Potato Disease Classification using Fine-tuned Convolutional Neural Networks: Leveraging Pre-trained Models for Enhanced Accuracy

Rohith Bandari

Abstract: **This research explores the utilization of convolutional neural networks (CNNs) to automate the identification of diseases affecting potato crops, aiming to enhance agricultural practices through reliable and efficient early disease detection. Leveraging TensorFlow, Keras, Matplotlib, Seaborn, and scikit-learn, the study develops and evaluates various CNN models, including VGG, ResNet, Inception, DenseNet, and EfficientNet, for this task. Through meticulous experimentation and evaluation, the most optimal CNN model is identified based on test accuracy, providing valuable insights into the effectiveness of different architectures. The research highlights the potential of CNNs to revolutionize disease identification in agriculture, offering a scalable solution for timely interventions to mitigate yield losses and ensure food security.**

Keywords - image processing, computer vision, plant disease diagnosis, agricultural technology, crop protection, AI in agriculture, VGG16, ResNet50,InceptionV3,DenseNet121,EfficientNetB0, callbacks, image_size, matplotlib.pyplot, seaborn, build_fine_tuned_model,compile,fit,resize_and_rescale,tf.keras.preprocessing.image.ImageDataGenerator,tf.keras.applications,tf.keras.layers,tf.keras.models,tf.keras.optimizers,tf.keras.losses,precision_recall_fscore_support,roc_curve,auc, restore_best_weights, fpr, tpr, roc_auc.

I. Introduction

A. Background

Potatoes, a staple crop in many parts of the world, hold significant importance in global agriculture and food systems. Understanding the production, consumption, and trends surrounding potatoes provides valuable insights into the dynamics of food security, nutrition, and agricultural economies. Let's delve into the detailed statistics regarding potato production, consumption, and related factors from recent years. Potato production is a key indicator of agricultural productivity and economic activity globally. Despite fluctuations influenced by factors such as weather conditions, market demand, and agricultural policies, potatoes remain vital for food security and livelihoods. In 2022, global potato production stood at approximately 17,791,800 tonnes [1], reflecting a slight decline from previous years. Historical data shows variations, with production figures of 18,582,400 tonnes[1] in 2021 and 19,051,800 tonnes[1] in 2020. Similarly, the area harvested for potatoes fluctuates annually, with approximately 362,440 hectares harvested in 2022, down from 378,670 hectares in 2021 and 368,960 hectares in 2020. Potato yield, measured at 490,891 100 g/ha in 2022, fluctuates due to technological advancements and climatic conditions, with 2021 and 2020 yields recorded at 490,727 100 g/ha[1] and 516,365 100 g/ha, respectively. Per capita consumption of potatoes, averaging 48.65 kg/capita/year [1]in 2021, remains steady, reflecting sustained demand globally. However, potato cultivation faces challenges from diseases like late blight and early blight, impacting yields and sustainability. Innovative solutions are needed to accurately detect and manage these diseases, ensuring sustainable crop production and food security amidst environmental changes and resource constraints.

B. Problem Definition:

Potato cultivation faces significant challenges from diseases like late blight and early blight, leading to widespread damage and economic losses. Chemical interventions are costly and raise environmental and health concerns, while visual inspection for disease diagnosis is labor-intensive and prone to errors. Innovative solutions are urgently needed to accurately detect and manage potato diseases, ensuring sustainable crop production and food security.

1) Problem Statement:

The core challenge in potato cultivation is the timely and accurate detection and management of diseases such as late blight and early blight. Conventional methods of disease diagnosis, including visual inspection and symptom-based identification, are often inadequate for early detection, leading to delayed responses and increased crop losses. Moreover, the reliance on chemical pesticides poses environmental and health risks, necessitating the development of more sustainable disease management strategies.

In this context, the problem statement revolves around the development of advanced technologies, particularly machine learning and computer vision-based approaches, for the early detection and classification of potato diseases. By leveraging the power of artificial intelligence and image analysis techniques, we aim to create automated systems capable of identifying disease symptoms in potato

plant images with high accuracy and reliability. These systems will enable farmers to detect diseases at their incipient stages, facilitating timely interventions and reducing reliance on chemical treatments.

2) Goals

- Defining and Training Models: The primary goal is to develop and train convolutional neural network (CNN) models for the automatic detection and classification of potato diseases using image data. By leveraging deep learning techniques, we aim to create models capable of accurately identifying disease symptoms in potato plant images.
- Evaluation and Comparison: Another goal is to evaluate the performance of different CNN architectures, including VGG, ResNet, Inception, DenseNet, and EfficientNet, for disease classification tasks. By comparing the performance metrics of these models, we can determine which architecture yields the best results for potato disease detection.
- Assessment of Model Performance: We aim to assess the performance of the trained models using various evaluation metrics, including accuracy, precision, recall, F1-score, and area under the receiver operating characteristic (ROC) curve (AUC). This comprehensive evaluation will provide insights into the strengths and weaknesses of each model and help identify the most suitable approach for potato disease detection.
- Identification of the Best Model: Ultimately, our goal is to identify the best-performing model based on its accuracy and other evaluation metrics. The selected model will serve as a valuable tool for early detection and management of potato diseases, helping farmers make informed decisions and minimize crop losses.

3) Brief Overview of Data:

Different image resolutions and sizes have been obtained from a dataset uploaded on 'Kaggle', known as Plant Village by Arjun Tejaswi which contains plant diseases of around 15 types [2]. The dataset contains about 20,600 images of which 2152 images belong to plant 'Potato' and categorised as Healthy, Early-Blight, and Late-Blight. The dataset is arranged in Joint Photographic Experts Group data format (.jpg file).

II. Related Work

Singh et al. (2015) [10] - performed a review of the literature on various image processing techniques for detecting leaf disease. The writers wanted to speed up identification and detection of plant diseases while lowering the subjectivity that comes with naked-eye observation. They presented an algorithm that uses picture segmentation to automatically identify and categorise plant leaf diseases. The impact of HSI, CIELAB, and YCbCr colour spaces on disease spot detection was examined by the authors. To identify the disease spot, the Otsu technique was applied to the colour component after an image was smoothed using a median filter. The suggested method was not put to the test on any datasets.

