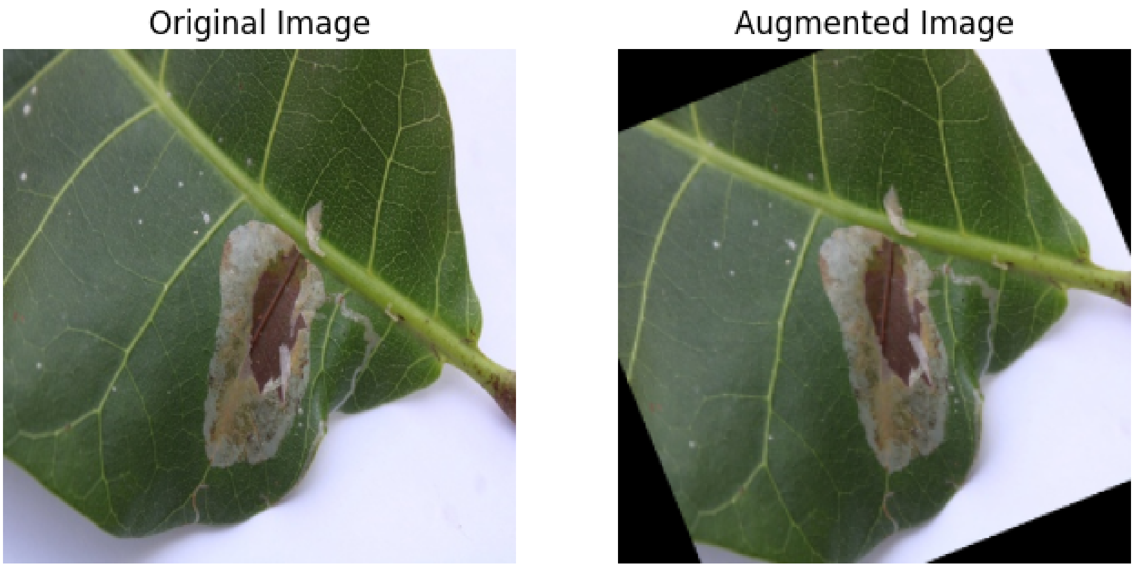
In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
import tensorflow as tf
import tensorflow.keras as k
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, AverageP
import random
```

```
WARNING:tensorflow:From c:\Users\mgssr\AppData\Local\Programs\Python
\Python311\Lib\site-packages\keras\src\losses.py:2976: The name tf.l
osses.sparse_softmax_cross_entropy is deprecated. Please use tf.comp
at.v1.losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```
path_folder = "Mod_Plant"
class_name = os.listdir(path_folder)
class_name.sort()
```

In [18]:
```
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt

def augment_data(images, labels, target_shape=(224, 224)):
    augmented_images = []
    augmented_labels = []

    for image, label in zip(images, labels):
        # Randomly select augmentation techniques
        augmentation_type = random.choice(['rotate', 'zoom'])

        if augmentation_type == 'rotate':
            # Randomly rotate the image by a degree between -20 and 20
            angle = random.randint(-90, 90)
            image = rotate_image(image, angle)
        elif augmentation_type == 'zoom':
            # Randomly zoom the image by a factor between 0.8 and 1.2
            zoom_factor = random.uniform(0.8, 1.25)
            image = zoom_image(image, zoom_factor)

        # Resize the image to the target shape
        image = cv2.resize(image, target_shape)
        augmented_images.append(image)
        augmented_labels.append(label)

    return np.array(augmented_images), np.array(augmented_labels)

# Define a function to rotate the image
def rotate_image(image, angle):
    height, width = image.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), ang
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, hei
    return rotated_image

# Define a function to zoom the image
def zoom_image(image, zoom_factor):
    height, width = image.shape[:2]
    zoomed_image = cv2.resize(image, (int(width * zoom_factor), int(he
    return zoomed_image

# Load an example image
image = cv2.imread("Mod_Plant\Cashew_leaf miner\leaf miner7_.jpg")

# Augment the image
augmented_image, _ = augment_data([image], [0])  # Assuming label 0 fo

# Plot the original and augmented images
plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(augmented_image[0], cv2.COLOR_BGR2RGB))
plt.title('Augmented Image')
plt.axis('off')
```

```
plt.show()
```



In [4]:
```
# Load images and labels
image_data = []
label_data = []
count = 0
for folder in class_name:
    images = os.listdir(path_folder + "/" + folder)
    print("Loading Folder -- {} " .format(folder), "The Count of Class
    for img in images:
        image = cv2.imread(path_folder + "/" + folder + "/" + img)
        image = cv2.resize(image, (224, 224))

        image_data.append(image)
        label_data.append(count)
    count += 1

print("---- Done ----------- ")
```

```
Loading Folder -- Cashew_leaf miner  The Count of Classes ==>  0
Loading Folder -- Cassava_brown spot  The Count of Classes ==>  1
Loading Folder -- Healthy  The Count of Classes ==>  2
Loading Folder -- Maize_leaf blight  The Count of Classes ==>  3
Loading Folder -- Tomato_septoria leaf spot  The Count of Classes ==
>  4
---- Done -----------
```

In [5]:
```
data = np.array(image_data)
data = data.astype("float32")
data = data/255.0

label = np.array(label_data)
```

In [6]:
```
print(data.shape)
```

```
(500, 224, 224, 3)
```

In [7]:
```
label_num = to_categorical(label, len(class_name))
```

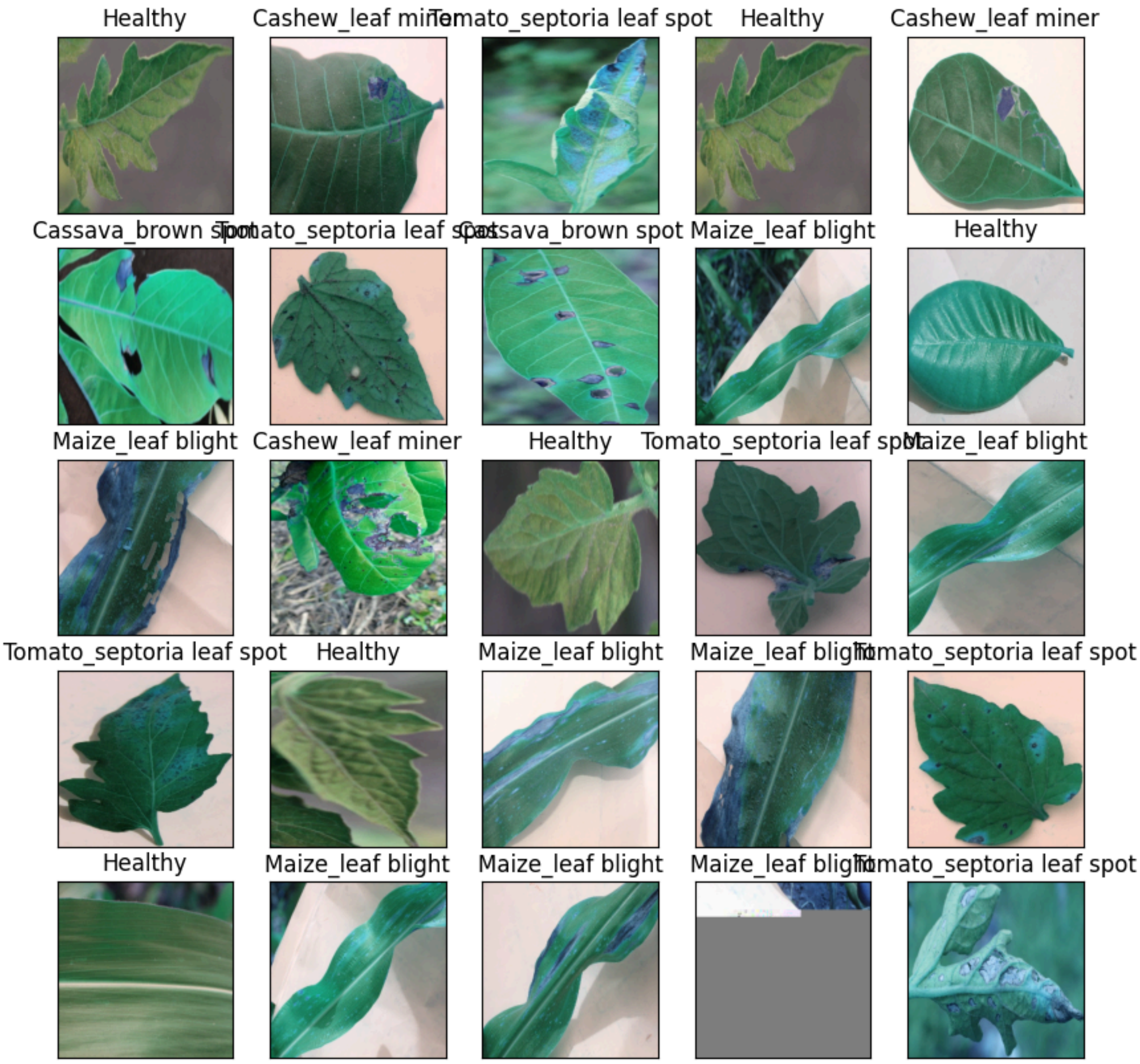
In [8]:
```
x_img, y_img = shuffle(data, label_num)
x_train, x_test, y_train, y_test = train_test_split(x_img, y_img, trai
```

In [9]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[9]:
```
((400, 224, 224, 3), (400, 5), (100, 224, 224, 3), (100, 5))
```

In [10]:
```
x_train_augmented, y_train_augmented = augment_data(x_train, y_train)
```

In [11]:
```
plt.figure(figsize=(10, 10))
for i in range(0, 25):
    plt.subplot(5, 5, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i])
    plt.title(class_name[np.argmax(y_train[i])])
```

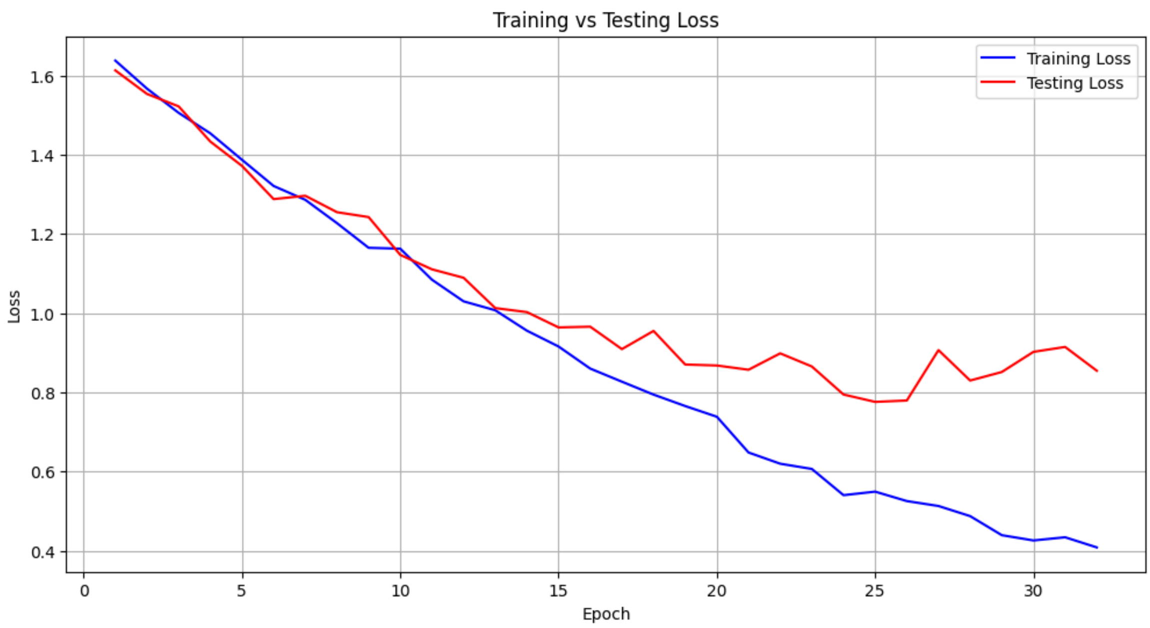


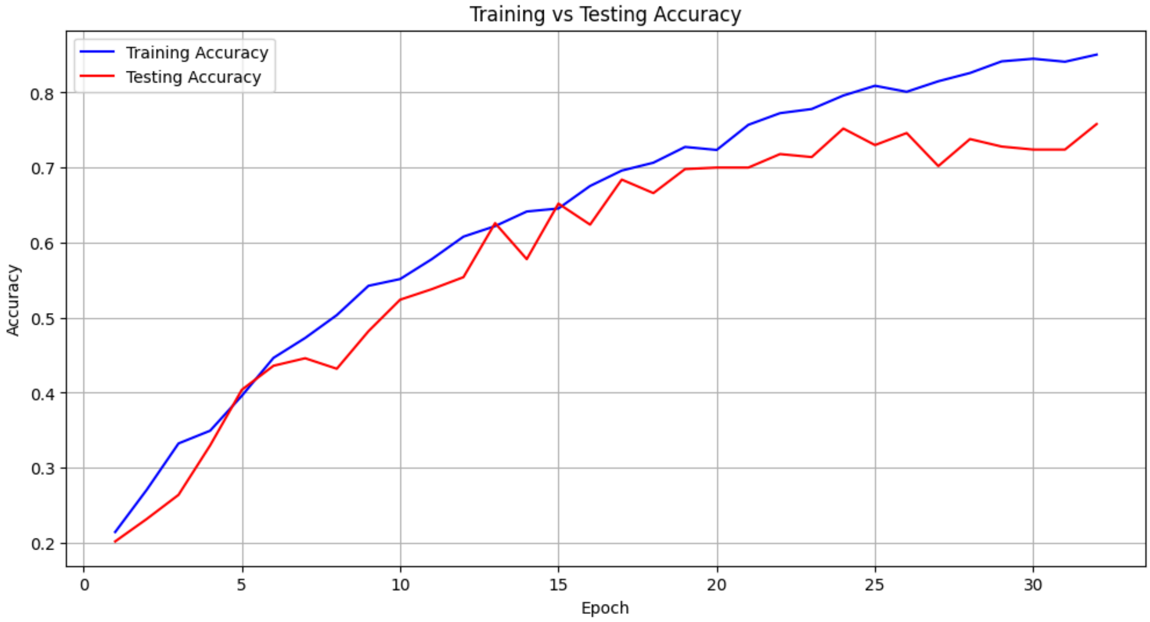
In [17]:
```
import numpy as np
from tensorflow.keras import optimizers  # Import optimizers from tens

