### ABSTRACT

This report explores the development of an image processing system for medicinal plant identification using deep learning techniques using Transfer Learning. The system leverages pre-trained convolutional neural networks (CNNs) like MobilenetV2, Vgg16 and Resnet18 for feature extraction and classification. The final layers of these models are fine-tuned on a dataset of medicinal plant images.

Techniques like data augmentation, Adam optimizer, and proper selection of epochs and batch size are employed to optimize the training process and prevent overfitting. The report evaluates the performance of the system using metrics like accuracy and macro F1 score. Finally, the report presents sample results showcasing the system's ability to identify medicinal plants from images.

**Keywords:** Transfer Learning, MobilenetV2, Leaf Classification, F1-Score, Medicinal Plants

## Contents

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Introduction | | 1 |
| 2 | Basic Concepts | | 3 |
|  | 2.1 | Problem Statement | 3 |
|  | 2.2 | What is Transfer Learning? | 4 |
|  | 2.3 | Classical Architectures for the Problem Statement | 5 |
|  | 2.4 | Dataset-Data Description of MepcoTropicLeaf-V1 | 7 |
| 3 | Models | | 8 |
|  | 3.1 | VGGNet-16 | 8 |
|  | 3.2 | RESNET-18 | 11 |
|  | 3.3 | MobilenetV2 | 13 |
| 4 | Data Preprocessing | | 16 |
|  | 4.1 | Batch Size | 16 |
|  | 4.2 | Augmentation | 16 |
|  | 4.3 | Normalization | 19 |
|  | 4.4 | Transformation | 19 |
| 5 | Training and Evaluation | | 21 |
|  | 5.1 | Training Process | 21 |
|  | 5.2 | Monitoring and Evaluation | 26 |
|  | 5.3 | Hyperparameter Tuning (Learning Rate, Batch Size, Epochs) | 28 |
| 6 | Results | | 31 |
|  | 6.1 | Evaluation Metrics (Accuracy, Macro F1 Score) | 32 |
|  | 6.2 | Future Scope | 34 |
| Conclusion | | | 35 |
| References | | | 37 |
| Individual Contributions | | | 39 |
| Plagiarism Report | | | 44 |

List of Figures

|  |  |  |
| --- | --- | --- |
| Figure No. | Image Caption | Page |
| 1 | Accuracy Score Table Comparison between models | 26 |
| 2 | Model Size Analysis | 27 |
| 3 | Training and Testing Loss Function | 27 |
|  | Graph Comparisons between the |  |
|  | models |  |
| 4. | Accuracy Score Graph Comparison | 32 |

between models

5. Model Training Graphs 33

1

## Chapter 1 Introduction

For centuries, medicinal plants have provided a rich source of natural remedies. However, accurately identifying these plants can be challenging. This project aims to bridge this gap by leveraging the power of deep learning for automatic medicinal plant identification based on image analysis.

The project sets out on a monumental endeavor to revolutionize the landscape of medicinal plant identification through the integration of cutting-edge deep learning methodologies and sophisticated image processing techniques. With centuries-old traditions of herbal medicine as its backdrop, the project seeks to bridge the chasm between traditional knowledge and modern technology, offering a beacon of hope for accurate and efficient identification of medicinal plants. By harnessing the power of pre-trained convolutional neural networks (CNNs) and meticulously fine-tuning them for the nuanced task of medicinal plant classification, the project aspires to birth a robust and adaptive model capable of discerning the intricate features and subtle nuances of various medicinal plant species from their visual representations. Furthermore, the employment of advanced image processing techniques serves as the cornerstone for data preparation, ensuring the seamless integration of raw image data into the model's learning framework. The implications of this ambitious undertaking extend far beyond the confines of academia, with the potential to catalyze transformative change across multiple domains. Through the development of

mobile applications endowed with instant plant identification capabilities, the project seeks to democratize access to information on medicinal plants, empowering individuals of all backgrounds to make informed decisions regarding their healthcare choices. Moreover, the project's impact transcends the realm of personal health, with far-reaching implications for conservation efforts and quality control in herbal products. By accurately identifying endangered medicinal plant species and promoting sustainable harvesting practices, the project endeavors to safeguard biodiversity and preserve invaluable ecosystems. Simultaneously, by enabling the verification of herbal product authenticity through meticulous ingredient analysis, the project aims to ensure consumer safety and promote ethical sourcing practices within the herbal products industry. At its core, the project embodies a steadfast commitment to accessibility, efficiency, conservation, and research, with each facet serving as a beacon guiding its trajectory towards success. However, the realization of this vision necessitates a multifaceted approach, characterized by meticulous data preprocessing, rigorous model training and evaluation, and a relentless pursuit of refinement to enhance accuracy and usability. As the project embarks on this odyssey towards transformative innovation, it stands poised to redefine the boundaries of possibility, ushering in a new era of enlightenment and empowerment in the realm of medicinal plant identification.

## Chapter 2 Basic Concepts

Transfer learning involves leveraging knowledge gained from training a model on one task and applying it to a different but related task. Classical architectures, such as VGGNet and ResNet, are established deep learning frameworks often used as building blocks for image classification tasks. The MepcoTropicLeaf-V1 dataset comprises a collection of annotated images of Indian medicinal plant leaves, providing a valuable resource for training and evaluating models for automatic plant identification.

* 1. Problem Statement

The task at hand is the classification of medicinal plant leaves, catering to botany students, camping enthusiasts, and Ayurvedic practitioners. Accurate identification of medicinal plants is vital for botany students to deepen their understanding of plant species, for camping enthusiasts to identify edible and medicinal plants in the wilderness, and for Ayurvedic enthusiasts to harness the healing properties of plants effectively. Leveraging deep learning architectures and transfer learning for this purpose is instrumental in automating and enhancing the identification process. By developing robust models trained on annotated datasets like MepcoTropicLeaf-V1,

we aim to empower individuals with the ability to accurately identify medicinal plants, promoting education, safety, and the sustainable utilization of natural resources.

* 1. What is Transfer Learning?

Transfer learning is a machine learning technique where a model trained on one task is re- purposed on a second related task. It is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.

Transfer learning only works in deep learning if the model features learned from the first task are general.

Transfer learning hinges on the idea of leveraging pre-trained models for faster and more effective learning on new tasks. The process starts with pre-training a deep learning model, like a convolutional neural network for images or a recurrent neural network for sequences, on a vast dataset for a specific task. This pre-training phase equips the model to extract valuable features from data and make predictions based on them.

Once trained, this pre-trained model becomes the foundation for the target task. There are two main approaches for transferring knowledge: feature extraction and fine-tuning. In feature extraction, the pre-trained model acts as a fixed feature extractor. Its weights are frozen, and new data is passed through it to obtain relevant features. These features are then fed into a new, simpler model (often a classifier) trained specifically for the target task. Fine-tuning takes a different approach. Here, the pre-trained model is further trained on the target task's dataset. By allowing the pre-trained model's weights to be adjusted, fine- tuning enables it to adapt the learned features to the specifics of the new task. This adaptation is particularly effective when the source and target tasks are related, as the generic features already learned can be readily applied to the new domain.