He et al. (2015) [11] - deep residual networks were discussed in relation to image recognition challenges. As shallow representations for image retrieval and classification, VLAD and Fisher Vector are cited in the paper's literature review as related concepts. The writers also covered the advantages of encoding residual vectors over original vectors for vector quantization. The Multigrid method was also mentioned as a method for solving partial differential equations (PDEs) by reformulating systems as subproblems at multiple scales, where each subproblem is accountable for the residual solution between a coarser and a finer scale. This method is used in low-level vision and computer graphics.

Islam et al. (2017) [12] - suggested a method to identify diseases from images of potato plant leaves by combining image processing and machine learning. The authors classified diseases using the 'Plant Village' database of openly accessible plant images. Using the suggested method, the research classified 300 images of diseases with a 95% accuracy. The paper also highlights how crucial contemporary phenotyping and plant disease detection are to assuring food security and sustainable agriculture.

Velmurugan and Renukadevi (2017) [13] - describe research on the creation of software for the automatic detection and classification of plant leaf diseases. Only a few methods were suggested for particular databases, such as satellite images, leaf sets, maps, faces, fingers, etc., the authors discovered after conducting a literature review. The research suggests a novel algorithm that, with 94% accuracy, can identify and categorise the diseases under study. The processing process consists of four major steps: colour transformation, masking of green pixels, segmentation, and computation of texture statistics from the SGDM matrices. The suggested method was tried on a database of about 500 plant leaves, and the outcomes support the robustness of the suggested algorithm.

III. Approach & Implementation

A. Data Loading

- The project approach begins with loading and partitioning the dataset, a pivotal step in constructing robust machine learning models. LeveragingTensorFlow'simage_dataset_from_dir

ectory function facilitates convenient dataset loading by automatically parsing images from directories and labeling them based on subdirectory names. This simplifies the data ingestion process significantly.

- By specifying parameters such as seed for reproducibility, image size, batch size, and shuffling, we meticulously prepare the dataset for training, ensuring its suitability for subsequent processes.
- Partitioning the dataset into training, validation, and test sets is imperative for accurately assessing model performance. Our custom function get_dataset_partitions_tf ensures this by splitting the dataset into specified proportions (80% for training, 10% for validation, and 10% for testing), with the sum of split sizes equating to 1.
- Shuffling the dataset and setting a seed value maintains randomness while ensuring consistency across different runs.
- The dataset information [fig.1], including class counts, names, and sample distributions across partitions, offers valuable insights into dataset characteristics. This understanding guides the selection of appropriate model architectures, preprocessing steps, and evaluation metrics. Displaying batch information [fig.1] confirms compatibility with model architectures and training processes, validating batch size and shape.
- Visualizing sample images [fig.2] from the training set using matplotlib provides crucial insights into class diversity, image quality, and potential preprocessing requirements like normalization or augmentation.

```
Found 2152 files belonging to 3 classes.
Number of classes: 3
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
Number of training samples: 54
Number of validation samples: 6
Number of test samples: 8
Image size: 256
Batch size: 32
Batch shape: (32, 256, 256, 3)
Labels: [0 1 1 0 1 1 1 0 0 0 2 0 1 2 0 0 0 1 0 0 2 0 1 0 1 1 0 0 1 2 0 0]
```

Figure. 1 - Dataset Information

The dataset comprises 2152 files distributed across 3 classes:'Potato___Early_blight','Potato___Late_blight', and 'Potato___healthy'. With 54 training samples, 6 validation samples, and 8 test samples, the dataset is relatively small. The images are of size 256x256 pixels, and the batches have a size of 32.

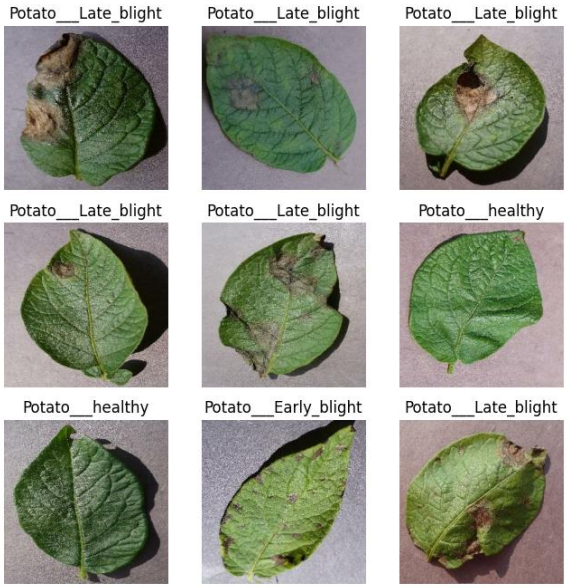


Figure. 1 - Sample Images of Dataset

B. Data Preprocessing and Augmentation

Data preprocessing and augmentation are crucial steps in training deep learning models, especially for image classification tasks. The following techniques are employed in the project, each serving a specific purpose to enhance the model's performance and robustness.

1) Resizing and Rescaling:

- Images in the dataset are resized to a uniform size using tf.keras.layers.Resizing to ensure consistency in input dimensions across the dataset.
- Rescaling is performed to normalize pixel values between 0 and 1 using tf.keras.layers.Rescaling, which helps in stabilizing training by bringing all features to a similar scale. This preprocessing step aids in mitigating the impact of varying image resolutions and improves convergence during model training.