# Define the number of random states and epochs
num_random_states = 5
num_epochs = 32

# Define lists to store accuracies and losses for each random state
train_losses_per_epoch = [[] for _ in range(num_epochs)]
val_losses_per_epoch = [[] for _ in range(num_epochs)]
train_accuracies_per_epoch = [[] for _ in range(num_epochs)]
val_accuracies_per_epoch = [[] for _ in range(num_epochs)]

for random_state in range(num_random_states):
    # Set random seed for reproducibility
    np.random.seed(random_state)

    # Build the model
    model = k.models.Sequential()
    model.add(k.layers.Conv2D(16, (5, 5), activation="relu", input_sha
    model.add(k.layers.AveragePooling2D((2, 2)))
    model.add(k.layers.Conv2D(32, (4, 4), activation="relu", padding='
    model.add(k.layers.AveragePooling2D((2, 2)))
    model.add(k.layers.Conv2D(64, (3, 3), activation="relu", padding='
    model.add(k.layers.AveragePooling2D((2, 2)))
    model.add(k.layers.Conv2D(128, (2, 2), activation="relu", padding=
    model.add(k.layers.MaxPool2D((2, 2)))
    model.add(k.layers.Flatten())
    model.add(k.layers.Dense(128, activation="relu"))
    model.add(k.layers.Dropout(0.5))
    model.add(k.layers.Dense(24, activation="relu"))
    model.add(k.layers.Dropout(0.1))
    model.add(k.layers.Dense(5, activation="softmax"))

    # Define the optimizer with the desired learning rate
    adam_optimizer = optimizers.Adam(learning_rate=0.001)

    # Compile the model
    model.compile(optimizer=adam_optimizer, loss=k.losses.Categorical(

    # Train the model with validation data
    history = model.fit(x_train_augmented, y_train_augmented, epochs=r

    # Store loss and accuracy for each epoch
    for epoch in range(num_epochs):
        train_losses_per_epoch[epoch].append(history.history['loss'][e
        val_losses_per_epoch[epoch].append(history.history['val_loss']
        train_accuracies_per_epoch[epoch].append(history.history['accu
        val_accuracies_per_epoch[epoch].append(history.history['val_ac

# Calculate average accuracies and losses for each epoch
avg_train_losses = [np.mean(losses) for losses in train_losses_per_epo
avg_val_losses = [np.mean(losses) for losses in val_losses_per_epoch]
avg_train_accuracies = [np.mean(accuracies) for accuracies in train_ac
avg_val_accuracies = [np.mean(accuracies) for accuracies in val_accura

# Print the loss for both training and validation along with average a
for epoch, (avg_train_loss, avg_val_loss, avg_train_acc, avg_val_acc)
    print(f"Epoch {epoch + 1}: Average Training Loss = {avg_train_loss
```

```
Epoch 1: Average Training Loss = 1.6380189180374145, Average Validat
ion Loss = 1.613264536857605, Average Training Accuracy = 0.21449999
809265136, Average Validation Accuracy = 0.2020000010728836
Epoch 2: Average Training Loss = 1.5672228574752807, Average Validat
ion Loss = 1.5538297176361084, Average Training Accuracy = 0.2710000
00834465, Average Validation Accuracy = 0.23200000077486038
Epoch 3: Average Training Loss = 1.5060779333114624, Average Validat
ion Loss = 1.5223712921142578, Average Training Accuracy = 0.3324999
988079071, Average Validation Accuracy = 0.26400000154972075
Epoch 4: Average Training Loss = 1.4542717218399048, Average Validat
ion Loss = 1.4335555553436279, Average Training Accuracy = 0.3494999
945163727, Average Validation Accuracy = 0.3300000011920929
Epoch 5: Average Training Loss = 1.3874922513961792, Average Validat
ion Loss = 1.3720906257629395, Average Training Accuracy = 0.3959999
978542328, Average Validation Accuracy = 0.4040000081062317
Epoch 6: Average Training Loss = 1.3213216304779052, Average Validat
ion Loss = 1.2881744861602784, Average Training Accuracy = 0.4465000
033378601, Average Validation Accuracy = 0.435999995470047
Epoch 7: Average Training Loss = 1.2863877534866333, Average Validat
ion Loss = 1.2965998888015746, Average Training Accuracy = 0.4730000
0190734864, Average Validation Accuracy = 0.4459999918937683
Epoch 8: Average Training Loss = 1.2275411367416382, Average Validat
ion Loss = 1.255146050453186, Average Training Accuracy = 0.50349999
66621398, Average Validation Accuracy = 0.4319999933242798
Epoch 9: Average Training Loss = 1.1652117252349854, Average Validat
ion Loss = 1.242788302898407, Average Training Accuracy = 0.54249998
92711639, Average Validation Accuracy = 0.4819999933242798
Epoch 10: Average Training Loss = 1.1629414319992066, Average Valida
tion Loss = 1.147287654876709, Average Training Accuracy = 0.5514999
92609024, Average Validation Accuracy = 0.5239999890327454
Epoch 11: Average Training Loss = 1.0846650719642639, Average Valida
tion Loss = 1.110866093635559, Average Training Accuracy = 0.5779999
971389771, Average Validation Accuracy = 0.5379999876022339
Epoch 12: Average Training Loss = 1.0299795985221862, Average Valida
tion Loss = 1.0895005702972411, Average Training Accuracy = 0.608000
0042915344, Average Validation Accuracy = 0.5539999961853027
Epoch 13: Average Training Loss = 1.0074589729309082, Average Valida
tion Loss = 1.0129522442817689, Average Training Accuracy = 0.622000
002861023, Average Validation Accuracy = 0.6260000109672547
Epoch 14: Average Training Loss = 0.9558757781982422, Average Valida
tion Loss = 1.0027688980102538, Average Training Accuracy = 0.641499
9961853027, Average Validation Accuracy = 0.5779999852180481
Epoch 15: Average Training Loss = 0.915880537033081, Average Validat
ion Loss = 0.9641321301460266, Average Training Accuracy = 0.6455000
162124633, Average Validation Accuracy = 0.6520000100135803
Epoch 16: Average Training Loss = 0.8601457118988037, Average Valida
tion Loss = 0.9659512758255004, Average Training Accuracy = 0.675500
0114440918, Average Validation Accuracy = 0.6240000128746033
Epoch 17: Average Training Loss = 0.8270924925804138, Average Valida
tion Loss = 0.9093442559242249, Average Training Accuracy = 0.695999
9918937683, Average Validation Accuracy = 0.6840000033378602
Epoch 18: Average Training Loss = 0.794433331489563, Average Validat
ion Loss = 0.9552288055419922, Average Training Accuracy = 0.7065000
057220459, Average Validation Accuracy = 0.6660000085830688
Epoch 19: Average Training Loss = 0.7655238270759582, Average Valida
tion Loss = 0.8704094648361206, Average Training Accuracy = 0.727500
0095367432, Average Validation Accuracy = 0.6980000138282776
Epoch 20: Average Training Loss = 0.7385206818580627, Average Valida
tion Loss = 0.8679668426513671, Average Training Accuracy = 0.723500
0014305115, Average Validation Accuracy = 0.700000011920929
Epoch 21: Average Training Loss = 0.6482043385505676, Average Valida
```

