(t) \leq \epsilon$) for p consecutive epochs, then stop training.

By stopping the training process when no significant improvement is observed, early stopping ensures that the model does not continue to learn beyond the point where it generalizes well, thus mitigating the risk of overfitting.

3.2. Binary Cross Entropy

Binary cross-entropy helps as a crucial damage purpose in machine learning, particularly suited for binary organization problems where the objective is to determine if an contribution fits to one of two distinct lessons. It quantifies the discrepancy amid the true tags and the predicted likelihoods for apiece class by evaluating the entropy change amid the predicted likelihoods and the actual binary tags. Formally, binary cross-entropy is expressed as the bad quantity of the right labels increased by the logarithm of the foretold likelihoods for the confident class, combined with the accompaniment of the true tags increased by the logarithm of the forecast chances for the negative class. This loss function is extensively used in neural network training, particularly once a sigmoid activation purpose is employed in the production layer, as it efficiently punishes inaccuracies and promotes precise classification(Ruby and Yendapalli, 2020).

3.3. Batch Size

The lot size denotes the quantity of training instances treated in a single onward and backward pass during neural network training. Larger batch sizes generally provide more stable and reliable gradient estimates but might demand extra memory and potentially unhurried down the training process. Determining the best batch size depends happening the specific perfect and dataset and often needs research to identify the most real value (Radiuk, 2017).

3.4. Batch Normalization Layer

Batch standardization is a vital method in deep learning aimed at improving neural network training. It operates by normalizing the output of the preceding activation layer, adjusting and scaling it according to the nasty and alteration of the current lot. This approach helps reduce inner covariate shift, leading to quicker convergence, enhanced accuracy, and improved stability throughout the exercise of deep neural networks(Bjorck et al., 2018).

3.5. Dense Layer

The thick layer is a key element of a neural network, anywhere respectively neuron in unique coating is linked to each neuron cutting-edge the following coating. It does a linear transformation proceeding the contribution information, shadowed by a non-linear start purpose, thereby increasing the model's learning capability. This layer is commonly employed in the final stages of neural networks for tasks such as reversion and organization (Javid et al., 2021).

3.5.1. Pooling Layer

Pooling layers, also recognized as down-sampling layers, are essential in convolutional neural networks (CNNs) within deep learning.Fig3.2 Show the diagram their main purpose is to reduce the 3-D sizes of the contribution data, exactly its width and height, while retaining important information (P. Singh et al., 2020).

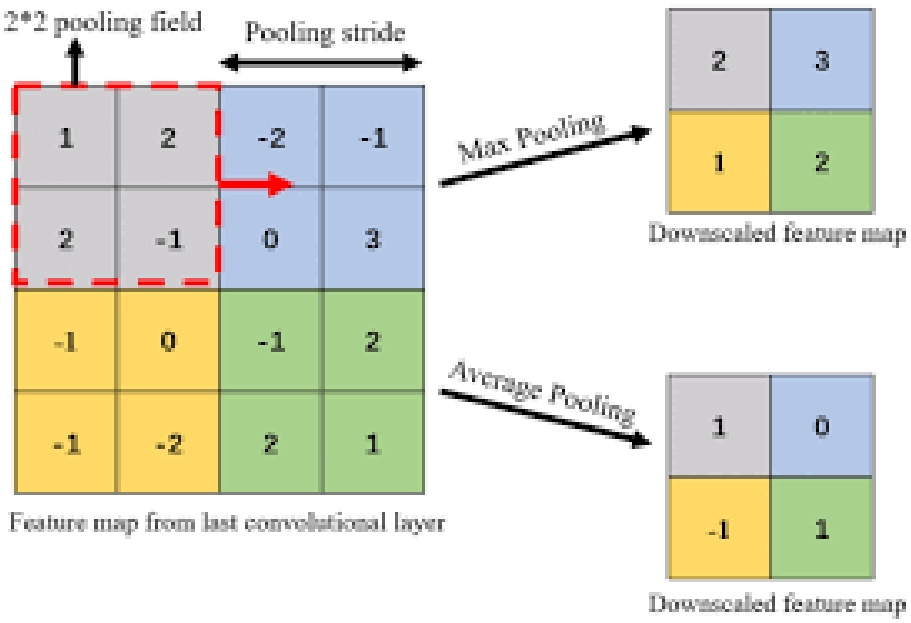


Fig. 3.2. Pooling Block Diagram (Yingge et al., 2020)

3.6. AlexNet Architecture

AlexNet is a ground-breaking deep convolutional neural network building that meaningfully advanced the arena of deep learning. It contains of eight coatings by trainable parameters: five convolutional coatings and three completely linked layers. A 224x224x3 RGB image is input to the network. Max-pooling follows the primary convolutional coating, which spreads 96 11x11 sieves with a pace of 4. The additional convolutional layer then utilizes 256 filters of size 5x5, also shadowed by max-pooling. In the third convolutional layer, 384 filters of size 3x3 are used. The quarter convolutional layer similarly applies

384 filters of size 3x3. The fifth and final convolutional layer employs 256 filters of size 3x3, followed by another max-pooling operation. The initial completely linked layer is composed of 4096 neurons, and the second entirely connected layer also contains 4096 neurons. The third totally associated layer contains of 1000 neurons, bring into line with the 1000 categories in the ImageNet dataset. The block diagram is shown Fig.3.3. The network's final layer is a softmax layer, which generates a likelihood distribution across the 1000 classes. AlexNet utilizes Rectified Linear Units (ReLU) as its start function and incorporates dropout in the fully connected layers to mitigate overfitting. Moreover, AlexNet leverages data augmentation and L2 regularization to enhance its generalization capabilities.

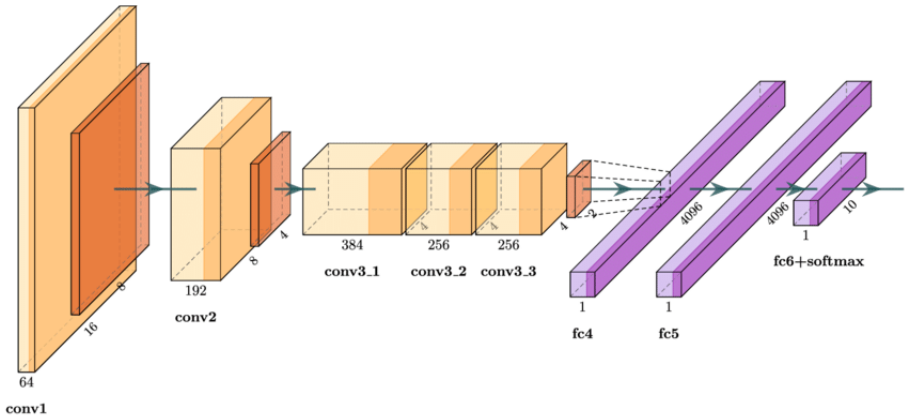


Fig. 3.3. Block Diagram AlexNet-Architecture

3.6.1. VGG19 is a deep convolutional neural network

The Visual Geometry Group at Oxford presented the VGG19 deep convolutional neural network in 2014. It buildings 19 weight layers, including 16 convolutional layers with small 3x3 filters and 3 fully connected layers, prepared into five blocks with max-pooling layers.Showing diagram in Fig.3.4. VGG19 builds on the VGG16 design by adding more layers, allowing it to learn more composite features and achieve high accuracy on the ImageNet dataset. Notwithstanding its improved computational and memory concerns, VGG19 is widely used for its operative and straightforward architecture, making it popular for transfer learning in numerous image classification tasks.

3.6.2. Inception V3

The Inception v3 deep convolutional neural network, developed by Google, was for the first time introduced in the paper titled "Rethinking the Beginning Architecture for Computer Vision(Szegedy et al., 2016). This network, which is a modification of the original Inception architecture shown in Fig.4.2 was skilled on a dataset covering 1.2 million pictures across 1000 different classes. Upon its release, the ideal set new standards in image

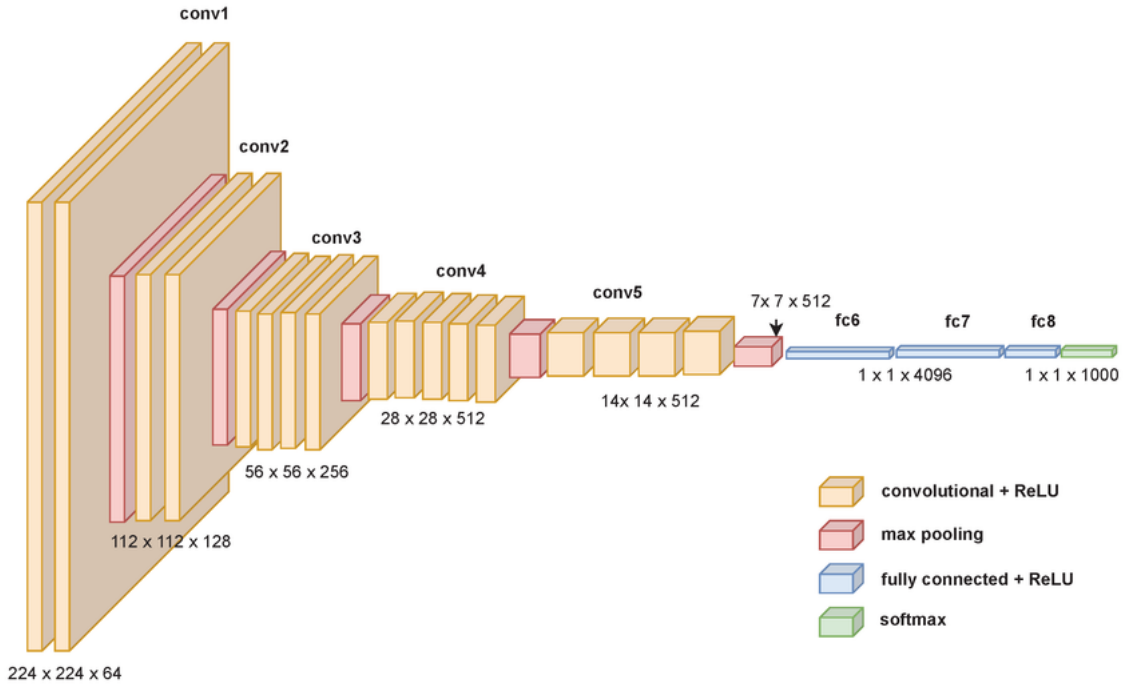


Fig. 3.4. Block Diagram VGG19 (Alibabaei et al., 2022)

classification tasks and remainders highly real for similar applications today (Xia et al., 2017).

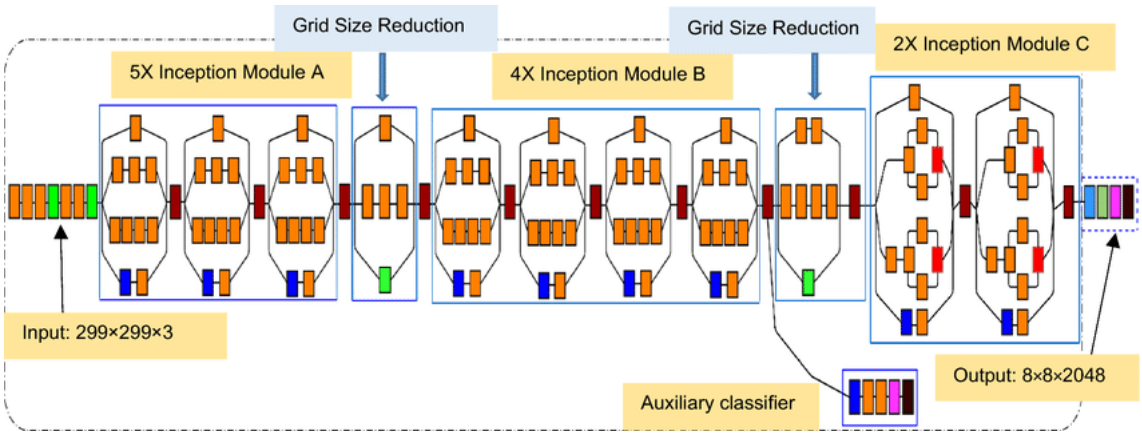


Fig. 3.5. Block Diagram Inception-v3(T. Singh and Vishwakarma, 2021)

3.7. MobileNetV2 Architecture

MobileNetV2 is a convolutional neural network architecture intended for mobile and resource-constrained environments, emphasizing efficiency and performance. It builds upon the original MobileNet architecture by introducing inverted residuals and linear bottlenecks. The key component is the reversed remaining block, where the effort and production are thin bottleneck layers, and the intermediate layer is expanded using depthwise separable convolutions.Showing diagram in Fig.3.6. This structure allows for significant

reduction in computational cost while maintaining performance. The architecture starts with a standard convolution layer shadowed by multiple bottleneck layers, each containing a depthwise separable convolution, batch normalization, and ReLU6 activation. The network consists of 17 of these bottleneck layers with varying expansion factors, followed by a final convolutional layer, a worldwide regular combining layer, and a fully linked layer with a softmax start for classification. MobileNetV2 efficiently balances model size and accuracy, making it suitable for moveable and entrenched vision applications.

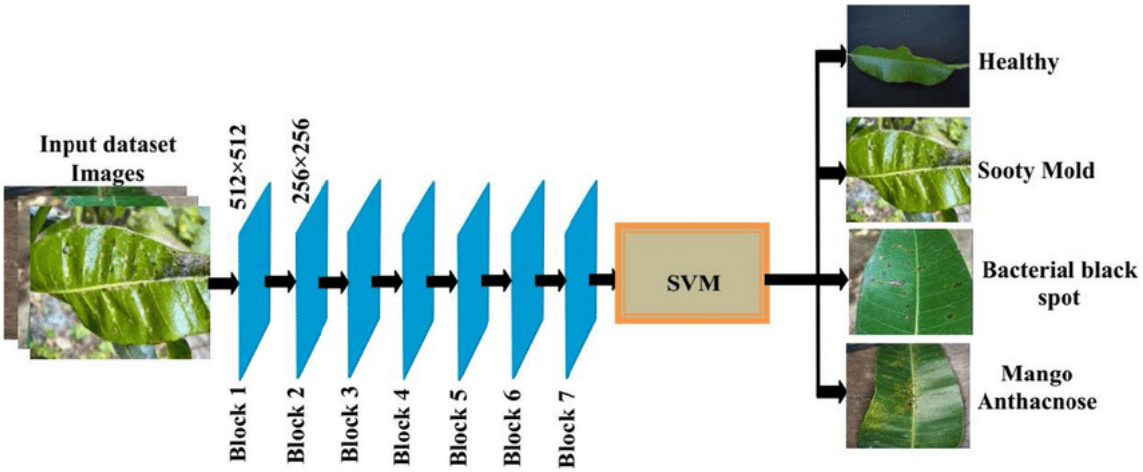


Fig. 3.6. Block Diagram Architectural-MobileNetV2-model

3.8. Rectified Linear Unit Layer

The Rectified Linear Unit (ReLU) start purpose is a popular choice in deep learning models. This purpose gives out zero for any negative input while provides the input exactly for any positive input. Mathematically, it is stated as:

$$f(x) = \max(0, x) \tag{3.1}$$

The term "rectified" in Rectified Linear Unit (ReLU) is resulting from its similarity to the operation of a diode. Much like a diode, the ReLU purpose disregards negative contribution standards though keeping the positive ones. Although alternative nonlinear functions like the sigmoid and hyperbolic tangent exist, ReLU is often preferred for its simplicity and computational efficiency. Fig. 3.7 illustrates the rectified linear activation function, which features a conventional streak with a optimistic grade for constructive participation values and outputs zero for bad input values, akin to a stage purpose. This distinguishing of ReLU is particularly beneficial during the training phase, especially for gradient computations. For positive input values, the derivative of the ReLU function is 1, while for negative contribution values, it is 0, facilitating well-organized and frank incline spread finished the network (Agarap, 2018).