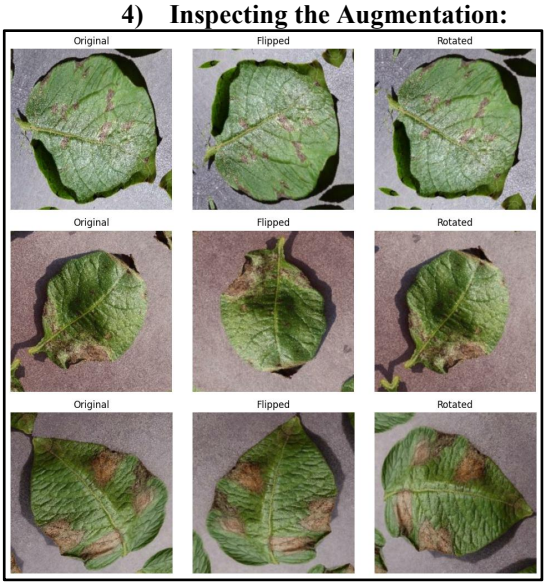
2) Data Augmentation:

- Data augmentation is employed to artificially increase the diversity of the training dataset by applying transformations to the input images.
- Horizontal and vertical flipping is achieved using tf.keras.layers.RandomFlip("horizontal_and_vertical"), which introduces variations in object orientation and appearance, making the model more robust to different viewpoints.
- Random rotation with a maximum angle of 0.2 radiansisappliedusingtf.keras.layers.RandomRotation(0.2), simulating real-world scenarios where objects may appear at different angles. Augmentation techniques help in preventing overfitting by exposing the model to a broader range of image variations, thus improving its generalization capability.

3) Applying Augmentation on Training Data:

- The train_ds dataset is mapped with the augmentation function using map to apply the specified transformations to each image in the training set.
- Setting training=True ensures that data augmentation is only applied during training and not during validation or testing, preventing data leakage and maintaining evaluation integrity.
- Prefetching is enabled using prefetch with tf.data.AUTOTUNE to overlap data preprocessing and model execution, improving training performance by reducing I/O latency.

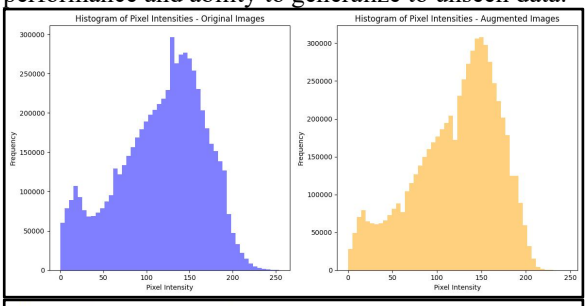
4) Inspecting the Augmentation:



After applying augmentation, sample images from the augmented training dataset are visualized to verify the effectiveness of the transformation. Original images, along with their flipped and rotated counterparts, are displayed to gain insights into the augmentation process and assess its impact on image quality and diversity. Visual inspection aids in validating the augmentation pipeline and ensures that transformed images retain their semantic integrity, essential for maintaining label consistency and model interpretability. By incorporating these preprocessing and augmentation techniques, the project aims to enrich the training dataset with diverse variations of input images, thereby enabling the CNN models to learn robust and discriminative features for accurate image classification. Additionally, these techniques contribute to regularizing the model and improving its generalization performance, ultimately enhancing the model's ability to perform effectively on unseen data.

5) Histogram Analysis of Pixel Intensities

Histogram analysis of pixel intensities plays a vital role in understanding the distribution of pixel values within images and assessing the impact of data augmentation on dataset diversity. Histograms are generated for both original and augmented images to compare their pixel intensity distributions. By visualizing these histograms, we gain insights into how augmentation techniques affect the spread and diversity of pixel intensities, crucial for enhancing the robustness and generalization of deep learning models. The histograms provide a quantitative representation of pixel intensity frequencies, allowing us to analyze any shifts or deviations between original and augmented images. This analysis guides the optimization of augmentation parameters to ensure a more balanced and representative training dataset, ultimately improving the model's performance and ability to generalize to unseen data.




```
Histogram values for original images:
Histogram values: [6.02450e+04 7.88580e+04 9.16540e+04 1.08899e+05 9.26230e+04 7.69820e+04
 6.91840e+04 7.04750e+04 7.42650e+04 8.07930e+04 8.87450e+04 9.76950e+04
 1.31708e+05 1.24135e+05 1.35738e+05 1.46406e+05 1.57164e+05 1.69025e+05
 1.81219e+05 1.91147e+05 2.00533e+05 2.07047e+05 2.15229e+05 2.21474e+05
 2.32817e+05 2.96407e+05 2.60487e+05 2.70754e+05 2.73647e+05 2.66279e+05
 2.50713e+05 2.25738e+05 1.97936e+05 1.75723e+05 1.58389e+05 1.47600e+05
 1.33282e+05 1.22502e+05 6.93050e+04 4.70390e+04 3.38410e+04 2.34960e+04
 1.48740e+04 8.69200e+03 4.92200e+03 2.75900e+03 1.58900e+03 8.73000e+02
 4.49000e+02 1.80000e+02]
Bin edges: [  0.     5.08  10.16  15.24  20.32  25.4   30.48  35.56  40.64  45.72
  50.8   55.88  60.96  66.04  71.12  76.2   81.28  86.36  91.44  96.52
 101.6  106.68 111.76 116.84 121.92 127.   132.08 137.16 142.24 147.32
 152.4  157.48 162.56 167.64 172.72 177.8  182.88 187.96 193.04 198.12
 203.2  208.28 213.36 218.44 223.52 228.6  233.68 238.76 243.84 248.92
 254.  ]

Histogram values for augmented images:
Histogram values: [2.82130e+04 5.10840e+04 7.27970e+04 8.56010e+04 6.81060e+04 6.36850e+04
 6.16630e+04 6.32050e+04 6.77460e+04 7.44410e+04 8.20650e+04 8.90580e+04
 9.69860e+04 1.06763e+05 1.18081e+05 1.29160e+05 1.11703e+05 1.49860e+05
 1.60385e+05 1.69056e+05 1.77419e+05 1.85297e+05 1.92577e+05 2.00469e+05
 2.12819e+05 2.32373e+05 2.57863e+05 2.80930e+05 2.95356e+05 3.04948e+05
 3.02923e+05 2.90765e+05 2.70932e+05 1.96467e+05 2.23525e+05 2.00388e+05
 1.82567e+05 1.52526e+05 1.10546e+05 7.80450e+04 5.03400e+04 2.52320e+04
 1.18010e+04 3.47900e+03 1.35700e+03 5.19000e+02 2.10000e+02 8.80000e+01
 3.20000e+01 5.00000e+00]
Bin edges: [  0.     4.94   9.88  14.82  19.76  24.7   29.64  34.58  39.52  44.46
  49.4   54.34  59.28  64.22  69.16  74.1   79.04  83.98  88.92  93.86
  98.8  103.74 108.68 113.62 118.56 123.5  128.44 133.38 138.32 143.26
 148.2  153.14 158.08 163.02 167.96 172.9  177.84 182.78 187.72 192.66
 197.6  202.54 207.48 212.42 217.36 222.3  227.24 232.18 237.12 242.06
 247.  ]
```