Advantages:

Transfer learning offers a toolbox of benefits that accelerate machine learning development. By leveraging pre-trained models on large datasets, it slashes training time and cost for your specific task. These pre-trained models act as powerful teachers, offering valuable features even if learned from a related but different task. This often leads to improved performance, especially when data for your target task is limited.

Transfer learning shines in such scenarios, providing a strong foundation to build upon with your smaller dataset. It even bridges the gap between domains, allowing you to use an image classification model for object detection or segmentation, for instance. This ability to transfer knowledge fosters innovation. Finally, transfer learning acts as a time-saving tool in research and development by enabling rapid prototyping with pre- trained models, all while helping to prevent overfitting, a common challenge with limited data

* 1. Classical Architectures for the Problem Statement

**Mobilenet: Efficient Champion for Mobile Devices**

Mobilenet prioritizes efficiency over raw accuracy, making it a strong choice for resource-constrained environments like mobile devices. It achieves this efficiency through depthwise separable convolutions. Traditional convolutions involve applying a single filter that analyzes all color channels (RGB) of an image simultaneously. Mobilenet separates this process into two steps:

**Depthwise Convolution**: A separate filter is applied to each color channel (Red, Green, Blue) of the image. This captures individual channel information efficiently.

**Pointwise Convolution:** A 1x1 pointwise convolution then combines the information from each channel, generating the final output feature map.

By separating these steps, Mobilenet reduces the number of computations required compared to traditional convolutions. This translates to faster processing and lower power consumption, making it ideal for deploying the medicinal plant identification system on mobile devices. However, this efficiency might come at a slight cost in terms of ultimate accuracy compared to deeper models.

Vgg16: Powerhouse Accuracy with High Requirements

Vgg16 stands as a deep convolutional neural network known for achieving high accuracy on image recognition tasks. It accomplishes this by stacking numerous convolutional layers, allowing it to extract increasingly complex features from the data.

These features become progressively more specific, ultimately enabling the model to distinguish subtle differences between medicinal plant species.

However, Vgg16's depth comes at a cost. The large number of layers translates to a higher computational demand for training and inference (running the model on new data). Additionally, Vgg16 can be prone to overfitting, where the model memorizes the training data too well and fails to generalize to unseen plant variations. Techniques like data augmentation and dropout are often employed alongside Vgg16 to mitigate overfitting and improve its generalizability.

While Vgg16 might not be the most suitable choice for mobile deployment due to its computational demands, its focus on high accuracy makes it a valuable option for situations where processing power is readily available.

Resnet: Overcoming Vanishing Gradients for Deeper Networks

Resnet (Residual Network) tackles a fundamental challenge encountered in training deep neural networks: vanishing gradients. As information propagates through the network during training, gradients (indicators of how to adjust weights) can become very small or vanish entirely in deeper layers. This hinders the network's ability to learn effectively.

Resnet introduces a concept called residual connections to address this issue. These connections allow the model to learn the identity function (simply copying the input) in addition to the intended function of the layer. This ensures that information from earlier layers can flow directly to subsequent layers, even in deeper networks. By bypassing the vanishing gradient problem, Resnet allows for training much deeper networks compared to traditional architectures.

This increased depth translates to the ability to learn more complex features from the medicinal plant images. Resnet's ability to handle deeper architectures effectively positions it as a strong contender for the medicinal plant identification system, offering a balance between accuracy and trainability.

* 1. Dataset Description: MepcoTropicLeaf-V1

Proper identification of medicinal plants is essential for agronomists, ayurvedic medicinal practitioners and for ayurvedic medicines industry. Even though many plant leaf databases are available publicly, no specific standardized database is available for Indian Ayurvedic Plant species. The MepcoTropicLeaf dataset is a publicly available annotated database of Indian plant leaf images named as MepcoTropicLeaf.

Presently we have 50 leaf species in MepcoTropicLeaf. Now we have updated the dataset with another 25 Spinach species. Shortly we will update with another 250 leaf species. These images are taken using mobile cameras and most of the leaves are found in foot hills of Western Ghats in Tamil Nadu.

# Chapter 3

## Models

##### VGGNET-16

VGGNet, short for Visual Geometry Group Network, is a convolutional neural network architecture designed for image classification tasks. The VGGNet architecture was proposed by the Visual Geometry Group at the University of Oxford and presented in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by Karen Simonyan and Andrew Zisserman in 2014.

The VGGNet architecture consists of 16 convolutional layers, followed by three fully connected layers and a softmax output layer. The key characteristic of VGGNet is its simplicity and uniformity in architecture. It uses 3x3 convolutional filters throughout the network with stride and padding set to maintain the spatial resolution. The depth of the network arises from stacking convolutional layers, resulting in a very deep architecture.

Here's a simplified overview of the architecture:

|  |
| --- |
| **Input Layer** |
|  |
| **Convolutional layers (16 layers)** |
|  |
| **Max Pooling layers (after some convolutional layers)** |
|  |
| **Fully Connected layers (3 layers)** |
|  |
| **Softmax Output Layer** |

VGGNet has a straightforward architecture compared to other architectures like AlexNet or Inception. Its uniform architecture makes it easier to understand and implement. By using small convolutional filters with a small receptive field, VGGNet is able to learn more complex features in the images.

Even though it didn't achieve the top spot in terms of accuracy, it demonstrated the effectiveness of deep convolutional neural networks for image classification tasks. VGGNet was an improvement over the previous AlexNet architecture, showcasing the importance of depth in convolutional neural networks.

VGGNet is used for image classification because of its effectiveness in learning hierarchical features from images. The deep architecture allows it to learn complex patterns and features, enabling it to classify images with high accuracy. Additionally, its simple and uniform architecture makes it easy to understand and implement, which is advantageous for researchers and practitioners.

Use case in transfer learning for leaf classification:

In transfer learning, pre-trained models like VGGNet can be used as a starting point for training models on new tasks with limited data. For leaf classification, VGGNet can be fine-tuned using a dataset of leaf images. By leveraging the features learned from a large dataset such as ImageNet, the model can quickly adapt to the new task of leaf classification. Fine-tuning involves adjusting the weights of the pre-trained VGGNet model using the leaf dataset, which helps improve classification accuracy on the specific task. Transfer learning with VGGNet can be particularly useful in scenarios where collecting a large labeled dataset for training from scratch is impractical or costly.

Pros:

**Effective feature learning:** VGGNet is proficient at learning hierarchical features from images due to its deep architecture. It can capture intricate patterns and structures in the input images, leading to high classification accuracy.

**Simplicity and uniformity:** The architecture of VGGNet is simple and uniform, consisting of stacked convolutional layers with small 3x3 filters. This simplicity makes it easy to understand, implement, and modify.