```
tion Loss = 0.8571983098983764, Average Training Accuracy = 0.757000
0052452087, Average Validation Accuracy = 0.7
Epoch 22: Average Training Loss = 0.6199059605598449, Average Valida
tion Loss = 0.8986073136329651, Average Training Accuracy = 0.772500
0023841858, Average Validation Accuracy = 0.7180000066757202
Epoch 23: Average Training Loss = 0.6066641807556152, Average Valida
tion Loss = 0.8654513478279113, Average Training Accuracy = 0.777999
997138977, Average Validation Accuracy = 0.7139999985694885
Epoch 24: Average Training Loss = 0.5404273390769958, Average Valida
tion Loss = 0.794665265083313, Average Training Accuracy = 0.7960000
038146973, Average Validation Accuracy = 0.7519999861717224
Epoch 25: Average Training Loss = 0.5493421912193298, Average Valida
tion Loss = 0.7760321259498596, Average Training Accuracy = 0.809000
0033378602, Average Validation Accuracy = 0.7300000071525574
Epoch 26: Average Training Loss = 0.5255245804786682, Average Valida
tion Loss = 0.7795137524604797, Average Training Accuracy = 0.800999
9990463257, Average Validation Accuracy = 0.7459999918937683
Epoch 27: Average Training Loss = 0.513078111410141, Average Validat
ion Loss = 0.9068312525749207, Average Training Accuracy = 0.8149999
976158142, Average Validation Accuracy = 0.7020000100135804
Epoch 28: Average Training Loss = 0.487531715631485, Average Validat
ion Loss = 0.8299837589263916, Average Training Accuracy = 0.8259999
990463257, Average Validation Accuracy = 0.7380000114440918
Epoch 29: Average Training Loss = 0.4393775224685669, Average Valida
tion Loss = 0.8515149831771851, Average Training Accuracy = 0.841500
0081062317, Average Validation Accuracy = 0.727999997138977
Epoch 30: Average Training Loss = 0.42608131766319274, Average Valid
ation Loss = 0.9023982048034668, Average Training Accuracy = 0.84500
00047683716, Average Validation Accuracy = 0.7239999890327453
Epoch 31: Average Training Loss = 0.43406533598899844, Average Valid
ation Loss = 0.9148020505905151, Average Training Accuracy = 0.84099
9984741211, Average Validation Accuracy = 0.7239999890327453
Epoch 32: Average Training Loss = 0.4085033297538757, Average Valida
tion Loss = 0.8547578454017639, Average Training Accuracy = 0.850499
975681305, Average Validation Accuracy = 0.7580000042915345
```

In [18]:
```
import matplotlib.pyplot as plt

# Plot training vs Testing loss
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), avg_train_losses, label='Training L
plt.plot(range(1, num_epochs + 1), avg_val_losses, label='Testing Loss
plt.title('Training vs Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot training vs Testing accuracy
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), avg_train_accuracies, label='Traini
plt.plot(range(1, num_epochs + 1), avg_val_accuracies, label='Testing 
plt.title('Training vs Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```





In [29]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_train)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_train, axis=1)  # Convert one-hot encoded labels

# Print classification report
print(classification_report(y_true, y_pred))
```

```
13/13 [==============================] - 1s 65ms/step
              precision    recall  f1-score   support

           0       0.70      0.97      0.81        79
           1       0.94      0.64      0.76        78
           2       0.94      0.86      0.89        84
           3       0.99      0.95      0.97        85
           4       0.90      0.95      0.92        74

    accuracy                           0.88       400
   macro avg       0.89      0.87      0.87       400
weighted avg       0.89      0.88      0.87       400
```

In [30]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels t

# Print classification report
print(classification_report(y_true, y_pred))
```

```
4/4 [==============================] - 0s 57ms/step
              precision    recall  f1-score   support

           0       0.44      0.67      0.53        21
           1       0.67      0.45      0.54        22
           2       0.64      0.56      0.60        16
           3       0.70      0.93      0.80        15
           4       0.95      0.69      0.80        26

    accuracy                           0.65       100
   macro avg       0.68      0.66      0.65       100
weighted avg       0.69      0.65      0.65       100
```

In [19]:
```
import numpy as np
from keras.utils import to_categorical
from keras.applications.vgg19 import VGG19
from keras.layers import Dense, Flatten
from keras.models import Model
import cv2
import os
import random
# Load pre-trained VGG19 model
vgg19 = VGG19(weights='imagenet', include_top=False, input_shape=(224,

# Freeze the layers
for layer in vgg19.layers:
    layer.trainable = False
```

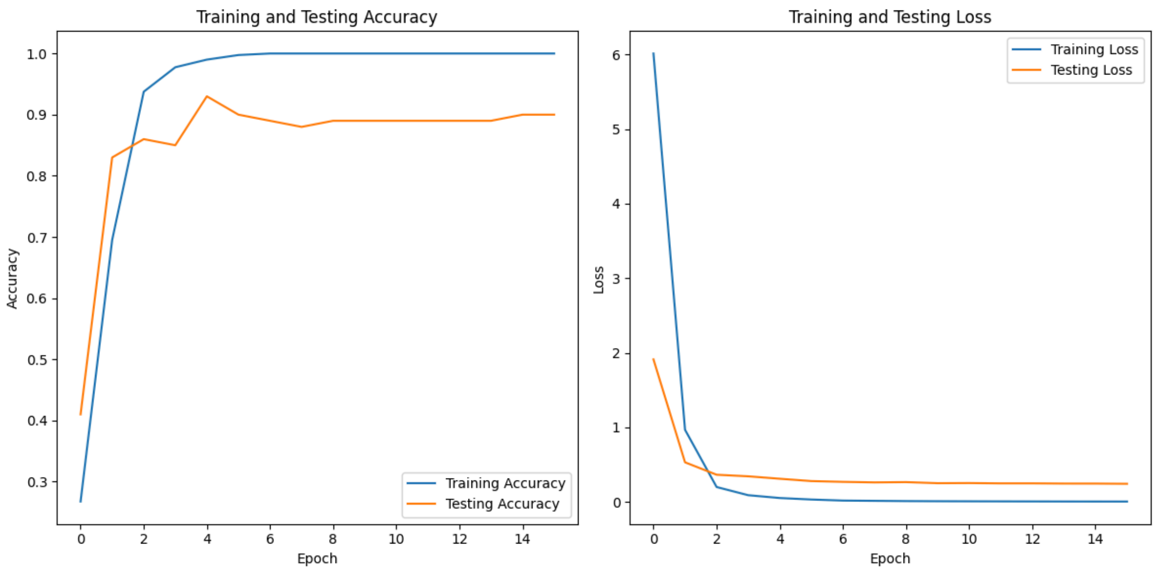
In [20]:
```
# Add custom layers for classification
x = Flatten()(vgg19.output)
x = Dense(512, activation='relu')(x)
predictions = Dense(len(class_name), activation='softmax')(x)

# Create model
model = Model(inputs=vgg19.input, outputs=predictions)
```

In [21]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metri
```

In [22]:
```
# Retrain the model with augmented data
history=model.fit(x_train_augmented, y_train_augmented, validation_dat
```

```
Epoch 1/16
13/13 [==============================] - 48s 4s/step - loss: 6.0132
- accuracy: 0.2675 - val_loss: 1.9112 - val_accuracy: 0.4100
Epoch 2/16
13/13 [==============================] - 46s 4s/step - loss: 0.9672
- accuracy: 0.6950 - val_loss: 0.5307 - val_accuracy: 0.8300
Epoch 3/16
13/13 [==============================] - 48s 4s/step - loss: 0.1996
- accuracy: 0.9375 - val_loss: 0.3642 - val_accuracy: 0.8600
Epoch 4/16
13/13 [==============================] - 46s 4s/step - loss: 0.0897
- accuracy: 0.9775 - val_loss: 0.3431 - val_accuracy: 0.8500
Epoch 5/16
13/13 [==============================] - 46s 4s/step - loss: 0.0518
- accuracy: 0.9900 - val_loss: 0.3104 - val_accuracy: 0.9300
Epoch 6/16
13/13 [==============================] - 46s 4s/step - loss: 0.0317
- accuracy: 0.9975 - val_loss: 0.2792 - val_accuracy: 0.9000
Epoch 7/16
13/13 [==============================] - 45s 4s/step - loss: 0.0176
- accuracy: 1.0000 - val_loss: 0.2689 - val_accuracy: 0.8900
Epoch 8/16
13/13 [==============================] - 46s 4s/step - loss: 0.0141
- accuracy: 1.0000 - val_loss: 0.2616 - val_accuracy: 0.8800
Epoch 9/16
13/13 [==============================] - 46s 4s/step - loss: 0.0106
- accuracy: 1.0000 - val_loss: 0.2653 - val_accuracy: 0.8900
Epoch 10/16
13/13 [==============================] - 46s 4s/step - loss: 0.0089
- accuracy: 1.0000 - val_loss: 0.2509 - val_accuracy: 0.8900
Epoch 11/16
13/13 [==============================] - 46s 4s/step - loss: 0.0078
- accuracy: 1.0000 - val_loss: 0.2526 - val_accuracy: 0.8900
Epoch 12/16
13/13 [==============================] - 46s 4s/step - loss: 0.0070
- accuracy: 1.0000 - val_loss: 0.2486 - val_accuracy: 0.8900
Epoch 13/16
13/13 [==============================] - 46s 4s/step - loss: 0.0062
- accuracy: 1.0000 - val_loss: 0.2488 - val_accuracy: 0.8900
Epoch 14/16
13/13 [==============================] - 46s 4s/step - loss: 0.0056
- accuracy: 1.0000 - val_loss: 0.2460 - val_accuracy: 0.8900
Epoch 15/16
13/13 [==============================] - 46s 4s/step - loss: 0.0051
- accuracy: 1.0000 - val_loss: 0.2458 - val_accuracy: 0.9000
Epoch 16/16
13/13 [==============================] - 45s 4s/step - loss: 0.0048
- accuracy: 1.0000 - val_loss: 0.2430 - val_accuracy: 0.9000
```

In [23]:
```
# Plot accuracy and loss
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Testing Accuracy')
plt.title('Training and Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Testing Loss')
plt.title('Training and Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