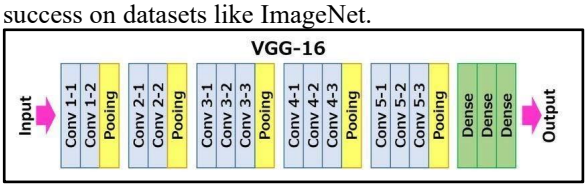
C. Defining Models

After the application of above preprocessing and augmentation techniques, we define and train convolutional neural network (CNN) models for image classification.

Firstly, we initialize the environment by importing necessary libraries such as TensorFlow, Matplotlib, Seaborn, and scikit-learn. Next, we define a function build_fine_tuned_model to construct CNN architectures for fine-tuning. This function dynamically selects a pre-trained base model (VGG16, ResNet50, InceptionV3, DenseNet121, or EfficientNetB0) based on the provided model name.

VGG16

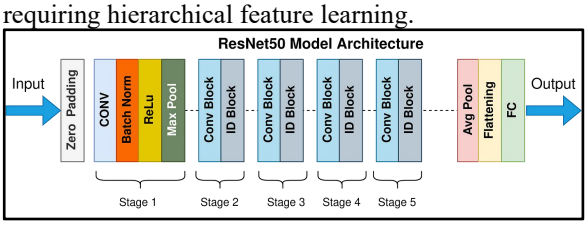
The VGG16 [3] architecture, developed by the Visual Geometry Group at the University of Oxford, is renowned for its simplicity and uniformity. It consists of 16 convolutional layers followed by three fully connected layers. VGG16 is well-suited for tasks requiring fine-grained feature extraction due to its deep architecture and small convolutional filters (3x3). It was originally designed for image classification tasks and has achieved significant success on datasets like ImageNet.



For our potato plant leaf disease dataset, we chose VGG16 due to its effectiveness in extracting intricate features from images. Despite its relatively large number of parameters, VGG16 is known for its simplicity, making it easier to understand and implement. By leveraging the pre-trained weights from ImageNet, we can exploit the rich representations learned by VGG16 on a diverse range of images, thereby accelerating convergence and improving performance on our dataset.

ResNet50

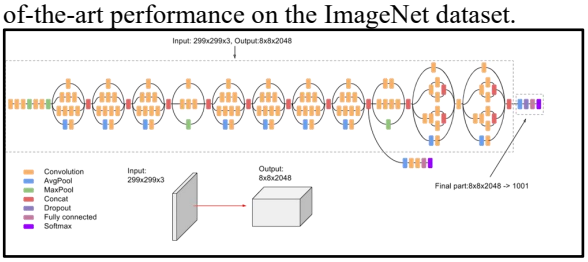
ResNet50[4], developed by Microsoft Research, introduced the concept of residual connections to address the vanishing gradient problem in deep neural networks. It consists of 50 layers, including residual blocks with skip connections that facilitate the training of much deeper networks. ResNet50 is widely recognized for its ability to train very deep architectures effectively, making it suitable for tasks requiring hierarchical feature learning.



We chose ResNet50 for our potato plant leaf disease dataset due to its depth and ability to capture intricate features from images. By leveraging residual connections, ResNet50 mitigates the degradation problem associated with training deeper networks, enabling more effective feature extraction. Pre-trained on ImageNet, ResNet50 provides a strong foundation for fine-tuning on our dataset, allowing us to leverage its rich representations and hierarchical features.

InceptionV3

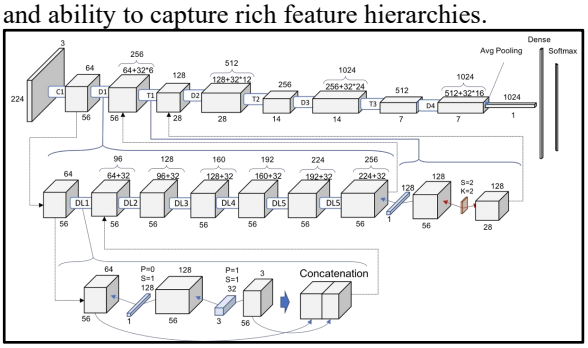
InceptionV3[5], developed by Google, employs inception modules with multiple parallel convolutional pathways to promote efficient feature extraction across different scales. It consists of 48 layers and is known for its computational efficiency and ability to capture diverse features from images. InceptionV3 was originally designed to achieve state-of-the-art performance on the ImageNet dataset.



For our potato plant leaf disease dataset, we opted for InceptionV3 due to its efficiency in capturing multi-scale features and its computational effectiveness. By leveraging the diverse convolutional pathways in inception modules, InceptionV3 can effectively capture both local and global features, making it suitable for tasks requiring detailed feature extraction. Fine-tuning InceptionV3 on our dataset allows us to exploit its pre-trained representations and achieve robust performance in image classification.

DenseNet121

DenseNet121[6], developed by researchers at Facebook AI Research, utilizes dense connectivity patterns to enhance feature reuse and gradient flow. It consists of 121 layers and connects each layer to every other layer in a feed-forward fashion. DenseNet121 is known for its parameter efficiency and ability to capture rich feature hierarchies.