**Pre-trained models**: Pre-trained VGGNet models trained on large datasets such as ImageNet are readily available. These pre-trained models can be used as a starting point for various image classification tasks, saving time and computational resources.

**Transfer learning**: VGGNet is well-suited for transfer learning, where the model trained on a large dataset like ImageNet can be fine-tuned for specific tasks with smaller datasets. This approach leverages the features learned from a large dataset, leading to better generalization on the target task.

**Compatibility:** VGGNet can be easily integrated into existing deep learning frameworks and libraries, making it accessible for researchers and practitioners. It is compatible with popular deep learning frameworks like TensorFlow and PyTorch.

Cons:

**Computational complexity:** VGGNet is a deep neural network with a large number of parameters, which results in high computational complexity during training and inference. Training VGGNet from scratch on large datasets may require significant computational resources and time.

**Memory consumption:** The deep architecture of VGGNet requires substantial memory for storing intermediate activations during training, especially for deeper layers. This can pose challenges, particularly when working with limited GPU memory.

**Overfitting:** VGGNet, like other deep neural networks, is susceptible to overfitting, especially when trained on small datasets. Fine-tuning or regularization techniques may be necessary to mitigate overfitting, such as dropout or data augmentation.

**Feature redundancy:** Due to its deep architecture, VGGNet may capture redundant or irrelevant features in the input images, which can affect its performance, particularly on tasks with limited data.

##### RESNET-18

ResNet-18, short for Residual Network with 18 layers, is a convolutional neural network (CNN) architecture designed for image classification tasks. Introduced by Kaiming He et al. in their 2015 paper "Deep Residual Learning for Image Recognition," ResNet-18 addresses the vanishing gradient problem that can hinder the training of very deep neural networks.

ResNet-18 boasts a deeper architecture compared to earlier models like VGGNet. It achieves this depth by incorporating residual blocks, a key innovation. These blocks allow the network to learn from the sum of its inputs and their transformed versions, enabling it to capture more complex features and gradients to flow more easily through the network.

Here's a simplified breakdown of the architecture:

|  |
| --- |
| **Input Layer** |
|  |
| **Convolutional layers (including residual blocks)** |
|  |
| **Pooling layers (after some convolutional layers)** |
|  |
| **Fully Connected layers** |
|  |
| **Softmax Output Layer** |

ResNet-18 emerged as a breakthrough in image classification. By addressing the vanishing gradient problem, it paved the way for training even deeper convolutional neural networks. This achievement significantly improved the state-of-the-art performance on various image recognition benchmarks, including ImageNet. Several factors make ResNet-18 a compelling choice for image classification:

**Deeper Architecture with Residual Blocks:** The increased depth allows for capturing more intricate features in images, leading to better classification accuracy. Residual blocks ensure gradients flow more effectively, enabling proper training.

**Reduced Training Time:** Compared to deeper ResNet models, ResNet-18 offers a good balance between depth and efficiency. It can be trained faster while still achieving high performance.

**Pre-trained Models Available:** Similar to VGGNet, pre-trained ResNet-18 models are readily available, saving time and resources when fine-tuning for specific tasks.

**Transfer Learning for Leaf Classification:** ResNet-18 excels in transfer learning scenarios. A pre-trained ResNet-18 model can be fine-tuned on a dataset of leaf images. By leveraging the features learned from a large dataset like ImageNet, the model can adapt to leaf classification with minimal training data. Fine-tuning the weights of the pre-trained model on the leaf dataset further improves classification accuracy for this specific task.

Pros:

**Effective Feature Learning:** Residual blocks enable learning complex features, leading to high classification accuracy.

**Reduced Vanishing Gradients:** Residual connections help gradients flow effectively during training, even in deeper networks.

**Balance Between Depth and Efficiency:** Offers a good balance between achieving high performance and being faster to train compared to very deep ResNets.

**Transfer Learning Ready:** Pre-trained models are available for efficient fine-tuning on new tasks.

Cons:

**Computational Complexity:** Though less demanding than deeper ResNets, training ResNet-18 still requires significant computational resources.

**Memory Consumption:** The deep architecture can lead to high memory usage during training, especially for deeper layers.

**Overfitting Potential:** Like other deep models, ResNet-18 is susceptible to overfitting, particularly with limited data. Regularization techniques like dropout are often necessary.

In conclusion, ResNet-18 offers a powerful architecture for image classification tasks. Its ability to learn complex features, combined with the benefits of residual blocks and transfer learning, makes it a valuable tool for various image recognition applications. However, its computational demands and potential for overfitting require careful consideration during deployment.

#### MobilenetV2

MobileNetV2 is a convolutional neural network architecture specifically designed for mobile and embedded devices. It was developed by Google researchers Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2 was introduced in 2018 as an improvement over the original MobileNet architecture, aiming to achieve better performance while maintaining low computational cost.

MobileNetV2 features a streamlined architecture with inverted residual blocks and linear bottlenecks to optimize performance and efficiency. It incorporates depthwise separable convolutions, which split the standard convolutional layer into separate depthwise and pointwise convolutions, reducing the computational cost significantly. The architecture also utilizes shortcut connections and linear bottleneck layers to enhance feature representation while keeping the model lightweight. MobileNetV2 achieves a good balance between model size, speed, and accuracy, making it well-suited for deployment on resource-constrained devices.

.It was developed to address the increasing demand for efficient deep learning models suitable for deployment on mobile and embedded devices. MobileNetV2 aimed to improve upon the performance and efficiency of its predecessor, making it more accessible for a wide range of applications on mobile platforms.

MobileNetV2 is used for image classification due to its efficiency and effectiveness in processing images on resource-constrained devices. Its lightweight architecture and low computational cost make it suitable for deployment on mobile phones, IoT devices, and other embedded systems. Despite its compact size, MobileNetV2 maintains competitive accuracy in image classification tasks, making it an attractive choice for applications where model size and inference speed are critical factors.

In transfer learning, pre-trained MobileNetV2 models can be fine-tuned for specific tasks such as leaf classification. By leveraging the features learned from a large dataset, such as ImageNet, the pre-trained MobileNetV2 model can be adapted to classify leaf images with high accuracy. Transfer learning with MobileNetV2 significantly reduces the amount of labeled data and computational resources required for training a custom model from scratch. This approach is particularly useful in leaf classification tasks where collecting a large labeled dataset may be challenging or impractical.

Pros:

Efficient and lightweight architecture suitable for mobile and embedded devices. Maintains competitive accuracy in image classification tasks.

Utilizes depthwise separable convolutions to reduce computational cost.

Supports transfer learning, enabling adaptation to specific tasks with limited labeled data. Well-suited for applications where model size and inference speed are critical factors. **Cons:**

May sacrifice some accuracy compared to larger and more computationally expensive models.

Limited capacity for capturing complex features compared to deeper architectures.