In [36]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_train)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_train, axis=1)  # Convert one-hot encoded labels

# Print classification report
print(classification_report(y_true, y_pred))
```

```
13/13 [==============================] - 37s 3s/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        79
           1       1.00      0.97      0.99        78
           2       0.98      1.00      0.99        84
           3       1.00      1.00      1.00        85
           4       1.00      1.00      1.00        74

    accuracy                           0.99       400
   macro avg       1.00      0.99      1.00       400
weighted avg       1.00      0.99      0.99       400
```

In [37]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels t

# Print classification report
print(classification_report(y_true, y_pred))
```

```
4/4 [==============================] - 10s 2s/step
              precision    recall  f1-score   support

           0       0.76      0.76      0.76        21
           1       0.82      0.82      0.82        22
           2       1.00      0.88      0.93        16
           3       1.00      1.00      1.00        15
           4       0.89      0.96      0.93        26

    accuracy                           0.88       100
   macro avg       0.89      0.88      0.89       100
weighted avg       0.88      0.88      0.88       100
```

In [24]:
```
import os
import torch
import torchvision.models as models
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset, random_split
from torchvision.datasets import ImageFolder
import torch.optim as optim
import torch.nn as nn
from torch.optim.lr_scheduler import StepLR
from PIL import Image

# Custom dataset class
class CustomImageFolder(Dataset):
    def __init__(self, root_dir, transform):
        self.dataset = ImageFolder(root_dir, transform=transform)
        self.corrupted_idx = []

    def __getitem__(self, index):
        try:
            return self.dataset[index]
        except OSError:
            self.corrupted_idx.append(index)
            return None  # Return None for corrupted file

    def __len__(self):
        return len(self.dataset)

    def get_corrupted_files(self):
        return [self.dataset.imgs[i] for i in self.corrupted_idx]

# Custom collate function to filter out None values
def custom_collate_fn(batch):
    batch = list(filter(lambda x: x is not None, batch))
    return torch.utils.data.dataloader.default_collate(batch)

# Data preprocessing
transform = transforms.Compose([
    transforms.Resize((227, 227)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224
])

# Custom dataset
custom_dataset = CustomImageFolder('Mod_Plant', transform=transform)
train_size = int(0.8 * len(custom_dataset))
test_size = len(custom_dataset) - train_size
train_dataset, test_dataset = random_split(custom_dataset, [train_size

# DataLoaders with custom collate function
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True,
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False, c

# Model setup
alexnet = models.alexnet(pretrained=True)
num_classes = len(custom_dataset.dataset.classes)
alexnet.classifier[6] = nn.Linear(alexnet.classifier[6].in_features, n

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(alexnet.parameters(), lr=0.001, momentum=0.9)
scheduler = StepLR(optimizer, step_size=7, gamma=0.1)
```

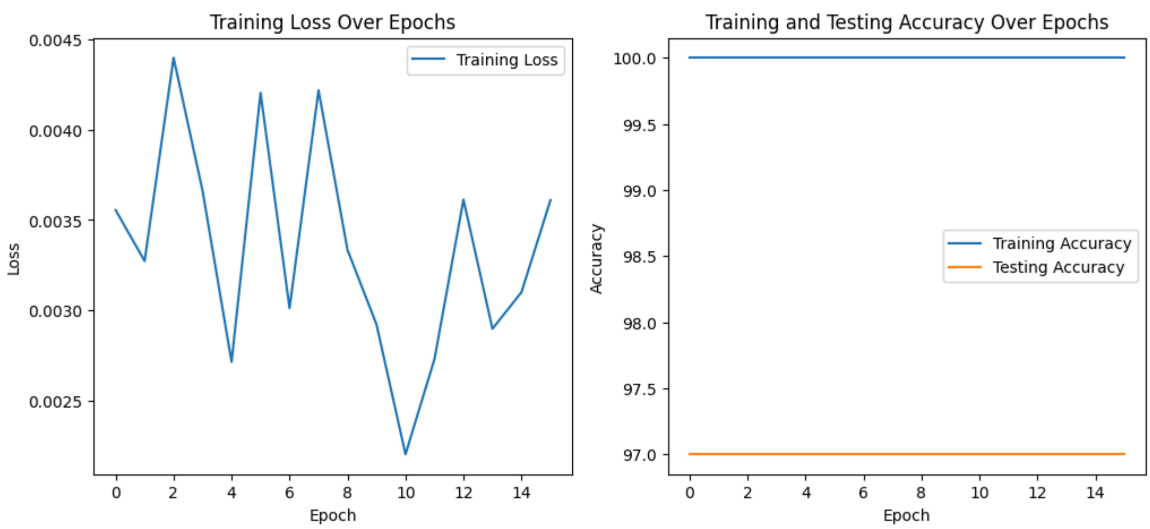
```
# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
alexnet.to(device)

# Training the model
num_epochs = 16
for epoch in range(num_epochs):
    alexnet.train()
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()

        outputs = alexnet(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    scheduler.step()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(train_loader)}")

    # Testing the model
    alexnet.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = alexnet(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    print(f'Accuracy of the network on the test images: {100 * correct

print('Finished Training')

# Print corrupted files
corrupted_files = custom_dataset.get_corrupted_files()
print("Corrupted files:", corrupted_files)
```

```
Epoch 1, Loss: 0.8613604914683562
Accuracy of the network on the test images: 90.0 %
Epoch 2, Loss: 0.22625552404385346
Accuracy of the network on the test images: 96.0 %
Epoch 3, Loss: 0.08228465828758019
Accuracy of the network on the test images: 95.0 %
Epoch 4, Loss: 0.044108745033064715
Accuracy of the network on the test images: 96.0 %
Epoch 5, Loss: 0.012466011121152686
Accuracy of the network on the test images: 99.0 %
Epoch 6, Loss: 0.011207065348011943
Accuracy of the network on the test images: 97.0 %
Epoch 7, Loss: 0.012707366997626824
Accuracy of the network on the test images: 99.0 %
Epoch 8, Loss: 0.005197566743635644
Accuracy of the network on the test images: 98.0 %
Epoch 9, Loss: 0.004622291688484928
Accuracy of the network on the test images: 99.0 %
Epoch 10, Loss: 0.004563153779599816
Accuracy of the network on the test images: 99.0 %
Epoch 11, Loss: 0.004616728939044361
Accuracy of the network on the test images: 99.0 %
Epoch 12, Loss: 0.005832659317932736
Accuracy of the network on the test images: 99.0 %
Epoch 13, Loss: 0.005168968654918269
Accuracy of the network on the test images: 99.0 %
Epoch 14, Loss: 0.004566358384461357
Accuracy of the network on the test images: 99.0 %
Epoch 15, Loss: 0.003990187251474708
Accuracy of the network on the test images: 99.0 %
Epoch 16, Loss: 0.00453592102544812
Accuracy of the network on the test images: 99.0 %
Finished Training
Corrupted files: [('Mod_Plant\\Maize_leaf blight\\leaf blight58_.jp
g', 3), ('Mod_Plant\\Maize_leaf blight\\leaf blight58_.jpg', 3), ('M
od_Plant\\Maize_leaf blight\\leaf blight58_.jpg', 3), ('Mod_Plant\\M
aize_leaf blight\\leaf blight58_.jpg', 3), ('Mod_Plant\\Maize_leaf b
light\\leaf blight58_.jpg', 3), ('Mod_Plant\\Maize_leaf blight\\leaf
blight58_.jpg', 3), ('Mod_Plant\\Maize_leaf blight\\leaf blight58_.j
pg', 3), ('Mod_Plant\\Maize_leaf blight\\leaf blight58_.jpg', 3),
('Mod_Plant\\Maize_leaf blight\\leaf blight58_.jpg', 3), ('Mod_Plant
\\Maize_leaf blight\\leaf blight58_.jpg', 3), ('Mod_Plant\\Maize_lea
f blight\\leaf blight58_.jpg', 3), ('Mod_Plant\\Maize_leaf blight\\l
eaf blight58_.jpg', 3), ('Mod_Plant\\Maize_leaf blight\\leaf blight5
8_.jpg', 3), ('Mod_Plant\\Maize_leaf blight\\leaf blight58_.jpg',
3), ('Mod_Plant\\Maize_leaf blight\\leaf blight58_.jpg', 3), ('Mod_P
lant\\Maize_leaf blight\\leaf blight58_.jpg', 3)]
```