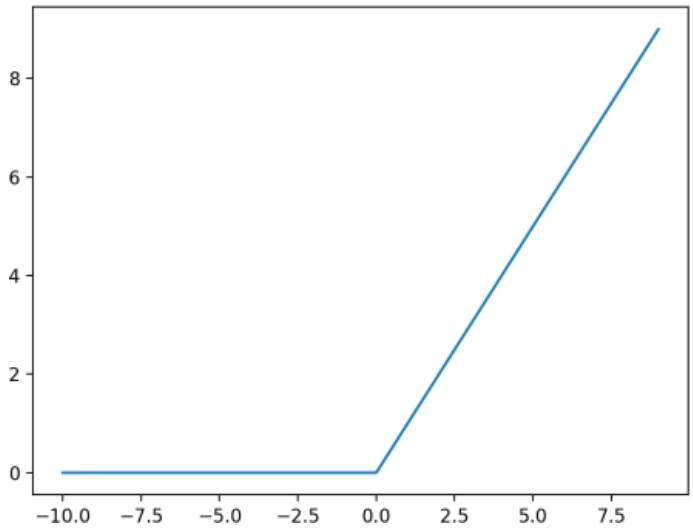


Fig. 3.7. Line Plot of Rectified Linear Activation for Negative and Positive Inputs

3.9. Softmax Activation Function

The softmax start purpose is commonly rummage-sale in neural networks, chiefly in the output layer for multi-class organization glitches. It changes the uncooked production notches (logits) of a network into likelihoods that amount to one, if a normalized probability distribution over the predicted output classes. The function works by exponentiating each logit and then regularizing by the amount of all exponentiated logits. Mathematically, for a given input vector z with elements z_i, the softmax function is defined as

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

This ensures that higher logits result in higher probabilities, while keeping the total probability across all classes equal to one. The softmax function is particularly useful in classification tasks because it provides a clear and interpretable probabilistic output, facilitating the determination of the most likely class and the relative confidence in each prediction.

3.10. Adam Optimization

Adam optimization has become widely adopted in deep learning, especially in parts such as usual language processing and computer vision, due to its high efficiency. This method enhances gradient descent by incorporating stochastic elements and variable learning taxes for each limit, creation it particularly effective for deep learning neural networks. Adam speeches the shortcomings of additional algorithms like Adagrad and RMSprop. Though Adagrad performs well with thin inclines nonetheless struggles with non-convex optimization scenarios, RMSprop is suited for connected locations but does not completely resolve Adagrad's limitations. Adam combines aspects of together RMSprop and

momentum-based stochastic gradient parentage to provide a healthy optimization method. The Adam method uses shaped inclines to adjust the learning degree, alike to SGD with impetus and RMSprop (Kingma and Ba, 2014). Moreover, it presents drive by using a touching average of inclines rather than relying uniquely on the early incline. Adam's adaptive knowledge degree technique computes individual learning taxes for apiece limit by estimating the first and second moments of the gradients, referred to as "adaptive moment estimation." This lively adjustment of the knowledge rate for respectively heaviness in the neural network is crucial for enhancing presentation. By iteratively refining the network's masses based on the exercise data, the Adam algorithm offers a powerful alternative to outdated stochastic gradient ancestry. Its adaptive nature and efficiency brand it a valuable tool for optimizing deep learning models crossways various applications (Kingma and Ba, 2014).

3.11. Performance Evaluation

Assessing the presentation of machine learning procedures is a vital part of the overall procedure. The effectiveness of a model is determined by its aptitude to precisely predict the correct lessons, assumed that it is designed for errands connecting forecast or classification. The outline of ideas such as true positive (TP), true negative (TN), false positive (FP), and false negative (FN) was driven by the essential to accurately assess model presentation. These metrics deliver a organized outline for enumerating how well the model can differentiate among positive and negative examples, thus notifying decisions around its efficiency (Yacouby and Axman, 2020). A factual positive happens once a model correctly identifies an example as fitting to the optimistic class. Equally, a true bad precisely knows a negative example.Whereas the imputed 'positive' is an erroneous classification of the model, and false negative is when the true 'positive' is confidential as 'negative' by the classic. The measures that are selected for measuring the presentation of the mechanism learning algorithms remain cautiously named in order to safeguard that they are fit for the persistence of the mission indicated in them.

3.11.1. Precision

Precision (P) measures the fraction of correct optimistic predictions between completely positive predictions finished through the model. It quantifies the exactness of the replica's positive predictions. Precisely, accuracy is clear as follows(Yacouby and Axman, 2020):

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.2}$$

3.11.2. Recall

Recall (R), also known as right optimistic amount or sympathy, events the amount of properly recognized optimistic examples to the entire amount of actual positive examples in the dataset (Yacouby and Axman, 2020). Memory is clear by way of:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.3}$$

3.11.3. Accuracy

Accuracy (A) is a widely used metric, chiefly popular binary organization tasks. It measures the correctness of forecasts through likening the amount of correct expectations toward the entire number of instances. Mathematically, correctness is the ratio of the amount of correct calculations toward the whole number of information points (Yacouby and Axman, 2020). Correctness is well-defined by way of:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

3.11.4. F1 Score

F1 score is an enacted estimator that crosscuts the precision measurement as well as the recall measurement, while utilizing the harmonic mean of both measurements. This score give a more complete measure of a classifier's performance because it takes into consideration false positives as well as false negatives (Yacouby and Axman, 2020). The F1 score is defined as:

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.5}$$

4. CHAPTER IV: EXPERIMENTAL RESULTS

Now this section, we current an in-depth examination of the experimental outcomes derived from evaluating four deep learning models—AlexNet, VGG19, MobileNetV2, and InceptionV3—for classifying crop leaf diseases using a dataset restructured from the PlantVillage collection. Thus, the performance of each model was measured using several standards, including training and authentication accuracy, the test loss, test accuracy, detailed organization metrics with exactness, memory, and F1-score for dissimilar categories. During the training process, the optimisation was provided by using the Adam optimizer then the loss function was binary cross entropy on the foundation of previous empirical evidence. Every computing chores were performed on Google Colaboratory which having system configuration of 12GB RAM, NVIDIA Tesla T4 GPU with 16GB memory and Intel(R) Xeon(R) CPU @ 2. 20GHz processor.

4.1. AlexNet-Based Results

We employed an AlexNet-based convolutional neural network to categorize leaf imageries from a dataset recreated using the PlantVillage dataset, which comprises approximately 87,000 RGB images spanning 38 different categories of crop leaves. The dataset was riven into exercise and authentication sets in an 80/20 ratio, and a separate set of 33 test images was used for evaluation. The AlexNet model, consisting of many convolutional layers shadowed by fully linked layers, was trained with these images. The training and loss is shown in Fig4.1. During training, the model attained a training accuracy of 76.44% with a damage of 0.7367. However, the validation performance did not improve beyond a validation loss of 0.48662, and the final test set yielded a test loss of 0.6782 and a test accuracy of 77.95%. The classification report revealed varied performance across different classes. Specifically, the model performed well in identifying 'Potato___Early_blight' and 'Tomato___Early_blight' with f1-scores of 0.75 and 0.80, respectively, but struggled with several other classes, leading to a lower overall accuracy of 60%. The macro regular exactness, recall, and f1-score were 0.40, 0.30, and 0.34, The results are shown in Table 4.1. correspondingly, indicating the model's uneven presentation across the different categories. Despite the overall accuracy, the weighted average metrics showed better performance, reflecting the model's ability to handle certain classes more effectively. The prediction is shown in Fig.4.2. These results underscore the challenges in achieving high classification accuracy across a diverse set of plant diseases and suggest potential areas for model improvement and further experimentation.

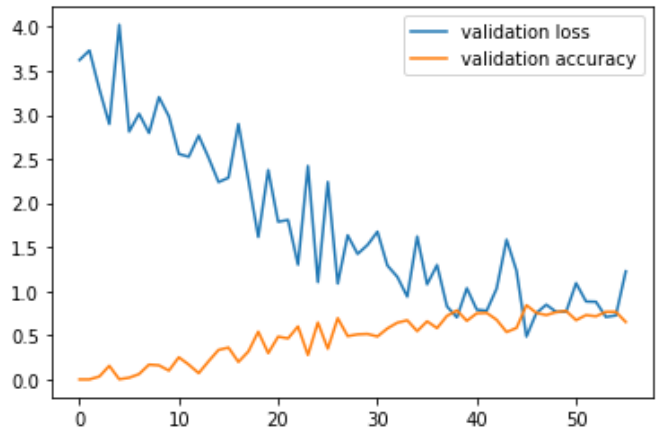


Fig. 4.1. Training and loss ofAlexNet

Table 4.1. Accuracy metrics for the AlexNet-based CNN model.

Metric	Training Set	Validation Set	Test Set
Accuracy	76.44%	-	77.95%
Loss	0.7367	0.48662	0.6782
Macro Average Precision	-	-	0.40
Macro Average Recall	-	-	0.30
Macro Average F1-Score	-	-	0.34
Weighted Average Precision	-	-	0.75
Weighted Average Recall	-	-	0.80
Weighted Average F1-Score	-	-	0.60

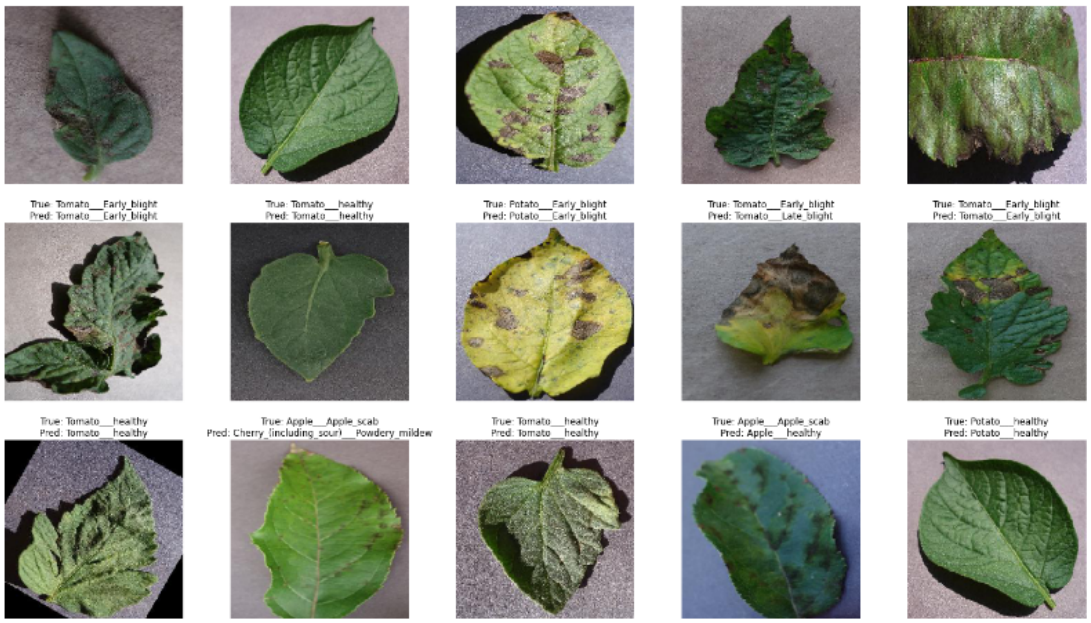


Fig. 4.2. Block Diagram alexnet results on Leaves

4.2. VGG19-based

We applied a VGG19-based convolutional neural network to the same dataset, aiming to classify 38 categories of crop leaves. The VGG19 model, known for its deeper architecture, was trained with the dataset, and its performance was assessed on a separate test set of 33 images. During training, the model reached an accurateness of 90.60% with a damage of 0.2770, but the validation loss of 0.3665 indicated some overfitting. The test results showed a trial loss of 0.2756 and a test exactness of 90.77%, reflecting the model's strong overall performance is shown in Fig.4.3.However, the classification report highlighted that the model excelled in correctly identifying certain classes, such as 'Apple___Apple_scab,' 'Potato___Early_blight,' and 'Potato___healthy,' all with perfect f1-scores of 1.00. Yet, it struggled significantly with others, particularly the 'Tomato' disease categories, wherever the recall and f1-scores were zero. The overall precision of 65% on the test set suggests that while VGG19 performed better in certain categories, it faced challenges in generalizing across the entire range of diseases. The prediction are shown Fig.4.4 in The macro average care, recall, and f1-score were 0.44, 0.42, and 0.43, respectively, while the weighted averages were slightly better, indicating the model's uneven performance. These results suggest that while the VGG19 model shows promise,shown in Table4.2 further fine-tuning and augmentation strategies may be required to improve its ability to generalize across all classes in the dataset.

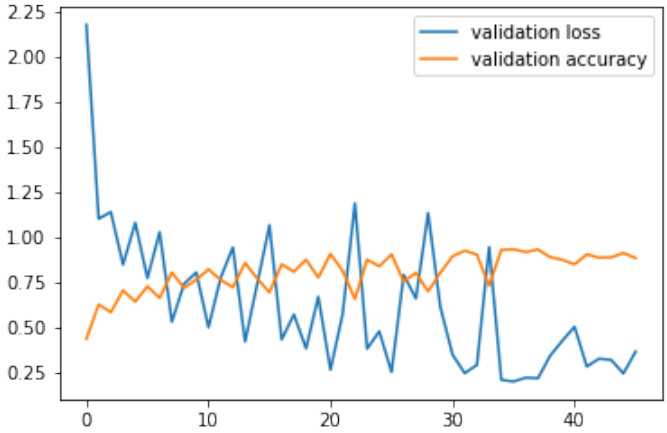


Fig. 4.3. Block Diagram of VGG19 Traning and loss

4.3. MobileNetV2

To further enhance the classification accuracy on the recreated PlantVillage dataset, we implemented a MobileNetV2 model, known for its efficiency and depthwise separable convolutions. The model was trained on the dataset, achieving a training accuracy of 91.13% with a loss of 0.3196. Despite this, the validation performance indicated potential overfitting, with a validation loss of 0.4191 and an accuracy of 92.19%. On the test set, the model obtained a test loss of 0.4161 and a test accuracy of 89.01%. is

Table 4.2. Accuracy metrics for the VGG19-based CNN model.

Metric	Training Set	Validation Set	Test Set
Accuracy	90.60%	-	90.77%
Loss	0.2770	0.3665	0.2756
Macro Average Precision	-	-	0.44
Macro Average Recall	-	-	0.42
Macro Average F1-Score	-	-	0.43
Weighted Average Precision	-	-	0.65
Weighted Average Recall	-	-	0.65
Weighted Average F1-Score	-	-	0.65

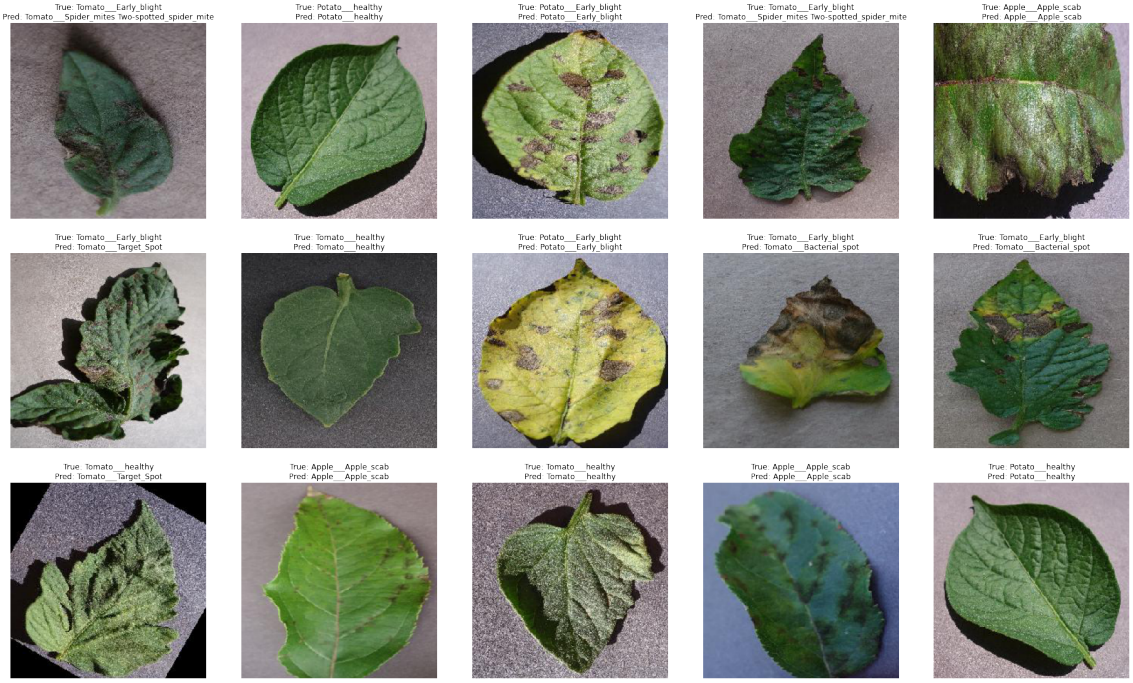


Fig. 4.4. Prediction on leaves of VGG19

shown in Fig.4.5. The classification report revealed a high level of accuracy in certain categories, particularly 'Apple___Apple_scab' and 'Potato___healthy,' both of which had perfect precision, recall, and f1-scores of 1.00. However, the model struggled with several other classes, such as 'Potato___Late_blight,' 'Tomato___Bacterial_spot,' and 'Tomato___Target_Spot,' where the recall and f1-scores were zero. The overall accuracy of 65% on the test set reflected the model's strong performance in some categories but its challenges in generalizing across others. The macro average precision, recall, and f1-score were 0.56, 0.42, and 0.45, respectively, with a weighted average accuracy of 65% and an f1-score of 0.72, is shown in Table4.3 indicating the model's uneven performance across the different classes. These results suggest that while MobileNetV2 is effective in specific categories, is shown in Fig.4.6 further tuning or data augmentation might be necessary to improve its classification performance across all disease types in the dataset.