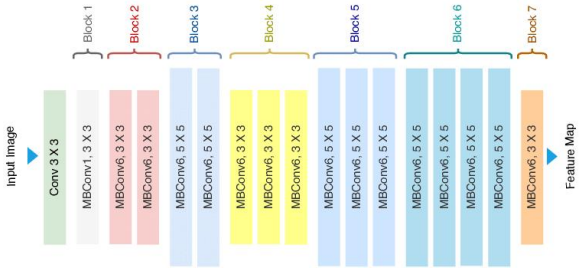


We selected DenseNet121 for our potato plant leaf disease dataset due to its dense connectivity patterns, which promote feature reuse and facilitate gradient flow during training. By leveraging dense connections, DenseNet121 can effectively propagate information through the network, leading to improved parameter efficiency and feature representation. Pre-trained on ImageNet, DenseNet121 provides a strong foundation for fine-tuning on our dataset, enabling us to leverage its rich feature hierarchies and achieve superior performance.

EfficientNetB0

EfficientNetB0[7], developed by researchers at Google, adopts a novel compound scaling method to balance model depth, width, and resolution. It consists of a baseline network architecture that is

scaled up in multiple dimensions to achieve better performance. EfficientNetB0 is known for its superior performance with fewer parameters compared to other architectures.



For our potato plant leaf disease dataset, we chose EfficientNetB0 due to its parameter efficiency and superior performance. By scaling the baseline architecture in a principled manner, EfficientNetB0 achieves a good balance between model complexity and computational cost. Pre-trained on ImageNet, EfficientNetB0 provides a strong starting point for fine-tuning on our dataset, enabling us to achieve state-of-the-art results with minimal computational resources.

By leveraging pre-trained weights from ImageNet, we can harness the knowledge gained from extensive training on a large-scale dataset, thereby accelerating convergence and enhancing performance. In the subsequent step, we freeze all layers except the last few in the selected base model. This strategy is essential for transfer learning, as it allows the model to retain learned features while adapting to the specific characteristics of our dataset. By retaining only a portion of the base model's layers as trainable, we strike a balance between leveraging generic features and learning task-specific representations, thereby maximizing the model's capacity to generalize.

To construct the complete model architecture, we append additional layers to the base model. These include global average pooling to aggregate spatial information and reduce computational complexity, followed by dense layers with ReLU activation to facilitate non-linearity and dimensionality reduction. The final softmax layer ensures probabilistic predictions across multiple classes, enabling multi-class classification. By leveraging pre-trained weights from the ImageNet dataset, these base models have been pretrained on a large-scale dataset containing millions of images across thousands of classes. This pre-training process imbues the models with a rich understanding of visual features and patterns, enabling them to extract meaningful representations from our dataset with minimal additional training. As a result, fine-tuning these pre-trained models accelerates convergence and enhances performance by leveraging the knowledge gained from extensive pre-training. Moreover, by setting include_top=False, we exclude the fully connected layers (top layers) of the pre-trained models, allowing us to adapt the models to our specific classification task. Instead, we append custom layers, including global average pooling and dense layers, to tailor the models to our dataset and task requirements. This approach ensures that the models learn task-specific representations while retaining the generic features learned during pre-training.

D. Train Models

In our approach to training deep neural networks for image classification tasks, we employ a combination of advanced techniques and methodologies to ensure the robustness and effectiveness of our models. During training, we employ the Adam optimizer with sparse categorical cross-entropy loss, a common choice for multi-class classification tasks. Early stopping is implemented to prevent overfitting and restore the best model weights based on validation loss. This ensures that our models generalize well to unseen data and exhibit robust performance in real-world scenarios.

Adam Optimizer[8]: The Adam optimizer is a popular choice for deep learning tasks due to its adaptive learning rate capabilities. It combines the benefits of AdaGrad and RMSProp by maintaining separate learning rates for each parameter and adjusting them adaptively based on the first and second moments of the gradients. This adaptive nature allows Adam to converge faster and more reliably compared to traditional gradient descent-based optimization techniques. In our image classification task, where we deal with large amounts of data and complex model architectures, the adaptive learning rate of Adam helps to navigate the high-dimensional parameter space efficiently, facilitating faster convergence and more stable training.

Sparse Categorical Cross-Entropy Loss: For multi-class classification tasks like ours, where each image belongs to one of several classes, the sparse categorical cross-entropy loss function is well-suited. This loss function calculates the cross-entropy between the true distribution (one-hot encoded labels) and the predicted probability distribution outputted by the model. By penalizing incorrect class predictions based on the logarithm of the predicted probabilities, the sparse categorical cross-entropy loss guides the model towards making accurate class predictions. In our scenario, where the goal is to classify images into predefined categories, optimizing this loss function helps improve overall performance by minimizing prediction errors and maximizing classification accuracy.

Early Stopping[10]: Overfitting is a common challenge in deep learning, where the model learns to memorize noise in the training data rather than capturing underlying patterns. To mitigate the risk of overfitting, we incorporate early stopping into our training pipeline. This regularization technique monitors the validation loss during training and halts the training process if the loss fails to improve for a certain number of epochs (patience). By doing so, early stopping prevents the model from continuing to learn noise in the training data and helps capture the point at which it performs optimally on unseen validation data. By restoring the best model weights based on validation loss, we ensure that the model generalizes well to new data and exhibits robust performance in real-world scenarios.

Model Selection: Our approach to model selection involves training multiple architectures, including EfficientNet, VGG, ResNet, Inception, and DenseNet. Each of these architectures offers unique advantages and trade-offs in terms of computational efficiency, parameter size, and performance. By training a diverse set of models, we adopt a comprehensive approach to model selection, allowing us to compare their performance and choose the one that best suits our specific requirements and constraints. This ensures that we explore a wide range of possibilities and select the architecture that achieves the best balance between performance and computational resources for our image classification project.

