UNIVERSITY OF PORTSMOUTH

Classification of Plant Diseases Using Deep Learning and the Impact of Data Augmentation on Model Performance

Team member 1 ID: 20031826

Team member 2 ID: 2294726

Overview:

Plant diseases are a serious concern in agriculture—they can wipe out entire fields and cause huge economic damage. In many rural areas, farmers may not have quick access to specialists, so spotting these diseases early is really important. The traditional approach involves experts examining the plants, but that can be slow, expensive, and not always available.Recently, I got interested in how deep learning—particularly Convolutional Neural Networks—can play a role in identifying plant diseases. What's fascinating is how these models catch subtle patterns in leaf images, whether it's the texture, shape, or color, and they often manage to classify diseases pretty accurately. For this project, I used the Plant Village dataset, which has lots of labeled images of healthy and diseased leaves. I tested how well different models performed, and I also looked at how data augmentation—things like flipping or rotating images—could improve the results. What I found is that augmentation can really help reduce overfitting and make the model more reliable when it sees new, unseen data.
Overall, this work shows that using deep learning in agriculture has a lot of potential, especially when resources are limited and time is critical.

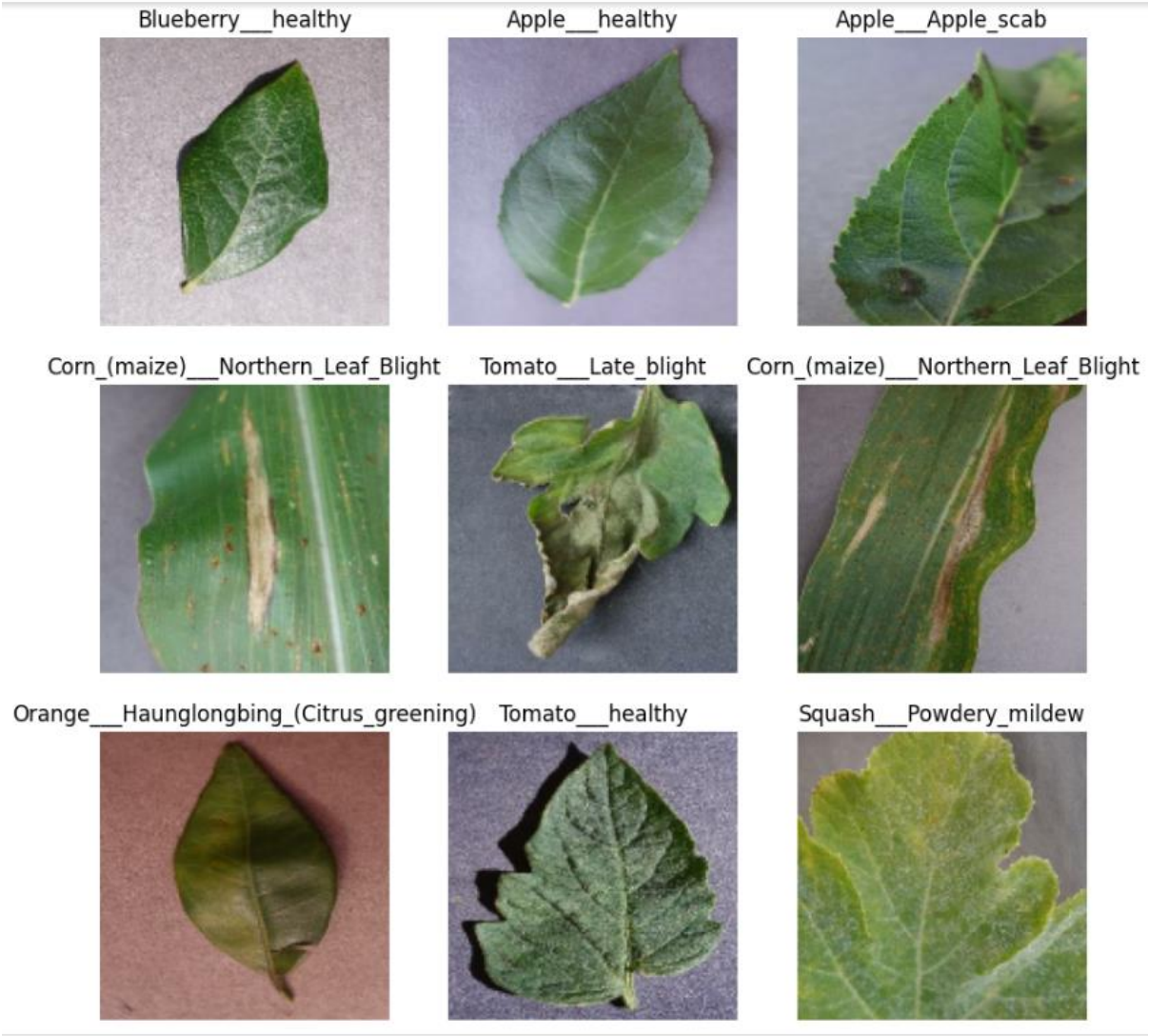
Loading and Splitting the Dataset:

We kicked things off by sorting the Plant Village dataset so that each plant type or disease had its own folder. After that, we split the data — about 80% was used for training the model, and the other 20% was kept for testing how well it performs. Each image is resized to 128×128 pixels before training, and the model works with them in batches of 16. The class names are taken straight from the folder names, which makes it easy to check that everything is set up right.

```python
import tensorflow as tf

# Parameters
img_size = (128, 128)
batch_size = 16

data_dir = '/content/PlantVillage/plantvillage dataset/color'

train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=img_size,
    batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=img_size,
    batch_size=batch_size
)

# Get class names
class_names = train_ds.class_names
print("Number of classes:", len(class_names))
print("Class names:", class_names)
```

Displaying Sample Images from the Training Dataset:

This figure includes nine leaf samples from the PlantVillage dataset, each labeled with its class name. The examples feature both healthy leaves and those showing symptoms of plant diseases like Apple Scab, Tomato Late Blight, and Northern Leaf Blight in corn. Reviewing these images helps confirm that the data has been loaded properly and gives a quick visual sense of how the leaf conditions differ..



CNN Architecture without data augmentation:

The diagram presents the structure of a Convolutional Neural Network (CNN) developed using TensorFlow Keras, without adding any data augmentation steps. This model is made up of three convolutional layers, each one followed by a max pooling layer to reduce dimensionality. After these layers, the data flows through two dense layers to complete the classification process. The training uses the Adam optimizer and relies on sparse categorical crossentropy as the loss function. Accuracy is used as the main performance metric. This version of the model acts as a starting point, making it easier to measure the impact of adding data augmentation later on.

```python
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Rescaling(1./255, input_shape=(128, 128, 3)),
    layers.Conv2D(16, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

model.summary()
```

Training Model:

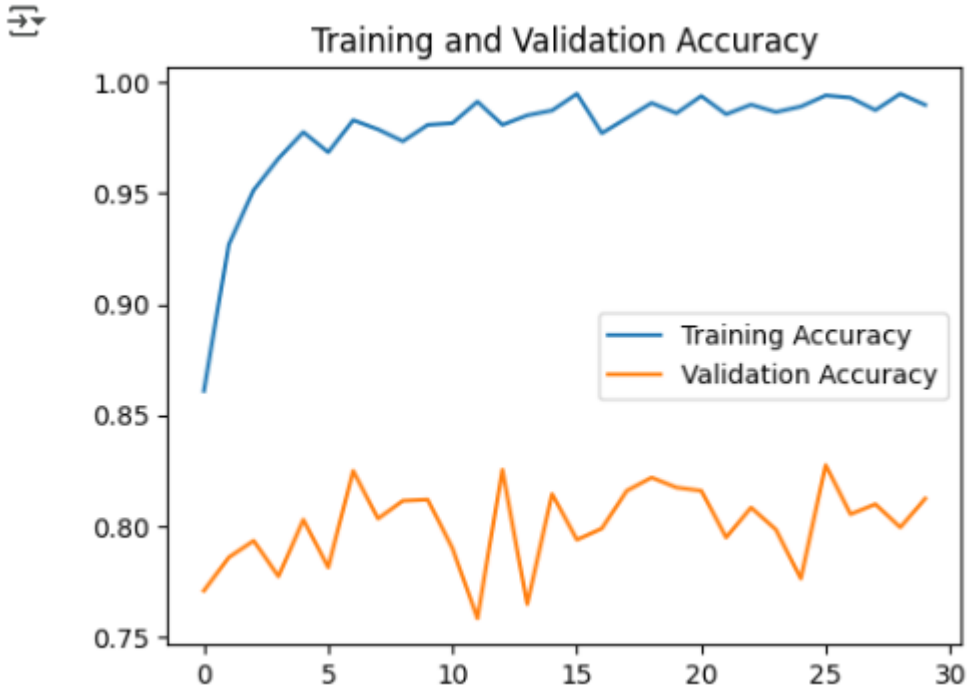
Training the Model

```python
[ ] history = model.fit(
        train_ds,
        validation_data=val_ds,
        epochs=30
    )
```

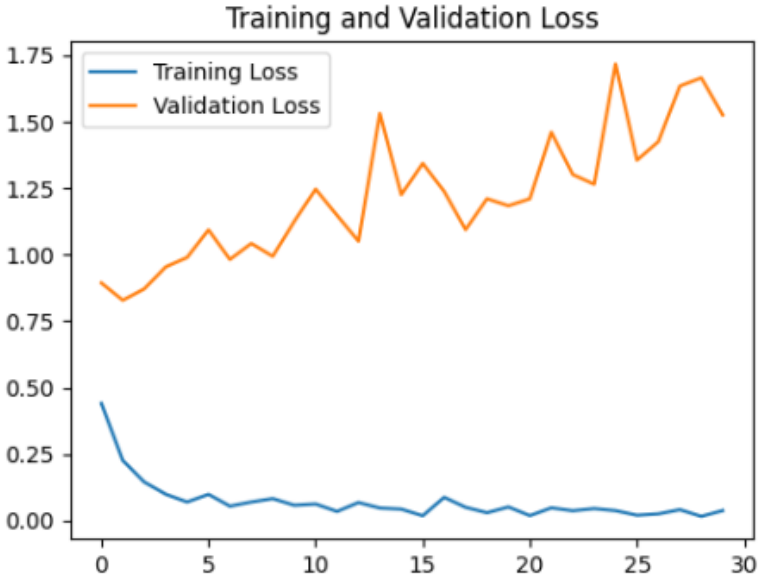
We began the training phase using the fit() function to help the CNN model learn from the data. The model went through 30 training cycles (epochs), working with both the training dataset (train_ds) and a separate validation dataset (val_ds). Throughout this process, the model adjusted its internal settings to reduce errors and improve how accurately it could recognize patterns.