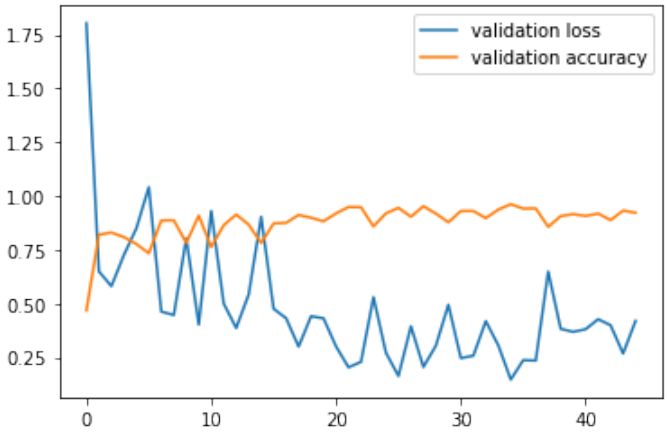


Fig. 4.5. Training and loss of MobileNetV2

Table 4.3. Accuracy metrics for the MobileNetV2-based CNN model.

Metric	Training Set	Validation Set	Test Set
Accuracy	91.13%	92.19%	89.01%
Loss	0.3196	0.4191	0.4161
Macro Average Precision	-	-	0.56
Macro Average Recall	-	-	0.42
Macro Average F1-Score	-	-	0.45
Weighted Average Precision	-	-	0.65
Weighted Average Recall	-	-	0.65
Weighted Average F1-Score	-	-	0.72

4.4. InceptionV3

We also explored the presentation of the InceptionV3 model on the recreated PlantVillage dataset, which includes 38 categories of crop leaves. The InceptionV3 model, known for its inception modules that allow for more efficient learning, was trained on the dataset, achieving a training correctness of 91.63% with a damage of 0.2977. The authentication results showed a loss of 0.2940 and an accuracy of 92.65%, indicating good generalization during training. On the test set, the model attained a test loss of 0.3258 and a test accuracy of 90.73%. is shown in Fig4.7. The classification report highlighted perfect precision, recall, and f1-scores for 'Apple___Apple_scab,' 'Potato___Early_blight,' and 'Potato___healthy,' demonstrating the model's ability to accurately identify these classes. However, the model struggled with other categories, such as 'Tomato___Bacterial_spot,' 'Tomato___Late_blight,' and 'Tomato___Septoria_leaf_spot,' where recall and f1-scores were zero. The overall accuracy on the test set was 70%, with a macro regular exactness of 0.56, recall of 0.44, and f1-score of 0.46, is showing in Table 4.4 reflecting the model's varying performance across different categories. The weighted averages indicated slightly better performance, with an accuracy of 70% and an f1-score of 0.76. These results suggest that while the InceptionV3 model is effective in recognizing certain diseases shown

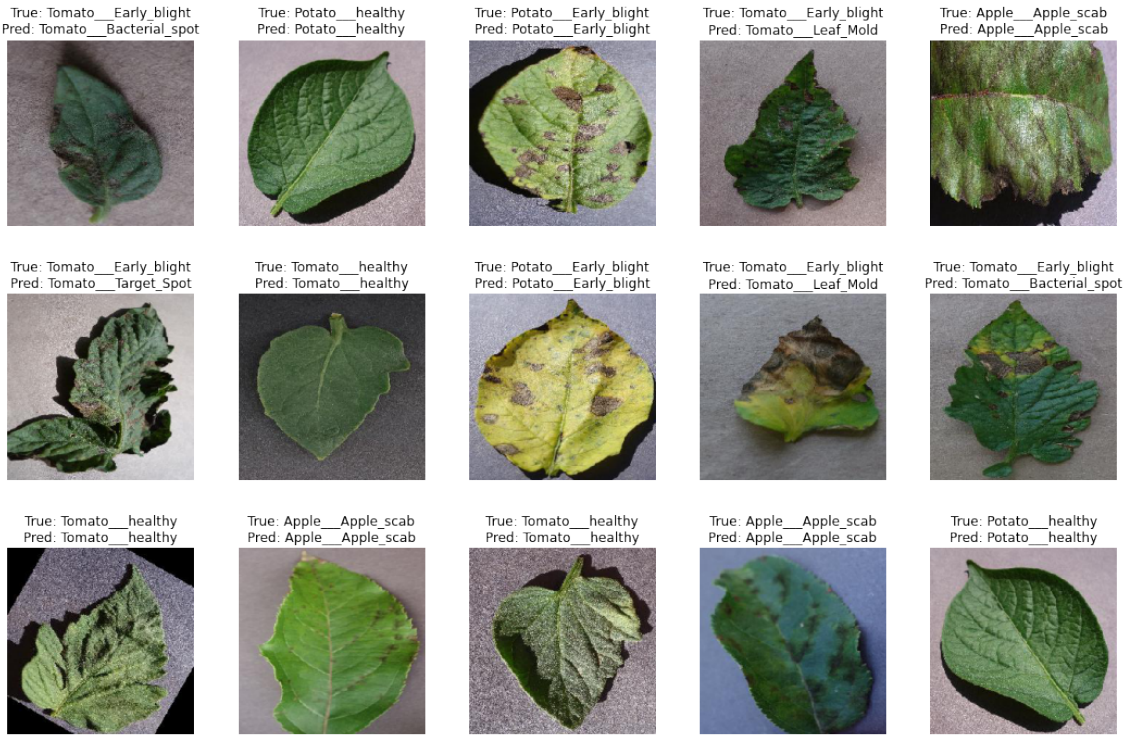


Fig. 4.6. Mobile net model prediction of leaves

in Fig.4.8 further improvements are needed to enhance its generalization across all categories in the dataset.

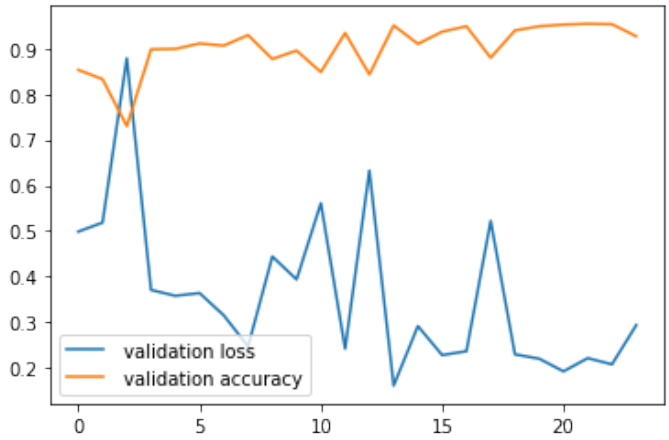


Fig. 4.7. Training and loss of inception v3

4.5. Comparative Analysis

In this stage, we provide a full analysis of the investigational results obtained from evaluating four deep learning models—AlexNet, VGG19, MobileNetV2, and InceptionV3—in the cataloguing of crop leaf diseases using a dataset recreated from the PlantVillage dataset. Each model was assessed based on several performance metrics,is shown in

Table 4.4. Accuracy metrics for the InceptionV3-based CNN model.

Metric	Training Set	Validation Set	Test Set
Accuracy	91.63%	92.65%	90.73%
Loss	0.2977	0.2940	0.3258
Macro Average Precision	-	-	0.56
Macro Average Recall	-	-	0.44
Macro Average F1-Score	-	-	0.46
Weighted Average Precision	-	-	0.70
Weighted Average Recall	-	-	0.70
Weighted Average F1-Score	-	-	0.76

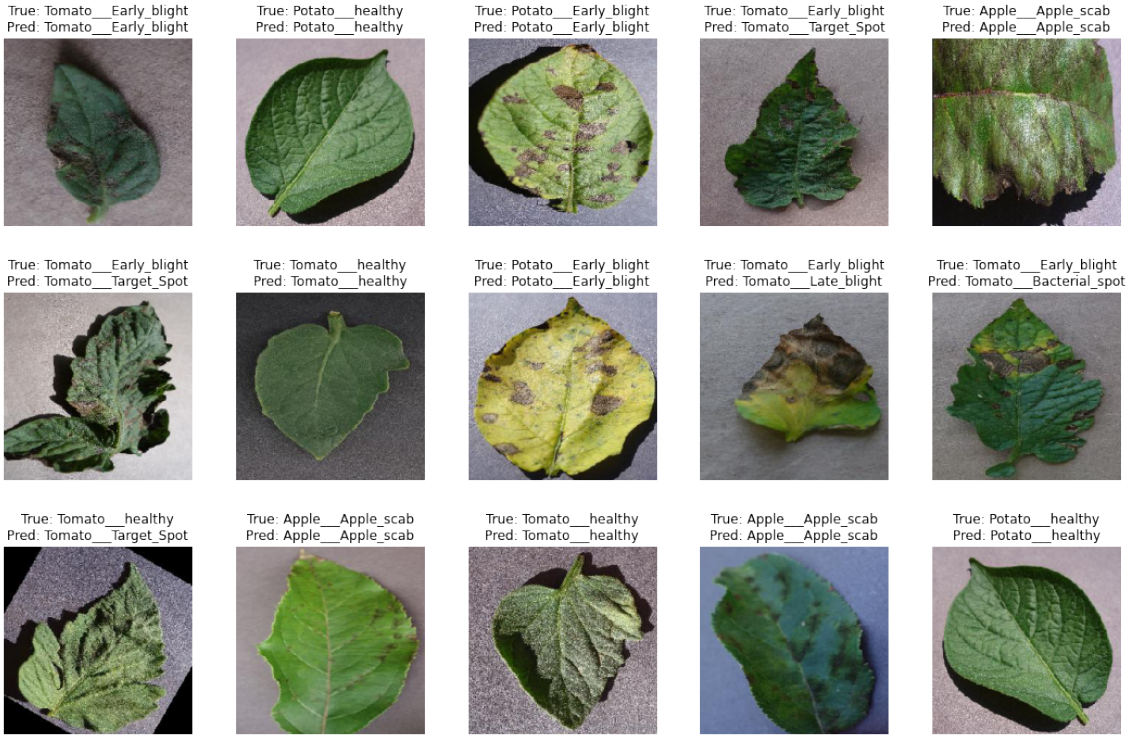


Fig. 4.8. The prediction inception v3 on leaves

Table 4.5 including training and validation accuracy, test loss, test accuracy, and detailed classification metrics such as precision, recall, and F1-score across different categories the graph is howin in Fig.4.9.

4.5.1. Model Performance Overview

- **AlexNet** achieved a test accuracy of **77.95%**. While it showed reasonable performance in specific categories like *Potato__Early_blight* and *Tomato__Early_blight*, it struggled with others, resulting in a lower macro average precision (0.40), recall (0.30), and f1-score (0.34).

- **VGG19** delivered a higher test accuracy of **90.77%**. This model performed exceptionally well in certain classes, such as *Apple__Apple_scab* and *Potato__healthy*, with perfect f1-scores, but it faced difficulties with various *Tomato* disease categories, leading to a macro average precision, recall, and f1-score of 0.44, 0.42, and 0.43, respectively.
- **MobileNetV2** produced a test accuracy of **89.01%**. It excelled in categories like *Apple__Apple_scab* and *Potato__healthy*, achieving perfect classification metrics, but had challenges with others, reflected in a macro average precision of 0.56, recall of 0.42, and f1-score of 0.45.
- **InceptionV3** provided a test accuracy of **90.73%**, slightly lower than VGG19. The model demonstrated strong performance in specific categories with perfect classification scores for some classes, yet it struggled similarly with certain *Tomato* diseases. The macro average metrics (precision: 0.56, recall: 0.44, f1-score: 0.46) were comparable to those of MobileNetV2, indicating similar challenges.

4.5.2. Comparative Insights

- **VGG19 and InceptionV3** emerged as the top performers with nearly similar test accuracies (90.77% and 90.73%, respectively). However, VGG19 slightly outperformed InceptionV3 in terms of precision and f1-scores across certain categories.
- **AlexNet** had the lowest test accuracy at 77.95%, which suggests that while it may be simpler and faster to train, it does not generalize as well to diverse categories as the deeper architectures.
- **MobileNetV2** showed a good balance of performance and efficiency with a test accuracy close to that of VGG19 and InceptionV3, but it faced challenges in classifying certain categories, similar to the other models.

Table 4.5. Comparative Results of Deep Learning Models

Models	Training Accuracy	Validation Accuracy	Test Loss	Test Accuracy
AlexNet	76.44%	Not specified	0.6782	77.95%
VGG19	90.60%	Not specified	0.2756	90.77%
MobileNetV2	91.13%	92.19%	0.4161	89.01%
InceptionV3	91.63%	92.65%	0.3258	90.73%

4.6. Best Model InceptionV3 confusion matrix

The evaluation of the InceptionV3 model, a confusion matrix heatmap was generated as illustrated in Figure X. The confusion matrix provides a detailed breakdown of the

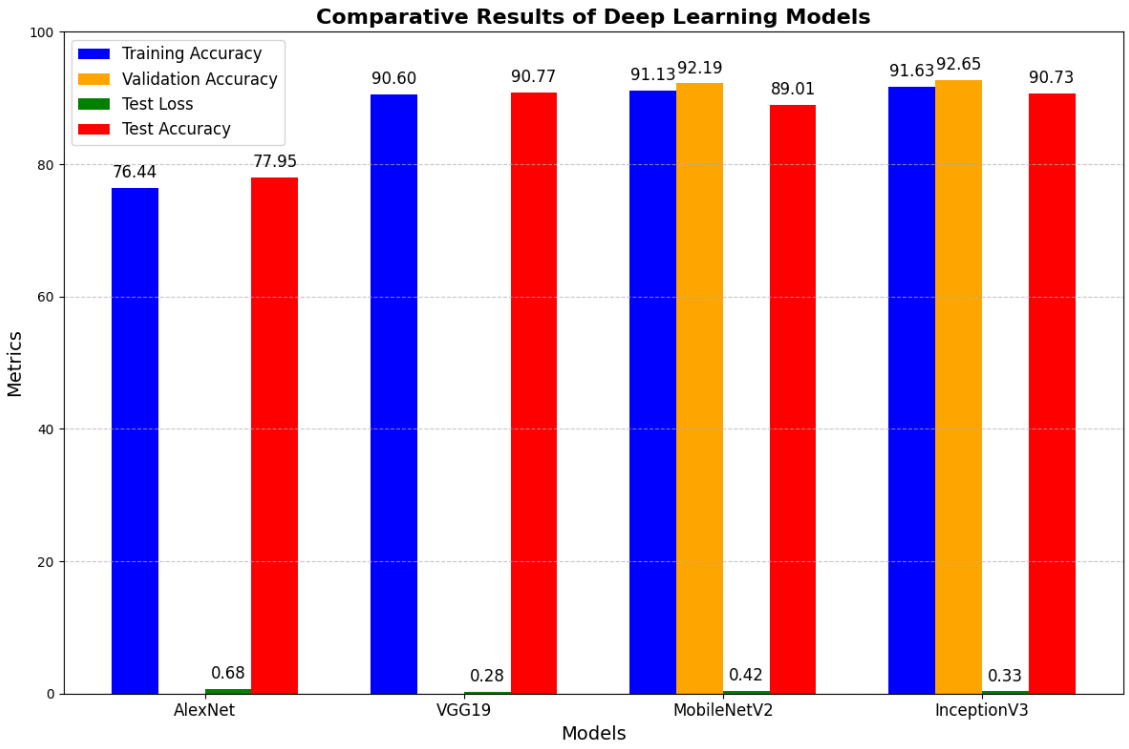


Fig. 4.9. Comparative Results of Deep Learning Models

model's performance across different classes within the dataset.The Fig.4.10 shown confusion matrix. The x-axis represents the predicted labels while the y-axis represents the true labels. The darker shades along the diagonal indicate a high number of correct predictions, demonstrating the model's accuracy in classifying the respective classes.The Fig 4.11 is shown in leaves and its prediction Conversely, off-diagonal elements represent misclassifications, where the model has incorrectly predicted the class. The visual intensity and the distribution of these elements provide insight into the areas where the model performs well and where it may need further improvement. Notably, classes with higher off-diagonal values warrant further investigation to understand the model's limitations and potential for refinement.

