1. Introduction

Deep convolutional neural networks (CNNs) have proven highly effective for image classification tasks. In this study, we focus on classifying olive leaf diseases using CNNs. A dataset of 3,400 olive leaf images was collected from Denizli, Turkey, during spring and summer. The dataset comprises three classes: leaves infected with Aculus olearius, leaves affected by Olive Peacock Spot, and healthy leaves. We employ both a custom CNN model and three pre-trained transfer learning models to evaluate their performance in disease detection. Data augmentation techniques are also applied to enhance model robustness. The aim is to determine the most effective approach for accurately classifying olive leaf diseases.

2.Dataset Description

The dataset consists of 3,400 olive leaf images, categorized into three classes: Healthy, Aculus olearius, and Olive Peacock Spot. The images are divided into training and testing sets, ensuring a balanced evaluation of model performance.

Dataset	**Total Images**	**Healthy(class 0)**	**Aculus olearius (class 1)**	**Olive Peacock Spot(class 2)**
Training Set	2,720	830	690	1,200
Testing Set	680	220	200	260

3.CNN Model Architectures

3.1. Custom CNN Model without Data Augmentation

This model consists of five convolutional blocks, each with increasing filter sizes (16, 32, 64, 128, and 256) and ReLU activation. Batch normalization is applied in the first block to stabilize training. Max-pooling layers reduce spatial dimensions, while fully connected layers with dropout regularization minimize overfitting. The final softmax layer classifies images into three categories.

```python
def build_custom_cnn():
    model = Sequential()

    # Block 1
    model.add(Conv2D(16, (3, 3), activation='relu', padding='same', input_shape=(224, 224, 3)))
    model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())  # Batch Normalization only in Block 1
    model.add(MaxPooling2D((2, 2)))

    # Block 2
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))

    # Block 3
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))

    # Block 4
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))
```

```python
    # Block 5
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPooling2D((2, 2)))

    # Fully Connected Layers
    model.add(Flatten())
    model.add(Dense(1000, activation='relu'))
    model.add(Dropout(0.2))  # Dropout to reduce overfitting
    model.add(Dense(1000, activation='relu'))
    model.add(Dense(3, activation='softmax'))

    return model
```

***Figure-1**: Custom CNN Model architecture*

3.2. Custom CNN Model with Data Augmentation

This model builds upon 3.1 but incorporates data augmentation techniques such as rotation, shifting, shearing, zooming, and horizontal flipping. These transformations improve generalization by exposing the model to varied versions of training images, reducing dependency on specific image orientations or conditions.

```
train_datagen1  ImageDataGenerator(
    rescale=1./255,  # Rescale image pixel values to [0, 1]
    rotation_range=50,  # Random rotations between -30 to 30 degrees
    width_shift_range=0.2,  # Randomly shift images horizontally by 20%
    height_shift_range=0.2,  # Randomly shift images vertically by 20%
    shear_range=0.2,  # Shear transformation
    zoom_range= 0.4, # Random zoom
    horizontal_flip=True,  # Randomly flip images horizontally
    fill_mode='nearest'  # Fill in missing pixels after transformations
)

test_datagen1= ImageDataGenerator(rescale=1./255)
```

***Figure-2**: Custom CNN Model architecture with data augmentation*

3.3. Pre-trained Transfer Learning Model without Data Augmentation

Instead of training from scratch, this model leverages pre-trained convolutional bases from VGG16, VGG19, and MobileNet (trained on ImageNet). The convolutional layers are frozen, meaning they retain their learned features, and only newly added fully connected layers are trained to adapt to the olive leaf classification task. This approach benefits from high-level feature extraction without requiring large datasets.

```
def build_model(base_model):
    for layer in base_model.layers:
        layer.trainable = False  # Freeze convolutional layers

    x = Flatten()(base_model.output)
    x = Dense(1000, activation="relu")(x)
    x = Dropout(0.5)(x)
    x = Dense(3, activation="softmax")(x)  # 3-class classification

    model = Model(inputs=base_model.input, outputs=x)
    model.compile(optimizer=Adam(learning_rate=0.0001), loss="categorical_crossentropy", metrics=["accuracy"])

    return model
```

***Figure-3**: Pre-trained CNN Model architecture*

```
# Load Pre-trained Models
vgg16_base = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
vgg19_base = VGG19(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
mobilenet_base = MobileNet(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
```

***Figure-4**: Loading 3 Pre-trained CNN Model*

3.4. Pre-trained Transfer Learning Model with Data Augmentation

Building upon 3.3, this model applies data augmentation to further enhance performance. The augmentation techniques help the model adapt to real-world variations by making it robust against different lighting conditions, orientations, and minor distortions.

```
# Define Data Augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=50,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode="nearest"
)

test_datagen = ImageDataGenerator(rescale=1./255)
```

***Figure-5**: Pre-trained CNN Model with augmentation*

3.5. Fine-tuning Pre-trained Transfer Learning Model with Data Augmentation

This approach unfreezes some convolutional layers (from a specific depth) in the pre-trained models, allowing them to be fine-tuned on the olive leaf dataset. A low learning rate is used to carefully adjust the pre-trained weights without overwriting valuable learned features. Data augmentation is applied to prevent overfitting and improve model generalization.

```
def build_model(base_model, unfreeze_from=30):

    for i, layer in enumerate(base_model.layers):
        if i < unfreeze_from:
            layer.trainable = False
        else:
            layer.trainable = True

    x = Flatten()(base_model.output)
    x = Dense(1000, activation="relu")(x)
    x = Dropout(0.5)(x)
    x = Dense(3, activation="softmax")(x)  # 3-class classification

    model = Model(inputs=base_model.input, outputs=x)
    # Use a low learning rate for fine-tuning
    model.compile(optimizer=Adam(learning_rate=1e-5),
                  loss="categorical_crossentropy",
                  metrics=["accuracy"])
    return model
```

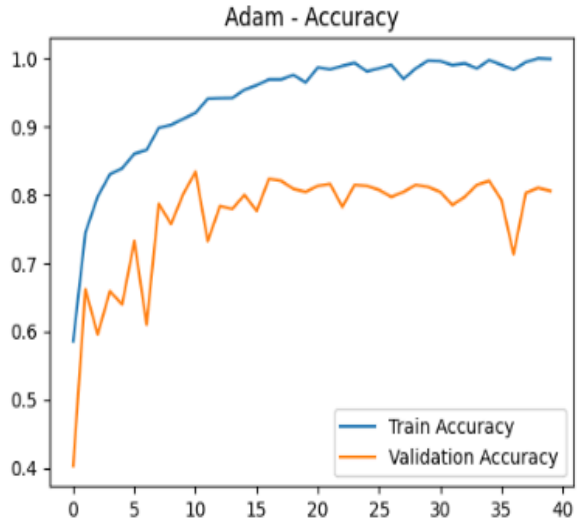
***Figure-6**: Pre-trained CNN Model with augmentation anf fine tune*

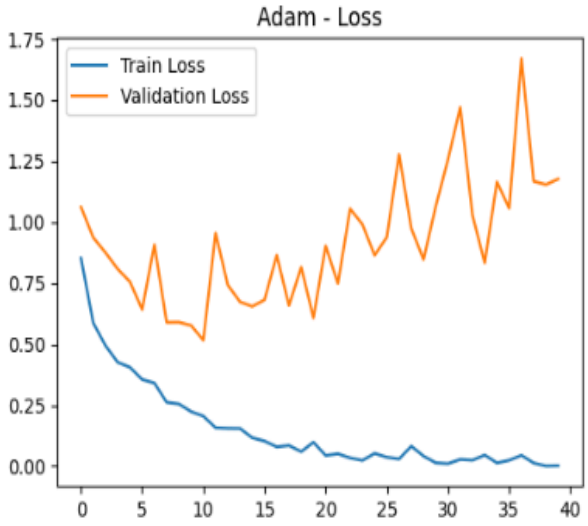
4.Results Analysis

All models were trained for ***40 epochs*** using categorical cross-entropy loss and Adam optimizer. Early stopping was applied to prevent overfitting, and batch normalization helped stabilize training.