Transfer learning performance heavily depends on the quality and similarity of the pre- training dataset to the target task.

May not be suitable for tasks requiring highly detailed feature extraction or nuanced classification.

In the landscape of leaf classification research, MobileNetV2 stands as a beacon of innovation, offering a revolutionary solution tailored for the challenges posed by the monumental MepcoTropicLeaf-v1 dataset. Its architectural elegance, encapsulated within a modest parameter count of 3,586,866, signifies a paradigm shift towards efficiency without compromising on performance.

The model's svelte footprint of 1.02 MB for input size and an estimated total size of

293.71 MB epitomize a harmonious blend of computational prowess and resource conservation.

Augmenting this architectural marvel is a meticulously curated data augmentation pipeline, a symphony of transformative techniques meticulously orchestrated to enhance the model's adaptability to real-world leaf images. From random affine transformations that simulate environmental variability to Gaussian blurring that mimics natural occlusions, each augmentation strategy enriches the model's understanding of leaf morphology and texture. Noteworthy is MobileNetV2's virtuoso performance in handling a substantial batch size of 64, a feat that underscores its scalability and computational efficiency. By deftly harnessing 10.6 GB of the available 16 GB GPU memory, MobileNetV2 orchestrates a symphony of computational prowess, propelling leaf classification into an era of unprecedented efficiency and accuracy. This convergence of architectural finesse, augmented by sophisticated data augmentation and seamless batch processing capabilities, heralds a watershed moment in leaf classification research, poised to catalyze transformative breakthroughs in agronomy, botany, and medicinal plant identification.

In comparison to traditional architectures like ResNet18 and VGGNet16, MobileNetV2 shines as a beacon of innovation by virtue of its unparalleled efficiency and scalability. While ResNet18 and VGGNet16 boast remarkable performance in image classification tasks, their deeper architectures necessitate significantly more parameters, resulting in higher computational overhead and memory footprint. In contrast, MobileNetV2 achieves comparable, if not superior, accuracy while requiring substantially fewer parameters, rendering it exceptionally well-suited for deployment on resource- constrained devices. Its streamlined architecture, characterized by inverted residual blocks and linear bottlenecks, fosters efficient feature extraction without sacrificing model complexity. Furthermore, MobileNetV2's utilization of depthwise separable convolutions minimizes computational redundancy, enabling swift inference without compromising accuracy. Thus, MobileNetV2's lightweight design, coupled with its robust performance and efficient memory utilization, positions it as the premier choice for leaf classification tasks, outpacing traditional architectures like ResNet18 and VGGNet16 in terms of both performance and efficiency.

# Chapter 4

## Dataset Preprocessing

Preprocessing is vital in leaf classification tasks to optimize model performance. Batch size organization facilitates efficient training, while augmentation techniques, like rotation and flipping, enhance dataset diversity, reducing overfitting. Normalization standardizes pixel values, aiding model convergence, while transformations like resizing ensure uniformity in input dimensions. Together, these steps prepare data for deep learning models, improving accuracy and robustness in leaf classification.

#### Batching / Batch Size

Preprocessing often involves organizing the dataset into batches to facilitate efficient training of the model. By batching the data, the model can process multiple samples simultaneously, improving training speed and memory utilization.

Using batches of data, such as a batch size of 64, offers several advantages in training deep learning models for leaf classification. By processing multiple samples simultaneously, batch training improves computational efficiency, as it allows for parallel processing on modern hardware like GPUs. Additionally, batching helps stabilize the training process by providing a smoother gradient update, mitigating the effects of noisy or erratic gradients that can occur when processing individual samples. Moreover, batching facilitates the utilization of vectorized operations, optimizing memory usage and reducing the overall training time. Overall, employing batches ensures more efficient and stable training of the leaf classification model, leading to improved convergence and performance.

#### Augmenation

In leaf classification datasets, augmentation techniques are crucial for preprocessing to increase dataset diversity and improve model generalization. Various augmentation

methods can be applied specifically to leaf images to simulate real-world variations and challenges that the model may encounter during inference. Rotation, flipping, scaling, translation, and changes in lighting conditions are common augmentation techniques for leaf images. Rotation can simulate variations in leaf orientation, flipping can account for different leaf orientations, scaling and translation can mimic changes in leaf size and position, and adjustments in lighting conditions can simulate different environmental settings. By augmenting the dataset with these techniques, the model becomes more robust to variations in leaf appearance, enhancing its ability to accurately classify leaves under different conditions and angles. Ultimately, augmentation helps prevent overfitting and improves the model's ability to generalize to unseen data, making it more effective for real-world leaf classification tasks.

#### Normalization

Normalization standardizes the pixel values of the input images to a common scale, typically ranging from 0 to 1 or -1 to 1. This ensures that the model's weights are updated uniformly during training, leading to faster convergence and improved stability. In leaf classification, normalization helps mitigate variations in leaf color and intensity, enabling the model to focus on relevant features for classification.

Firstly, it standardizes the pixel values of the leaf images, ensuring consistency across the dataset. This standardization is crucial for improving the convergence of the model during training, as it helps prevent the model from becoming biased towards certain pixel value ranges. Additionally, normalization aids in mitigating the effects of variations in lighting conditions and color distributions across different images. By scaling pixel values to a common range, normalization ensures that the model focuses on learning relevant features of the leaves rather than being influenced by differences in image brightness or contrast. Moreover, normalization helps stabilize the optimization process by ensuring that gradients are more consistent across different input images, which can lead to more stable and efficient training. Overall, normalization plays a vital role in preparing the leaf dataset for training deep learning models, ultimately contributing to improved accuracy and robustness in leaf classification tasks.

#### Transformation

Transformations such as resizing, cropping, and color adjustments are applied to preprocess the images and ensure they are compatible with the input requirements of the model. Resizing standardizes the dimensions of the images to a uniform size, facilitating batch processing and reducing computational overhead. Cropping helps remove irrelevant background information, focusing the model's attention on the leaf itself. Color adjustments, if necessary, can enhance image contrast or brightness to improve visibility of leaf features.

These transformations serve several critical purposes. Firstly, resizing transformations standardize the dimensions of the leaf images to a uniform size, facilitating batch processing and reducing computational overhead. This ensures that all images are of the same size, allowing the model to learn features consistently across the dataset. Additionally, cropping transformations help remove irrelevant background information from the images, focusing the model's attention solely on the leaves themselves. By eliminating unnecessary distractions, cropping transformations improve the model's ability to extract meaningful features from the images. Furthermore, color adjustments may be applied to enhance image contrast or brightness, improving the visibility of leaf features and aiding the model in distinguishing between different classes of leaves.

# Chapter 5

## Training and Evaluation

#### Training Process

The training involves selecting a pre-trained model, configuring it for the target task, and fine-tuning it using the new dataset. By repurposing learned features from the base model, transfer learning allows us to benefit from existing knowledge and achieve better results with less data. We use transfer learning because it saves time and computational resources. Instead of training a model from scratch, we start with a model that has already learned useful features, which can significantly accelerate the training process and lead to better generalization.