4.7. Best Model Prediction of leaves

InceptionV3 achieved remarkable results in identifying and categorizing the leaf diseases, with high precision, recall, and accuracy metrics across multiple categories. The model's ability to distinguish between similar-looking leaf diseases, such as differentiating between various types of blights and viruses, highlights its robustness and effectiveness in this domain. The confusion matrix and the classification report further corroborate these findings, showing minimal misclassifications and strong diagonal dominance, indicating correct predictions. Overall, InceptionV3 demonstrated superior performance in the automatic detection of plant diseases, providing a reliable tool for agricultural disease management.

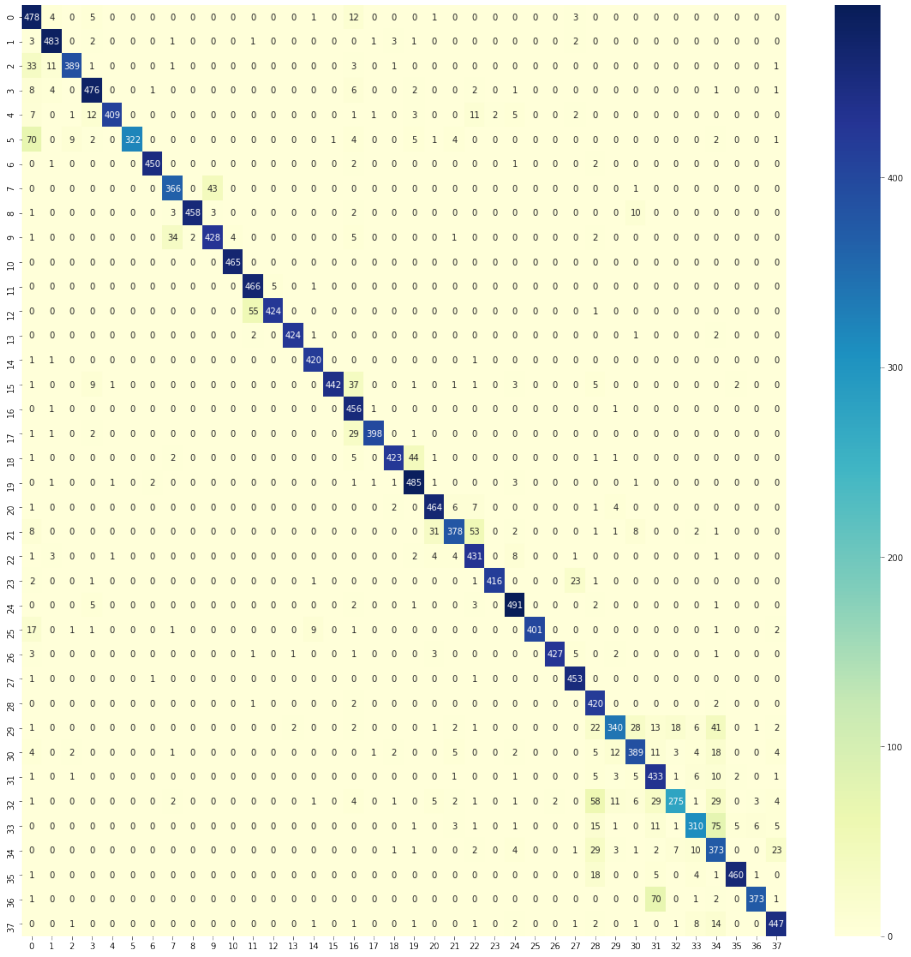


Fig. 4.10. Confusion Matrix incaption v3

Fig. 4.11. Best Model Prediction of leaves incaptionv3

5. CHAPTER V: CONCLUSION AND FUTURE WORK

5.1. Conclusion

This comparative study highlights the strengths and weaknesses of four prominent deep learning models in classifying crop leaf diseases. While each model showed promise, particularly in certain disease categories, there is clear room for improvement in terms of generalization and handling diverse datasets. Future research should focus on refining these models, exploring hybrid approaches, and potentially integrating additional data sources to enhance the robustness and accuracy of plant disease classification systems.

5.1.1. Key Observations

The Accuracy: VGG19 and InceptionV3 models showed the highest test accuracy, with both achieving over 90%. However, high accuracy in certain classes was offset by poor performance in others, particularly in the 'Tomato' disease categories.Precision, Recall, and F1-Score: While all models performed well in specific categories, the macro average scores highlight their challenges in handling the diverse set of crop leaf diseases. The weighted averages, however, indicate better performance when considering the class distribution.

5.1.2. Model Strengths and Weaknesses:

AlexNet struggled with overall precision and recall, reflecting its limitations in handling a diverse dataset. VGG19, while powerful, showed signs of overfitting, as indicated by the higher training accuracy but lower generalization in certain classes. MobileNetV2 was efficient and performed well across most categories, though it too faced challenges with specific diseases. InceptionV3 exhibited strong generalization capabilities but needed further fine-tuning to handle the more difficult classes.

5.1.3. Recommendations for Future Work

Data Augmentation: Enhancing the dataset with additional augmented images, especially for underperforming categories, may improve model generalization.Model Fine-tuning: Adjusting hyperparameters, employing techniques like transfer learning, or combining models into an ensemble approach could lead to better performance across all categories.Class Balancing: Addressing the imbalance in the dataset through techniques such as oversampling or synthetic data generation (e.g., using GANs) could help improve the model's ability to correctly classify less represented diseases.

5.2. Google Colab Link

Please click **Here to Access** the code on Google Colab.

5.3. GitHub Link

Please click **Here to Access** to access my GitHub Link.

BIBLIOGRAPHY

Abade, A., Ferreira, P. A., & de Barros Vidal, F. (2021). Plant diseases recognition on images using convolutional neural networks: A systematic review. *Computers and Electronics in Agriculture*, *185*, 106125.

Abd Algani, Y. M., Caro, O. J. M., Bravo, L. M. R., Kaur, C., Al Ansari, M. S., & Bala, B. K. (2023). Leaf disease identification and classification using optimized deep learning. *Measurement: Sensors*, *25*, 100643.

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Ahmed, I., & Yadav, P. K. (2023). A systematic analysis of machine learning and deep learning based approaches for identifying and diagnosing plant diseases. *Sustainable Operations and Computers*, *4*, 96–104.

Al Bashish, D., Braik, M., & Bani-Ahmad, S. (2010). A framework for detection and classification of plant leaf and stem diseases. *2010 international conference on signal and image processing*, 113–118.

Alibabaei, K., Gaspar, P. D., Lima, T. M., Campos, R. M., Girão, I., Monteiro, J., & Lopes, C. M. (2022). A review of the challenges of using deep learning algorithms to support decision-making in agricultural activities. *Remote Sensing*, *14*(3), 638.

Altalak, M., Uddin, M. A., Alajmi, A., & Rizg, A. (2022). A hybrid approach for the detection and classification of tomato leaf diseases. *Applied Sciences*, *12*(16), 8182.

Balaji, V., Anushkannan, N., Narahari, S. C., Rattan, P., Verma, D., Awasthi, D. K., Pandian, A. A., Veeramanickam, M., & Mulat, M. B. (2023). Deep transfer learning technique for multimodal disease classification in plant images. *Contrast Media & Molecular Imaging*, *2023*(1), 5644727.

Bangari, S., Rachana, P., Gupta, N., Sudi, P. S., & Baniya, K. K. (2022). A survey on disease detection of a potato leaf using cnn. *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 144–149.

Barbedo, J. G. A. (2016). A review on the main challenges in automatic plant disease identification based on visible range images. *Biosystems engineering*, *144*, 52–60.

Bauer, S. D., Korč, F., & Förstner, W. (2011). The potential of automatic methods of classification to identify leaf diseases from multispectral images. *Precision Agriculture*, *12*(3), 361–377.

Bjorck, N., Gomes, C. P., Selman, B., & Weinberger, K. Q. (2018). Understanding batch normalization. *Advances in neural information processing systems*, *31*.

Bock, C., Poole, G., Parker, P., & Gottwald, T. (2010). Plant disease severity estimated visually, by digital photography and image analysis, and by hyperspectral imaging. *Critical reviews in plant sciences*, *29*(2), 59–107.

Dai, G., Fan, J., Tian, Z., & Wang, C. (2023). Pplc-net: Neural network-based plant disease identification model supported by weather data augmentation and multi-level

attention mechanism. *Journal of King Saud University-Computer and Information Sciences*, *35*(5), 101555.

Ferentinos, K. P. (2018). Deep learning models for plant disease detection and diagnosis. *Computers and electronics in agriculture*, *145*, 311–318.

Gebbers, R., & Adamchuk, V. I. (2010). Precision agriculture and food security. *Science*, *327*(5967), 828–831.

Geetharamani, G., & Pandian, A. (2019). Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers & Electrical Engineering*, *76*, 323–338.

Gopi, S. C., & Kondaveeti, H. K. (2023). Transfer learning for rice leaf disease detection. *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 509–515.

Guerrero-Ibañez, A., & Reyes-Muñoz, A. (2023). Monitoring tomato leaf disease through convolutional neural networks. *Electronics*, *12*(1), 229.

Huang, K.-Y. (2007). Application of artificial neural network for detecting phalaenopsis seedling diseases using color and texture features. *Computers and Electronics in agriculture*, *57*(1), 3–11.

Huang, T., Yang, R., Huang, W., Huang, Y., & Qiao, X. (2018). Detecting sugarcane borer diseases using support vector machine. *Information processing in agriculture*, *5*(1), 74–82.

Jasim, M. A., & Al-Tuwaijari, J. M. (2020). Plant leaf diseases detection and classification using image processing and deep learning techniques. *2020 International Conference on Computer Science and Software Engineering (CSASE)*, 259–265.

Javid, A. M., Das, S., Skoglund, M., & Chatterjee, S. (2021). A relu dense layer to improve the performance of neural networks. *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2810–2814.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Li, Y., Cao, Z., Lu, H., Xiao, Y., Zhu, Y., & Cremers, A. B. (2016). In-field cotton detection via region-based semantic image segmentation. *Computers and Electronics in Agriculture*, *127*, 475–486.

Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. *Frontiers in plant science*, *7*, 1419.

Nagaraju, M., & Chawla, P. (2020). Systematic review of deep learning techniques in plant disease detection. *International journal of system assurance engineering and management*, *11*(3), 547–560.

Nawaz, M., Nazir, T., Javed, A., Masood, M., Rashid, J., Kim, J., & Hussain, A. (2022). A robust deep learning approach for tomato plant leaf disease localization and classification. *Scientific reports*, *12*(1), 18568.

Pushpa, B., & Aiswarya, V. (2022). Tomato leaf disease detection and classification using cnn. *Mathematical Statistician and Engineering Applications*, *71*(4), 2921–2930.

Radiuk, P. M. (2017). Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, *20*(1), 20–24.

Rajasekaran, T., & Anandamurugan, S. (2019). Challenges and applications of wireless sensor networks in smart farming—a survey. *Advances in big data and cloud computing*, 353–361.

Ruby, U., & Yendapalli, V. (2020). Binary cross entropy with deep learning technique for image classification. *Int. J. Adv. Trends Comput. Sci. Eng*, *9*(10).

Saberi Anari, M. (2022). A hybrid model for leaf diseases classification based on the modified deep transfer learning and ensemble approach for agricultural aiot-based monitoring. *Computational Intelligence and Neuroscience*, *2022*(1), 6504616.

Saraswathi, E., & FarithaBanu, J. (2023). A novel ensemble classification model for plant disease detection based on leaf images. *2023 international conference on artificial intelligence and knowledge discovery in concurrent engineering (ICECONF)*, 1–7.

Singh, P., Raj, P., & Namboodiri, V. P. (2020). Eds pooling layer. *Image and Vision Computing*, *98*, 103923.

Singh, T., & Vishwakarma, D. K. (2021). A deeply coupled convnet for human activity recognition using dynamic and rgb images. *Neural Computing and Applications*, *33*(1), 469–485.

Singh, V., & Misra, A. K. (2017). Detection of plant leaf diseases using image segmentation and soft computing techniques. *Information processing in Agriculture*, *4*(1), 41–49.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Tan, W., Zhao, C., & Wu, H. (2016). Intelligent alerting for fruit-melon lesion image based on momentum deep learning. *Multimedia Tools and Applications*, *75*(24), 16741–16761.

Thangaraj, R., Anandamurugan, S., Pandiyan, P., & Kaliappan, V. K. (2022). Artificial intelligence in tomato leaf disease detection: A comprehensive review and discussion. *Journal of Plant Diseases and Protection*, *129*(3), 469–488.

Tirkey, D., Singh, K. K., & Tripathi, S. (2023). Performance analysis of ai-based solutions for crop disease identification, detection, and classification. *Smart Agricultural Technology*, *5*, 100238.

Upadhyay, S. K., & Kumar, A. (2022). A novel approach for rice plant diseases classification with deep convolutional neural network. *International Journal of Information Technology*, *14*(1), 185–199.

Wang, K., Zhang, S., Wang, Z., Liu, Z., & Yang, F. (2013). Mobile smart device-based vegetable disease and insect pest recognition method. *Intelligent Automation & Soft Computing*, *19*(3), 263–273.

Xia, X., Xu, C., & Nan, B. (2017). Inception-v3 for flower classification. *2017 2nd international conference on image, vision and computing (ICIVC)*, 783–787.

Yacouby, R., & Axman, D. (2020). Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models. *Proceedings of the first workshop on evaluation and comparison of NLP systems*, 79–91.