In [40]:
```
import matplotlib.pyplot as plt

train_losses = []
train_accuracies = []
test_accuracies = []

# Train the model
for epoch in range(num_epochs):
    alexnet.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for i, data in enumerate(train_loader, 0):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = alexnet(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()

    train_loss = running_loss / len(train_loader)
    train_accuracy = 100 * correct_train / total_train
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    # Test the model
    alexnet.eval()
    correct_test = 0
    total_test = 0

    with torch.no_grad():
        for data in test_loader:
            images, labels = data
            images, labels = images.to(device), labels.to(device)
            outputs = alexnet(images)
            _, predicted = torch.max(outputs.data, 1)
            total_test += labels.size(0)
            correct_test += (predicted == labels).sum().item()

    test_accuracy = 100 * correct_test / total_test
    test_accuracies.append(test_accuracy)

    print(f"Epoch {epoch+1}, Loss: {train_loss}, Train Accuracy: {trai

# Plotting Loss and Accuracy
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss Over Epochs')
plt.legend()
```

```
plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Training Accuracy')
plt.plot(test_accuracies, label='Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy Over Epochs')
plt.legend()

plt.show()
```

```
Epoch 1, Loss: 0.0035552675208936515, Train Accuracy: 100.0%, Test A
ccuracy: 97.0%
Epoch 2, Loss: 0.003272762293748271, Train Accuracy: 100.0%, Test Ac
curacy: 97.0%
Epoch 3, Loss: 0.004398179876331527, Train Accuracy: 100.0%, Test Ac
curacy: 97.0%
Epoch 4, Loss: 0.0036613778569377386, Train Accuracy: 100.0%, Test A
ccuracy: 97.0%
Epoch 5, Loss: 0.0027148133063187394, Train Accuracy: 100.0%, Test A
ccuracy: 97.0%
Epoch 6, Loss: 0.004203962274074841, Train Accuracy: 100.0%, Test Ac
curacy: 97.0%
Epoch 7, Loss: 0.003012547708259752, Train Accuracy: 100.0%, Test Ac
curacy: 97.0%
Epoch 8, Loss: 0.004218278349771236, Train Accuracy: 100.0%, Test Ac
curacy: 97.0%
Epoch 9, Loss: 0.0033346827974757897, Train Accuracy: 100.0%, Test A
ccuracy: 97.0%
Epoch 10, Loss: 0.0029219738924159454, Train Accuracy: 100.0%, Test
Accuracy: 97.0%
Epoch 11, Loss: 0.00220318276506777, Train Accuracy: 100.0%, Test Ac
curacy: 97.0%
Epoch 12, Loss: 0.0027304209562806557, Train Accuracy: 100.0%, Test
Accuracy: 97.0%
Epoch 13, Loss: 0.0036129730601365175, Train Accuracy: 100.0%, Test
Accuracy: 97.0%
Epoch 14, Loss: 0.002898138885099727, Train Accuracy: 100.0%, Test A
ccuracy: 97.0%
Epoch 15, Loss: 0.003100269729307351, Train Accuracy: 100.0%, Test A
ccuracy: 97.0%
Epoch 16, Loss: 0.0036096685977939228, Train Accuracy: 100.0%, Test
Accuracy: 97.0%
```



In [41]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_train)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_train, axis=1)  # Convert one-hot encoded labels

# Print classification report
print(classification_report(y_true, y_pred))
```

```
13/13 [==============================] - 35s 3s/step
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        79
           1       1.00      0.97      0.99        78
           2       0.98      1.00      0.99        84
           3       1.00      1.00      1.00        85
           4       1.00      1.00      1.00        74

    accuracy                           0.99       400
   macro avg       1.00      0.99      1.00       400
weighted avg       1.00      0.99      0.99       400
```

In [42]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels t

# Print classification report
print(classification_report(y_true, y_pred))
```

```
4/4 [==============================] - 9s 2s/step
              precision    recall  f1-score   support

           0       0.76      0.76      0.76        21
           1       0.82      0.82      0.82        22
           2       1.00      0.88      0.93        16
           3       1.00      1.00      1.00        15
           4       0.89      0.96      0.93        26

    accuracy                           0.88       100
   macro avg       0.89      0.88      0.89       100
weighted avg       0.88      0.88      0.88       100
```

In [43]:
```
torch.save(alexnet.state_dict(), '/content/drive/MyDrive/Plant_alexnet

torch.save(alexnet, 'Plant_alexnet_model.pth')
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent cal
l last)
Cell In[43], line 1
----> 1 torch.save(alexnet.state_dict(), '/content/drive/MyDrive/Pla
nt_alexnet_model.pth')
      3 torch.save(alexnet, 'Plant_alexnet_model.pth')

File c:\Users\mgssr\AppData\Local\Programs\Python\Python311\Lib\site
-packages\torch\serialization.py:440, in save(obj, f, pickle_module,
pickle_protocol, _use_new_zipfile_serialization)
    437 _check_save_filelike(f)
    439 if _use_new_zipfile_serialization:
--> 440     with _open_zipfile_writer(f) as opened_zipfile:
    441         _save(obj, opened_zipfile, pickle_module, pickle_pro
tocol)
    442         return

File c:\Users\mgssr\AppData\Local\Programs\Python\Python311\Lib\site
-packages\torch\serialization.py:315, in _open_zipfile_writer(name_o
r_buffer)
    313 else:
    314     container = _open_zipfile_writer_buffer
--> 315 return container(name_or_buffer)

File c:\Users\mgssr\AppData\Local\Programs\Python\Python311\Lib\site
-packages\torch\serialization.py:288, in _open_zipfile_writer_file._
_init__(self, name)
    287 def __init__(self, name) -> None:
--> 288     super().__init__(torch._C.PyTorchFileWriter(str(name)))

RuntimeError: Parent directory /content/drive/MyDrive does not exis
t.
```

In [ ]:
```
alexnet = models.alexnet()
alexnet.classifier[6] = nn.Linear(alexnet.classifier[6].in_features, 5
alexnet.load_state_dict(torch.load('/content/drive/MyDrive/Plant_alexn
alexnet.to(device)  # Move the model to the device
```

Out[60]:
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=
(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, c
eil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=
(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, c
eil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding
=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), paddin
g=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=5, bias=True)
  )
)
```

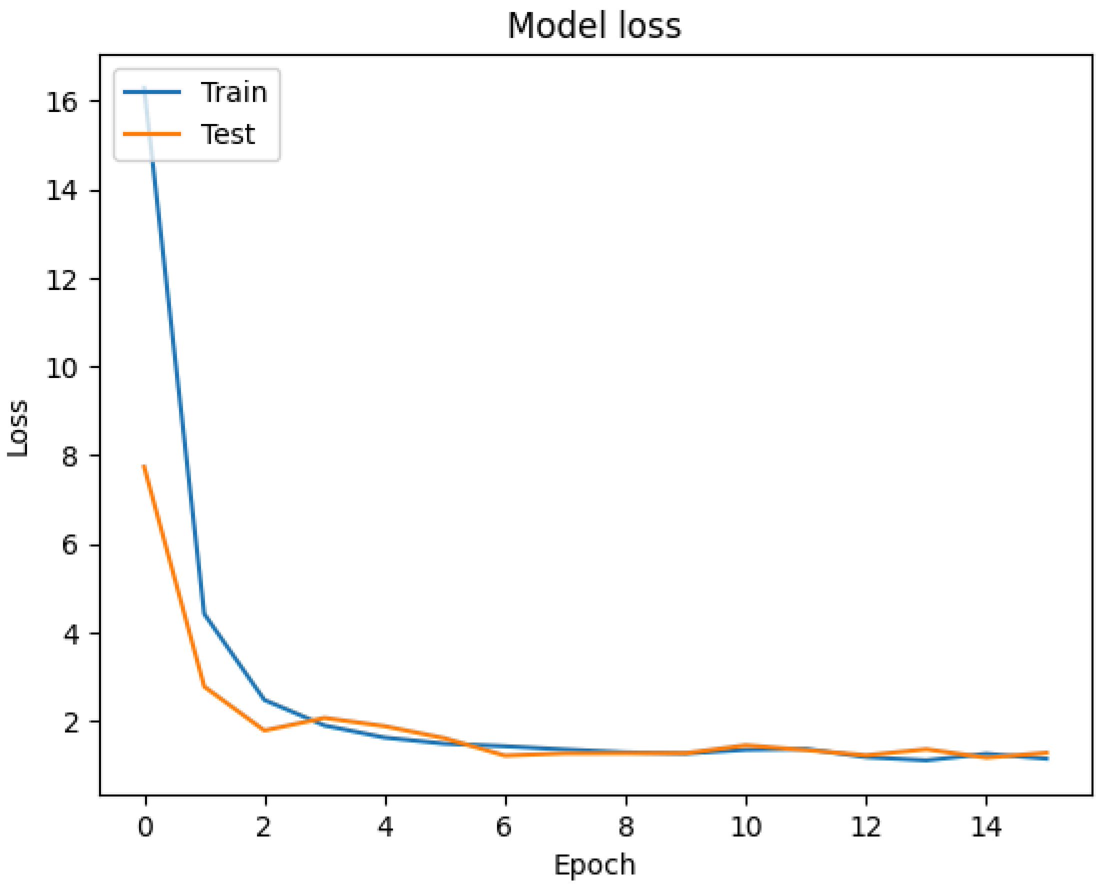
In [ ]:
```
from PIL import Image
import torch
import torchvision.transforms as transforms

# Define your transformation pipeline
transform = transforms.Compose([
    transforms.Resize((227, 227)),  # AlexNet uses 227x227 input s
    transforms.ToTensor(),
    # Normalization values for pretrained models are usually the m
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.2
])

def predict_single_image(image_path, model, transform, device):
    # Open the image file
    image = Image.open(image_path)

    # Convert to RGB if not already (assumes that the input image
    if image.mode != 'RGB':
        image = image.convert('RGB')

    # Apply the transformations to the image
    image = transform(image)

    # Add a batch dimension since pytorch expects a batch, not a s
    image = image.unsqueeze(0).to(device)

    # Set the model to evaluation mode
    model.eval()

    # No need to track gradients for validation, hence wrap in tor
    with torch.no_grad():
        outputs = model(image)
        _, predicted = torch.max(outputs, 1)