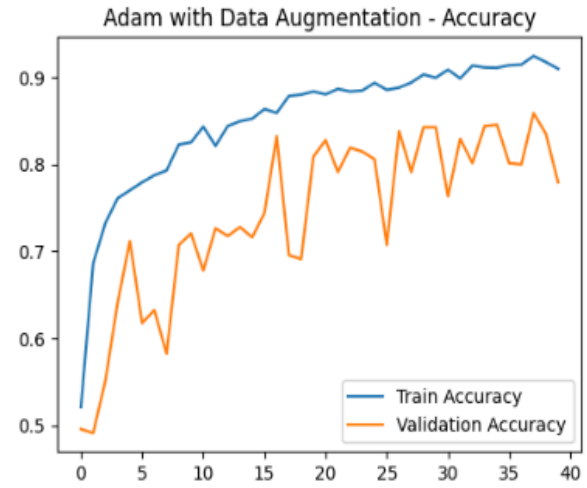
4.1. Accuracy curves for each model

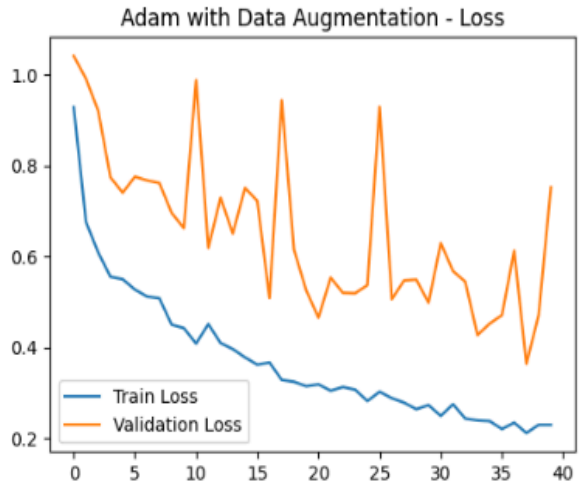
Custom CNN Model without Data Augmentation (Section 3.1)



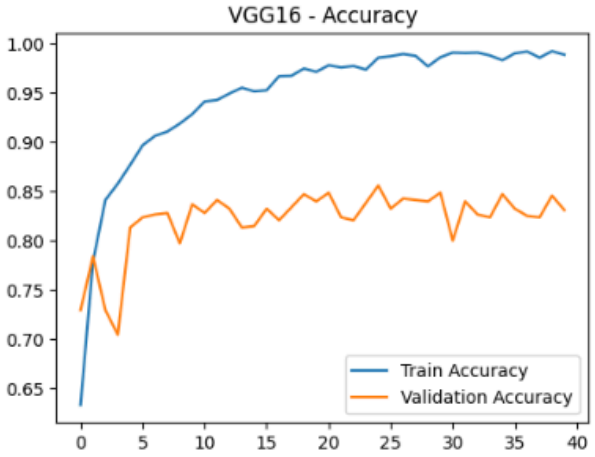


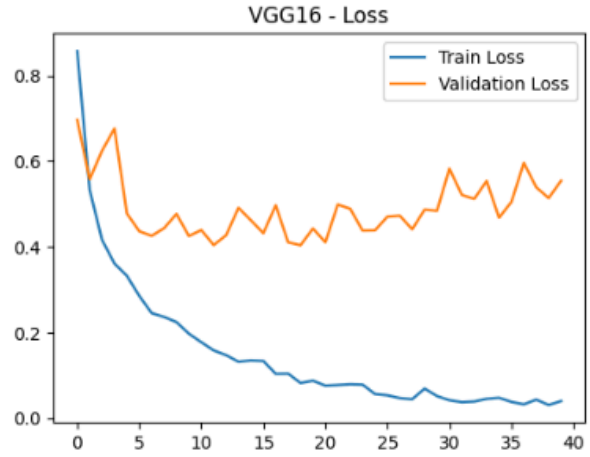
Custom CNN Model with Data Augmentation (Section 3.2)

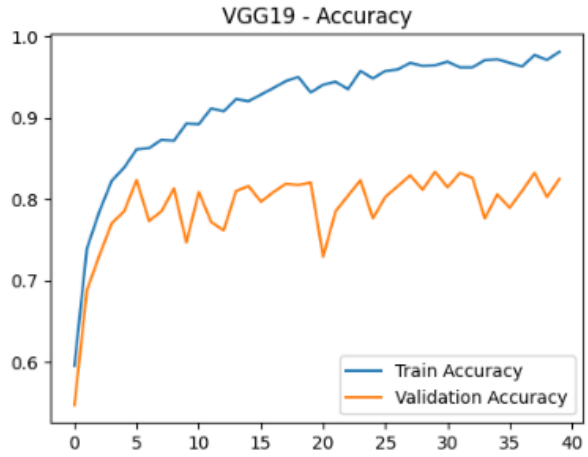


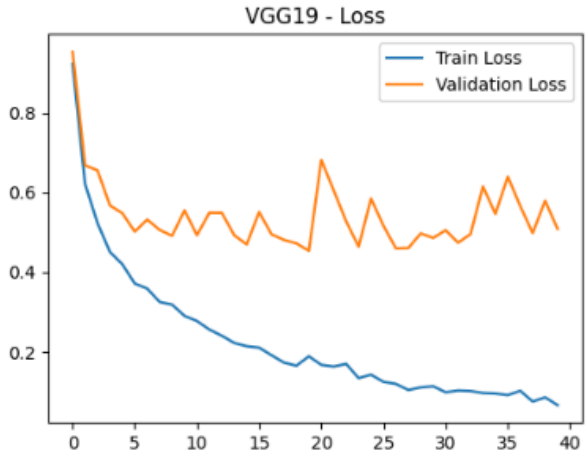


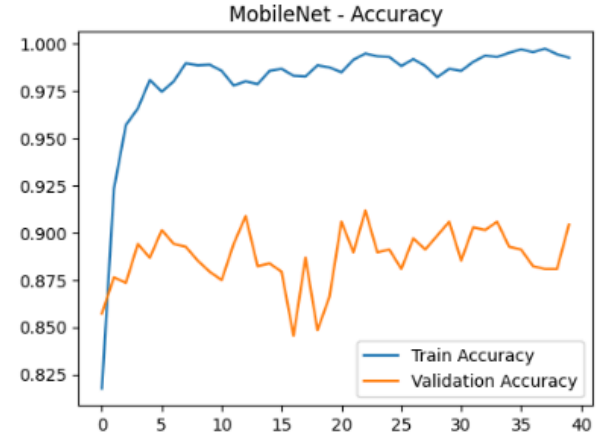
Pre-trained Transfer Learning Model without Data Augmentation (Section 3.3)