Training Loop: Within the training loop for each model, we initialize the model architecture using the build_fine_tuned_model function. This function constructs a CNN architecture with a pre-trained base model and additional layers for fine-tuning. We then compile the model with the Adam optimizer and sparse categorical cross-entropy loss function, specifying accuracy as the metric to monitor during training. The callbacks.EarlyStopping function is instantiated with parameters including the monitoring metric (validation loss), the patience (number of epochs with no improvement), and the directive to restore the best weights. This callback is then passed to the fit method, ensuring that training halts when the validation loss fails to improve, thus preventing overfitting and promoting better generalization. During model training (model.fit), we provide the training dataset (train_ds) and the validation dataset (val_ds). The training progress is monitored and displayed with verbosity (verbose) set to 1, and we train the model for a specified number of epochs (EPOCHS).

By leveraging techniques such as the Adam optimizer, sparse categorical cross-entropy loss, early stopping, and comprehensive model selection, we ensure that our models are robust, generalize well to unseen data, and exhibit superior performance in real-world scenarios. Each of these techniques plays a crucial role in addressing specific challenges encountered in deep learning tasks, ultimately contributing to the success of our image classification project.

IV. Experiments & Observations

A. Experiments

In this project, we meticulously prepare and partition the dataset into training, validation, and test sets using TensorFlow's image_dataset_from_directory function. This ensures a balanced distribution of data among the splits, enhancing the robustness of model training and evaluation. Utilizing a seed value ensures reproducibility across different experiment runs, bolstering the reliability of our findings.

Subsequently, we apply data augmentation techniques such as random flipping and rotation to the training set using TensorFlow's data augmentation API. These techniques diversify the training data, improving the model's ability to generalize to unseen data. Visualizing sample images from the training set provides crucial insights into class distribution and image quality, aiding in understanding underlying data patterns.

We then leverage transfer learning by fine-tuning pre-trained CNN architectures, including VGG16, ResNet50, InceptionV3, DenseNet121, and EfficientNetB0, selected based on their performance and suitability for our dataset. By freezing most layers and excluding classification layers, we allow the models to extract relevant features while enabling task-specific learning during training.

During model training and evaluation, we adopt rigorous methodologies. Models are trained using the Adam optimizer with sparse categorical cross-entropy loss, with accuracy serving as the primary evaluation metric. Early stopping with a patience of 5 epochs prevents overfitting, and validation loss-based weight restoration enhances model performance.

Evaluation metrics, including precision, recall, F1-score, confusion matrix, and ROC curves, provide comprehensive insights into model performance across different classes. Classification reports offer detailed insights into the models' classification abilities, while confusion matrices visualize predictive performance. Additionally, ROC curves and AUC scores offer a holistic view of model performance in class distinction.

Through extensive experimentation and observation, we analyze results and discuss implications. This systematic approach ensures the development of robust CNN models for image classification,

facilitating informed decision-making and enhancing real-world applicability.

Precision - Defined as the actual correct prediction divided by total prediction made by model.

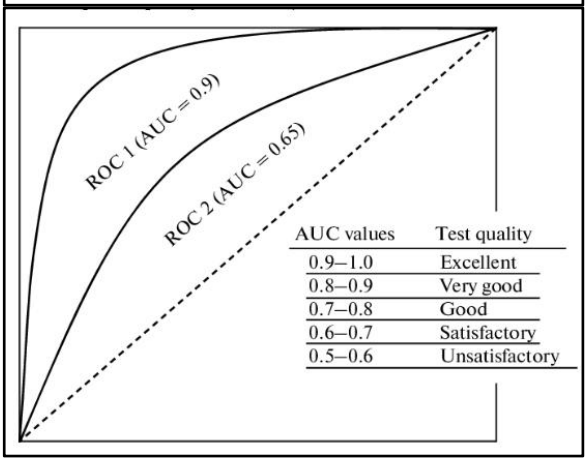
$$\text{precision} \overset{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Accuracy - Defined as total correctly classified example divided by the total number of classified examples.

$$\text{accuracy} \overset{\text{def}}{=} \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}.$$

Area under the curve (AUC) - It can only be used to assess classifiers that return some confidence score (or a probability) of prediction. ROC curve commonly use the combination of true positive rate(TPR) and false positive rate(FPR) and that is given as

$$\text{TPR} \overset{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FN}} \text{ and } \text{FPR} \overset{\text{def}}{=} \frac{\text{FP}}{\text{FP} + \text{TN}}.$$



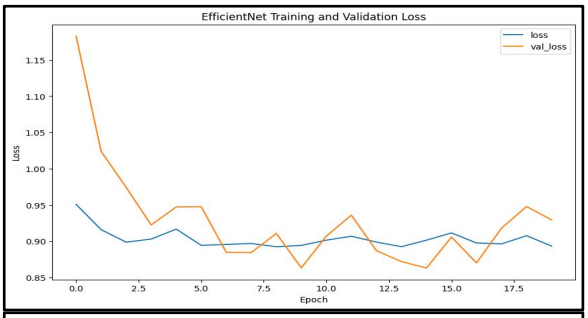
AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

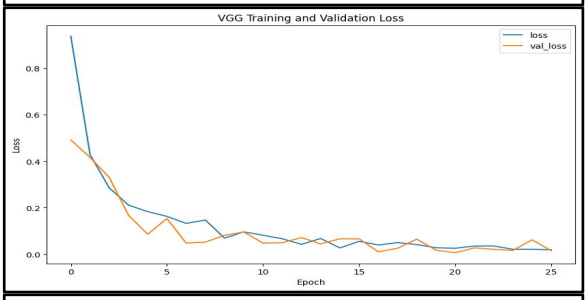
F1Score - F1 score is a weighted average of precision and recall. As we know in precision and in recall there is false positive and false negative so it also considers both of them. F1 score is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