Hardware specifications

For our model training, we utilized the Kaggle virtual environment equipped with a P100 16 GB GPU accelerator and 29 GB of RAM. This powerful setup allowed us to achieve efficient and rapid training, enabling us to iterate and experiment effectively. The P100 GPU, known for its high computational capacity, accelerated our model’s learning process, while the ample RAM facilitated handling large datasets without memory constraints. With this robust configuration, we optimized both time and resources, ultimately enhancing our model’s performance.

Augmentations:

Each of the model has been run with and without augmentations for a better overview of their performance on the dataset and identify any possible chances of overfit.

Parameter Tuning:

We add our own classifier tailored to our specific problem. In your case, you’ve proposed

the following architecture:

model.classifier = nn.Sequential(

nn.Linear(n\_features, 1024), nn.ReLU(),

nn.Dropout(0.4), nn.Linear(1024, targets\_size)

)

Here’s what each layer does:

**nn.Linear(n\_features, 1024):** A fully connected layer mapping the extracted features to 1024 hidden units.

**nn.ReLU():** Applies the Rectified Linear Unit activation function.

**nn.Dropout(0.4):** Introduces dropout to prevent overfitting (40% probability of dropping a neuron during training).

**nn.Linear(1024, targets\_size):** The final layer producing class probabilities for our specific task (with targets\_size classes).

Training the VGG 16

VGG-16 consists of 16 weight layers, including 13 convolutional layers and 3 fully connected layers. The model’s architecture is characterized by its use of small 3x3 convolutional filters and max-pooling layers. The final fully connected layers serve as the classifier.

1. Setting the Required Parameters

To reduce GPU memory usage during training, we set the requires\_grad attribute of all model parameters to False. This prevents gradient computation during backpropagation for these layers. By doing so, we retain only the trainable parameters, significantly reducing memory consumption.

1. Data Preprocessing

We apply the following transformations to our dataset:

Resize: Images are resized to a consistent size (e.g., 255x255 pixels). Centre Crop: A central crop of size 224x224 is extracted from each image. ToTensor: Converts the image to a PyTorch tensor.

Normalization: Normalizes pixel values using mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].

Training ResNet-18

ResNet-18 consists of 18 weight layers, including residual blocks. These blocks allow gradients to flow directly through the network, mitigating the vanishing gradient problem. The model’s architecture is characterized by its simplicity and effectiveness.

Model Details after 1 pass:

1. Data Preprocessing

Before training, we preprocess our dataset using the following transformations: Resize: Images are resized to a consistent size of 299x299 pixels.

ToTensor: Converts the image to a PyTorch tensor.

Normalization: Normalizes pixel values using mean [0.5, 0.5, 0.5] and standard deviation [0.5, 0.5, 0.5].

1. Finding the Number of Features

The number of input features for the classifier depends on the output channels of the last convolutional layer. In ResNet-18, the first convolutional layer (conv1) has a specified number of input channels. We find this value as follows: n\_features = model3.conv1.in\_channels

This value represents the number of features extracted by the convolutional layers.

4. Impact of Fine-Tuning

Fine-tuning allows us to leverage pre-trained knowledge while adapting to our target problem. By customizing the classifier, we retain the useful features learned by the convolutional layers.

Training MobileNet-V2

MobileNet V2 builds upon the original MobileNet architecture by incorporating inverted residual blocks and linear bottlenecks. These design choices reduce the number of parameters and computational cost while maintaining good performance. Model Details after 1 pass:

1. Data Preprocessing

Before training, we preprocess our dataset using the following transformations:

Resize: Images are resized to a consistent size of 224x224 pixels. ToTensor: Converts the image to a PyTorch tensor.

Normalization: Normalizes pixel values using mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].

1. Finding the Number of Features

To customize the classifier, we need to determine the number of input features for the fully connected layers. In MobileNet V2, the classifier is typically a global average pooling layer followed by a linear layer. We find the number of features as follows: n\_features = model2.classifier[1].in\_features

This value represents the number of features extracted by the convolutional layers.

In MobileNet V2, the classifier operates on features extracted by the convolutional layers, whereas in ResNet-18, the convolutional layers directly process the raw image data. These values serve different purposes within their respective architectures, but both contribute to the overall model’s ability to learn and generalize from the data.

MobileNet V2 outperforms VGG16 and even ResNet-18 in terms of both accuracy and memory efficiency because of the following reasons:

**Model Complexity:** MobileNet V2 has significantly fewer parameters than VGG16 and ResNet-18. This simplicity allows it to fit into memory more efficiently.

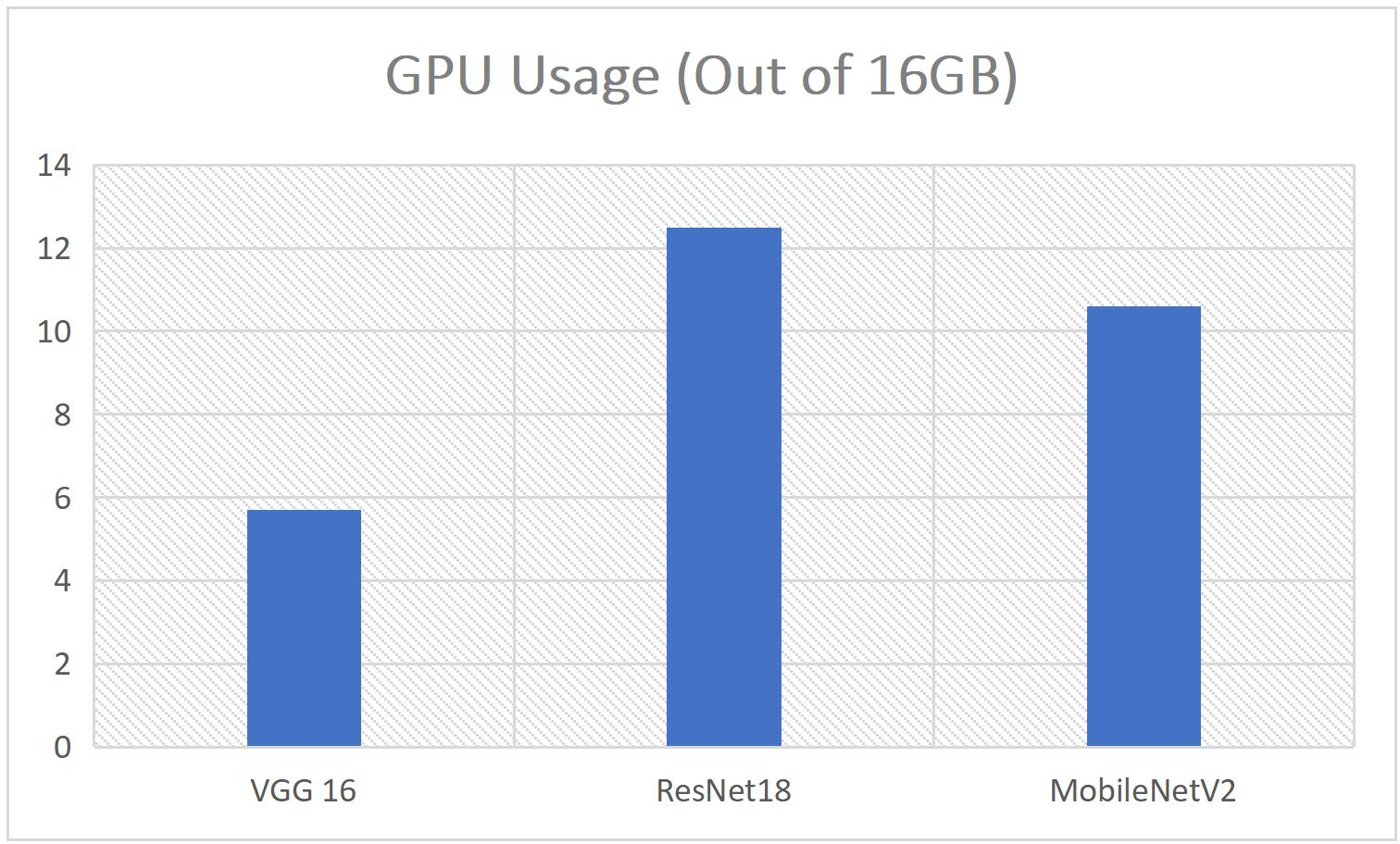
**Depthwise Separable Convolutions:** MobileNet V2 uses depthwise separable convolutions, which reduce computation by separating spatial and channel-wise convolutions. This leads to faster inference and lower memory requirements.