Accuracy and Loss Performance:

Accuracy: Throughout the training process, the model improved its ability to identify patterns in the training data, eventually reaching an accuracy exceeding 99%. While it fluctuated from one round to another, the accuracy on validation data remained approximately 80%. This difference indicates that the model might have overfitted, performing exceptionally on the training set but failing to understand new, unseen data.



Loss: Even though the training loss went down steadily, the loss on the validation set was higher and jumped around a lot. This kind of gap supports the idea that the model performs well on data it's seen before but runs into trouble with new data.



Evaluate the Model:

```
loss, accuracy = model.evaluate(val_ds)
print(f"Validation Accuracy: {accuracy * 100:.2f}%")
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8176 - loss: 1.5934
Validation Accuracy: 81.05%
```

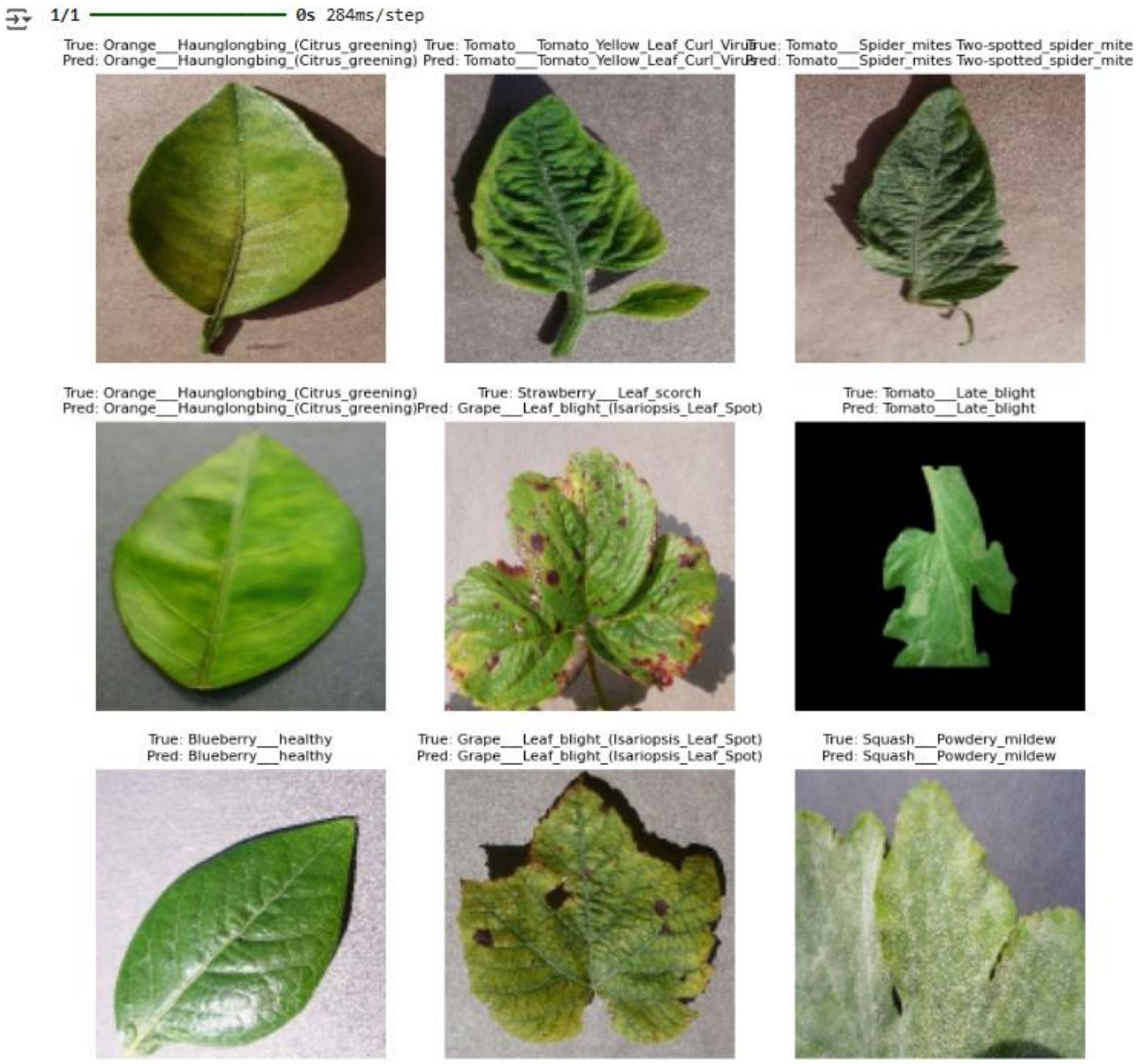
This code block evaluates the generalization ability of the trained CNN model through the validation dataset. The results show:
81. 05% is the accuracy on validation.
Validation Loss: 1. 5934
While the model achieves remarkable performance on the training data, its effectiveness drops when evaluated on new data, highlighting overfitting again. These metrics reinforce the observations made in the training and validation curves. This evaluation stage is crucial to any machine learning workflow as it provides a numerical assessment of the model's ability to perform well on real-world data.