    # Get the index of the predicted class
    return predicted.item()

# Make sure to define `alexnet`, `device`, and `dataset.classes` a
# Example usage:
image_path = '/content/drive/MyDrive/Mod_Plant/Cassava_brown spot/
predicted_class_index = predict_single_image(image_path, alexnet,
predicted_class = class_name[predicted_class_index]
print(f'Predicted Class: {predicted_class}')
```

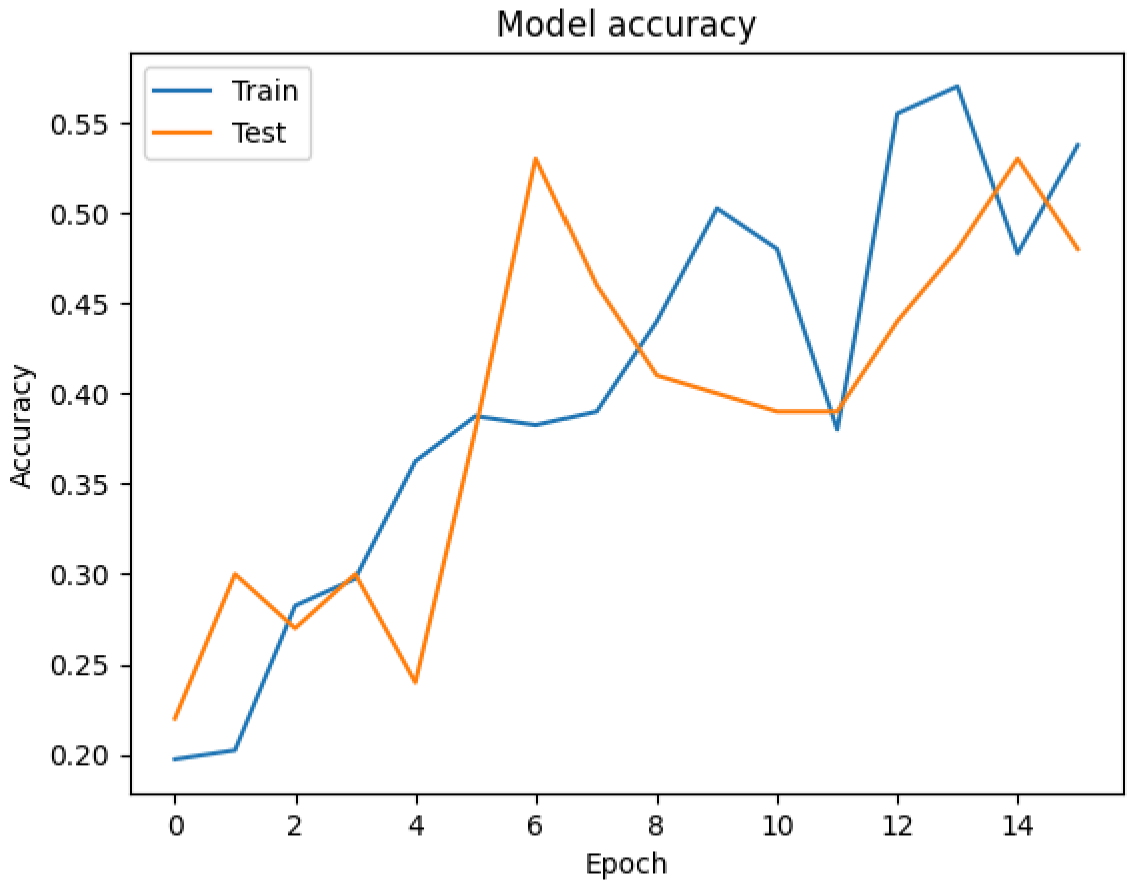
```
Predicted Class: Cassava_brown spot
```

In [44]:
```
##RESNET
import numpy as np
from keras.utils import to_categorical
from keras.applications.resnet50 import ResNet50
from keras.layers import Dense, Flatten
from keras.models import Model
import cv2
import os
import random

# Load pre-trained ResNet50 model
resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(

# Freeze the layers
for layer in resnet.layers:
    layer.trainable = False

# Add custom layers for classification
x = Flatten()(resnet.output)
x = Dense(512, activation='relu')(x)
predictions = Dense(len(class_name), activation='softmax')(x)  # Ensur

# Create model
model = Model(inputs=resnet.input, outputs=predictions)
model.compile(optimizer='adam', loss='categorical_crossentropy', metri

# Assuming x_train_augmented, y_train_augmented, x_test, y_test are de

history = model.fit(x_train_augmented, y_train_augmented, validation_d
```

```
Epoch 1/16
13/13 [==============================] - 25s 2s/step - loss: 16.2886
- accuracy: 0.1975 - val_loss: 7.7465 - val_accuracy: 0.2200
Epoch 2/16
13/13 [==============================] - 21s 2s/step - loss: 4.4219
- accuracy: 0.2025 - val_loss: 2.7822 - val_accuracy: 0.3000
Epoch 3/16
13/13 [==============================] - 21s 2s/step - loss: 2.4796
- accuracy: 0.2825 - val_loss: 1.7917 - val_accuracy: 0.2700
Epoch 4/16
13/13 [==============================] - 21s 2s/step - loss: 1.9031
- accuracy: 0.2975 - val_loss: 2.0761 - val_accuracy: 0.3000
Epoch 5/16
13/13 [==============================] - 21s 2s/step - loss: 1.6303
- accuracy: 0.3625 - val_loss: 1.8894 - val_accuracy: 0.2400
Epoch 6/16
13/13 [==============================] - 21s 2s/step - loss: 1.4923
- accuracy: 0.3875 - val_loss: 1.6170 - val_accuracy: 0.3800
Epoch 7/16
13/13 [==============================] - 21s 2s/step - loss: 1.4348
- accuracy: 0.3825 - val_loss: 1.2250 - val_accuracy: 0.5300
Epoch 8/16
13/13 [==============================] - 21s 2s/step - loss: 1.3610
- accuracy: 0.3900 - val_loss: 1.2760 - val_accuracy: 0.4600
Epoch 9/16
13/13 [==============================] - 21s 2s/step - loss: 1.2961
- accuracy: 0.4400 - val_loss: 1.2776 - val_accuracy: 0.4100
Epoch 10/16
13/13 [==============================] - 21s 2s/step - loss: 1.2688
- accuracy: 0.5025 - val_loss: 1.2813 - val_accuracy: 0.4000
Epoch 11/16
13/13 [==============================] - 22s 2s/step - loss: 1.3552
- accuracy: 0.4800 - val_loss: 1.4587 - val_accuracy: 0.3900
Epoch 12/16
13/13 [==============================] - 22s 2s/step - loss: 1.3734
- accuracy: 0.3800 - val_loss: 1.3507 - val_accuracy: 0.3900
Epoch 13/16
13/13 [==============================] - 23s 2s/step - loss: 1.1883
- accuracy: 0.5550 - val_loss: 1.2332 - val_accuracy: 0.4400
Epoch 14/16
13/13 [==============================] - 22s 2s/step - loss: 1.1155
- accuracy: 0.5700 - val_loss: 1.3678 - val_accuracy: 0.4800
Epoch 15/16
13/13 [==============================] - 21s 2s/step - loss: 1.2632
- accuracy: 0.4775 - val_loss: 1.1795 - val_accuracy: 0.5300
Epoch 16/16
13/13 [==============================] - 22s 2s/step - loss: 1.1538
- accuracy: 0.5375 - val_loss: 1.2890 - val_accuracy: 0.4800
```

In [45]:
```
import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Plot training & validation accuracy values
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```





In [46]:

```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_train)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_train, axis=1)  # Convert one-hot encoded labels

# Print classification report
print(classification_report(y_true, y_pred))
```

```
13/13 [==============================] - 16s 1s/step
              precision    recall  f1-score   support

           0       1.00      0.14      0.24        79
           1       0.35      0.83      0.49        78
           2       0.67      0.57      0.62        84
           3       0.70      0.82      0.76        85
           4       0.97      0.42      0.58        74

    accuracy                           0.56       400
   macro avg       0.74      0.56      0.54       400
weighted avg       0.73      0.56      0.54       400
```

In [47]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels t

# Print classification report
print(classification_report(y_true, y_pred))
```

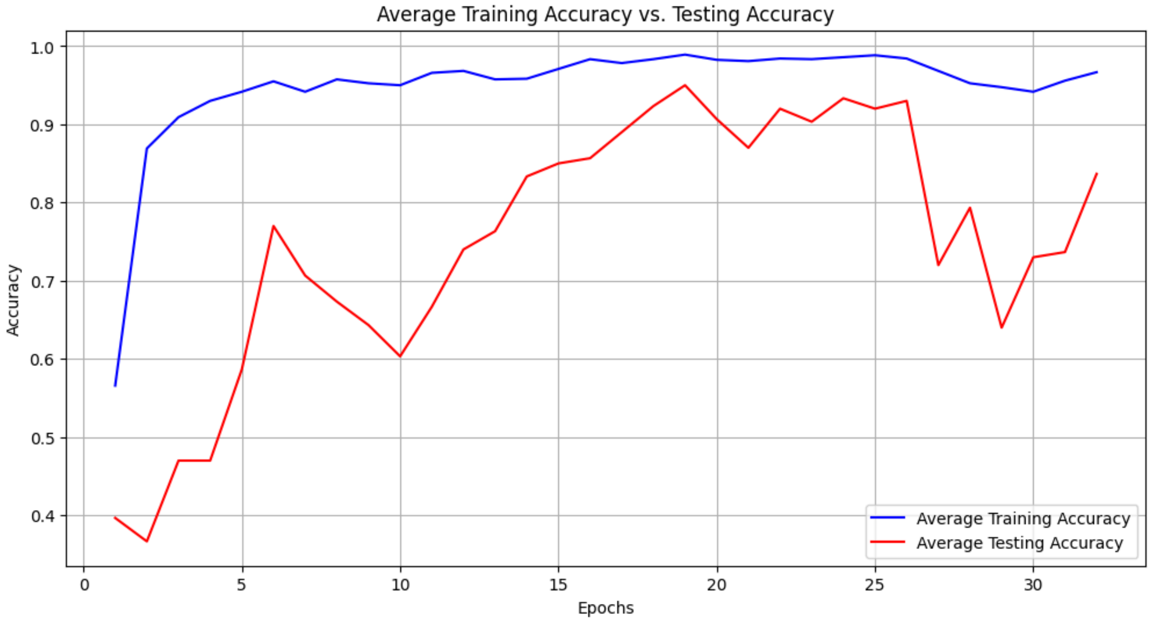
```
4/4 [==============================] - 4s 840ms/step
              precision    recall  f1-score   support

           0       1.00      0.19      0.32        21
           1       0.35      0.73      0.47        22
           2       0.50      0.50      0.50        16
           3       0.48      0.73      0.58        15
           4       0.82      0.35      0.49        26

    accuracy                           0.48       100
   macro avg       0.63      0.50      0.47       100
weighted avg       0.65      0.48      0.46       100
```

In [25]:
```
import numpy as np
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split

# Define the number of random states and epochs
num_random_states = 3
num_epochs = 32

# Define lists to store accuracies and losses for each random state
train_losses_per_epoch = [[] for _ in range(num_epochs)]
val_losses_per_epoch = [[] for _ in range(num_epochs)]
train_accuracies_per_epoch = [[] for _ in range(num_epochs)]
val_accuracies_per_epoch = [[] for _ in range(num_epochs)]

# Assuming x_data and y_data are your data
for random_state in range(num_random_states):
    # Set random seed for reproducibility
    np.random.seed(random_state)