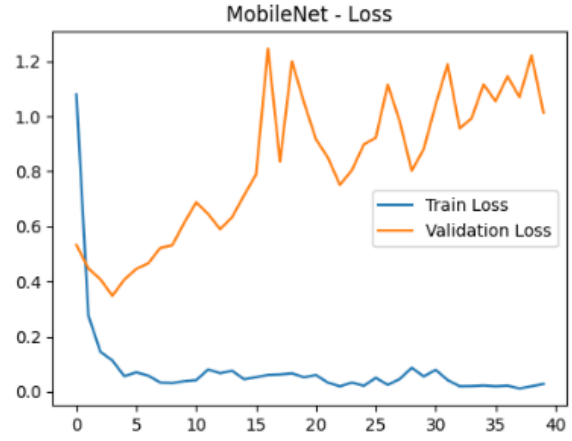




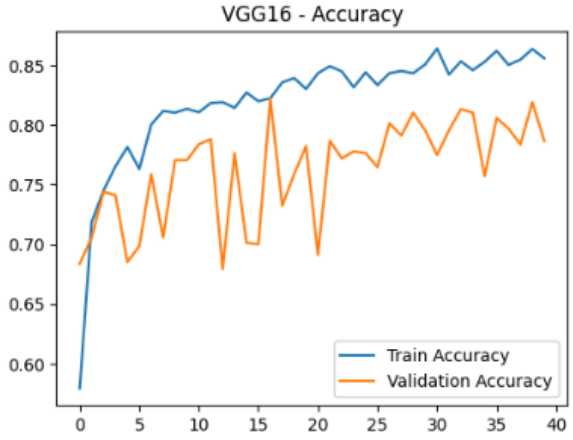


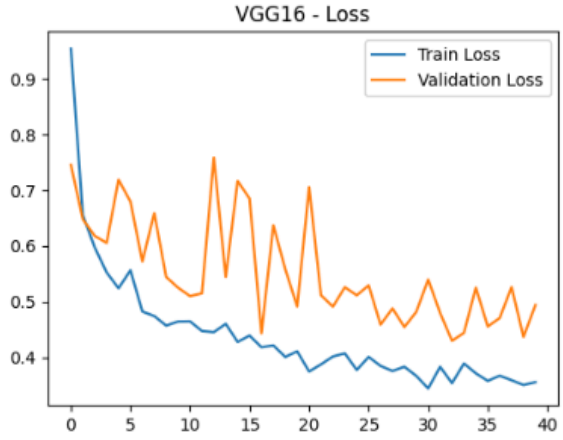


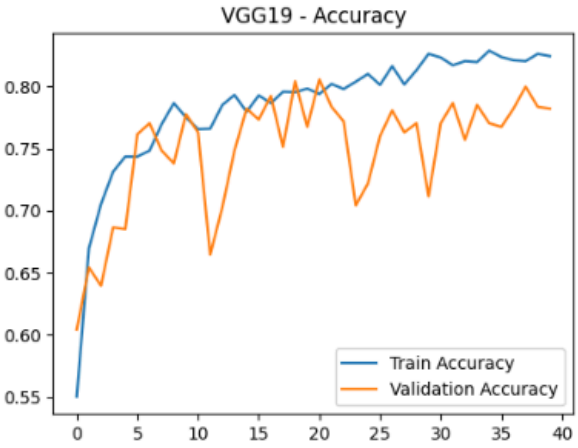


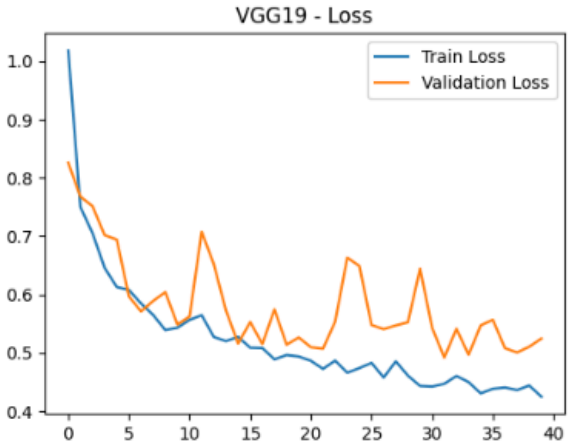


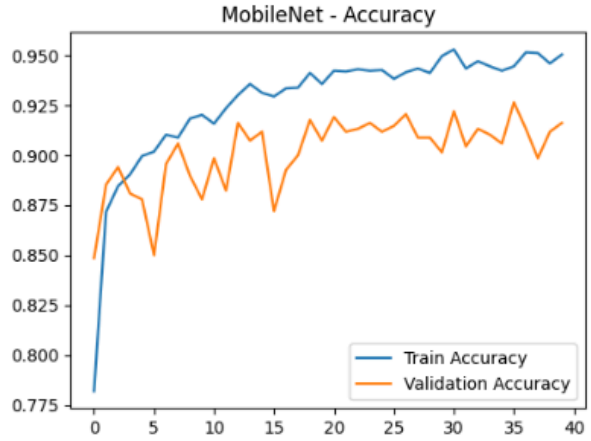
Pre-trained Transfer Learning Model with Data Augmentation (Section 3.4)

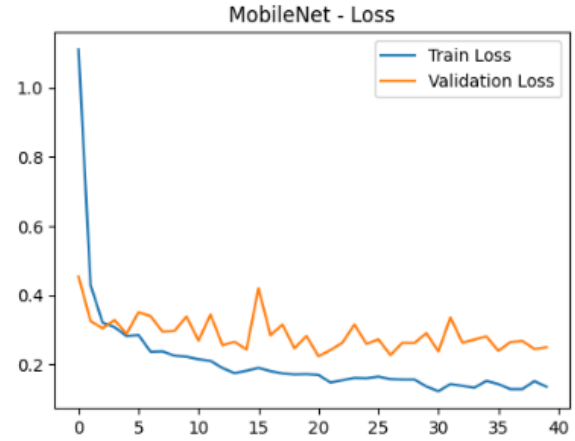




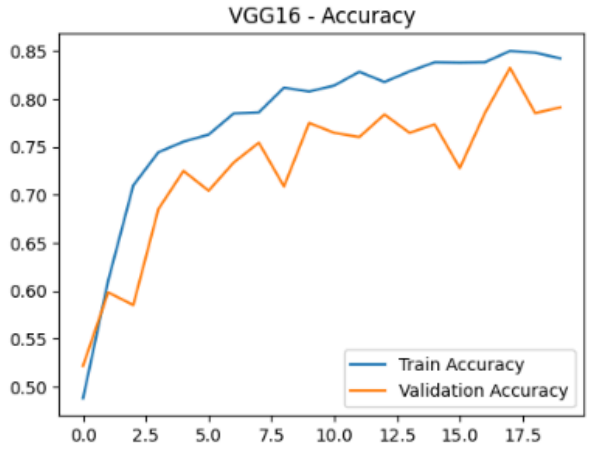


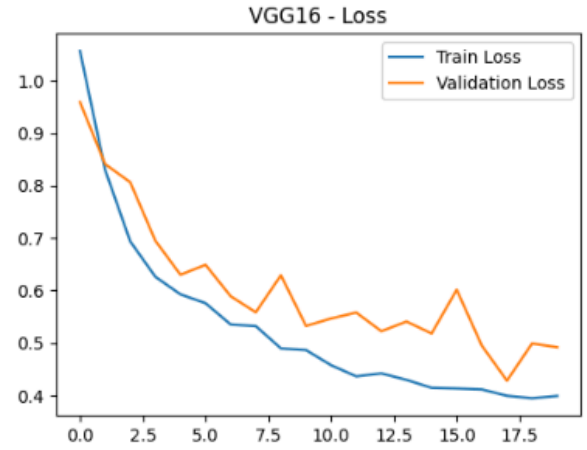


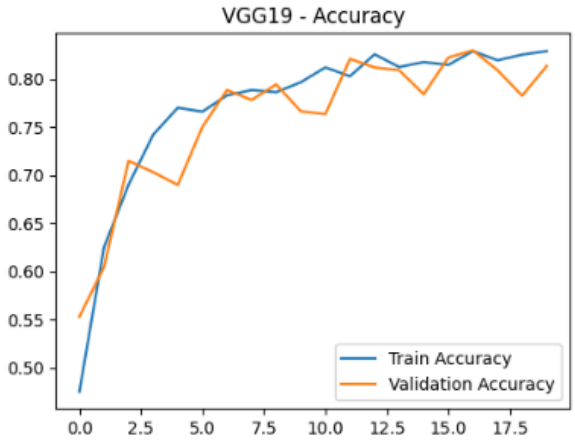


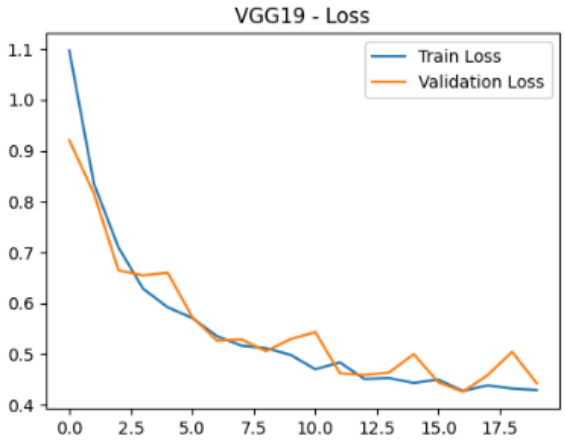


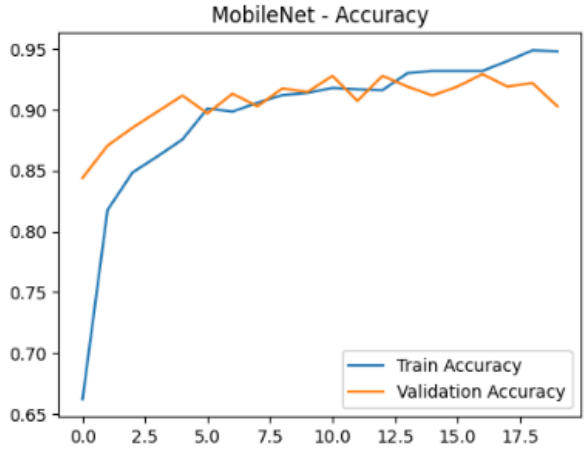
Fine-tuning Pre-trained Transfer Learning Model with Data Augmentation (Section 3.5)