Predicting and Visualizing Model Outputs on Validation Data:

The diagram illustrates the model's performance in predicting images it has not previously encountered from the validation set. By utilizing model. predict(), the code creates predictions and contrasts them with the true class labels. Each image showcases the correct label above and the predicted label beneath. This direct comparison simplifies the evaluation of how accurately the model identifies various categories. In many instances, the predictions align with the actual labels, indicating that the CNN has effectively learned significant features from the dataset.

Predicting on New Images from Validation DataSet

```
for images, labels in val_ds.take(1):
    predictions = model.predict(images)
    predicted_labels = tf.argmax(predictions, axis=1)

    plt.figure(figsize=(10, 10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        true_label = class_names[labels[i]]
        pred_label = class_names[predicted_labels[i]]
        plt.title(f"True: {true_label}\nPred: {pred_label}", fontsize=8)
        plt.axis("off")
```



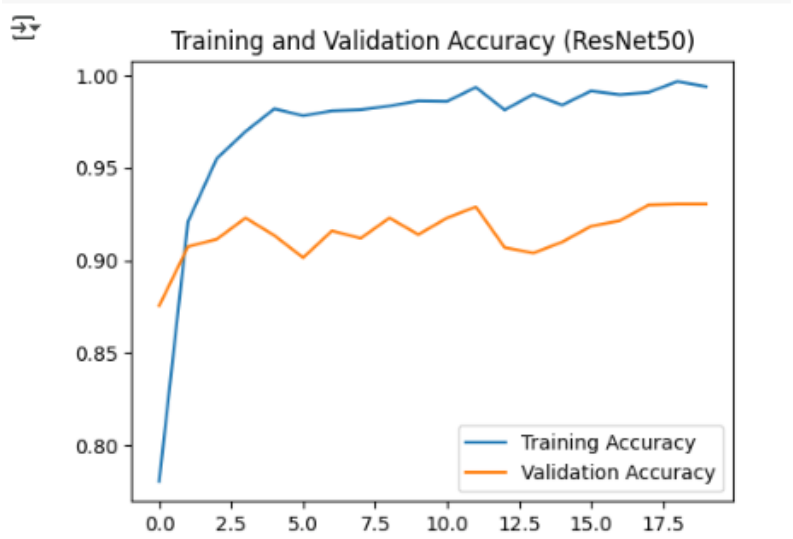
ResNet50 Model:

This block of code shows how transfer learning is used to modify a ResNet50 architecture for the leaf disease classification job. To preserve learnt low-level features, the model's convolutional base is frozen (trainable=False) and initialized using pre-trained ImageNet weights (weights='imagenet').GlobalAveragePooling2D is then used to construct a custom classifier on top, which is followed by fully connected layers with ReLU and softmax activations. This configuration makes use of the robustness of ResNet50's deep feature extraction while enabling the model to adjust to the distinct classes in the PlantVillage dataset. Compatibility with integer-labeled multiclass targets is ensured by compiling the model using sparse categorical cross-entropy and the Adam optimizer.

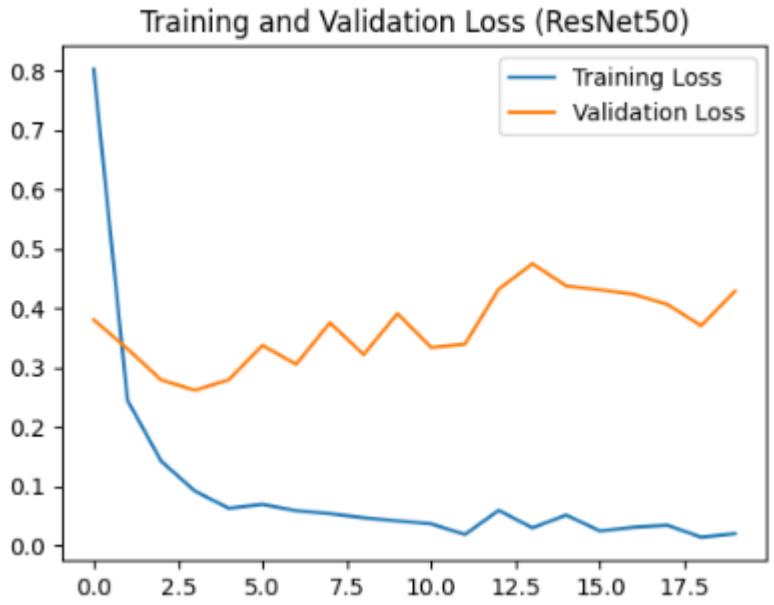
Building a ResNet50 Model

```
[ ] from tensorflow.keras.applications import ResNet50
    from tensorflow.keras import layers, models

    # Load ResNet50 base
    resnet_base = ResNet50(
        input_shape=(128, 128, 3),
        include_top=False,   # We will add our own classifier
        weights='imagenet'   # Use pretrained ImageNet weights
    )

    # Freeze the base model
    resnet_base.trainable = False

    # Build the full model
    resnet_model = models.Sequential([
        resnet_base,
        layers.GlobalAveragePooling2D(),
        layers.Dense(128, activation='relu'),
        layers.Dense(len(class_names), activation='softmax')
    ])

    # Compile
    resnet_model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    resnet_model.summary()
```