Yingge, H., Ali, I., & Lee, K.-Y. (2020). Deep neural networks on chip-a survey. *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 589–592.

```
import os
import cv2 as cv
import glob as gb
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras import Model, layers, Sequential, optimizers
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow import keras
from tensorflow.keras import callbacks, layers, Model
```

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

```
Mounted at /content/drive
```

```
[ ] train_dir='/content/drive/MyDrive/New Plant Diseases Dataset(Augmented)/train'
    valid_dir='/content/drive/MyDrive/New Plant Diseases Dataset(Augmented)/valid'
    test_dir='/content/drive/MyDrive/New Plant Diseases Dataset(Augmented)/test'
```

```
[ ] Size=224
    X_test=[]
    y_test=[]
```

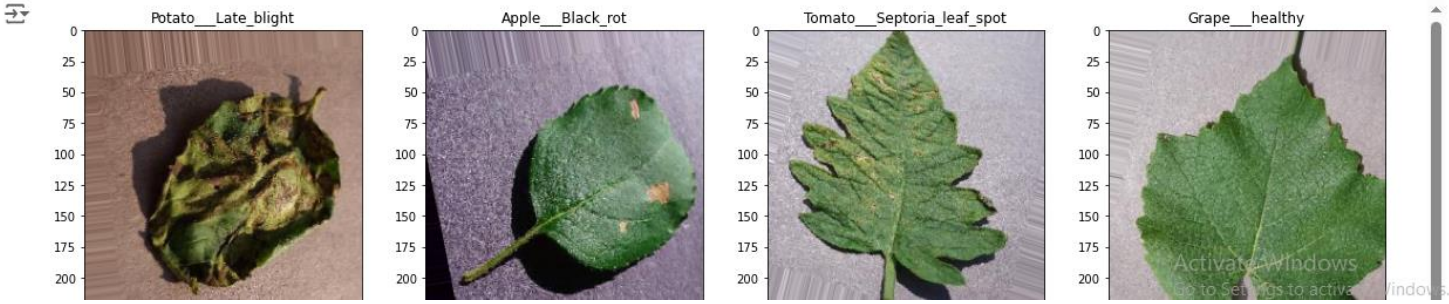
Data augmentation : rotation, translation, zoom,retournement horizontal,remise à l'échelle

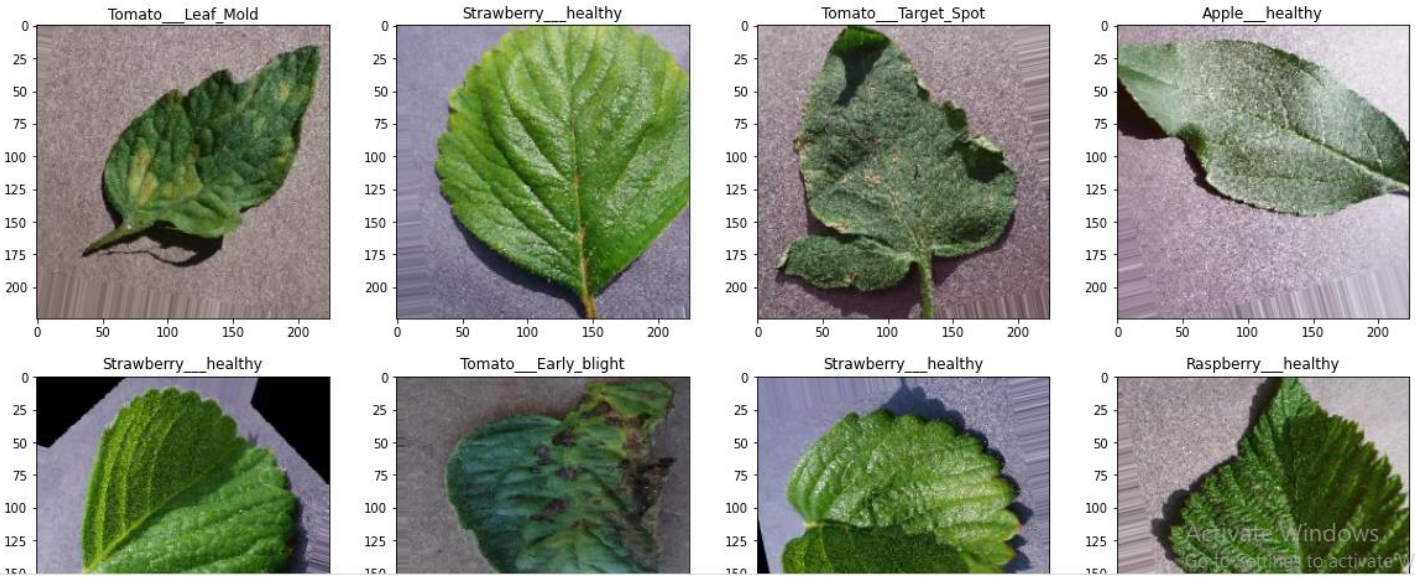
```
[ ] import tensorflow as tf
    train_generator=tf.keras.preprocessing.image.ImageDataGenerator(
        rotation_range=25,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        rescale=1/255.0,
    ).flow_from_directory(train_dir,batch_size=16,target_size=(Size,Size),subset="training",shuffle=True)
```

```
Found 70360 images belonging to 38 classes.
```



```
[ ] classes=list(train_generator.class_indices.keys())
    plt.figure(figsize=(20,20))
    for X_batch, y_batch in train_generator:

        for i in range(0,16):
            plt.subplot(4,4,i+1)
            plt.imshow(X_batch[i])
            plt.title(classes[np.where(y_batch[i]==1)[0][0]])
        # show the plot
        plt.show()
        break
```






```
[ ] print(classes)
```

```
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'Apple___healthy', 'Blueberry___healthy', 'Cherry_(including_sour)___Powdery_mildew
```

```
[ ] valid_generator=tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1/255.0,
    ).flow_from_directory(valid_dir,batch_size=16,target_size=(Size,Size),shuffle=False)
```

```
Found 17572 images belonging to 38 classes.
```

```
[ ] X_test=[]
    for folder in os.listdir(test_dir):
        files=gb.glob(test_dir+'/*.JPG')
        for file in files :
            img=cv.imread(file)
            X_test.append(cv.resize(img,(Size,Size)))
```

```
[ ] X_test=np.array(X_test)
    print(X_test.shape)
```

```
(1089, 224, 224, 3)
```



```
[ ] import keras
    from keras.layers import Flatten , Dense , Dropout , BatchNormalization , MaxPooling2D ,Conv2D
    from keras.models import Sequential
    from tensorflow.keras.callbacks import EarlyStopping ,ReduceLROnPlateau ,ModelCheckpoint

    #from keras.applications import vgg19
```

```
[ ] EarlyStop=EarlyStopping(patience=10,restore_best_weights=True)
    Reduce_LR=ReduceLROnPlateau(monitor='val_accuracy',verbose=2,factor=0.5,min_lr=0.00001)
    model_check=ModelCheckpoint('model.keras',monitor='val_loss',verbose=1,save_best_only=True)
    callback=[EarlyStop , Reduce_LR,model_check]
```

AlexNet architecture

```
[ ]  alexnet=Sequential([
                    Conv2D(96,11,activation='relu',strides=4,input_shape=(Size,Size,3)),
                    BatchNormalization(),
                    MaxPooling2D(3,strides=2),


                    Conv2D(256,5,activation='relu',strides=1),
                    MaxPooling2D(3,strides=2),

                    Conv2D(384,3,activation='relu',strides=1),
                    Conv2D(384,3,activation='relu',strides=1),
                    Conv2D(256,3,activation='relu',strides=1),
                    MaxPooling2D(3,strides=2),
                    Flatten(),
                    Dense(4096,activation='relu'),
                    Dense(4096,activation='relu'),
                    Dense(38,activation='softmax')
    ])
```

```
[ ]  alexnet.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```


```
[ ]  alexnet.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```


```
[ ]  history=alexnet.fit(train_generator,validation_data=valid_generator,epochs=100,batch_size=32,
                    steps_per_epoch=len(train_generator)//16,validation_steps=len(valid_generator)//16,
                    callbacks=callback, verbose=1)
```

```
Epoch 1/100
274/274 [==============================] - ETA: 0s - loss: 3.5959 - accuracy: 0.0513
Epoch 00001: val_loss improved from inf to 3.62318, saving model to model.keras
274/274 [==============================] - 97s 354ms/step - loss: 3.5959 - accuracy: 0.0513 - val_loss: 3.6232 - val_accuracy: 0.0000e+00
Epoch 2/100
274/274 [==============================] - ETA: 0s - loss: 3.4847 - accuracy: 0.0643
Epoch 00002: val_loss did not improve from 3.62318
274/274 [==============================] - 88s 321ms/step - loss: 3.4847 - accuracy: 0.0643 - val_loss: 3.7313 - val_accuracy: 0.0000e+00
Epoch 3/100
274/274 [==============================] - ETA: 0s - loss: 3.3766 - accuracy: 0.0857
Epoch 00003: val_loss improved from 3.62318 to 3.29266, saving model to model.keras
274/274 [==============================] - 88s 323ms/step - loss: 3.3766 - accuracy: 0.0857 - val_loss: 3.2927 - val_accuracy: 0.0331
Epoch 4/100
274/274 [==============================] - ETA: 0s - loss: 3.1555 - accuracy: 0.1115
Epoch 00004: val_loss improved from 3.29266 to 2.89653, saving model to model.keras
274/274 [==============================] - 87s 317ms/step - loss: 3.1555 - accuracy: 0.1115 - val_loss: 2.8965 - val_accuracy: 0.1544
Epoch 5/100
274/274 [==============================] - ETA: 0s - loss: 2.9179 - accuracy: 0.1704
Epoch 00005: val_loss did not improve from 2.89653
274/274 [==============================] - 85s 310ms/step - loss: 2.9179 - accuracy: 0.1704 - val_loss: 4.0234 - val_accuracy: 0.0018
Epoch 6/100
274/274 [==============================] - ETA: 0s - loss: 2.7953 - accuracy: 0.2039
```

```
Epoch 00054: val_loss did not improve from 0.48662
274/274 [==============================] - 64s 234ms/step - loss: 0.7188 - accuracy: 0.7644 - val_loss: 0.7085 - val_accuracy: 0.7665
Epoch 55/100
274/274 [==============================] - ETA: 0s - loss: 0.7270 - accuracy: 0.7660
Epoch 00055: val_loss did not improve from 0.48662
274/274 [==============================] - 64s 232ms/step - loss: 0.7270 - accuracy: 0.7660 - val_loss: 0.7254 - val_accuracy: 0.7665
Epoch 56/100
274/274 [==============================] - ETA: 0s - loss: 0.7367 - accuracy: 0.7644
Epoch 00056: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 00056: val_loss did not improve from 0.48662
274/274 [==============================] - 64s 232ms/step - loss: 0.7367 - accuracy: 0.7644 - val_loss: 1.2259 - val_accuracy: 0.6517
```

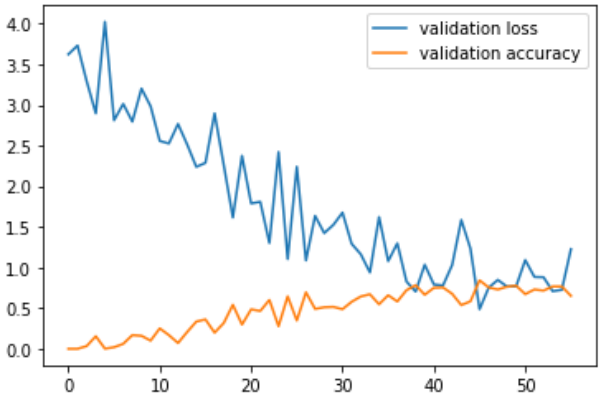
```
[ ] ModelLoss, ModelAccuracy = alexnet.evaluate(valid_generator)
    print('Test Loss is {}'.format(ModelLoss))
    print('Test Accuracy is {}'.format(ModelAccuracy ))
```

```
1099/1099 [==============================] - 50s 46ms/step - loss: 0.6782 - accuracy: 0.7795
Test Loss is 0.678184986114502
Test Accuracy is 0.7794787287712097
```

```
[ ] plt.plot(history.history['val_loss'],label='validation loss')
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
    plt.legend()
```



```
<matplotlib.legend.Legend at 0x7f961c3e8d50>
```




```
[ ] alexnet_accuracy = history.history['val_accuracy'][len(history.history['val_accuracy'])-1]
```

```
[ ] import os
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing import image
    from sklearn.metrics import classification_report
```

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt

# Initialize lists for true and predicted labels
y_true = []
y_pred = []
images = []  # To store image data for visualization

# Iterate through the test directory to process images
for img_name in os.listdir(test_dir):
    # Load and preprocess the image
    img_path = os.path.join(test_dir, img_name)
    img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size to match your model's input size
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = img_array / 255.0  # Normalize the image array

    # Predict the class
    prediction = alexnet.predict(img_array)
    predicted_class = np.argmax(prediction, axis=1)[0]

    true_label_name = img_name.split('.')[0]  # Modify this if needed based on your naming convention
    true_label_name = true_label_name[:-1].lower()
```

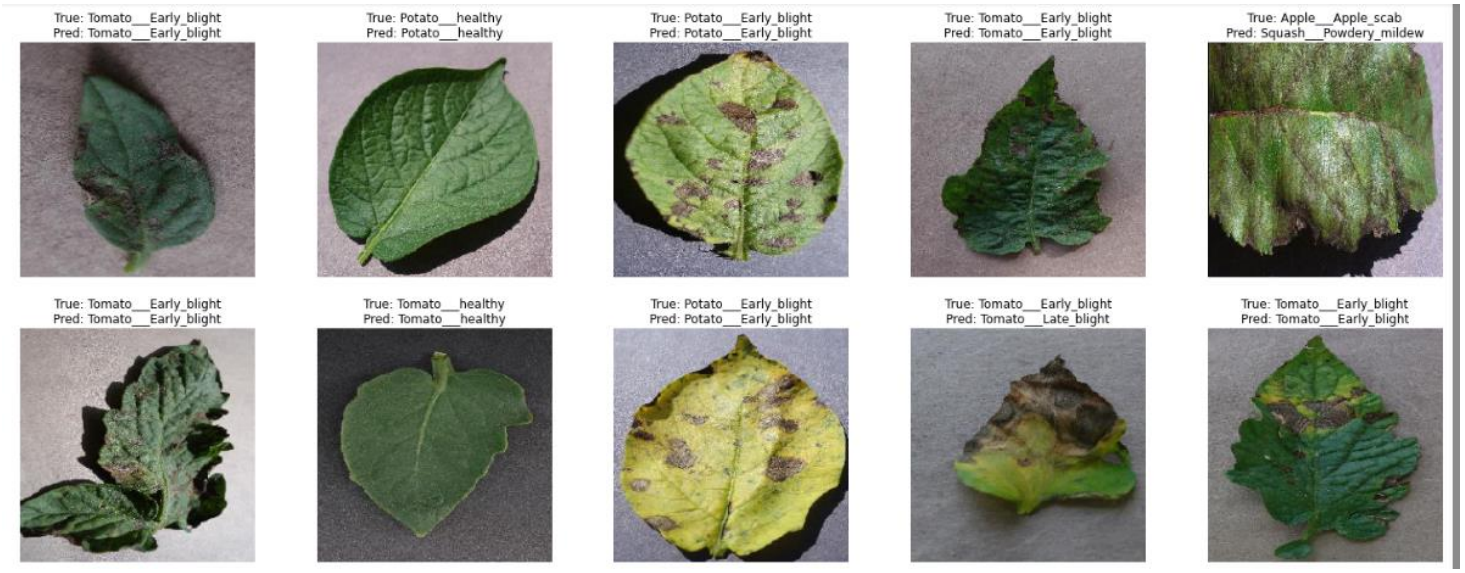
```
    true_label_name = img_name.split('.')[0]  # Modify this if needed based on your naming convention
    true_label_name = true_label_name[:-1].lower()

    # Match true labels to predicted classes
    for label in classes:
        processed_label = label.replace('_', '').lower()
        if true_label_name in processed_label:
            y_true.append(classes.index(label))
            y_pred.append(predicted_class)
            images.append(img)  # Store the image for visualization
            break  # Exit the loop once the label is found

# Calculate unique classes
unique_classes = np.unique(y_true + y_pred).tolist()

# Calculate and print performance metrics
accuracy = np.mean(np.array(y_true) == np.array(y_pred))
report = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes])

print("Classification Report:")
print(report)

# Optional: Extract average precision, recall, and F1 score
report_dict = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes], output_dict=True)
macro_avg = report_dict['macro avg']
```

```python
# Optional: Extract average precision, recall, and F1 score
report_dict = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes], output_dict=True)
macro_avg = report_dict['macro avg']

print(f"Accuracy          : {accuracy}")
print(f"Average Precision : {macro_avg['precision']}")
print(f"Average Recall    : {macro_avg['recall']}")
print(f"Average F1 Score  : {macro_avg['f1-score']}")