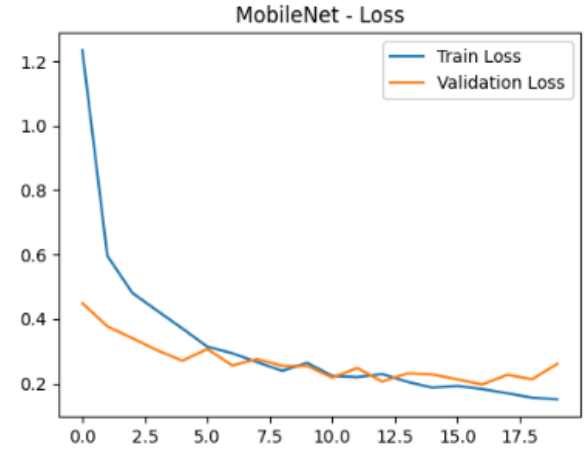












4.2. Confusion matrices for each model

A confusion matrix is a key evaluation metric used in classification tasks to analyze model performance by comparing predicted labels with actual labels. It is a square matrix where:

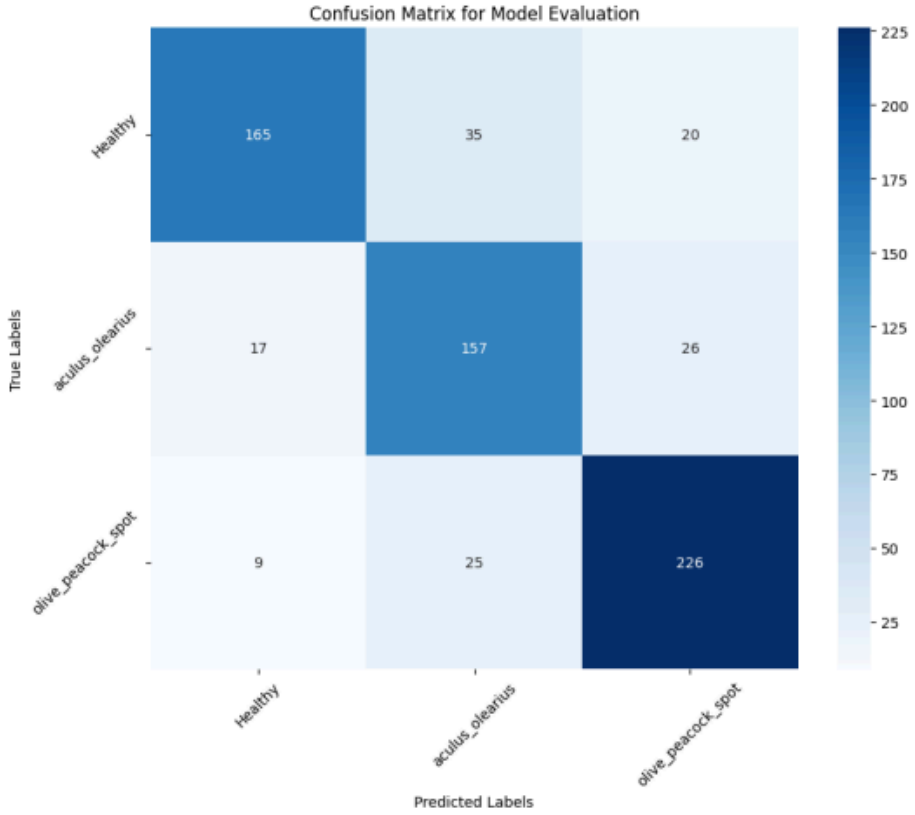
True Positives (TP): Correctly predicted positive cases.

True Negatives (TN): Correctly predicted negative cases.

False Positives (FP): Incorrectly predicted positive cases (Type I error).

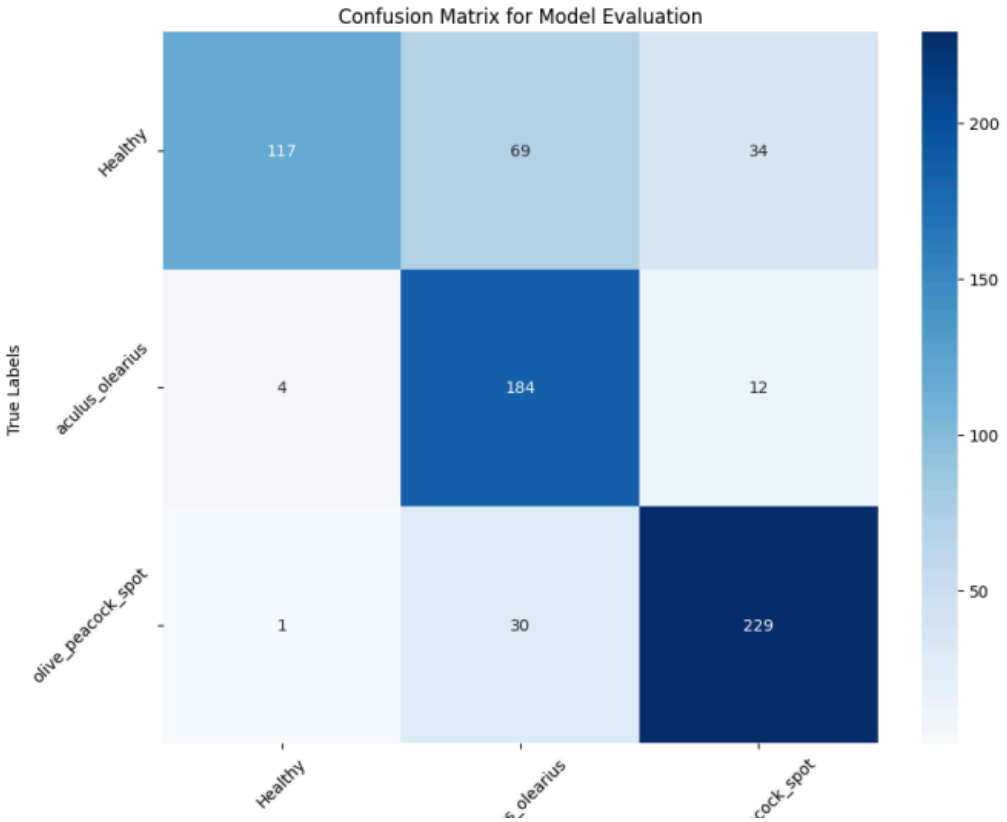
False Negatives (FN): Incorrectly predicted negative cases (Type II error).