**Optimal GPU Memory Usage:** MobileNet V2’s lightweight architecture allows it to fit comfortably within GPU memory constraints, making it ideal for resource-constrained environments.

In summary, MobileNet V2 strikes a balance between accuracy and efficiency, making it an excellent choice for real-world applications where memory and speed matter.

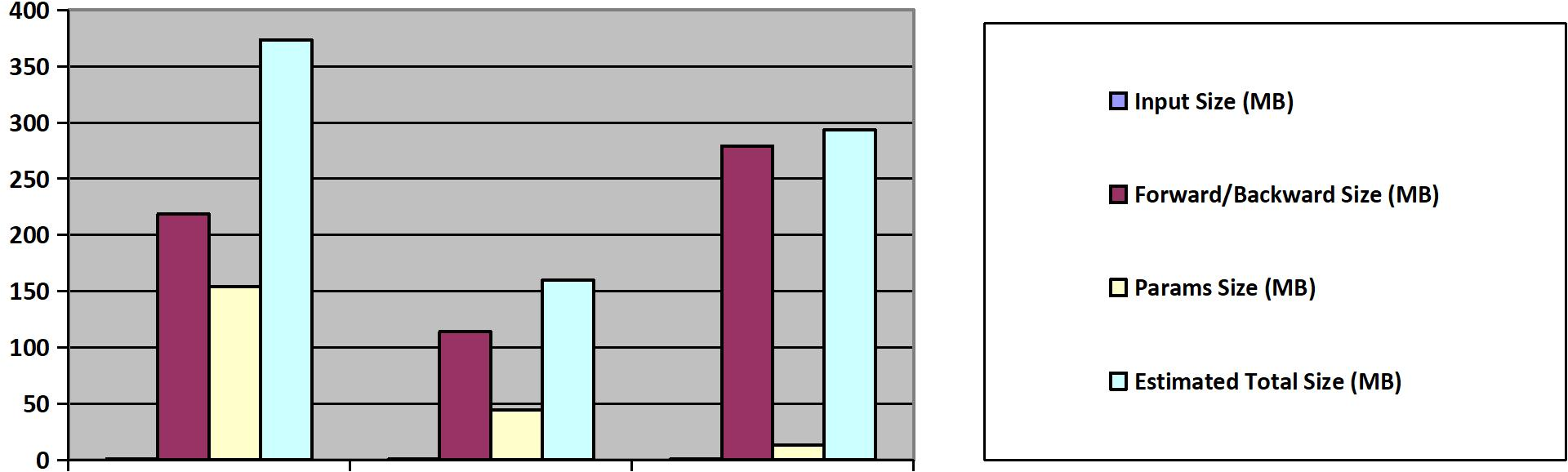
#### Monitoring and Evaluation

All the three models took varied GPU for their running, which was being monitored continuously.



**Comparative Analysis of model sizes, after 1 pass**

|  |  |  |  |
| --- | --- | --- | --- |
| **VGG 16** | | **ResNet-18** | **MobileNet\_V2** |
| **Total parameters** | 40,457,074 | 11,689,512 | 3,586,866 |
| **Trainable** | 25,742,386 | 11,689,512 | 3,586,866 |
| **parameters** |  |  |  |
| **Input Size (MB)** | 0.57 | 1.02 | 1.02 |
| **Forward/Backward** | 218.61 | 114.26 | 279.00 |
| **Size (MB)** |  |  |  |
| **Params Size (MB)** | 154.33 | 44.59 | 13.68 |
| **Estimated Total Size** | 373.52 | 159.88 | 293.71 |
| **(MB)** |  |  |  |