And then, the accuracy curve indicates strong generalization and minimal overfitting, showing that the model quickly reaches a high training accuracy of 99–100% while the validation accuracy levels off at 93%.



This behavior is reinforced by the loss curves, which indicate that the validation loss remains consistent and generally close with only minor fluctuations, while the training loss decreases significantly and remains low. This suggests that the pretrained ResNet50 architecture effectively learned the primary patterns in the dataset without undergoing excessive overfitting.



Evaluate the Model ResNet50:

The image shows the latest evaluation of the model using the data being validated. The result indicates excellent performance, with an accuracy of 93.30 percent and a loss of 0.3203. This model provides very accurate predictions on plant leaf images.

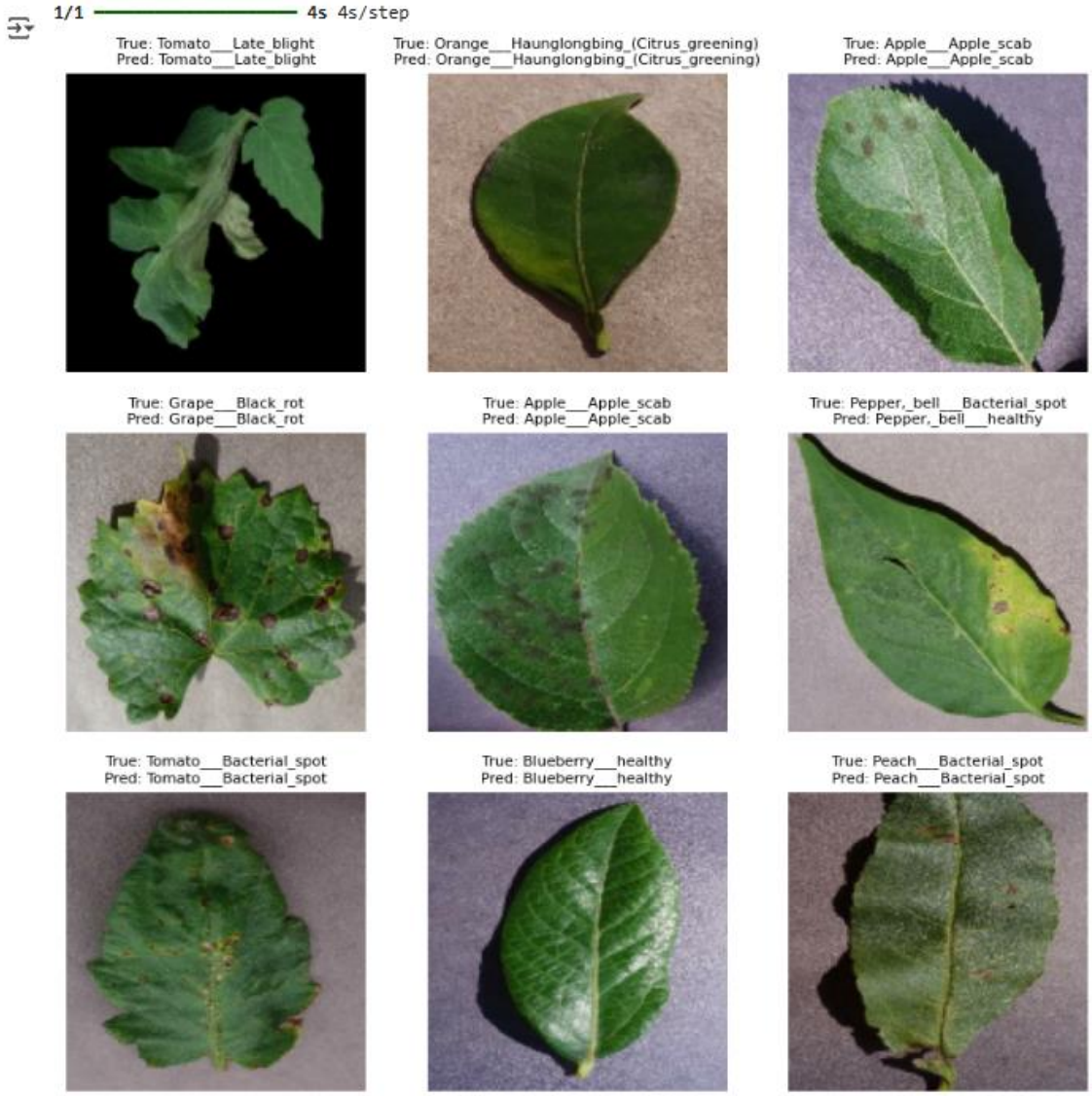
Evaluate the ResNet50 Model

```
loss, accuracy = resnet_model.evaluate(val_ds)
print(f"Validation Accuracy (ResNet50): {accuracy * 100:.2f}%")
```

```
125/125 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9381 - loss: 0.3203
Validation Accuracy (ResNet50): 93.30%
```