<u>Custom CNN Model without Data Augmentation (Section 3.1)</u>



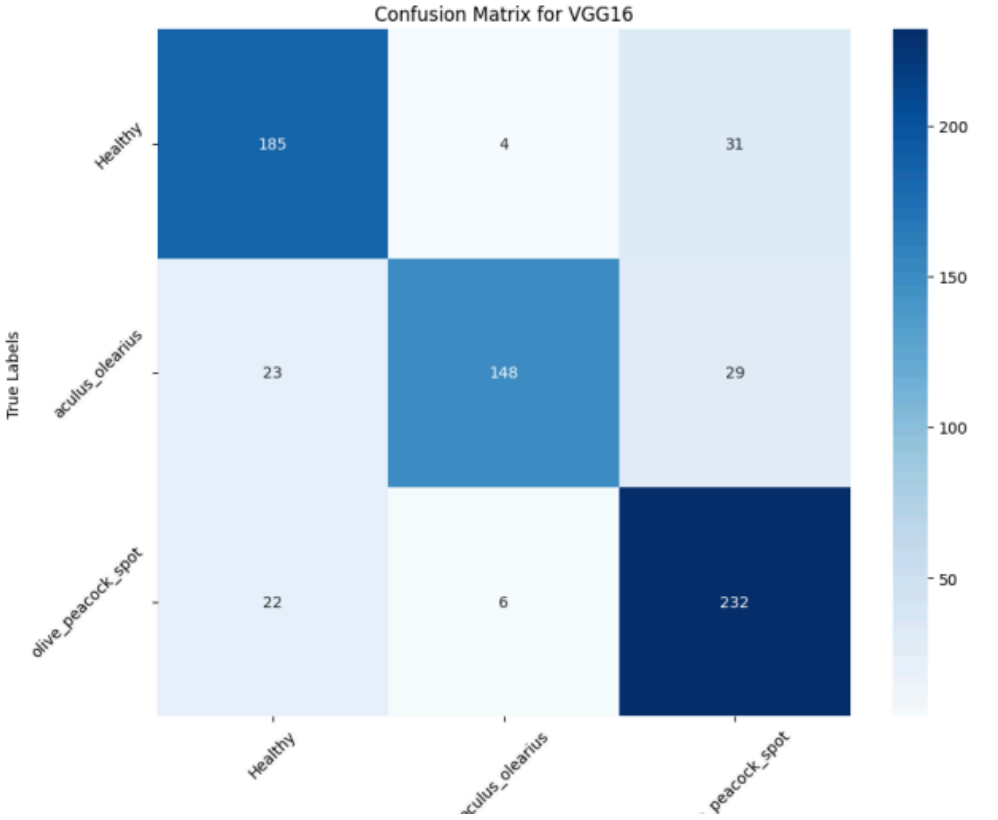

```
🔹 Confusion Matrix:
[[165  35  20]
 [ 17 157  26]
 [  9  25 226]]
```

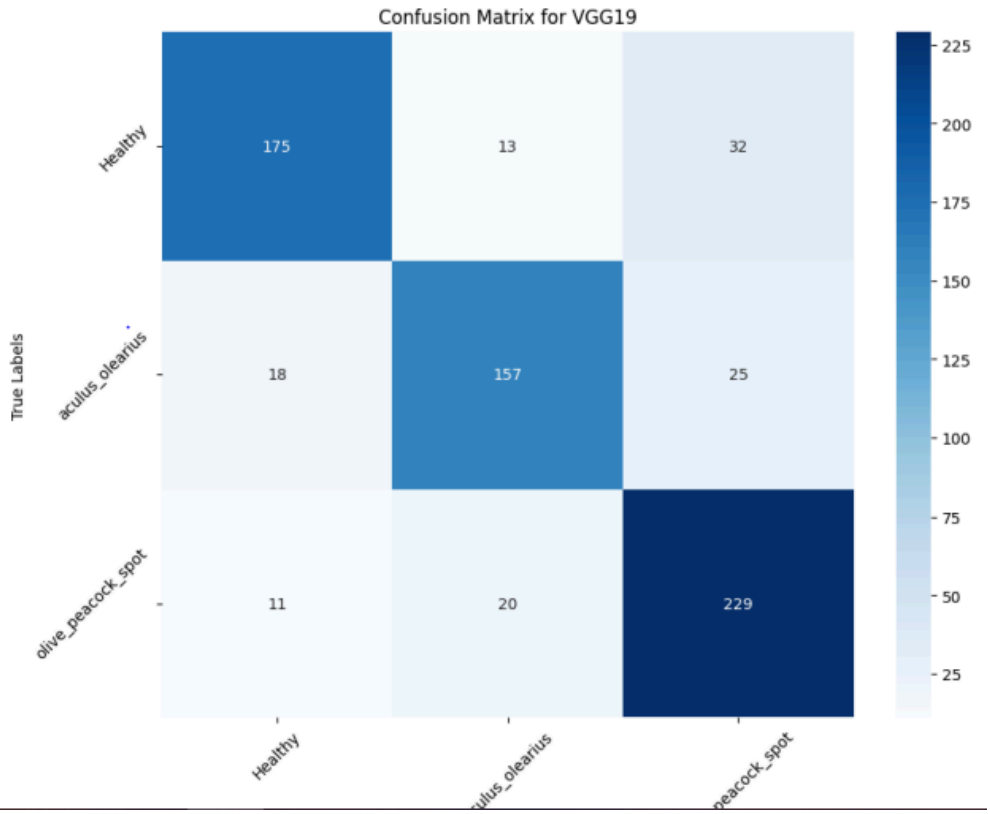
Custom CNN Model with Data Augmentation (Section 3.2)

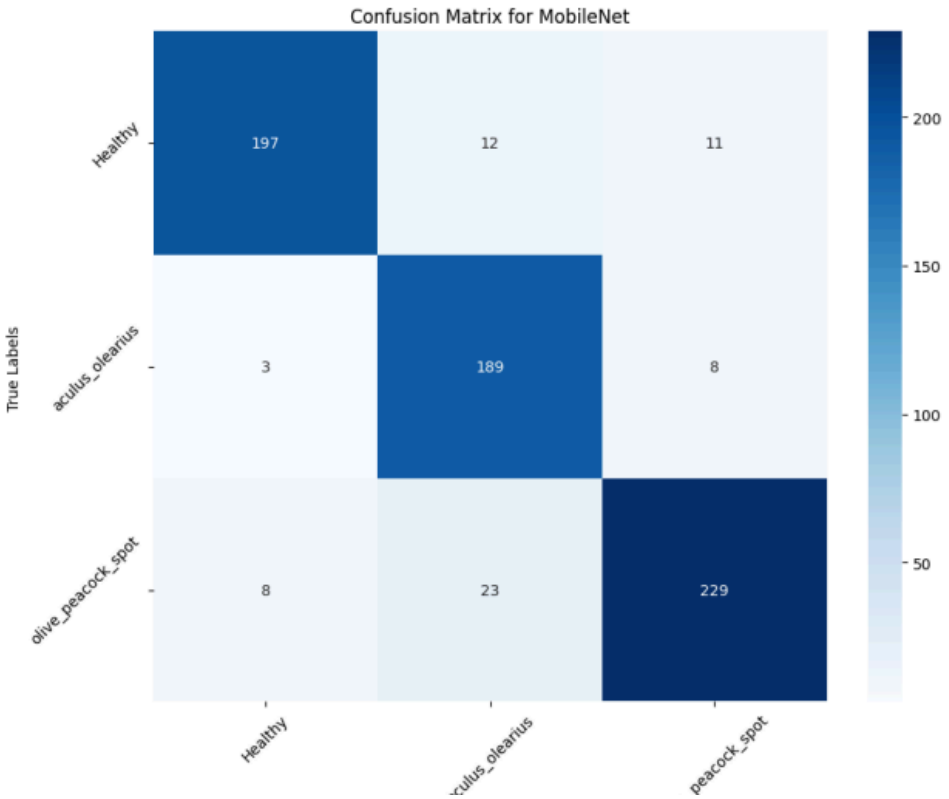



```
🔹 Confusion Matrix:
[[117  69  34]
 [  4 184  12]
 [  1  30 229]]
```