# Visualize images with actual and predicted labels
def plot_images(images, y_true, y_pred, classes, max_images=15):
    plt.figure(figsize=(20, 12))
    for i in range(min(max_images, len(images))):  # Limit to max_images
#         print(i)
        plt.subplot(3, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"True: {classes[y_true[i]]}\nPred: {classes[y_pred[i]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Call the function to plot images
plot_images(images, y_true, y_pred, classes)
```

[ ]

Classification Report:

	precision	recall	f1-score	support
Apple___Apple_scab	0.00	0.00	0.00	3
Apple___healthy	0.00	0.00	0.00	0
Cherry_(including_sour)___Powdery_mildew	0.00	0.00	0.00	0
Potato___Early_blight	1.00	0.60	0.75	5
Potato___healthy	1.00	1.00	1.00	2
Squash___Powdery_mildew	0.00	0.00	0.00	0
Tomato___Early_blight	1.00	0.67	0.80	6
Tomato___Late_blight	0.00	0.00	0.00	0
Tomato___Target_Spot	0.00	0.00	0.00	0
Tomato___healthy	1.00	0.75	0.86	4
accuracy			0.60	20
macro avg	0.40	0.30	0.34	20
weighted avg	0.85	0.60	0.70	20



VGG19

```
[ ] from keras.applications import VGG19

    # Instantiate the VGG19 model
    vgg19 = VGG19(weights='imagenet', include_top=False, input_shape=(Size,Size,3))

    vgg19.trainable = False
```

```
[ ] # Define the layers
    inputs = keras.Input(shape=(Size,Size,3))

    # Get the layer
    x = vgg19(inputs, training = False)
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)
```

```
[ ] # Define the layers
    inputs = keras.Input(shape=(Size,Size,3))

    # Get the layer
    x = vgg19(inputs, training = False)
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)

    x = tf.keras.layers.Dense(38, activation="softmax")(x)

    vgg19 = Model(inputs=inputs, outputs=x)
```

```
[ ] vgg19.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
[ ] history=vgg19.fit(train_generator,validation_data=valid_generator,epochs=100,batch_size=32,
                      steps_per_epoch=len(train_generator)//16,validation_steps=len(valid_generator)//16,
                      callbacks=callback, verbose=1)
```

```
Epoch 1/100
274/274 [==============================] - ETA: 0s - loss: 2.9066 - accuracy: 0.3403
Epoch 00001: val_loss did not improve from 0.48662
274/274 [==============================] - 71s 258ms/step - loss: 2.9066 - accuracy: 0.3403 - val_loss: 2.1753 - val_accuracy: 0.4375
Epoch 2/100
274/274 [==============================] - ETA: 0s - loss: 1.2660 - accuracy: 0.5847
Epoch 00002: val_loss did not improve from 0.48662
```



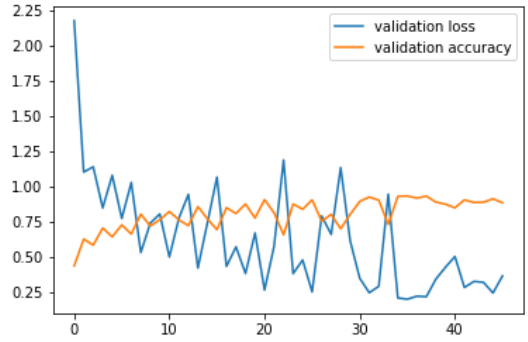
```
274/274 [==============================] - ETA: 0s - loss: 0.3214 - accuracy: 0.8891
Epoch 00043: val_loss did not improve from 0.20124
274/274 [==============================] - 68s 248ms/step - loss: 0.3214 - accuracy: 0.8891 - val_loss: 0.3274 - val_accuracy: 0.8869
Epoch 44/100
274/274 [==============================] - ETA: 0s - loss: 0.2919 - accuracy: 0.9033
Epoch 00044: val_loss did not improve from 0.20124
274/274 [==============================] - 68s 248ms/step - loss: 0.2919 - accuracy: 0.9033 - val_loss: 0.3210 - val_accuracy: 0.8888
Epoch 45/100
274/274 [==============================] - ETA: 0s - loss: 0.3055 - accuracy: 0.9015
Epoch 00045: val_loss did not improve from 0.20124
274/274 [==============================] - 70s 256ms/step - loss: 0.3055 - accuracy: 0.9015 - val_loss: 0.2454 - val_accuracy: 0.9127
Epoch 46/100
274/274 [==============================] - ETA: 0s - loss: 0.2770 - accuracy: 0.9060
Epoch 00046: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.

Epoch 00046: val_loss did not improve from 0.20124
274/274 [==============================] - 70s 256ms/step - loss: 0.2770 - accuracy: 0.9060 - val_loss: 0.3665 - val_accuracy: 0.8851
```

```
[ ] ModelLoss, ModelAccuracy = vgg19.evaluate(valid_generator)
    print('Test Loss is {}'.format(ModelLoss))
    print('Test Accuracy is {}'.format(ModelAccuracy ))
```

```
1099/1099 [==============================] - 89s 81ms/step - loss: 0.2756 - accuracy: 0.9077
Test Loss is 0.27555862069129944
Test Accuracy is 0.9076940417289734
```

```
[ ] plt.plot(history.history['val_loss'],label='validation loss')
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
    plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f961e354b50>
```



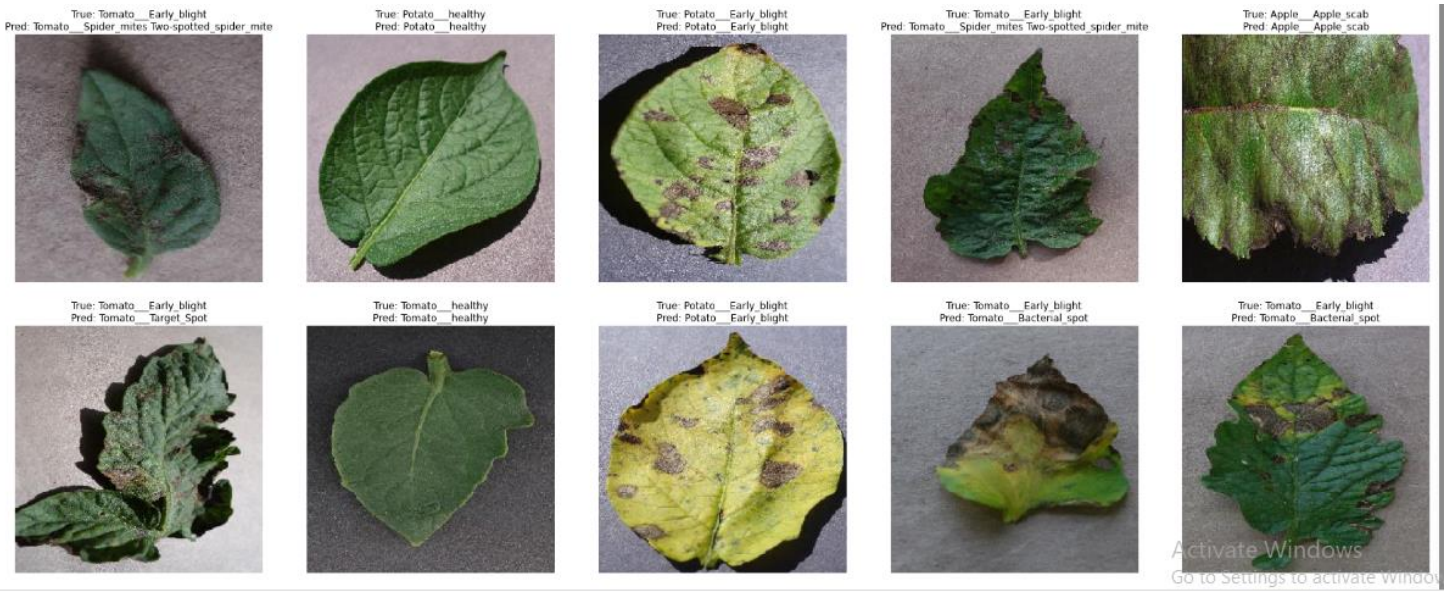

```
[ ] vgg19_accuracy = history.history['val_accuracy'][len(history.history['val_accuracy'])-1]
```

```
[ ] import os
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing import image
    from sklearn.metrics import classification_report
    import matplotlib.pyplot as plt

    # Initialize lists for true and predicted labels
    y_true = []
    y_pred = []
    images = []  # To store image data for visualization

    # Iterate through the test directory to process images
    for img_name in os.listdir(test_dir):
        # Load and preprocess the image
        img_path = os.path.join(test_dir, img_name)
        img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size to match your model's input size
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0  # Normalize the image array

        # Predict the class
        prediction = vgg19.predict(img_array)
        predicted_class = np.argmax(prediction, axis=1)[0]

        true_label_name = img_name.split('_')[0]  # Modify this if needed based on your naming convention
```

```
        true_label_name = img_name.split('.')[0]  # Modify this if needed based on your naming convention
        true_label_name = true_label_name[:-1].lower()

        # Match true labels to predicted classes
        for label in classes:
            processed_label = label.replace('_', '').lower()
            if true_label_name in processed_label:
                y_true.append(classes.index(label))
                y_pred.append(predicted_class)
                images.append(img)  # Store the image for visualization
                break  # Exit the loop once the label is found

    # Calculate unique classes
    unique_classes = np.unique(y_true + y_pred).tolist()

    # Calculate and print performance metrics
    accuracy = np.mean(np.array(y_true) == np.array(y_pred))
    report = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes])

    print("Classification Report:")
    print(report)

    # Optional: Extract average precision, recall, and F1 score
    report_dict = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes], output_dict=True)
    macro_avg = report_dict['macro avg']

    print(f"Accuracy          : {accuracy}")
```

```
print(report)

# Optional: Extract average precision, recall, and F1 score
report_dict = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes], output_dict=True)
macro_avg = report_dict['macro avg']

print(f"Accuracy          : {accuracy}")
print(f"Average Precision : {macro_avg['precision']}")
print(f"Average Recall    : {macro_avg['recall']}")
print(f"Average F1 Score  : {macro_avg['f1-score']}")

# Visualize images with actual and predicted labels
def plot_images(images, y_true, y_pred, classes, max_images=15):
    plt.figure(figsize=(25, 15))
    for i in range(min(max_images, len(images))):  # Limit to max_images
#         print(i)
        plt.subplot(3, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"True: {classes[y_true[i]]}\nPred: {classes[y_pred[i]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Call the function to plot images
plot_images(images, y_true, y_pred, classes)
```

```
Classification Report:
                                        precision    recall  f1-score   support

                           Apple___Apple_scab       1.00      1.00      1.00         3
                        Potato___Early_blight       1.00      1.00      1.00         5
                             Potato___healthy       1.00      1.00      1.00         2
                       Tomato___Bacterial_spot       0.00      0.00      0.00         0
                         Tomato___Early_blight       0.00      0.00      0.00         6
                          Tomato___Late_blight       0.00      0.00      0.00         0
Tomato___Spider_mites Two-spotted_spider_mite       0.00      0.00      0.00         0
                        Tomato___Target_Spot        0.00      0.00      0.00         0
                             Tomato___healthy       1.00      0.75      0.86         4

                                    accuracy                           0.65        20
                                   macro avg       0.44      0.42      0.43        20
                                weighted avg       0.70      0.65      0.67        20
```



MobileNetV2

```
[ ]
    mbnet_v2 = tf.keras.applications.MobileNetV2(
        weights="imagenet",
        include_top=False,
        input_shape=(Size,Size,3)
    )
    mbnet_v2.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_224_no_
9412608/9406464 [==============================] - 0s 0us/step
```

```
[ ]  # Define the layers
    inputs = keras.Input(shape=(Size,Size,3))

    # Get the layer
    x = mbnet_v2(inputs, training = False)

    # Stack layers further
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)
```



```
[ ]  # Define the layers
    inputs = keras.Input(shape=(Size,Size,3))

    # Get the layer
    x = mbnet_v2(inputs, training = False)

    # Stack layers further
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)
    x = tf.keras.layers.Dense(4096, activation="relu")(x)

    x = tf.keras.layers.Dense(38, activation="softmax")(x)


    # Combine the model
    mobilenet2 = Model(inputs=inputs, outputs=x)

    # Summary
    mobilenet2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
mobilenetv2_1.00_224 (Functi (None, 7, 7, 1280)        2257984
_________________________________________________________________
flatten_2 (Flatten)          (None, 62720)             0
_________________________________________________________________
dense_6 (Dense)              (None, 4096)              256905216
_________________________________________________________________
dense_7 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_8 (Dense)              (None, 38)                155686
=================================================================
Total params: 276,100,198
Trainable params: 273,842,214
Non-trainable params: 2,257,984
_________________________________________________________________
```

```
mobilenet2.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
history=mobilenet2.fit(train_generator,validation_data=valid_generator,epochs=100,batch_size=32,
                       steps_per_epoch=len(train_generator)//16,validation_steps=len(valid_generator)//16,
                       callbacks=callback, verbose=1)
```


```
                       steps_per_epoch=len(train_generator)//16,validation_steps=len(valid_generator)//16,
                       callbacks=callback, verbose=1)
```


```
Epoch 1/100
274/274 [==============================] - ETA: 0s - loss: 5.9018 - accuracy: 0.4843
Epoch 00001: val_loss did not improve from 0.29294
274/274 [==============================] - 60s 220ms/step - loss: 5.9018 - accuracy: 0.4843 - val_loss: 1.8033 - val_accuracy: 0.4678
Epoch 2/100
274/274 [==============================] - ETA: 0s - loss: 1.1255 - accuracy: 0.6941
Epoch 00002: val_loss did not improve from 0.29294
274/274 [==============================] - 59s 217ms/step - loss: 1.1255 - accuracy: 0.6941 - val_loss: 0.6494 - val_accuracy: 0.8199
Epoch 3/100
274/274 [==============================] - ETA: 0s - loss: 0.8012 - accuracy: 0.7641
Epoch 00003: val_loss did not improve from 0.29294
274/274 [==============================] - 60s 218ms/step - loss: 0.8012 - accuracy: 0.7641 - val_loss: 0.5803 - val_accuracy: 0.8300
Epoch 4/100
274/274 [==============================] - ETA: 0s - loss: 0.7114 - accuracy: 0.7972
Epoch 00004: val_loss did not improve from 0.29294
274/274 [==============================] - 58s 212ms/step - loss: 0.7114 - accuracy: 0.7972 - val_loss: 0.7253 - val_accuracy: 0.8088
Epoch 5/100
274/274 [==============================] - ETA: 0s - loss: 0.6375 - accuracy: 0.8173
Epoch 00005: val_loss did not improve from 0.29294
274/274 [==============================] - 59s 217ms/step - loss: 0.6375 - accuracy: 0.8173 - val_loss: 0.8494 - val_accuracy: 0.7767
Epoch 6/100
274/274 [==============================] - ETA: 0s - loss: 0.6209 - accuracy: 0.8177
Epoch 00006: val_loss did not improve from 0.29294
274/274 [==============================] - 59s 214ms/step - loss: 0.6209 - accuracy: 0.8177 - val_loss: 1.0409 - val_accuracy: 0.7335
Epoch 7/100
274/274 [==============================] - ETA: 0s - loss: 0.5830 - accuracy: 0.8269
Epoch 00007: val_loss did not improve from 0.29294
```

```
274/274 [==============================] - 57s 208ms/step - loss: 0.3182 - accuracy: 0.9056 - val_loss: 0.3804 - val_accu
Epoch 42/100
274/274 [==============================] - ETA: 0s - loss: 0.3569 - accuracy: 0.9044
Epoch 00042: val_loss did not improve from 0.14606
274/274 [==============================] - 57s 209ms/step - loss: 0.3569 - accuracy: 0.9044 - val_loss: 0.4268 - val_accuracy: 0.9182
Epoch 43/100
274/274 [==============================] - ETA: 0s - loss: 0.3305 - accuracy: 0.9092
Epoch 00043: val_loss did not improve from 0.14606
274/274 [==============================] - 57s 209ms/step - loss: 0.3305 - accuracy: 0.9092 - val_loss: 0.3986 - val_accuracy: 0.8879
Epoch 44/100
274/274 [==============================] - ETA: 0s - loss: 0.3146 - accuracy: 0.9124
Epoch 00044: val_loss did not improve from 0.14606
274/274 [==============================] - 57s 208ms/step - loss: 0.3146 - accuracy: 0.9124 - val_loss: 0.2672 - val_accuracy: 0.9320
Epoch 45/100
274/274 [==============================] - ETA: 0s - loss: 0.3196 - accuracy: 0.9113
Epoch 00045: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.