Testing The Mode (ResNet50):

In this section, the ResNet50 model's performance is shown on unseen validation images through a visual comparison of its predictions vs the actual correct labels. The code that was utilized to create a 3x3 grid of leaf images from the data, produce predictions with the trained ResNet50 model, and handle a batch of validation data is shown in the figure on the left. These images are displayed alongside their true and predicted class labels in the corresponding figure on the right. This display allows us to visually assess the model's classification accuracy and pinpoint any misclassifications. It illustrates how proficiently the model operates across various classes, accurately detecting most diseases and healthy leaves.



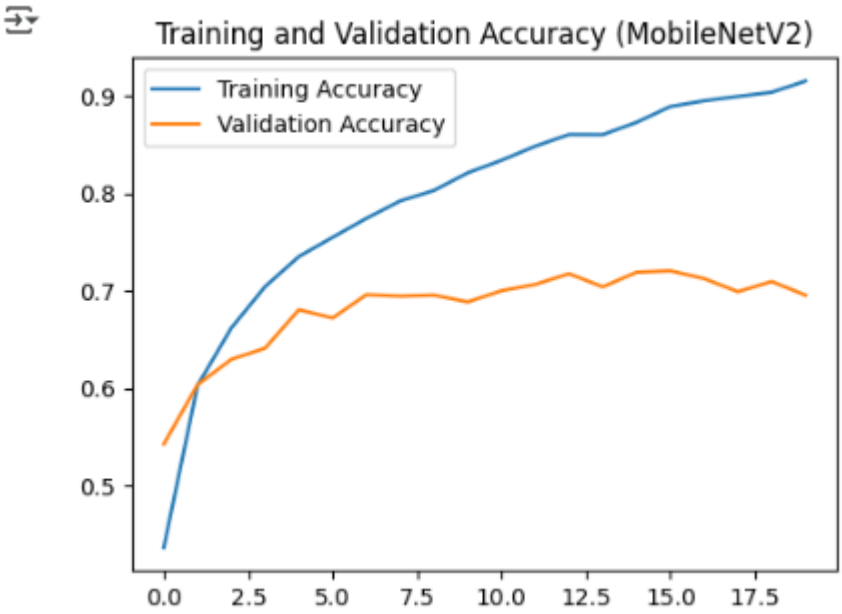
Building Model MobileNetV2 without data augmentation:

In this step, the MobileNetV2 architecture was used to classify the plant leaf diseases. It is a portable and effective deep learning model. Pretrained weights from ImageNet have been importer into the model in order to tale advantage of transfer learning and to accelerate convergence. The top layers of the model have been removed to provide a unique classification head that would work well for the dataset. ReLU and SoftMax activations on Dense layers are used to produce class probabilities after implementing a GlobalAveragePooling2D layer is added to minimize dimensionality The model is then prepared for training by compiling it using the sparse categorical crossentropy loss function and the Adam optimizer. This method strikes a balance between computational efficiency and precision, making it particularly helpful for deployment on mobile or edge devices.

```python
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import layers, models

# Load MobileNetV2 base
mobilenet_base = MobileNetV2(
    input_shape=(128, 128, 3),
    include_top=False,    # We will add our own custom classifier
    weights='imagenet'    # Pretrained weights
)

# Freeze the base model
mobilenet_base.trainable = False

# Build full model
mobilenet_model = models.Sequential([
    mobilenet_base,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.Dense(len(class_names), activation='softmax')
])

# Compile
mobilenet_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

mobilenet_model.summary()
```

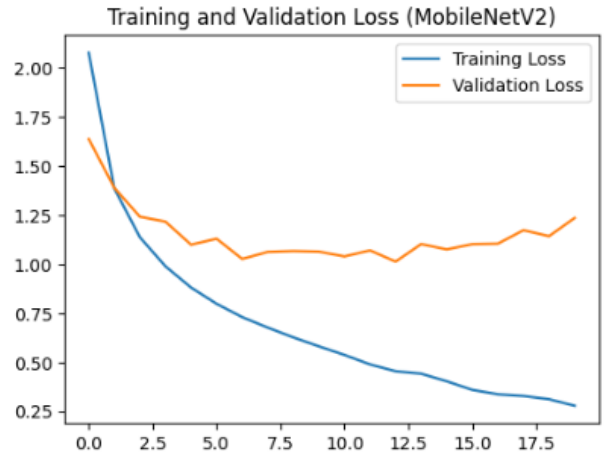
The Accuracy of Model:

This figure represents the MobileNetV2 model's training and validation accuracy over 20 epochs of training. The training accuracy gradually increases and surpasses 90% while the validation accuracy peaks at about 70%.