    # Load the MobileNet model without the top layer and use pre-trair
    base_model = MobileNet(weights='imagenet', include_top=False, inpu

    # Add custom top layers for classification
    x = Flatten()(base_model.output)
    x = Dense(256, activation='relu')(x)
    x = Dropout(0.5)(x)
    x = Dense(5, activation='softmax')(x)

    # Create the model
    model = Model(inputs=base_model.input, outputs=x)

    # Compile the model
    model.compile(optimizer=Adam(learning_rate=0.001), loss='categoric

    # Train the model with validation data
    history = model.fit(x_train_augmented, y_train_augmented, epochs=r

    # Store loss and accuracy for each epoch
    for epoch in range(num_epochs):
        train_losses_per_epoch[epoch].append(history.history['loss'][e
        val_losses_per_epoch[epoch].append(history.history['val_loss']
        train_accuracies_per_epoch[epoch].append(history.history['accu
        val_accuracies_per_epoch[epoch].append(history.history['val_ac

# Calculate average accuracies and losses for each epoch
avg_train_losses = [np.mean(losses) for losses in train_losses_per_epo
avg_val_losses = [np.mean(losses) for losses in val_losses_per_epoch]
avg_train_accuracies = [np.mean(accuracies) for accuracies in train_ac
avg_val_accuracies = [np.mean(accuracies) for accuracies in val_accura

# Print the loss for both training and validation along with average a
for epoch, (avg_train_loss, avg_val_loss, avg_train_acc, avg_val_acc)
    print(f"Epoch {epoch + 1}: Average Training Loss = {avg_train_loss
```

```
Epoch 1: Average Training Loss = 11.156750361124674, Average Validat
ion Loss = 23.264310836791992, Average Training Accuracy = 0.5658333
202203115, Average Validation Accuracy = 0.39666666587193805
Epoch 2: Average Training Loss = 1.9339452187220256, Average Validat
ion Loss = 43.558082580566406, Average Training Accuracy = 0.8691666
523615519, Average Validation Accuracy = 0.36666666467984516
Epoch 3: Average Training Loss = 1.4607891043027241, Average Validat
ion Loss = 39.044787089029946, Average Training Accuracy = 0.9091666
73819224, Average Validation Accuracy = 0.4700000087420146
Epoch 4: Average Training Loss = 1.134154697259267, Average Validati
on Loss = 34.79811668395996, Average Training Accuracy = 0.930000007
1525574, Average Validation Accuracy = 0.4699999988079071
Epoch 5: Average Training Loss = 0.6583217680454254, Average Validat
ion Loss = 14.511480331420898, Average Training Accuracy = 0.9416666
626930237, Average Validation Accuracy = 0.5866666833559672
Epoch 6: Average Training Loss = 0.4770967165629069, Average Validat
ion Loss = 5.454058885574341, Average Training Accuracy = 0.95499998
33106995, Average Validation Accuracy = 0.7700000007947286
Epoch 7: Average Training Loss = 0.6462103724479675, Average Validat
ion Loss = 8.502470016479492, Average Training Accuracy = 0.94166664
28248087, Average Validation Accuracy = 0.7066666682561239
Epoch 8: Average Training Loss = 0.5388818581899008, Average Validat
ion Loss = 9.433899720509848, Average Training Accuracy = 0.95750000
07947286, Average Validation Accuracy = 0.6733333269755045
Epoch 9: Average Training Loss = 0.2960502952337265, Average Validat
ion Loss = 14.187627951304117, Average Training Accuracy = 0.9525000
055631002, Average Validation Accuracy = 0.6433333357175192
Epoch 10: Average Training Loss = 0.46728219588597614, Average Valid
ation Loss = 20.140477180480957, Average Training Accuracy = 0.95000
0007947286, Average Validation Accuracy = 0.6033333241939545
Epoch 11: Average Training Loss = 0.40505970517794293, Average Valid
ation Loss = 19.608489453792572, Average Training Accuracy = 0.96583
3306312561, Average Validation Accuracy = 0.6666666666666666
Epoch 12: Average Training Loss = 0.27389497061570484, Average Valid
ation Loss = 14.911057035128275, Average Training Accuracy = 0.96833
33436648051, Average Validation Accuracy = 0.7400000095367432
Epoch 13: Average Training Loss = 0.572069875895977, Average Validat
ion Loss = 12.549346605936686, Average Training Accuracy = 0.9575000
007947286, Average Validation Accuracy = 0.7633333404858907
Epoch 14: Average Training Loss = 0.3886292800307274, Average Valida
tion Loss = 3.0491596162319183, Average Training Accuracy = 0.958333
3333333334, Average Validation Accuracy = 0.8333333333333334
Epoch 15: Average Training Loss = 0.1296361784140269, Average Valida
tion Loss = 3.0087282061576843, Average Training Accuracy = 0.970833
3412806193, Average Validation Accuracy = 0.850000003973643
Epoch 16: Average Training Loss = 0.10171402245759964, Average Valid
ation Loss = 2.9340948363145194, Average Training Accuracy = 0.98333
33492279053, Average Validation Accuracy = 0.8566666642824808
Epoch 17: Average Training Loss = 0.09171188126007716, Average Valid
ation Loss = 1.513121058543523, Average Training Accuracy = 0.978333
3341280619, Average Validation Accuracy = 0.8900000055631002
Epoch 18: Average Training Loss = 0.10449689999222755, Average Valid
ation Loss = 1.0640411873658497, Average Training Accuracy = 0.98333
33492279053, Average Validation Accuracy = 0.9233333269755045
Epoch 19: Average Training Loss = 0.1522105485200882, Average Valida
tion Loss = 0.8203253448009491, Average Training Accuracy = 0.989166
6769981384, Average Validation Accuracy = 0.950000007947286
Epoch 20: Average Training Loss = 0.10259490708510081, Average Valid
ation Loss = 0.8840660750865936, Average Training Accuracy = 0.98249
99968210856, Average Validation Accuracy = 0.9066666762034098
Epoch 21: Average Training Loss = 0.05948632831374804, Average Valid
```

```
ation Loss = 0.8684227814277014, Average Training Accuracy = 0.98083
33317438761, Average Validation Accuracy = 0.8700000047683716
Epoch 22: Average Training Loss = 0.07958683868249257, Average Valid
ation Loss = 0.711875299612681, Average Training Accuracy = 0.984166
68176651, Average Validation Accuracy = 0.9199999968210856
Epoch 23: Average Training Loss = 0.06336737424135208, Average Valid
ation Loss = 1.5645332584778469, Average Training Accuracy = 0.98333
33492279053, Average Validation Accuracy = 0.903333326180776
Epoch 24: Average Training Loss = 0.05586510089536508, Average Valid
ation Loss = 1.6168708267311256, Average Training Accuracy = 0.98583
33468437195, Average Validation Accuracy = 0.9333333373069763
Epoch 25: Average Training Loss = 0.09333205098907153, Average Valid
ation Loss = 0.8381495773792267, Average Training Accuracy = 0.98833
33444595337, Average Validation Accuracy = 0.9199999968210856
Epoch 26: Average Training Loss = 0.19384370744228363, Average Valid
ation Loss = 1.4908939053614934, Average Training Accuracy = 0.98416
668176651, Average Validation Accuracy = 0.9300000071525574
Epoch 27: Average Training Loss = 0.38337049384911853, Average Valid
ation Loss = 10.502147694428762, Average Training Accuracy = 0.96833
33436648051, Average Validation Accuracy = 0.7200000087420145
Epoch 28: Average Training Loss = 0.38212838520606357, Average Valid
ation Loss = 8.118413408597311, Average Training Accuracy = 0.952500
0055631002, Average Validation Accuracy = 0.7933333317438761
Epoch 29: Average Training Loss = 0.5440186771253744, Average Valida
tion Loss = 35.25304331382116, Average Training Accuracy = 0.9475000
103314718, Average Validation Accuracy = 0.6399999856948853
Epoch 30: Average Training Loss = 0.4842752665281296, Average Valida
tion Loss = 8.582316716512045, Average Training Accuracy = 0.9416666
825612386, Average Validation Accuracy = 0.7299999992052714
Epoch 31: Average Training Loss = 0.27624693512916565, Average Valid
ation Loss = 5.083118120829265, Average Training Accuracy = 0.955833
3357175192, Average Validation Accuracy = 0.7366666793823242
Epoch 32: Average Training Loss = 0.18429172659913698, Average Valid
ation Loss = 1.729980707168579, Average Training Accuracy = 0.966666
6587193807, Average Validation Accuracy = 0.8366666634877523
```

In [26]:
```
import numpy as np
import matplotlib.pyplot as plt

# Plot average training vs. average Testing loss
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), avg_train_losses, label='Average Tr
plt.plot(range(1, num_epochs + 1), avg_val_losses, label='Average Test
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Average Training vs. Testing Loss')
plt.legend()
plt.grid(True)
plt.show()

# Plot average training accuracy vs. average Testing accuracy
plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), avg_train_accuracies, label='Averag
plt.plot(range(1, num_epochs + 1), avg_val_accuracies, label='Average
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Average Training Accuracy vs. Testing Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