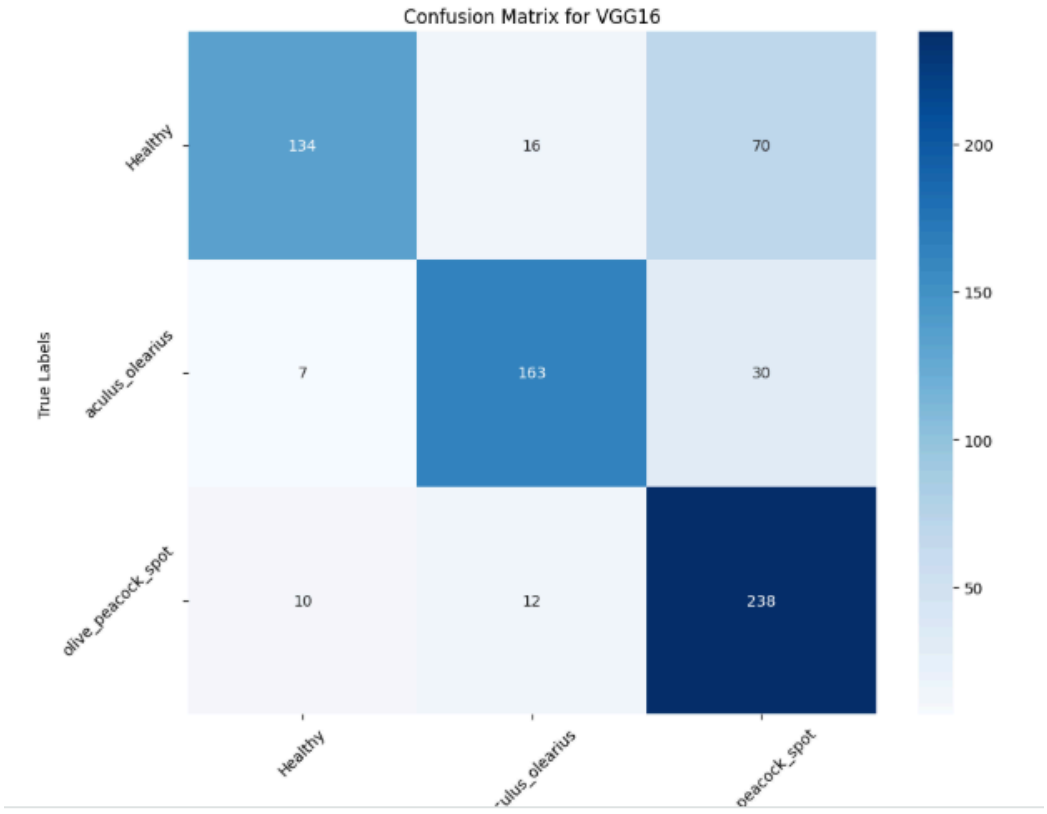
Pre-trained Transfer Learning Model without Data Augmentation (Section 3.3)

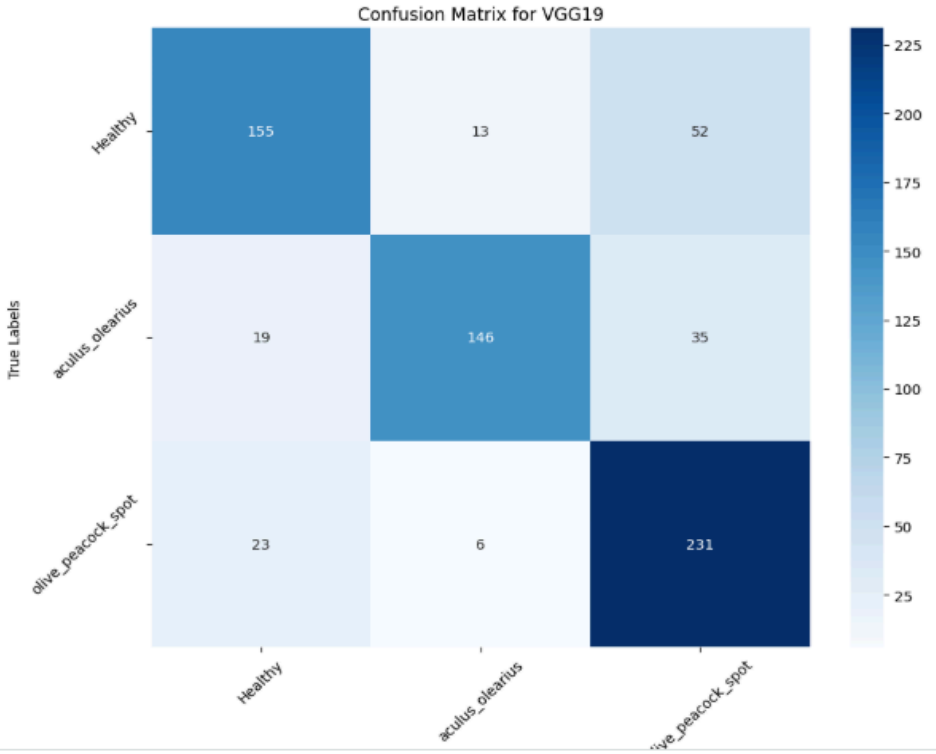


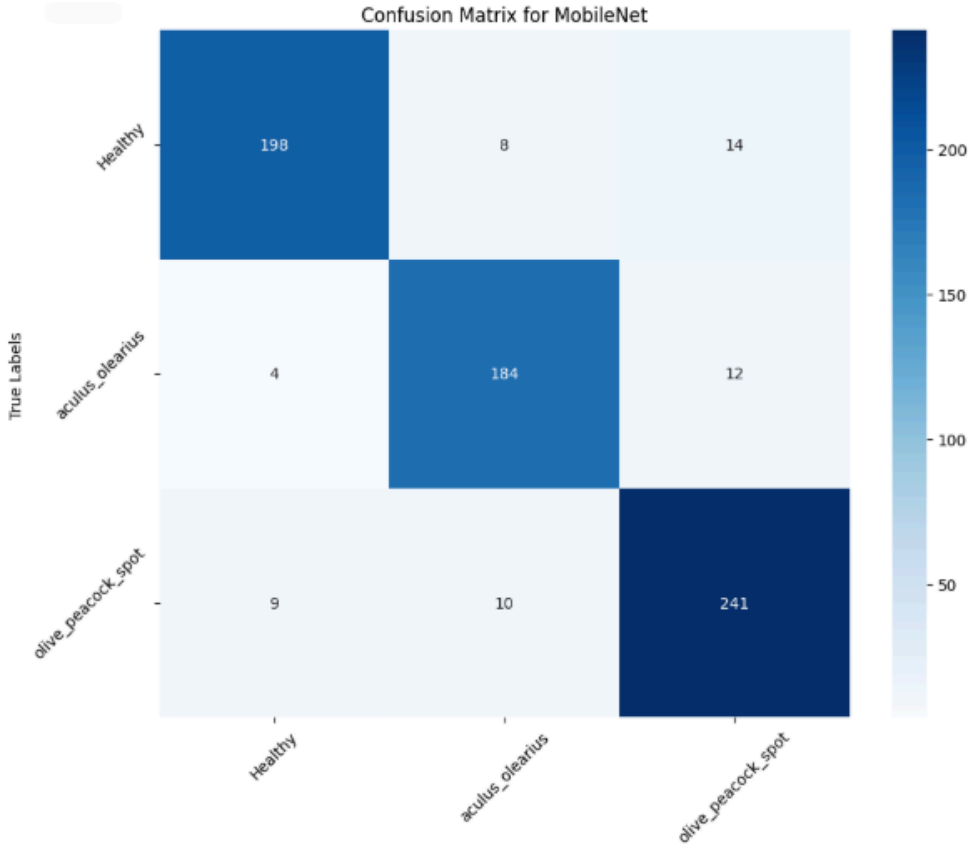




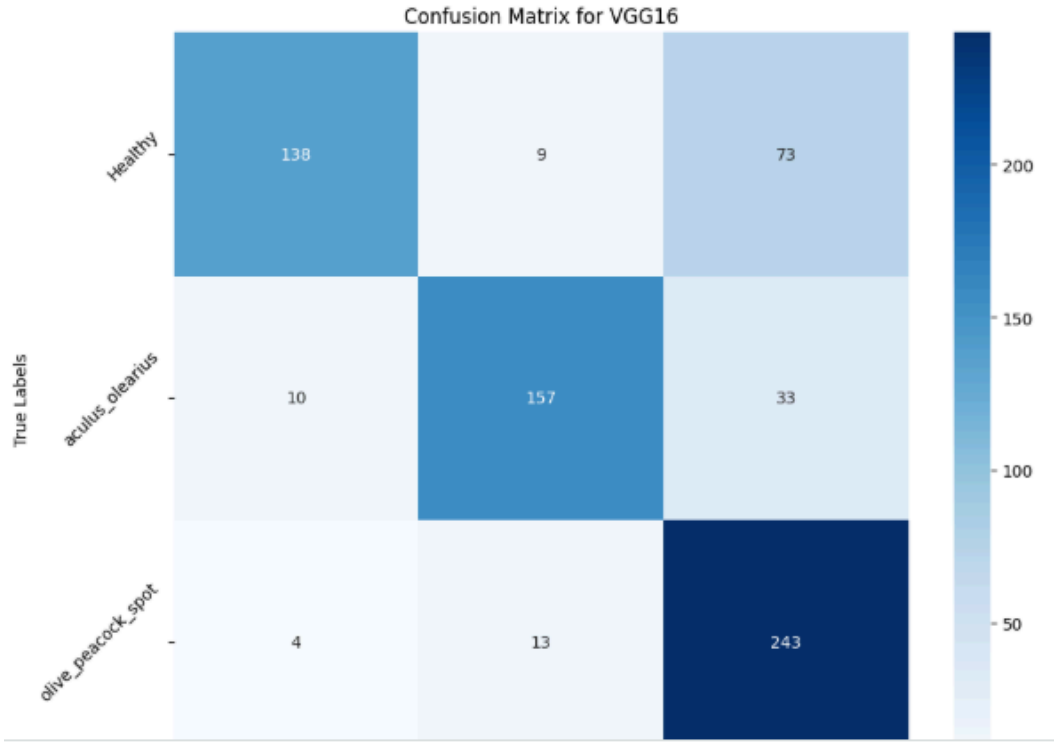
Pre-trained Transfer Learning Model with Data Augmentation (Section 3.4)