Loss Of Model:

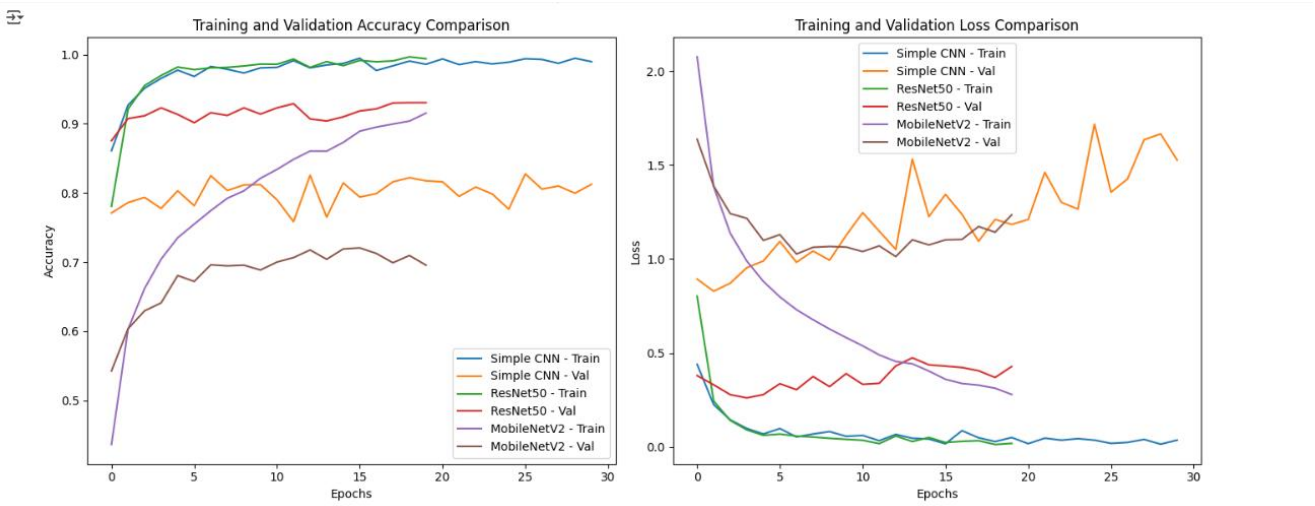
This figure shows the MobileNetV2 model's training and validation loss over 20 epochs of training. The model is learning and fitting the training data quite well as the training loss gradually declines over the 20 epochs. However, the validation loss does not show as big of progress and floats over a larger value. It is clear that the model is overfitting as seen by the comparison between the training loss and validation loss. It basically can see training data but struggles with generalizing new, unseen data.



Comparison The Models Accuracy and loss without Data Augmentation:

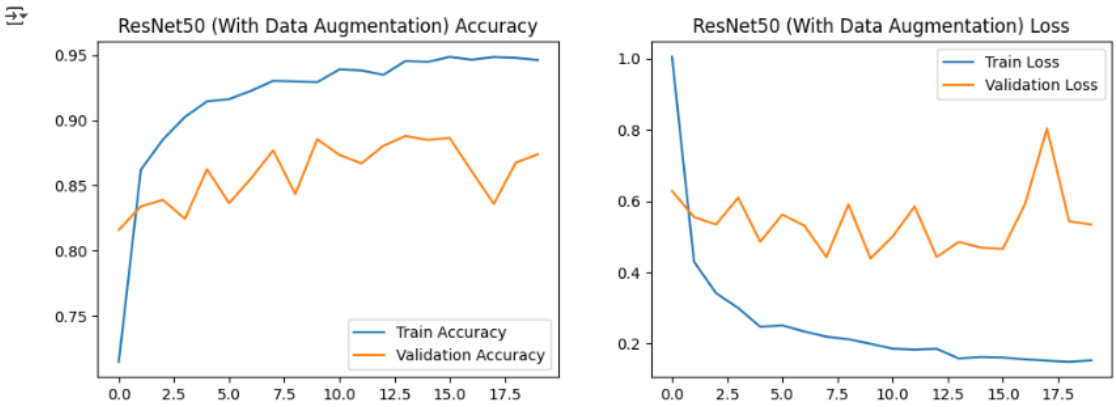
Among the three models, ResNet50 shows the strongest generalization. It reaches the highest validation accuracy (around 93%) and follows its training accuracy closely, indicating reliable learning. On the other hand, the simple CNN reaches nearly perfect accuracy during training, but its validation accuracy drops to about 81%, revealing clear signs of overfitting.

MobileNetV2 exhibits weaker performance relative to the other models, demonstrating moderate training accuracy and a lower validation accuracy of approximately 70%. This difference indicates that the model is underfitting and lacks the ability to fully learn from the provided data.