Epoch 00045: val_loss did not improve from 0.14606
274/274 [==============================] - 59s 215ms/step - loss: 0.3196 - accuracy: 0.9113 - val_loss: 0.4191 - val_accuracy: 0.9219
```

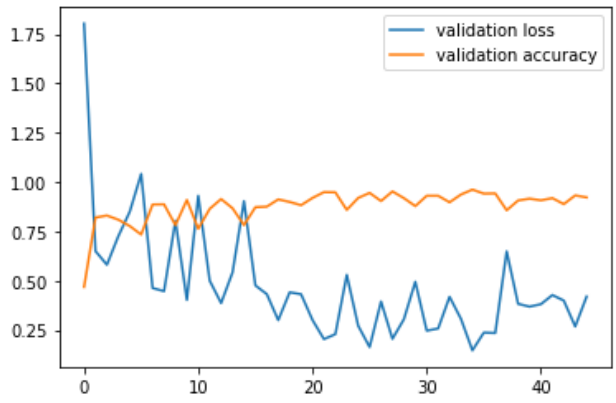
```
[ ] ModelLoss, ModelAccuracy = mobilenet2.evaluate(valid_generator)
    print('Test Loss is {}'.format(ModelLoss))
    print('Test Accuracy is {}'.format(ModelAccuracy))
```

```
1099/1099 [==============================] - 42s 38ms/step - loss: 0.4161 - accuracy: 0.8901
Test Loss is 0.41611775755882263
Test Accuracy is 0.8901092410087585
```



```
[ ] plt.plot(history.history['val_loss'],label='validation loss')
    plt.plot(history.history['val_accuracy'],label='validation accuracy')
    plt.legend()
```

```
<matplotlib.legend.Legend at 0x79a8fa748450>
```




```
[ ] mobilenetv2_accuracy = history.history['val_accuracy'][len(history.history['val_accuracy'])-1]
```

```
[ ] import os
    import numpy as np
    import tensorflow as tf
```

```
[ ] mobilenetv2_accuracy = history.history['val_accuracy'][len(history.history['val_accuracy'])-1]
```

```
[ ] import os
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing import image
    from sklearn.metrics import classification_report
    import matplotlib.pyplot as plt

    # Initialize lists for true and predicted labels
    y_true = []
    y_pred = []
    images = []  # To store image data for visualization

    # Iterate through the test directory to process images
    for img_name in os.listdir(test_dir):
        # Load and preprocess the image
        img_path = os.path.join(test_dir, img_name)
        img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size to match your model's input size
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0  # Normalize the image array
```

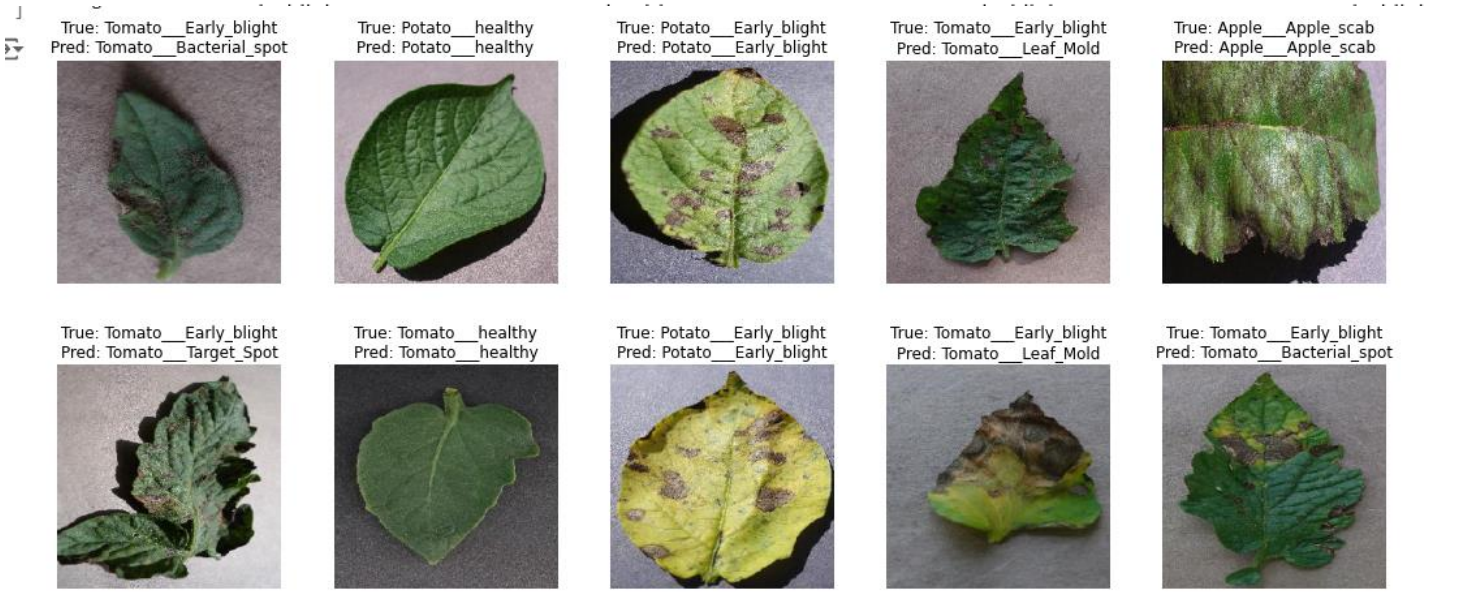
```
[ ]     prediction = mobilenet2.predict(img_array)
        predicted_class = np.argmax(prediction, axis=1)[0]

        true_label_name = img_name.split('.')[0]  # Modify this if needed based on your naming convention
        true_label_name = true_label_name[:-1].lower()

        # Match true labels to predicted classes
        for label in classes:
            processed_label = label.replace('_', '').lower()
            if true_label_name in processed_label:
                y_true.append(classes.index(label))
                y_pred.append(predicted_class)
                images.append(img)  # Store the image for visualization
                break  # Exit the loop once the label is found

    # Calculate unique classes
    unique_classes = np.unique(y_true + y_pred).tolist()

    # Calculate and print performance metrics
    accuracy = np.mean(np.array(y_true) == np.array(y_pred))
    report = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes])

    print("Classification Report:")
    print(report)

    # Optional: Extract average precision, recall, and F1 score
    report_dict = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes], output_dict=True)
    macro_avg = report_dict['macro avg']
```

```
print(f"Accuracy          : {accuracy}")
print(f"Average Precision : {macro_avg['precision']}")
print(f"Average Recall    : {macro_avg['recall']}")
print(f"Average F1 Score  : {macro_avg['f1-score']}")

# Visualize images with actual and predicted labels
def plot_images(images, y_true, y_pred, classes, max_images=15):
    plt.figure(figsize=(15, 10))
    for i in range(min(max_images, len(images))):  # Limit to max_images
#         print(i)
        plt.subplot(3, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"True: {classes[y_true[i]]}\nPred: {classes[y_pred[i]]}")
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Call the function to plot images
plot_images(images, y_true, y_pred, classes)
```

Classification Report:

	precision	recall	f1-score	support
Apple___Apple_scab	1.00	1.00	1.00	3
Potato___Early_blight	1.00	0.60	0.75	5
Potato___Late_blight	0.00	0.00	0.00	0
Potato___healthy	1.00	1.00	1.00	2
Tomato___Bacterial_spot	0.00	0.00	0.00	0
Tomato___Early_blight	1.00	0.17	0.29	6
Tomato___Leaf_Mold	0.00	0.00	0.00	0
Tomato___Target_Spot	0.00	0.00	0.00	0
Tomato___healthy	1.00	1.00	1.00	4
accuracy			0.65	20
macro avg	0.56	0.42	0.45	20
weighted avg	1.00	0.65	0.72	20

```
Accuracy          : 0.65
Average Precision : 0.5555555555555556
Average Recall    : 0.4185185185185185
Average F1 Score  : 0.4484126984126984
```



Inception model

```
[ ]  inception_model = InceptionV3(input_shape= (Size, Size, 3),
                                   include_top = False,
                                   weights = "imagenet")



    for layer in inception_model.layers:
        layer.trainable = False


    # Taking output from 'mixed8' layer
    last_layer = inception_model.get_layer('mixed9')
    print('Last Layer Output Shape:', last_layer.output_shape)
    last_output = last_layer.output


    x = layers.Flatten()(last_output)

    x = layers.Dense(2048, activation='relu')(x)
```

```
for layer in inception_model.layers:
    layer.trainable = False


# Taking output from 'mixed8' layer
last_layer = inception_model.get_layer('mixed9')
print('Last Layer Output Shape:', last_layer.output_shape)
last_output = last_layer.output


x = layers.Flatten()(last_output)

x = layers.Dense(2048, activation='relu')(x)

x = layers.Dense(1024, activation='relu')(x)

x = layers.Dropout(0.2)(x)

x = layers.Dense(38, activation='softmax')(x)

inception3 = Model(inception_model.input, x)
```

```
[ ] inception3.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
[ ] history=inception3.fit(train_generator,validation_data=valid_generator,epochs=100,batch_size=32,
                           steps_per_epoch=len(train_generator)//16,validation_steps=len(valid_generator)//16,
                           callbacks=callback, verbose=1)
```

```
Epoch 1/100
274/274 [==============================] - ETA: 0s - loss: 1.6590 - accuracy: 0.5935
Epoch 00001: val_loss did not improve from 0.14606
274/274 [==============================] - 61s 223ms/step - loss: 1.6590 - accuracy: 0.5935 - val_loss: 0.4988 - val_accuracy: 0.8529
Epoch 2/100
274/274 [==============================] - ETA: 0s - loss: 0.6898 - accuracy: 0.7790
Epoch 00002: val_loss did not improve from 0.14606
274/274 [==============================] - 60s 219ms/step - loss: 0.6898 - accuracy: 0.7790 - val_loss: 0.5184 - val_accuracy: 0.8327
Epoch 3/100
274/274 [==============================] - ETA: 0s - loss: 0.5936 - accuracy: 0.8166
Epoch 00003: val_loss did not improve from 0.14606
274/274 [==============================] - 60s 219ms/step - loss: 0.5936 - accuracy: 0.8166 - val_loss: 0.8784 - val_accuracy: 0.7298
Epoch 4/100
274/274 [==============================] - ETA: 0s - loss: 0.5073 - accuracy: 0.8481
Epoch 00004: val_loss did not improve from 0.14606
274/274 [==============================] - 60s 219ms/step - loss: 0.5073 - accuracy: 0.8481 - val_loss: 0.3710 - val_accuracy: 0.8980
Epoch 5/100
274/274 [==============================] - ETA: 0s - loss: 0.4672 - accuracy: 0.8634
Epoch 00005: val_loss did not improve from 0.14606
274/274 [==============================] - 58s 213ms/step - loss: 0.3236 - accuracy: 0.9115 - val_loss: 0.1927 - val_accuracy: 0.9522
Epoch 22/100
274/274 [==============================] - ETA: 0s - loss: 0.3440 - accuracy: 0.9037
Epoch 00022: val_loss did not improve from 0.14606
274/274 [==============================] - 59s 214ms/step - loss: 0.3440 - accuracy: 0.9037 - val_loss: 0.2216 - val_accuracy: 0.9540
Epoch 23/100
274/274 [==============================] - ETA: 0s - loss: 0.2957 - accuracy: 0.9122
Epoch 00023: val_loss did not improve from 0.14606
274/274 [==============================] - 59s 215ms/step - loss: 0.2957 - accuracy: 0.9122 - val_loss: 0.2080 - val_accuracy: 0.9531
Epoch 24/100
274/274 [==============================] - ETA: 0s - loss: 0.2977 - accuracy: 0.9163
Epoch 00024: val_loss did not improve from 0.14606
274/274 [==============================] - 59s 215ms/step - loss: 0.2977 - accuracy: 0.9163 - val_loss: 0.2940 - val_accuracy: 0.9265
```

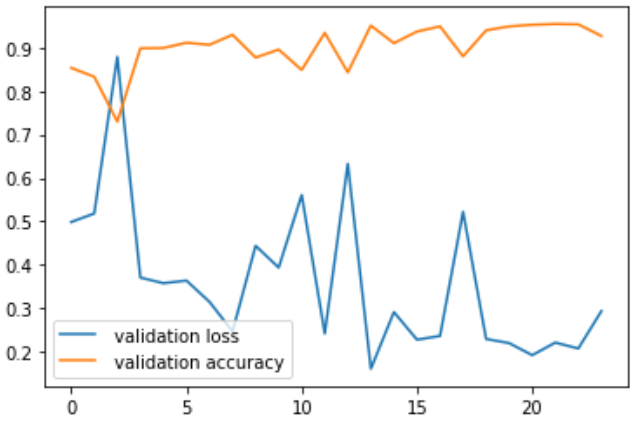


```
[ ] ModelLoss, ModelAccuracy = inception3.evaluate(valid_generator)
    print('Test Loss is {}'.format(ModelLoss))
    print('Test Accuracy is {}'.format(ModelAccuracy))
```

```
1099/1099 [==============================] - 44s 40ms/step - loss: 0.3258 - accuracy: 0.9073
Test Loss is 0.3258468210697174
Test Accuracy is 0.9072957038879395
```

```
[ ] plt.plot(history.history['val_loss'],label=' validation loss')
    plt.plot(history.history['val_accuracy'],label=' validation accuracy')
    plt.legend()
```

```
<matplotlib.legend.Legend at 0x79a5555974d0>
```




```
[ ] inceptionv3_accuracy = history.history['val_accuracy'][len(history.history['val_accuracy'])-1]
```

```
[ ] inceptionv3_accuracy = history.history['val_accuracy'][len(history.history['val_accuracy'])-1]
```

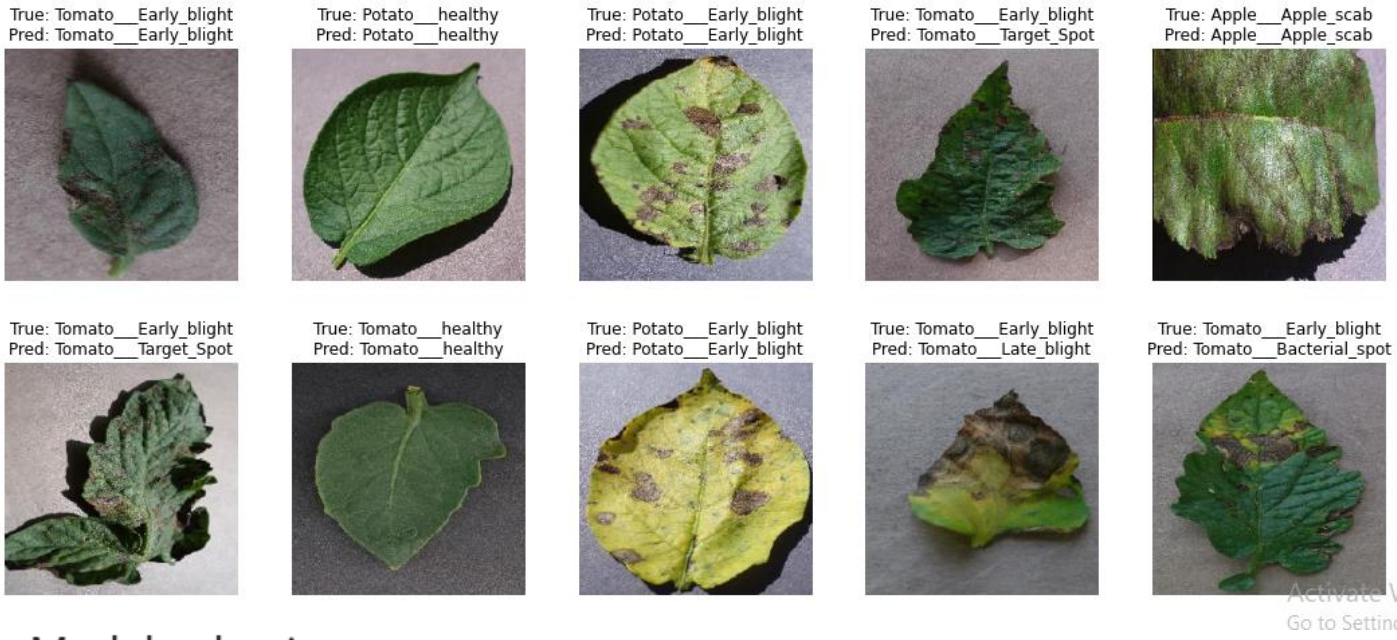
```
[ ] import os
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing import image
    from sklearn.metrics import classification_report
    import matplotlib.pyplot as plt