In [13]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels t

# Print classification report
print(classification_report(y_true, y_pred))
```

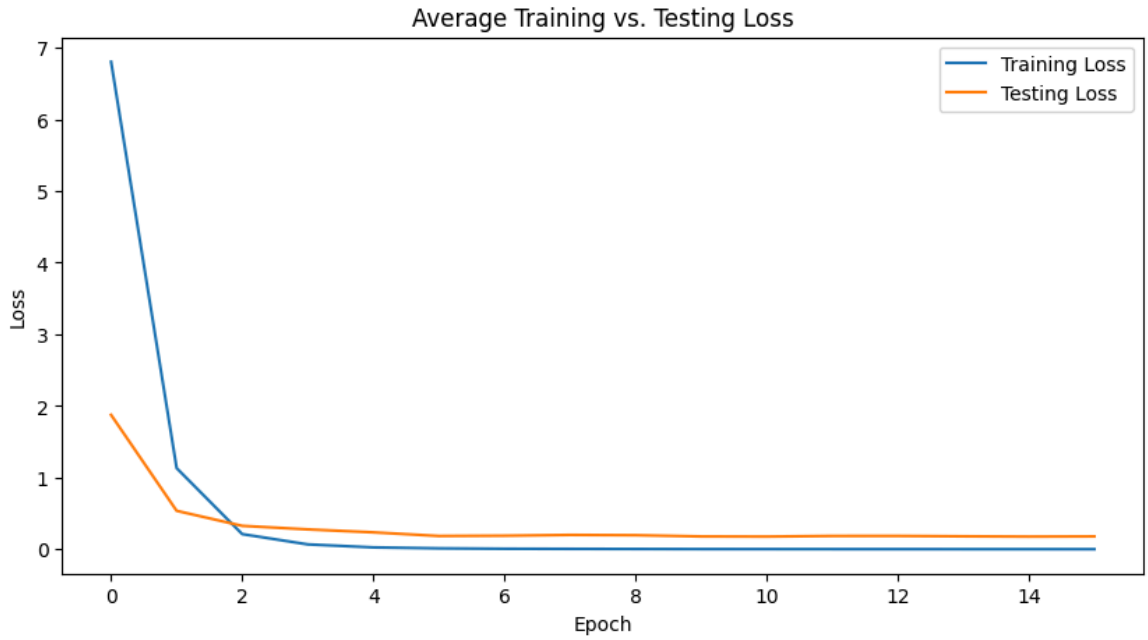
```
4/4 [==============================] - 2s 207ms/step
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        24
           1       1.00      1.00      1.00        16
           2       1.00      0.92      0.96        26
           3       0.92      1.00      0.96        12
           4       0.95      0.91      0.93        22

    accuracy                           0.96       100
   macro avg       0.96      0.97      0.96       100
weighted avg       0.96      0.96      0.96       100
```

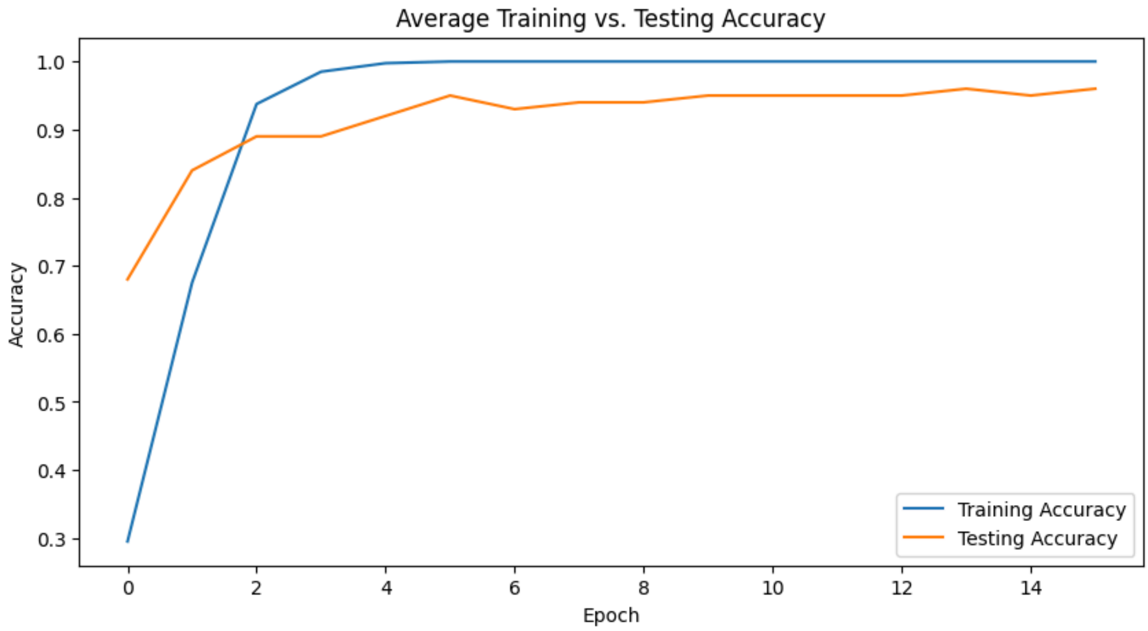
In [14]:
```
import numpy as np
from keras.utils import to_categorical
from keras.applications.vgg16 import VGG16  # Import VGG16 instead of
from keras.layers import Dense, Flatten
from keras.models import Model
import cv2
import os
import random

# Load pre-trained VGG16 model
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224,

# Freeze the layers
for layer in vgg16.layers:
    layer.trainable = False

# Add custom layers for classification
x = Flatten()(vgg16.output)
x = Dense(512, activation='relu')(x)
predictions = Dense(len(class_name), activation='softmax')(x)  # Ensur

# Create model
model = Model(inputs=vgg16.input, outputs=predictions)  # Change input
model.compile(optimizer='adam', loss='categorical_crossentropy', metri

# Assuming x_train_augmented, y_train_augmented, x_test, y_test are de

history = model.fit(x_train_augmented, y_train_augmented, validation_d
```

```
WARNING:tensorflow:From c:\Users\mgssr\AppData\Local\Programs\Python
\Python311\Lib\site-packages\keras\src\optimizers\__init__.py:309: T
he name tf.train.Optimizer is deprecated. Please use tf.compat.v1.tr
ain.Optimizer instead.

Epoch 1/16
13/13 [==============================] - 38s 3s/step - loss: 6.8031
- accuracy: 0.2950 - val_loss: 1.8758 - val_accuracy: 0.6800
Epoch 2/16
13/13 [==============================] - 36s 3s/step - loss: 1.1333
- accuracy: 0.6750 - val_loss: 0.5377 - val_accuracy: 0.8400
Epoch 3/16
13/13 [==============================] - 37s 3s/step - loss: 0.2128
- accuracy: 0.9375 - val_loss: 0.3280 - val_accuracy: 0.8900
Epoch 4/16
13/13 [==============================] - 37s 3s/step - loss: 0.0692
- accuracy: 0.9850 - val_loss: 0.2785 - val_accuracy: 0.8900
Epoch 5/16
13/13 [==============================] - 37s 3s/step - loss: 0.0273
- accuracy: 0.9975 - val_loss: 0.2371 - val_accuracy: 0.9200
Epoch 6/16
13/13 [==============================] - 37s 3s/step - loss: 0.0140
- accuracy: 1.0000 - val_loss: 0.1864 - val_accuracy: 0.9500
Epoch 7/16
13/13 [==============================] - 37s 3s/step - loss: 0.0082
- accuracy: 1.0000 - val_loss: 0.1904 - val_accuracy: 0.9300
Epoch 8/16
13/13 [==============================] - 37s 3s/step - loss: 0.0065
- accuracy: 1.0000 - val_loss: 0.2012 - val_accuracy: 0.9400
Epoch 9/16
13/13 [==============================] - 38s 3s/step - loss: 0.0051
- accuracy: 1.0000 - val_loss: 0.1974 - val_accuracy: 0.9400
Epoch 10/16
13/13 [==============================] - 37s 3s/step - loss: 0.0038
- accuracy: 1.0000 - val_loss: 0.1800 - val_accuracy: 0.9500
Epoch 11/16
13/13 [==============================] - 37s 3s/step - loss: 0.0035
- accuracy: 1.0000 - val_loss: 0.1780 - val_accuracy: 0.9500
Epoch 12/16
13/13 [==============================] - 37s 3s/step - loss: 0.0031
- accuracy: 1.0000 - val_loss: 0.1866 - val_accuracy: 0.9500
Epoch 13/16
13/13 [==============================] - 37s 3s/step - loss: 0.0028
- accuracy: 1.0000 - val_loss: 0.1869 - val_accuracy: 0.9500
Epoch 14/16
13/13 [==============================] - 37s 3s/step - loss: 0.0026
- accuracy: 1.0000 - val_loss: 0.1816 - val_accuracy: 0.9600
Epoch 15/16
13/13 [==============================] - 37s 3s/step - loss: 0.0024
- accuracy: 1.0000 - val_loss: 0.1777 - val_accuracy: 0.9500
Epoch 16/16
13/13 [==============================] - 37s 3s/step - loss: 0.0023
- accuracy: 1.0000 - val_loss: 0.1803 - val_accuracy: 0.9600
```

In [16]:

```
# Get training history
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Calculate average training and Testing loss
avg_train_loss = np.mean(train_loss)
avg_val_loss = np.mean(val_loss)

# Calculate average training and Testing accuracy
avg_train_acc = np.mean(train_acc)
avg_val_acc = np.mean(val_acc)

# Plot average training vs. average Testing loss
plt.figure(figsize=(10, 5))
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Testing Loss')
plt.title('Average Training vs. Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot average training vs. average Testing accuracy
plt.figure(figsize=(10, 5))
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Testing Accuracy')
plt.title('Average Training vs. Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Print average training and Testing loss and accuracy
print(f'Average Training Loss: {avg_train_loss}')
print(f'Average Testing Loss: {avg_val_loss}')
print(f'Average Training Accuracy: {avg_train_acc}')
print(f'Average Testing Accuracy: {avg_val_acc}')
```






```
Average Training Loss: 0.5187471565877786
Average Testing Loss: 0.3314676694571972
Average Training Accuracy: 0.9306250009685755
Average Testing Accuracy: 0.9156249910593033
```

In [56]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_train)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_train, axis=1)  # Convert one-hot encoded labels

# Print classification report
print(classification_report(y_true, y_pred))
```

```
13/13 [==============================] - 33s 3s/step
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        79
           1       1.00      0.97      0.99        78
           2       0.98      0.99      0.98        84
           3       1.00      1.00      1.00        85
           4       1.00      1.00      1.00        74

    accuracy                           0.99       400
   macro avg       0.99      0.99      0.99       400
weighted avg       0.99      0.99      0.99       400
```

In [15]:
```
from sklearn.metrics import classification_report
import numpy as np

# Assuming 'model' is your trained model
y_pred_probabilities = model.predict(x_test)
y_pred = np.argmax(y_pred_probabilities, axis=1)  # Convert probabilit

# Assuming y_test is one-hot encoded
y_true = np.argmax(y_test, axis=1)  # Convert one-hot encoded labels t

# Print classification report
print(classification_report(y_true, y_pred))
```

```
4/4 [==============================] - 8s 2s/step
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        24
           1       1.00      0.81      0.90        16
           2       0.93      1.00      0.96        26
           3       1.00      1.00      1.00        12
           4       0.95      0.95      0.95        22

    accuracy                           0.96       100
   macro avg       0.97      0.95      0.96       100
weighted avg       0.96      0.96      0.96       100
```