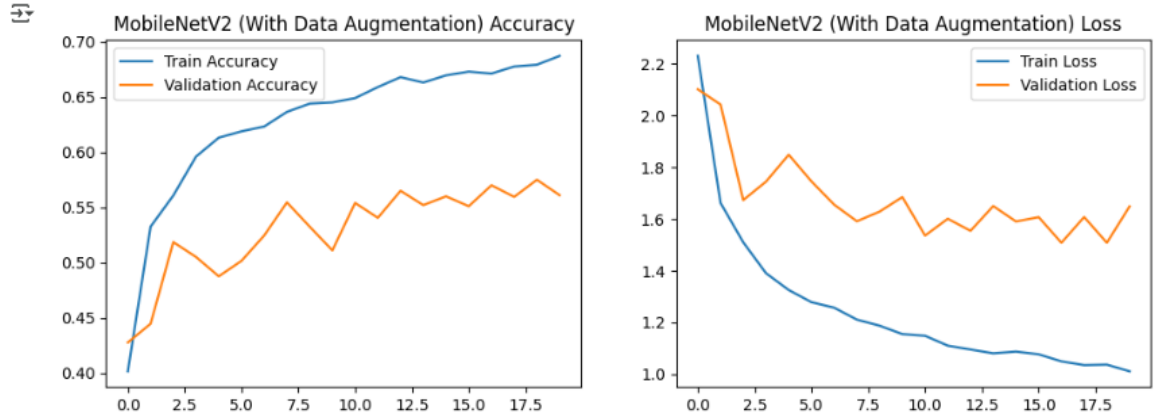
ResNet50 With Data Augmentation:

ResNet50 benefits greatly from data augmentation, which exposes the model to a wider variety of training examples. This helps improve its ability to generalize and leads to a more stable loss curve along with better validation accuracy. These results show that augmentation is a valuable strategy for improving performance in challenging image classification tasks.



MobileNetV2 With Data Augmentation:

Even after applying data augmentation, MobileNetV2 only shows a small improvement. Its validation accuracy still stays on the lower side, which suggests that the model's lightweight structure might not be strong enough to fully capture the details in the dataset—unless it's fine-tuned or redesigned a bit.



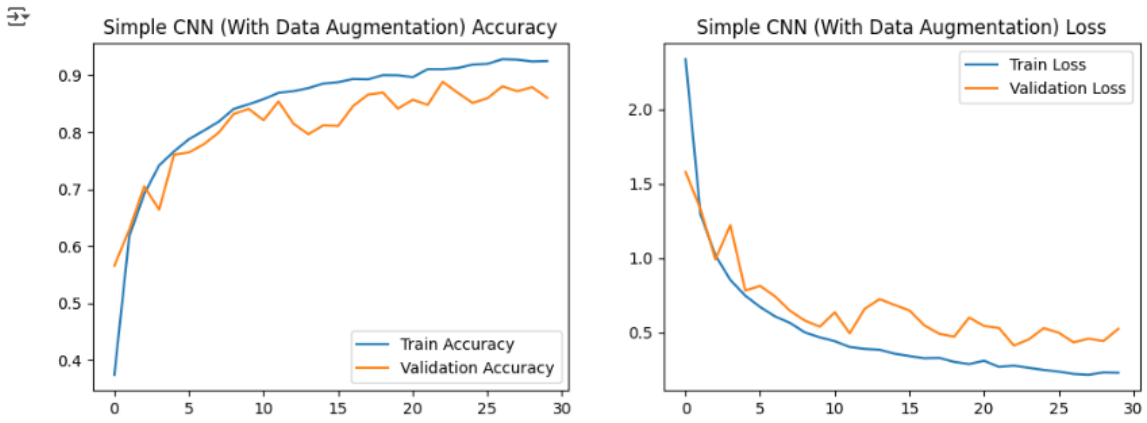
CNN Model with Data Augmentation:

Accuracy

As the training progresses, the model's accuracy continues to enhance and ultimately exceeds 90%. The validation accuracy also shows good results, attaining about 87% to 89%. Given that both training and validation scores are similar, this indicates that the model is generalizing effectively and not experiencing excessive overfitting.

Loss

Throughout the training process, the loss for both training and validation steadily diminishes. The validation loss primarily stays constant with only slight variations, indicating that the model is acquiring knowledge effectively without considerable overfitting. Overall, the inclusion of data augmentation greatly enhanced the performance of the Simple CNN.



Final Visualization Comparing all the Augmented Models (Simple CNN, ResNet50, and MobileNetV2):

Comparison of Accuracy:

Here, the accuracy of the resnet50 model is shown to be 90%. With the training and validation data, the CNN model also showed relatively strong performance, roughly equivalent to the accuracy of the resnet50 model. On the other hand, the mobilenetv2 model does not achieve the same performance with even more data, as its validation accuracy ranges between 58 and 60 percent, which reflects a slight improvement.

Comparison of Losses:

The resnet50 model emphasizes maintaining minimal loss during the training process and achieves more balanced results. The CNN model has consistent and decreasing results, which indicates a reduction in data overfitting. The mobilenet v2 model also reflects the difficulty of learning complex patterns.