    # Initialize lists for true and predicted labels
    y_true = []
    y_pred = []
    images = []  # To store image data for visualization

    # Iterate through the test directory to process images
    for img_name in os.listdir(test_dir):
        # Load and preprocess the image
        img_path = os.path.join(test_dir, img_name)
        img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size to match your model's input size
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0  # Normalize the image array
```

```
[ ]     img_path = os.path.join(test_dir, img_name)
        img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size to match your model's input size
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0  # Normalize the image array

        # Predict the class
        prediction = inception3.predict(img_array)
        predicted_class = np.argmax(prediction, axis=1)[0]

        true_label_name = img_name.split('.')[0]  # Modify this if needed based on your naming convention
        true_label_name = true_label_name[:-1].lower()

        # Match true labels to predicted classes
        for label in classes:
            processed_label = label.replace('_', '').lower()
            if true_label_name in processed_label:
                y_true.append(classes.index(label))
                y_pred.append(predicted_class)
                images.append(img)  # Store the image for visualization
                break  # Exit the loop once the label is found

    # Calculate unique classes
    unique_classes = np.unique(y_true + y_pred).tolist()
```

```
[ ] report_dict = classification_report(y_true, y_pred, target_names=[classes[i] for i in unique_classes], output_dict=True)
    macro_avg = report_dict['macro avg']

    print(f"Accuracy          : {accuracy}")
    print(f"Average Precision : {macro_avg['precision']}")
    print(f"Average Recall    : {macro_avg['recall']}")
    print(f"Average F1 Score  : {macro_avg['f1-score']}")

    # Visualize images with actual and predicted labels
    def plot_images(images, y_true, y_pred, classes, max_images=15):
        plt.figure(figsize=(15, 10))
        for i in range(min(max_images, len(images))):  # Limit to max_images
    #         print(i)
            plt.subplot(3, 5, i + 1)
            plt.imshow(images[i])
            plt.title(f"True: {classes[y_true[i]]}\nPred: {classes[y_pred[i]]}")
            plt.axis('off')
        plt.tight_layout()
        plt.show()

    # Call the function to plot images
    plot_images(images, y_true, y_pred, classes)
```

Classification Report:

	precision	recall	f1-score	support
Apple___Apple_scab	1.00	1.00	1.00	3
Potato___Early_blight	1.00	1.00	1.00	5
Potato___healthy	1.00	1.00	1.00	2
Tomato___Bacterial_spot	0.00	0.00	0.00	0
Tomato___Early_blight	1.00	0.17	0.29	6
Tomato___Late_blight	0.00	0.00	0.00	0
Tomato___Septoria_leaf_spot	0.00	0.00	0.00	0
Tomato___Target_Spot	0.00	0.00	0.00	0
Tomato___healthy	1.00	0.75	0.86	4
accuracy			0.70	20
macro avg	0.56	0.44	0.46	20
weighted avg	1.00	0.70	0.76	20



Model selection

```
[ ]  models = ["alexnet","vgg19","mobilenetv2","inceptionv3"]
     Models = [alexnet,vgg19,mobilenet2,inception3]

     accuracies = [alexnet_accuracy,vgg19_accuracy,mobilenetv2_accuracy,inceptionv3_accuracy]

     best_model_index = accuracies.index(max(accuracies))
     print("best performing model :",models[best_model_index])
     best_model_name = models[best_model_index]
     best_model = Models[best_model_index]
```

```
best performing model : inceptionv3
```

metrics applied on best model

```
[ ]  predictions=best_model.predict(valid_generator)
```

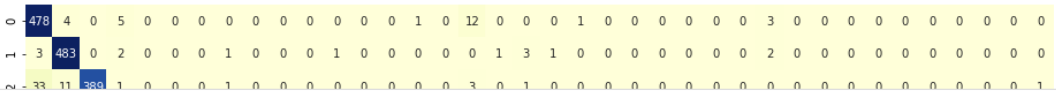
```
predictions=best_model.predict(valid_generator)
```

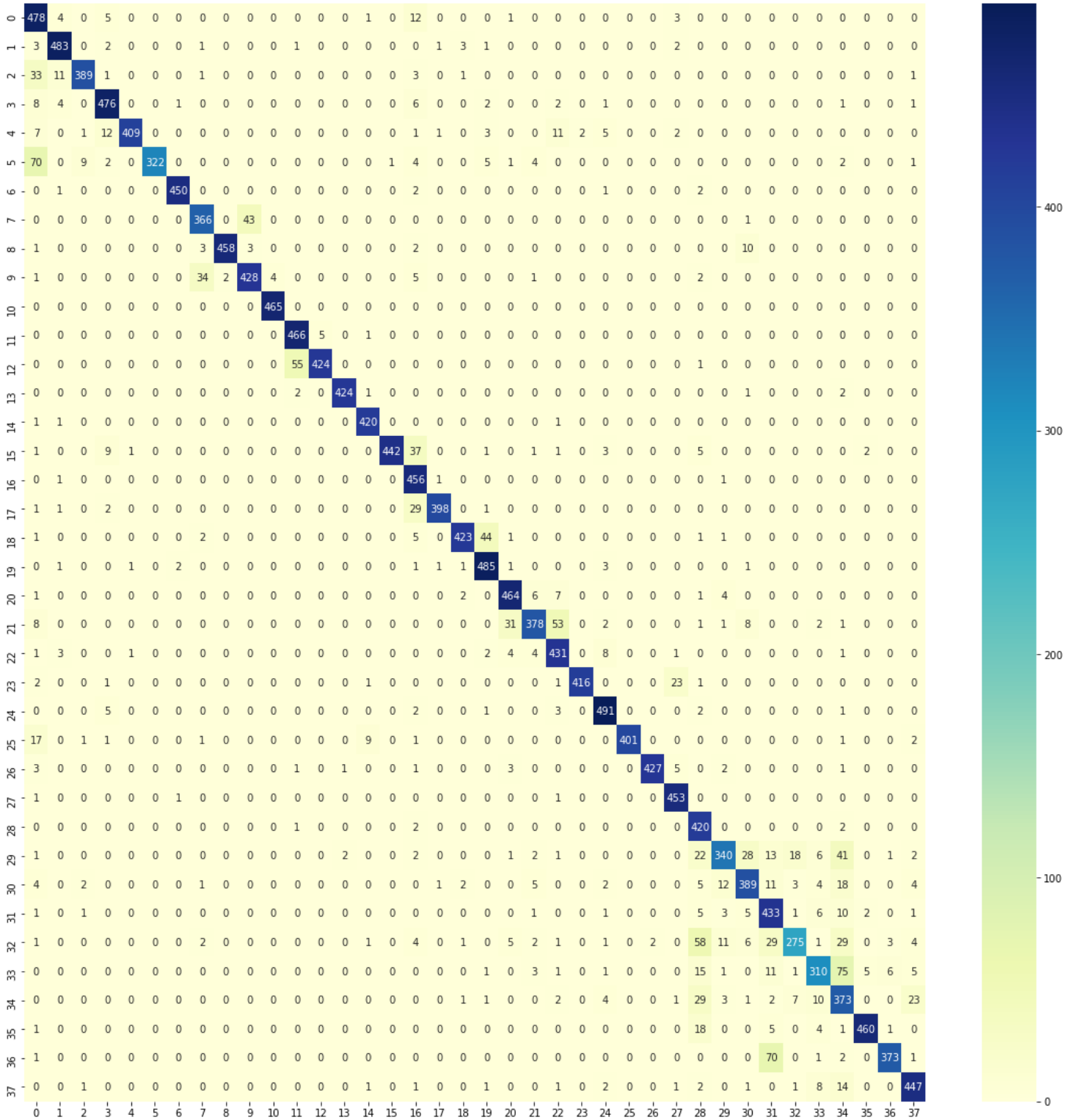
```
from sklearn.metrics import classification_report
print(classification_report(valid_generator.classes, predictions.argmax(axis=1)))
```

```
              precision    recall  f1-score   support

           0       0.74      0.95      0.83       504
           1       0.95      0.97      0.96       497
           2       0.96      0.88      0.92       440
           3       0.92      0.95      0.94       502
           4       0.99      0.90      0.94       454
           5       1.00      0.76      0.87       421
           6       0.99      0.99      0.99       456
           7       0.89      0.89      0.89       410
           8       1.00      0.96      0.98       477
           9       0.90      0.90      0.90       477
          10       0.99      1.00      1.00       465
          11       0.89      0.99      0.93       472
          12       0.99      0.88      0.93       480
          13       0.99      0.99      0.99       430
          14       0.97      0.99      0.98       423
          15       1.00      0.88      0.93       503
          16       0.79      0.99      0.88       459
          33       0.88      0.71      0.79       435
          34       0.65      0.82      0.72       457
          35       0.98      0.94      0.96       490
          36       0.97      0.83      0.90       448
          37       0.91      0.93      0.92       481

    accuracy                           0.91     17572
   macro avg       0.92      0.91      0.91     17572
weighted avg       0.92      0.91      0.91     17572
```

```
[ ] import seaborn as sns
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(valid_generator.classes, predictions.argmax(axis=1))
    plt.figure(figsize=(20,20))
    sns.heatmap(cm,  annot=True, fmt="d" ,cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x79a5548e1810>
```




0 478 4 0 5 0 0 0 0 0 0 0 0 0 0 1 0 12 0 0 0 1 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0
1 3 483 0 2 0 0 0 1 0 0 0 1 0 0 0 0 0 0 1 3 1 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
2 33 11 389 1 0 0 0 1 0 0 0 0 0 0 0 0 3 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
3 8 4 0 476 0 0 1 0 0 0 0 0 0 0 0 0 6 0 0 2 0 0 2 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1
4 7 0 1 12 409 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 3 0 0 11 2 5 0 0 2 0 0 0 0 0 0 0 0 0 0
5 70 0 9 2 0 322 0 0 0 0 0 0 0 0 0 0 1 4 0 0 5 1 4 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1
6 0 1 0 0 0 0 450 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 0 0 0 0
7 0 0 0 0 0 0 0 366 0 43 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
8 1 0 0 0 0 0 0 3 458 3 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 10 0 0 0 0 0 0 0
9 1 0 0 0 0 0 0 34 2 428 4 0 0 0 0 0 5 0 0 0 0 1 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 465 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
11 0 0 0 0 0 0 0 0 0 0 0 466 5 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
12 0 0 0 0 0 0 0 0 0 0 0 55 424 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
13 0 0 0 0 0 0 0 0 0 0 0 2 0 424 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0
14 1 1 0 0 0 0 0 0 0 0 0 0 0 0 420 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
15 1 0 0 9 1 0 0 0 0 0 0 0 0 0 0 442 37 0 0 1 0 1 1 0 3 0 0 0 5 0 0 0 0 0 0 2 0 0
16 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 456 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
17 1 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0 29 398 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
18 1 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 5 0 423 44 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
19 0 1 0 0 1 0 2 0 0 0 0 0 0 0 0 0 1 1 1 485 1 0 0 0 3 0 0 0 0 0 1 0 0 0 0 0 0 0
20 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 464 6 7 0 0 0 0 0 1 4 0 0 0 0 0 0 0 0
21 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 31 378 53 0 2 0 0 0 1 1 8 0 0 2 1 0 0 0
22 1 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 4 4 431 0 8 0 0 1 0 0 0 0 0 0 1 0 0 0
23 2 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 416 0 0 0 23 1 0 0 0 0 0 0 0 0
24 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1 0 0 3 0 491 0 0 0 2 0 0 0 0 0 1 0 0 0
25 17 0 1 1 0 0 0 1 0 0 0 0 0 0 9 0 1 0 0 0 0 0 0 0 0 0 401 0 0 0 0 0 0 0 0 1 0 0 2
26 3 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 3 0 0 0 0 0 427 5 0 2 0 0 0 0 1 0 0 0
27 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 453 0 0 0 0 0 0 0 0 0 0
28 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 420 0 0 0 0 0 2 0 0 0
29 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 2 0 0 0 1 2 1 0 0 0 0 0 22 340 28 13 18 6 41 0 1 2
30 4 0 2 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 2 0 0 5 0 0 2 0 0 0 5 12 389 11 3 4 18 0 0 4
31 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 5 3 5 433 1 6 10 2 0 1
32 1 0 0 0 0 0 0 2 0 0 0 0 0 0 1 0 4 0 1 0 5 2 1 0 1 0 2 0 58 11 6 29 275 1 29 0 3 4
33 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 3 1 0 1 0 0 0 15 1 0 11 1 310 75 5 6 5
34 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 2 0 4 0 0 1 29 3 1 2 7 10 373 0 0 23
35 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 18 0 0 5 0 4 1 460 1 0
36 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 70 0 1 2 0 373 1
37 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 2 0 0 1 2 0 1 0 1 8 14 0 0 447
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
400
300
200
100
0

Inference on new examples

test data are already resized, no additional preprocessing needed

```
[ ] y_pred=best_model.predict(X_test)
```

```
[ ] y_pred[0]
```

```
array([3.6466660e-04, 1.6229467e-07, 1.7708426e-06, 1.1910176e-06,
       1.2175467e-08, 3.5688981e-07, 1.6628610e-10, 1.4510526e-06,
       1.1093270e-08, 5.2518849e-06, 1.6175438e-08, 1.2337470e-06,
       5.8173332e-06, 1.3574771e-05, 3.7434969e-08, 1.1108717e-06,
       5.3037348e-07, 6.7135124e-09, 1.6460954e-06, 2.5003457e-08,
       5.5071760e-05, 8.6923259e-01, 2.0348685e-05, 9.7169888e-08,
       7.5733446e-08, 2.7815380e-08, 6.2108862e-05, 2.9966418e-07,
       3.2519307e-05, 7.8481667e-02, 4.8926070e-02, 2.1653008e-03,
       1.3339783e-04, 1.3421451e-04, 2.3919459e-04, 1.1444669e-04,
       3.6727126e-06, 1.7294903e-07], dtype=float32)
```

```
[ ] predictions=y_pred.argmax(axis=1)
```

```
[ ] classes
```


[ ] predictions=y_pred.argmax(axis=1)

```
[ ] classes
```

```
['Apple___Apple_scab',
 'Apple___Black_rot',
 'Apple___Cedar_apple_rust',
 'Apple___healthy',
 'Blueberry___healthy',
 'Cherry_(including_sour)___Powdery_mildew',
 'Cherry_(including_sour)___healthy',
 'Corn_(maize)___Cercospora_leaf_spot Gray_leaf_spot',
 'Corn_(maize)___Common_rust_',
 'Corn_(maize)___Northern_Leaf_Blight',
 'Corn_(maize)___healthy',
 'Grape___Black_rot',
 'Grape___Esca_(Black_Measles)',
 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)',
 'Grape___healthy',
 'Orange___Haunglongbing_(Citrus_greening)',
 'Peach___Bacterial_spot',
 'Peach___healthy',
 'Pepper,_bell___Bacterial_spot',
 'Pepper,_bell___healthy',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy',
 'Raspberry___healthy',
 'Soybean___healthy',
```

```
[ ]  predictions
```

```
array([21, 35, 35, ...,  2, 20, 34])
```

```
[ ]  predictions[0]
```

```
[ ]  import matplotlib.pyplot as plt

     # Assuming X_test contains your test images, predictions contain your
     plt.figure(figsize=(20, 10))

     for i in range(10):
         plt.subplot(2, 5, i + 1)  # 2 rows, 5 columns
         plt.imshow(X_test[i])
         plt.title(classes[predictions[i]])
         plt.axis('off')

     plt.show()
```