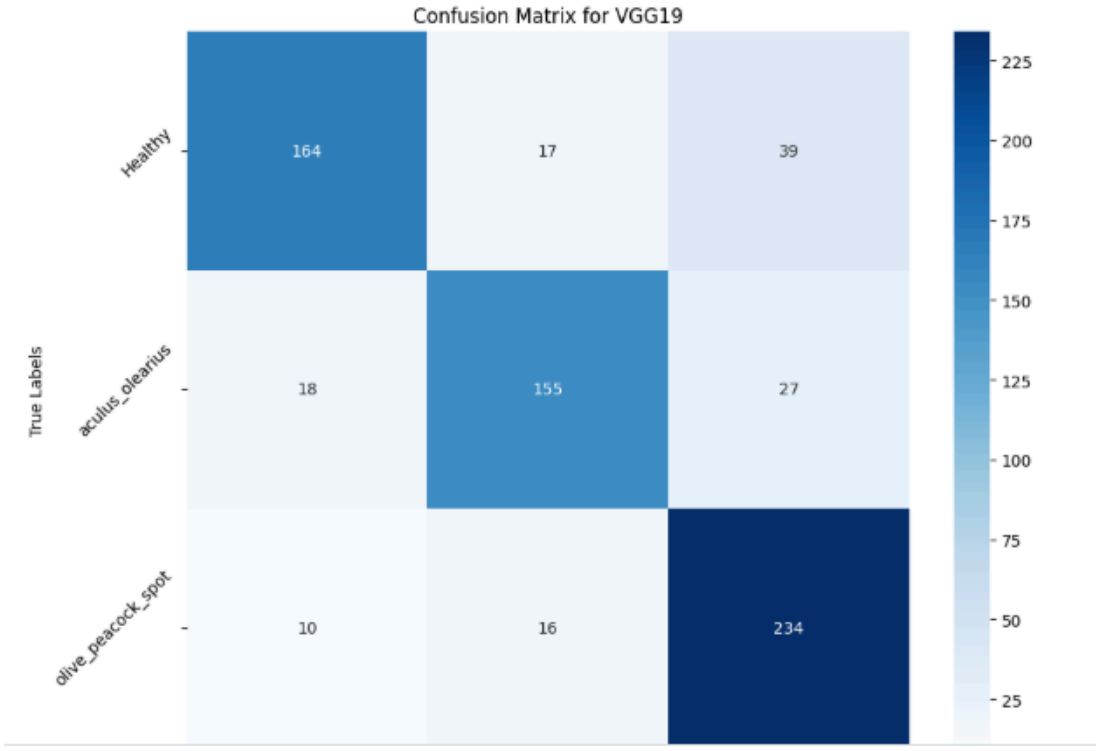


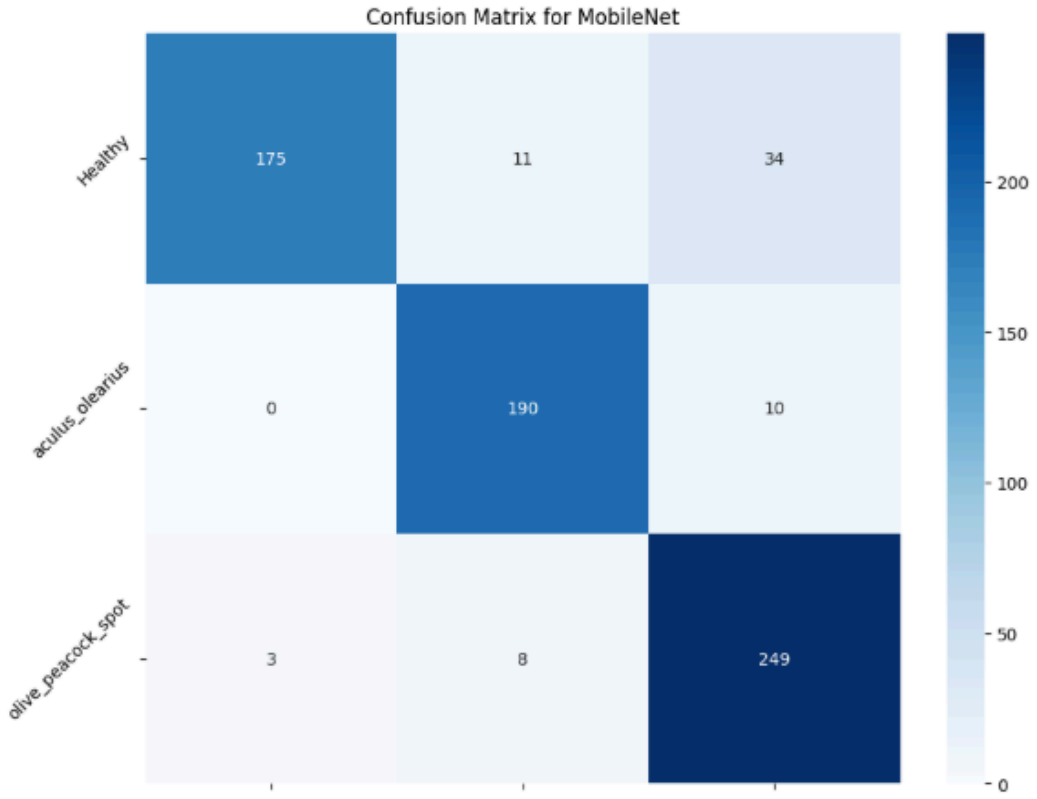

Confusion Matrix for VGG19
True Labels
Healthy
aculus_olearius
olive_peacock_spot
155
13
52
19
146
35
23
6
231
225
200
175
150
125
100
75
50
25
Healthy
aculus_olearius


Confusion Matrix for MobileNet
True Labels
Healthy
aculus_olearius
olive_peacock_spot
198
8
14
4
184
12
9
10
241
200
150
100
50
Healthy
aculus_olearius
olive_peacock_spot

Fine-tuning Pre-trained Transfer Learning Model with Data Augmentation (Section 3.5)







4.3. Classification reports for each model

The classification report evaluates model performance using key metrics:
Accuracy: Overall correctness of the model.
Precision: How many predicted positives are actually correct.
Recall (Sensitivity): How many actual positives were correctly predicted.
F1-score: A balance between precision and recall.
Each model's report provides these metrics for three classes (Healthy, Aculus Olearius, Olive Peacock Spot), helping compare model effectiveness.