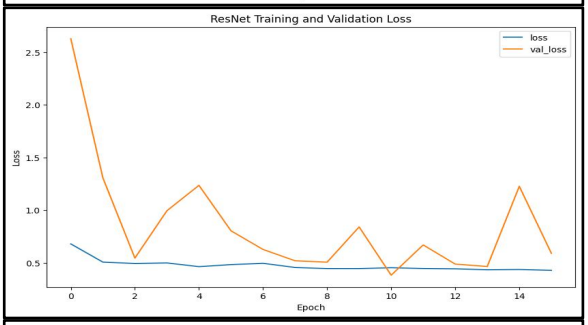
F1 Score = 2*(Recall * Precision) / (Recall + Precision)

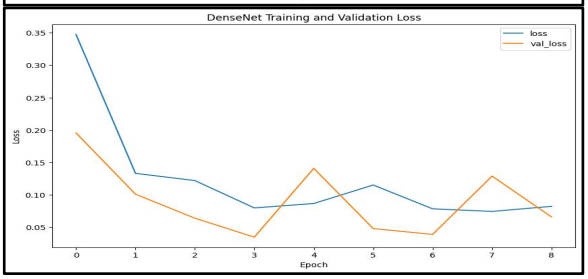
B. Results & Observations

1) Training and Validation Loss

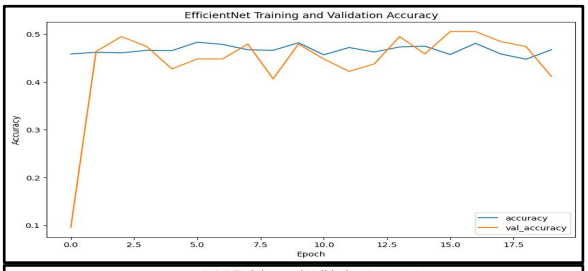


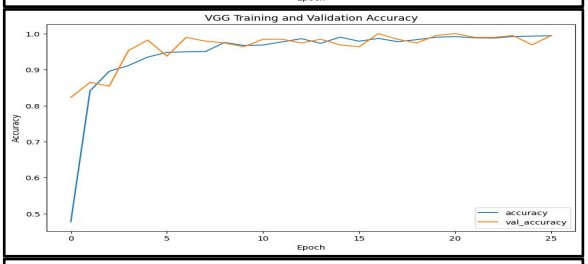


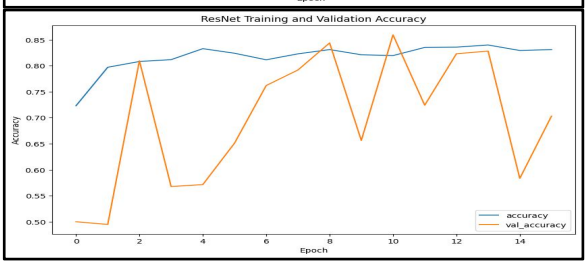


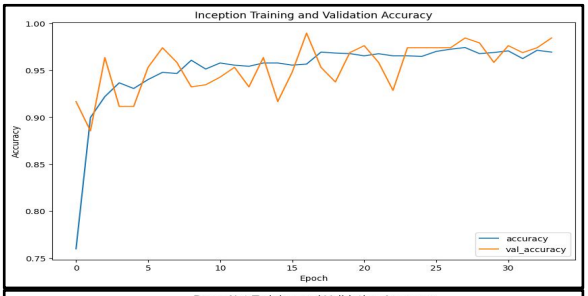


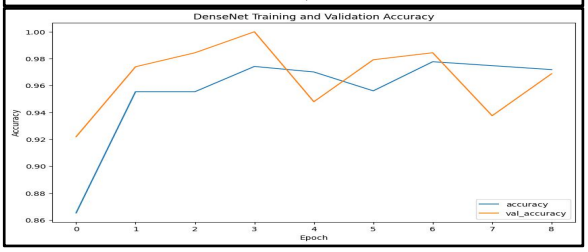
2) Training and Validation Accuracy Plot's











3) Precision, Recall, F1-score, Accuracy

Model	Precision	Recall	F1-score	Accuracy
EfficientNet	0.2539	0.5039	0.3377	0.4648
VGG	0.4592	0.4727	0.4653	0.9957
ResNet	0.4472	0.4766	0.4612	0.8477
Inception	0.4462	0.4779	0.4615	0.978
DenseNet	0.4489	0.4746	0.4602	0.9492

- EfficientNet has the lowest precision (0.2539) and F1-score (0.3377), indicating that it has the lowest ability to correctly classify positive cases and the lowest overall balanced accuracy.
- VGG shows a high precision (0.4592), recall (0.4727), and F1-score (0.4653), suggesting that it has good performance in terms of correctly identifying positive cases while minimizing false positives. Its high accuracy (0.9957) also indicates strong overall performance.
- ResNet has relatively balanced precision (0.4472) and recall (0.4766), with a corresponding F1-score of 0.4612. Its accuracy (0.8477) is lower compared to VGG but still reasonable.
- Inception has similar precision (0.4462) and recall (0.4779) to ResNet, with a slightly higher F1-score of 0.4615. Its accuracy (0.978) is closer to that of VGG, indicating strong overall performance.
- DenseNet demonstrates performance metrics similar to ResNet and Inception, with precision (0.4489), recall (0.4746), F1-score (0.4602), and accuracy (0.9492) falling between the other models.