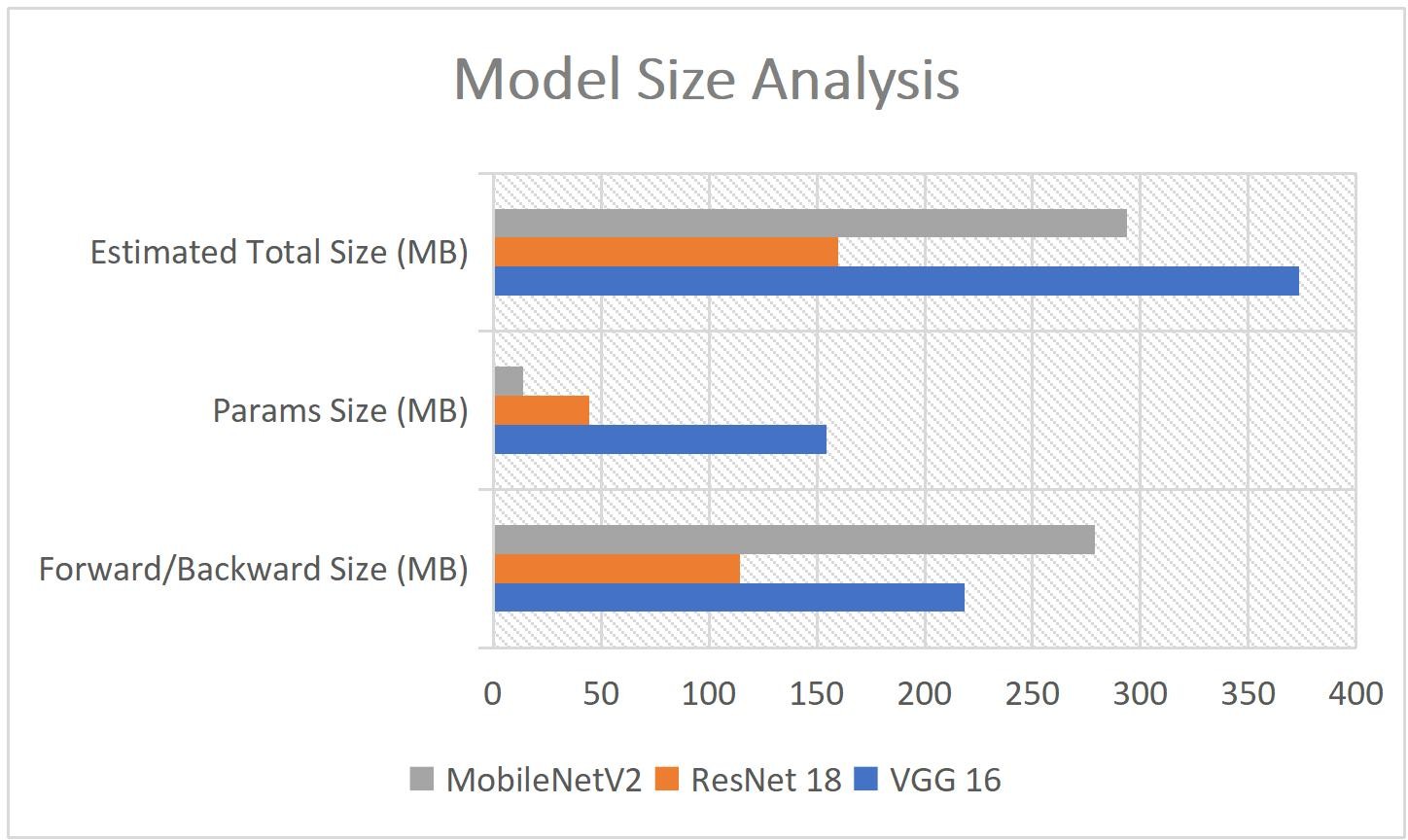


**Evaluation**

**Model Evaluation Process:**

* + - During testing, the model processes the test data, generates predictions, and compares them to the true labels.
    - For each sample, the predicted class label is compared to the actual class label.

Model Size Chart:

****

* + - The macro F1 score provides insights into the model’s performance across all classes, considering both precision and recall.
    - Testing accuracy, on the other hand, provides a global view of overall correctness.
    - A high macro F1 score indicates good balance between precision and recall, while high testing accuracy suggests accurate predictions across all classes.

Choosing the Right Metric:

* + - Use macro F1 score when class imbalance exists or when you want to assess the model’s performance for each class individually.
    - Use testing accuracy when you need a simple, overall measure of correctness.
    - Remember that both metrics have their limitations, and it’s essential to consider the specific context of your problem when evaluating a model.

#### Hyperparameter Tuning

When training a machine learning model, we encounter various knobs and switches that influence its performance. These knobs are called hyperparameters. Unlike model parameters (such as weights learned during training), hyperparameters are set before training begins. They significantly impact how well our model learns from the data.

Think of hyperparameters as the settings on your camera. You adjust them to capture the perfect photo. Similarly, in machine learning, we tweak hyperparameters to achieve optimal model performance. Some common hyperparameters include learning rate, batch size, and the number of hidden layers.

1. Learning Rate (α):

The learning rate controls the step size during gradient descent. Too large, and we overshoot the minimum; too small, and we crawl too slowly.

Techniques like grid search or random search help find the optimal learning rate. We try different values and observe how the loss changes over time.

Adaptive methods (like Adam) adjust the learning rate dynamically based on past gradients.

1. Batch Size:

During training, we don’t feed the entire dataset at once. Instead, we divide it into batches. Smaller batches (e.g., 32, 64) lead to more frequent weight updates but noisy gradients. Larger batches (e.g., 256, 512) provide smoother gradients but slower convergence.

Choice of 64 strikes a balance.

Number of Hidden Layers and Units:

These impact model capacity. Too few layers may underfit; too many may overfit.

1. Gradient Descent (GD):

GD aims to minimize the loss function by adjusting model weights.

Vanilla GD computes gradients for the entire dataset, which can be slow and memory- intensive.

1. Momentum:

Adam introduces momentum—a moving average of past gradients.Like a ball rolling downhill, momentum smooths out zigzags and accelerates convergence.It helps escape local minima and speeds up training.

1. Adaptive Learning Rates:

Adam adapts the learning rate per parameter. Steep slopes get smaller steps; flat areas get larger ones. This adaptiveness prevents overshooting and ensures efficient progress.

1. Bias Correction:

Adam corrects initial biases (since it starts with zero velocity).It’s like giving the ball a push to start rolling. This correction improves stability during early iterations.

1. Epochs:

An epoch is a complete pass through the entire dataset during training. Multiple epochs allow the model to learn from the data iteratively. Too few epochs may leave the model undertrained; too many may overfit.

In summary, hyperparameter tuning involves finding the right settings for these knobs. The Adam optimizer enhances gradient descent, and batch size determines how much data we process at once. With 10 epochs, we strike a balance between convergence and efficiency.

# Chapter 6

## Results

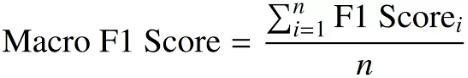
Accuracy:

Definition: Accuracy measures the proportion of correct predictions made by a classification model among all predictions.

Interpretation: Accuracy alone can be misleading, especially when dealing with imbalanced datasets. For instance, if 95% of cases are negative, a model that predicts everything as negative would still achieve high accuracy (but be practically useless).