Custom CNN Model without Data Augmentation (Section 3.1)

🔷 Classification Report for Adam:

	precision	recall	f1-score	support
0	0.86	0.75	0.80	220
1	0.72	0.79	0.75	200
2	0.83	0.87	0.85	260
accuracy			0.81	680
macro avg	0.81	0.80	0.80	680
weighted avg	0.81	0.81	0.81	680

Custom CNN Model with Data Augmentation (Section 3.2)

🔷 Classification Report for Adam with Data Augmentation:

	precision	recall	f1-score	support
0	0.96	0.53	0.68	220
1	0.65	0.92	0.76	200
2	0.83	0.88	0.86	260
accuracy			0.78	680
macro avg	0.81	0.78	0.77	680
weighted avg	0.82	0.78	0.77	680

Pre-trained Transfer Learning Model without Data Augmentation (Section 3.3)

🔷 Classification Report for VGG16:

	precision	recall	f1-score	support
0	0.80	0.84	0.82	220
1	0.94	0.74	0.83	200
2	0.79	0.89	0.84	260
accuracy			0.83	680
macro avg	0.85	0.82	0.83	680
weighted avg	0.84	0.83	0.83	680

🔷 Classification Report for VGG19:

	precision	recall	f1-score	support
0	0.86	0.80	0.83	220
1	0.83	0.79	0.81	200
2	0.80	0.88	0.84	260
accuracy			0.82	680
macro avg	0.83	0.82	0.82	680
weighted avg	0.83	0.82	0.82	680

🔷 Classification Report for MobileNet:

	precision	recall	f1-score	support
0	0.95	0.90	0.92	220
1	0.84	0.94	0.89	200
2	0.92	0.88	0.90	260
accuracy			0.90	680
macro avg	0.90	0.91	0.90	680
weighted avg	0.91	0.90	0.90	680

Pre-trained Transfer Learning Model with Data Augmentation (Section 3.4)

◆ Classification Report for VGG16:

	precision	recall	f1-score	support
0	0.89	0.61	0.72	220
1	0.85	0.81	0.83	200
2	0.70	0.92	0.80	260
accuracy			0.79	680
macro avg	0.81	0.78	0.78	680
weighted avg	0.81	0.79	0.78	680

◆ Classification Report for VGG19:

	precision	recall	f1-score	support
0	0.79	0.70	0.74	220
1	0.88	0.73	0.80	200
2	0.73	0.89	0.80	260
accuracy			0.78	680
macro avg	0.80	0.77	0.78	680
weighted avg	0.79	0.78	0.78	680

◆ Classification Report for MobileNet:

	precision	recall	f1-score	support
0	0.94	0.90	0.92	220
1	0.91	0.92	0.92	200
2	0.90	0.93	0.91	260
accuracy			0.92	680
macro avg	0.92	0.92	0.92	680
weighted avg	0.92	0.92	0.92	680

Fine-tuning Pre-trained Transfer Learning Model with Data Augmentation (Section 3.5)

🔷 Classification Report for VGG16:

	precision	recall	f1-score	support
0	0.91	0.63	0.74	220
1	0.88	0.79	0.83	200
2	0.70	0.93	0.80	260
accuracy			0.79	680
macro avg	0.83	0.78	0.79	680
weighted avg	0.82	0.79	0.79	680

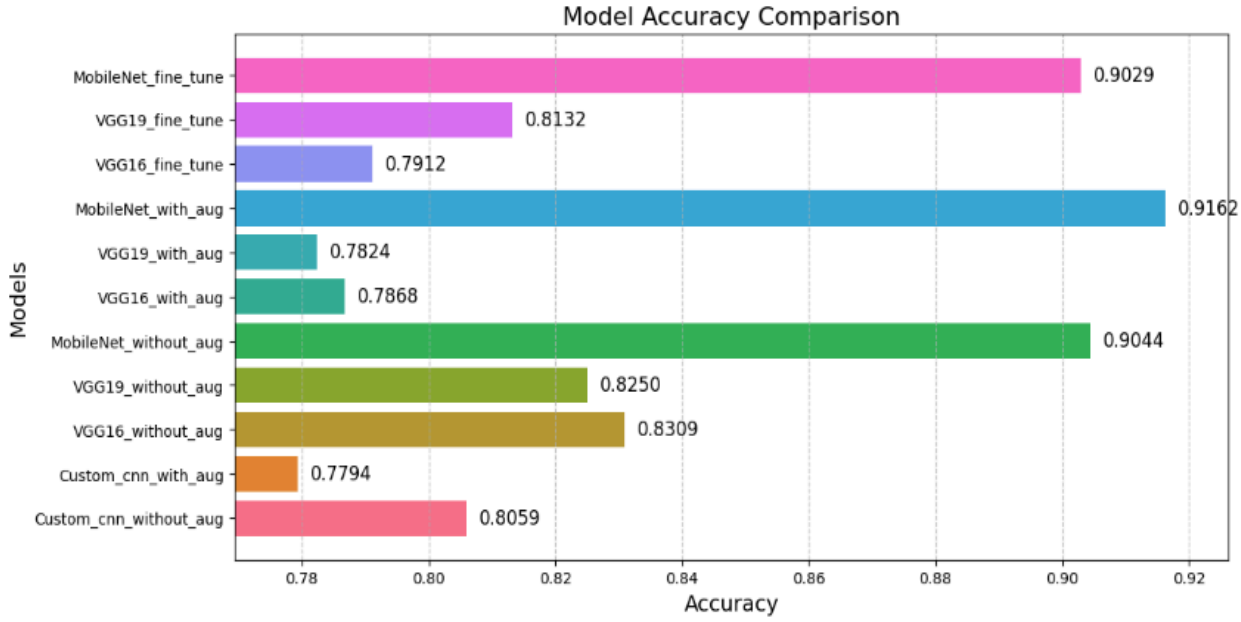
🔷 Classification Report for VGG19:

	precision	recall	f1-score	support
0	0.85	0.75	0.80	220
1	0.82	0.78	0.80	200
2	0.78	0.90	0.84	260
accuracy			0.81	680
macro avg	0.82	0.81	0.81	680
weighted avg	0.82	0.81	0.81	680

🔷 Classification Report for MobileNet:

	precision	recall	f1-score	support
0	0.98	0.80	0.88	220
1	0.91	0.95	0.93	200
2	0.85	0.96	0.90	260
accuracy			0.90	680
macro avg	0.91	0.90	0.90	680
weighted avg	0.91	0.90	0.90	680

5.Model Comparison



6.Discussion

The study evaluated various CNN approaches for classifying olive leaf diseases, revealing key insights into model performance and data handling. Pre-trained models, particularly MobileNet, outperformed the custom CNN by a significant margin. MobileNet achieved the highest accuracy **(91.62%)** when combined with data augmentation, showcasing its ability to adapt to the olive leaf dataset. This success can be attributed to transfer learning, where pre-trained models leverage features learned from large datasets like ImageNet, reducing the need for extensive training data.

Data augmentation proved beneficial for MobileNet but had mixed effects on VGG models. For instance, VGG16's accuracy dropped slightly with augmentation, suggesting that aggressive transformations might not align well with VGG's architecture or that the model was already prone to overfitting. Fine-tuning further highlighted MobileNet's adaptability, as its performance remained robust even after adjusting pre-trained layers. In contrast, VGG models saw only marginal improvements, indicating that their deeper architectures might require more data or careful hyperparameter tuning to avoid overfitting.

The custom CNN, while simpler, struggled to match pre-trained models, achieving a maximum accuracy of **80.59%**. Its lower performance likely stems from limited depth and fewer parameters, which restricted its ability to capture complex features. However, its decent performance without augmentation suggests potential for improvement with architectural adjustments or larger datasets.

Class-wise metrics revealed that MobileNet excelled in distinguishing all three classes (Healthy, Aculus olearius, Olive Peacock Spot), with balanced precision and recall. In contrast, the custom model showed uneven performance, particularly for class 1 (Aculus olearius), where precision and recall were lower, indicating misclassifications.

7.Conclusion

In conclusion, MobileNet combined with data augmentation emerged as the most effective model for olive leaf disease classification, achieving 91.62% accuracy. This highlights the practicality of transfer learning for agricultural tasks with limited datasets. Pre-trained models like MobileNet, designed for efficiency and adaptability, are well-suited for real-world applications where computational resources and data availability are constraints. While the custom CNN showed promise, its performance lagged behind pre-trained models, underscoring the challenge of building effective architectures from scratch without extensive data. Future work could explore hybrid approaches, integrating domain-specific augmentations or testing newer architectures like EfficientNet. Overall, this study demonstrates that leveraging pre-trained models with strategic augmentation offers a reliable path toward accurate and scalable plant disease detection systems.

8.References

[1] S. Uguz, "Classification of olive leaf diseases using deep convolutional neural networks," ResearchGate, 2020. Available: https://www.researchgate.net/profile/Sinan-Uguz/publication/343376465_Classification_of_olive_leaf_diseases_using_deep_convolutional_neural_networks/links/64097c6e0cf1030a5685ce89/Classification-of-olive-leaf-diseases-using-deep-convolutional-neural-networks.pdf.

[2] S. Uguz, "CNN Olive Dataset," GitHub Repository, 2020. Available: https://github.com/sinanuguz/CNN_olive_dataset.