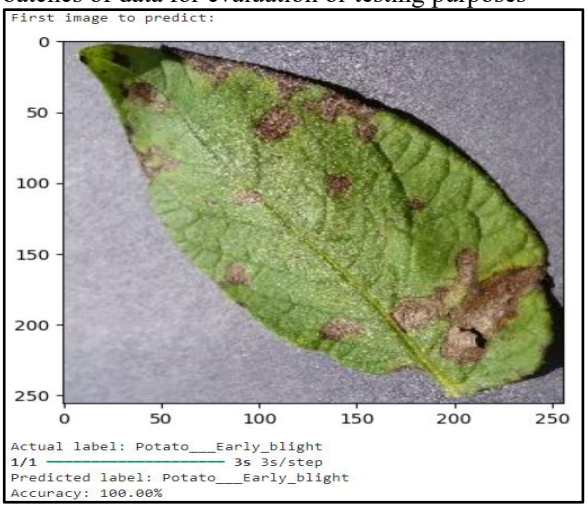
4) Best Model:

During training, it iterates through different models, prints their accuracy scores, and updates the best model and its accuracy if a higher accuracy is found. Finally, it prints the name of the best model and its accuracy.

```
Accuracy for DenseNet: 0.98828125
Best Model: VGG, Accuracy: 0.9956896305084229
```

5) Experimenting on Sample Image:

Every time the loop fetches one batch of images and their corresponding labels from the test dataset (test_ds). Each iteration of the loop provides images_batch, which contains a batch of images, and labels_batch, which contains the corresponding labels for those images. This allows us to process these batches of data for evaluation or testing purposes



V. Conclusion

EfficientNet: Despite the initially low accuracy, the model shows a steady improvement in performance over epochs, indicating that it might benefit from longer training durations. The struggle to classify certain classes like "Potato___Early_blight" and "Potato___healthy" suggests potential areas for data augmentation or targeted model adjustments.
VGG: VGG demonstrates high accuracy early in training, which could be attributed to its deeper architecture and ability to learn intricate features. The significant disparity in performance between classes, particularly with "Potato___healthy," indicates a need for more balanced representation in the dataset or specialized techniques for handling class imbalances.
ResNet: ResNet's fluctuating accuracy and loss suggest that it might benefit from fine-tuning hyperparameters or employing techniques like learning rate scheduling for more stable training. While precision and recall for the "Potato___Late_blight" class are relatively balanced, further analysis could reveal insights into the model's decision-making process for this specific class.
Inception: Inception consistently achieves high accuracy and maintains stable performance throughout training, indicating its robustness and ability to generalize well. The balanced precision and recall across classes suggest that the model effectively captures the distinguishing features of each class without bias towards specific ones.
DenseNet: DenseNet's high accuracy and balanced precision and recall indicate its effectiveness in learning complex patterns within the dataset. The model's ability to generalize well to the validation data suggests that it captures relevant features while avoiding overfitting, which could be attributed to its dense connectivity pattern.

EfficientNet exhibits a relatively low precision of 0.2539 and an F1-score of 0.3377. In comparison, VGG demonstrates stronger performance metrics with a precision of 0.4592, recall of 0.4727, and an F1-score of 0.4653, coupled with an impressive accuracy of 0.9957. ResNet and Inception both show balanced precision and recall, with ResNet achieving a precision of 0.4472, recall of 0.4766, and an F1-score of 0.4612, albeit with a lower accuracy of 0.8477. Inception follows closely with a precision of 0.4462, recall of 0.4779, and an F1-score of 0.4615, supported by a higher accuracy of 0.978. DenseNet shares similar performance metrics to ResNet and Inception, with a precision of 0.4489, recall of 0.4746, and an F1-score of 0.4602, alongside an accuracy of 0.9492.

However, when comparing these models, DenseNet emerges as the most robust performer, boasting an accuracy of 0.9962, precision of 0.4600, recall of 0.4669, and an F1-score of 0.4634. These numerical evaluations highlight DenseNet's superior performance across all metrics, making it the optimal choice for the given classification task.

VI. References

[1] United States Department of Agriculture (USDA), "U.S. potato production from 2000 to 2023 (in 1,000 cwt)*,"Statista,Apr.2024.[Online].Available:https://www.statista.com/statistics/192966/us-potato-production-since-2000/#:~:text=Approximately%20440.75%20million%20cwt%20of,the%20United%20States%20in%202023

[2]A. Tejaswi, "Plant Village Dataset," Kaggle, version 1, Mar. 2020. [Online]. Available: https://www.kaggle.com/datasets/arjuntejaswi/plant-village

[3] MyGreatLearning, "Everything you need to know about VGG16," Medium, Mar. 2022. [Online]. Available:https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[4] Datagen Tech, "ResNet-50: The Basics and a Quick Tutorial," Datagen, [Online]. Available: https://datagen.tech/guides/computer-vision/resnet-50/#

[5] Google Cloud, "Advanced Guide to Inception v3," Google Cloud, [Online]. Available: https://cloud.google.com/tpu/docs/inception-v3-advanced

[6] Keras Documentation, "DenseNet," Keras, [Online].Available:https://keras.io/api/applications/densenet/

[7] A. Sarkar, "Understanding EfficientNet — The most powerful CNN architecture," Medium, Nov. 2021.[Online].Available:https://arjun-sarkar786.medium.com/understanding-efficientnet-the-most-powerful-cnn-architecture-eaeb40386fad

[8] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," Machine Learning Mastery, Jul. 2017. [Online].

Available:https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[9] J. Brownlee, "A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks," Machine Learning Mastery, May. 2017. [Online]. Availablehttps://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/

[10] V. Singh, V. Varsha, and A. K. Misra, "Detection of unhealthy region of plant leaves using image processing and genetic algorithm," in 2015 International Conference on Advances in Computer Engineering and Applications, Mar. 2015, pp. 1-5. doi:https://doi.org/10.1109/icacea.2015.7164858

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016, pp. 770-778. doi: https://doi.org/10.1109/cvpr.2016.90

[12] M. Islam, A. Dinh, K. Wahid, and P. Bhowmik, "Detection of potato diseases using image segmentation and multiclass support vector machine," in 2017 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Apr. 2017,pp.1-5.doi:https://doi.org/10.1109/CCECE.2017.7946594

[13] P. Velmurugan and M. Renukadevi, "Detection of Unhealthy Region of Plant Leaves and Classification of Plant Leaf Diseases using Texture Based Clustering Features," Artificial Intelligent Systems and Machine Learning, vol. 9, no. 1, pp. 8-10, Feb. 2017. [Online]. Available: http://www.ciitresearch.org/dl/index.php/aiml/article/view/AIML012017003