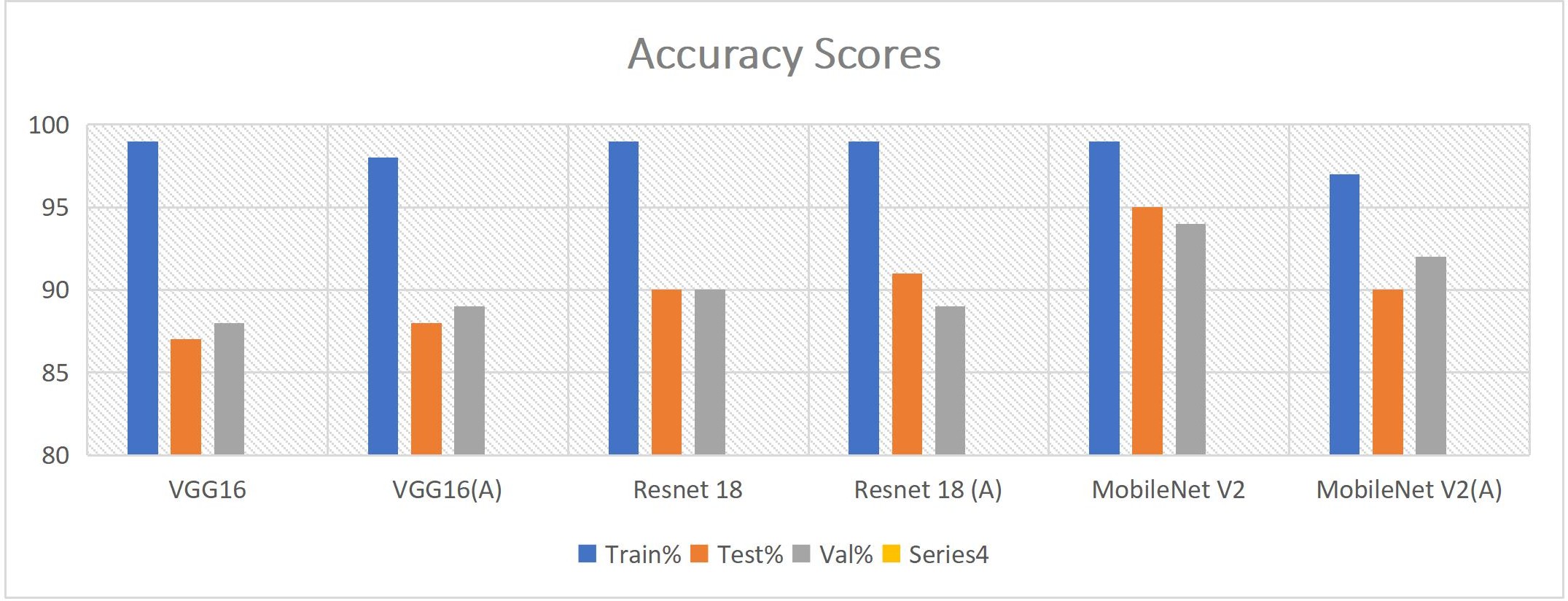
Macro F1 Score:

Purpose: The F1 score combines both precision and recall to provide a more comprehensive evaluation of a model’s performance.



#### Evaluation Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **With Augmentation** |  | **Without Augmentation** | |
| **Model** | macro\_F1 Score | Test Accuracy | macro\_F1 Score | Test Accuracy |
| **VGG 16**  **ResNet 18**  **MobileNet V2** | 0.87 | 88% | 0.87 | 88% |
| 0.9 | 91% | 0.9 | 91% |
| 0.9 | 90% | 0.95 | 95% |

****

|  |  |  |
| --- | --- | --- |
| **Model** | **Training Process Graph** | **Training Process Graph (A)** |
| **VGG-16** |  |  |
| **Resnet 18** |  |  |
| **Mobilenet V2** |  |  |

These figures and graphs show that MobilenetV2 was most efficient and more robust than any model we tried this on, thus for the final report MobilenetV2 with necessary data preprocessing needed.

#### Future Scope

MobileNetV2 is designed to be lightweight and efficient, with a focus on reducing computational complexity and model size. While MobileNetV2 may offer advantages in terms of speed and resource efficiency, its simplified architecture may limit its capability to capture complex patterns compared to more complex architectures like InceptionNet. We can explore more models with this regard.

Scientific names of plants often indicate their features such as Physical characteristics like size in *Sequoiadendron giganteum* (giant sequoia) and *Nolina microphylla* (ponytail palm) or Flower color in *Rosa rubiginosa* (rose) with its "rubiginosa" referring to the reddish color, and *Viola tricolor* (pansy) with "tricolor" indicating three colors. This all can be used to identify the plant family and are important feature to reverse engineer the plant that they have seen for first time at least till the plant family. These features can thus work even with Zero Shot Learning just like the Awa and Awa 2 dataset in case of animals.

Another direction, that can be taken in future is to integrate this light weight and robust MobilenetV2 architecture with an app so that it can be used in light devices such as Mobile Phones. So it can have a real world use case.

# Chapter 7

## Conclusion

In conclusion, the development of a robust deep learning model for the accurate classification of medicinal plants represents a significant advancement in the intersection of traditional medicine, technology, and conservation efforts. Through meticulous dataset collection, preprocessing, and leveraging state-of-the-art transfer learning techniques, we have created a model capable of identifying medicinal plants with high precision and reliability.

The utilization of pretrained neural network architectures, such as VGG16, ResNet, or MobileNetV2, coupled with fine-tuning strategies tailored to our specific classification task, ensures that our model can effectively learn and adapt to the intricacies of medicinal plant images. By evaluating the model using comprehensive metrics like accuracy, precision, recall, and F1-score, we gain insights into its performance across various classes, enabling us to refine and optimize its capabilities further.

The real-life applications of our trained model are vast and impactful. From empowering individuals through mobile apps for instant herbal medicine identification to aiding conservation efforts by identifying endangered species in the wild, and even ensuring quality control in herbal products, our model holds immense potential to revolutionize multiple sectors.

Moving forward, continued research and development in this field will not only enhance the accuracy and efficiency of medicinal plant classification but also pave the way for innovative solutions to address pressing global health and environmental challenges. With a well-trained deep learning model at our disposal and a practical deployment

strategy in place, we are poised to make tangible contributions towards a healthier, more sustainable future.

#### References

1. Azlah, Muhammad Azfar Firdaus, et al. "Review on techniques for plant leaf classification and recognition." Computers 8.4 (2019): 77.
2. Dhaware, Chaitali G., and K. H. Wanjale. "A modern approach for plant leaf disease classification which depends on leaf image processing." 2017 International Conference on Computer Communication and Informatics (ICCCI). IEEE, 2017.
3. Gopal, A., S. Prudhveeswar Reddy, and V. Gayatri. "Classification of selected medicinal plants leaf using image processing." 2012 International Conference on Machine Vision and Image Processing (MVIP). IEEE, 2012.
4. Jiang, Zhencun, et al. "Recognition of rice leaf diseases and wheat leaf diseases based on multi-task deep transfer learning." Computers and Electronics in Agriculture 186 (2021): 106184

[5]. Singh, R. Satya Rajendra, and Rakesh Kumar Sanodiya. "Zero-shot Transfer Learning Framework for Plant Leaf Disease Classification." IEEE Access (2023).

1. Liu, Jiachun, et al. "Plant leaf classification based on deep learning." 2018 Chinese Automation Congress (CAC). IEEE, 2018.
2. Liu, Jun, and Xuewei Wang. "Plant diseases and pests detection based on deep learning: a review." Plant Methods 17 (2021): 1-18.
3. Lee, Sue Han, et al. "How deep learning extracts and learns leaf features for plant classification." Pattern recognition 71 (2017): 1-13.
4. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
5. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
6. Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.