```
from PIL import Image
import os

def resize_images(input_dir, output_dir, new_size):
    # Ensure the output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Loop through each class folder in the input directory
    for class_folder in os.listdir(input_dir):
        if os.path.isdir(os.path.join(input_dir, class_folder)):
            class_input_dir = os.path.join(input_dir, class_folder)
            class_output_dir = os.path.join(output_dir, class_folder)
            os.makedirs(class_output_dir, exist_ok=True)

            # Loop through each image file in the class folder
            for filename in os.listdir(class_input_dir):
                if filename.endswith(".jpg") or filename.endswith(".png"):  # Add more formats if needed
                    # Open the image
                    image_path = os.path.join(class_input_dir, filename)
                    with Image.open(image_path) as img:
                        # Resize the image
                        resized_img = img.resize(new_size)

                        # Save the resized image
                        output_path = os.path.join(class_output_dir, filename)
                        resized_img.save(output_path)

    print("Image resizing complete.")

# Set the directory containing the images
input_dir = "/content/drive/MyDrive/DATASET"
output_dir = "/content/drive/MyDrive/DATA299"
new_size = (299, 299)  # Define the new size of the images

# Resize the images
resize_images(input_dir, output_dir, new_size)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!pip install split-folders
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
```

```
import splitfolders

# Split with a ratio.
# To only split into training and validation set, set a tuple to ratio, i.e, (.8, .2).
splitfolders.ratio("/content/drive/MyDrive/DATA299", output="/content/drive/MyDrive/DATA299SPLIT",
    seed=1337, ratio=(.7, .15, .15), group_prefix=None, move=False) # default values
```

```
Copying files: 1409 files [11:24,  2.06 files/s]
```

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
from itertools import cycle

# Define the dataset directories
dataset_dirs = ['/content/drive/MyDrive/DATA299SPLIT']
dataset_names = ['Image_Data']

# Define common hyperparameters
img_width, img_height = 299, 299  # InceptionV3 requires input shape of 299x299
```

```
input_shape = (img_width, img_height, 3)
batch_size = 32

# Data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=20,
    horizontal_flip=True,
    fill_mode='nearest')

# Data augmentation for validation and testing (only rescale)
val_test_datagen = ImageDataGenerator(rescale=1.0 / 255)

# Define a list of colors for plotting ROC curves
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green'])

# Loop over datasets
for dataset_dir, dataset_name in zip(dataset_dirs, dataset_names):
    print(f"Training and evaluating on {dataset_name}...")

    # Load and preprocess data for the current dataset
    train_data_dir ='/content/drive/MyDrive/DATA299SPLIT/train'
    val_data_dir = '/content/drive/MyDrive/DATA299SPLIT/val'
    test_data_dir = '/content/drive/MyDrive/DATA299SPLIT/test'

    train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')  # Set class_mode to 'binary'

    val_generator = val_test_datagen.flow_from_directory(
        val_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')  # Set class_mode to 'binary'

    test_generator = val_test_datagen.flow_from_directory(
        test_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')  # Set class_mode to 'binary'

    # Build and train the model
    base_model = InceptionV3(input_shape=input_shape, include_top=False, weights='imagenet')

    custom_model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')  # Change the number of output units to 1 for binary classification
    ])

    custom_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Change loss to 'binary_crossentropy'

    history = custom_model.fit(train_generator, epochs=100, validation_data=val_generator)
    validation_accuracy = history.history['val_accuracy'][-1]

    # Plot training and validation loss
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    # Plot training and validation accuracy
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()
```

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

# Evaluate the model on testing data
loss, testing_accuracy = custom_model.evaluate(test_generator)

# Make predictions using the model on the testing set
preds = custom_model.predict(test_generator)
class_preds = np.round(preds).flatten().astype(int)  # Round predictions for binary classification

# Print classification report
print("Classification Report:")
print(classification_report(test_generator.classes, class_preds))

# Compute the confusion matrix
cm = confusion_matrix(test_generator.classes, class_preds)
print("Confusion Matrix:")
print(cm)

# Calculate ROC curve and AUC for the binary class
fpr, tpr, _ = roc_curve(test_generator.classes, preds)
roc_auc = auc(fpr, tpr)
print("ROC and AUC for Binary Class:")
print(f"FPR: {fpr}")
print(f"TPR: {tpr}")
print(f"AUC: {roc_auc}")

# Plot ROC curve for the binary class
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'InceptionV3 (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve - InceptionV3 - {dataset_name}')
plt.legend(loc='lower right')
plt.show()
```

```
Training and evaluating on Image_Data...
Found 986 images belonging to 2 classes.
Found 211 images belonging to 2 classes.
Found 212 images belonging to 2 classes.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_
87910968/87910968 [==============================] - 3s 0us/step
Epoch 1/100
31/31 [==============================] - 384s 11s/step - loss: 0.1711 - accuracy: 0.9199 - val_loss: 36.9814 - val_accuracy: 0.4550
Epoch 2/100
31/31 [==============================] - 30s 972ms/step - loss: 0.0662 - accuracy: 0.9757 - val_loss: 31.6727 - val_accuracy: 0.4550
Epoch 3/100
31/31 [==============================] - 30s 966ms/step - loss: 0.0613 - accuracy: 0.9797 - val_loss: 82.4403 - val_accuracy: 0.4550
Epoch 4/100
31/31 [==============================] - 30s 964ms/step - loss: 0.0868 - accuracy: 0.9706 - val_loss: 2.0867 - val_accuracy: 0.8341
Epoch 5/100
31/31 [==============================] - 30s 976ms/step - loss: 0.0545 - accuracy: 0.9817 - val_loss: 6.0483 - val_accuracy: 0.7014
Epoch 6/100
31/31 [==============================] - 31s 968ms/step - loss: 0.0453 - accuracy: 0.9858 - val_loss: 0.1252 - val_accuracy: 0.9810
Epoch 7/100
31/31 [==============================] - 30s 960ms/step - loss: 0.0321 - accuracy: 0.9868 - val_loss: 3.5828 - val_accuracy: 0.5545
Epoch 8/100
31/31 [==============================] - 30s 958ms/step - loss: 0.0246 - accuracy: 0.9919 - val_loss: 18.3128 - val_accuracy: 0.4550
Epoch 9/100
31/31 [==============================] - 30s 966ms/step - loss: 0.0409 - accuracy: 0.9848 - val_loss: 0.0479 - val_accuracy: 0.9810
Epoch 10/100
31/31 [==============================] - 31s 999ms/step - loss: 0.0269 - accuracy: 0.9909 - val_loss: 7.9401 - val_accuracy: 0.5213
Epoch 11/100
31/31 [==============================] - 30s 961ms/step - loss: 0.0289 - accuracy: 0.9888 - val_loss: 1.3486 - val_accuracy: 0.9052
Epoch 12/100
31/31 [==============================] - 30s 961ms/step - loss: 0.0263 - accuracy: 0.9919 - val_loss: 1.2226 - val_accuracy: 0.6351
Epoch 13/100
31/31 [==============================] - 30s 959ms/step - loss: 0.0428 - accuracy: 0.9888 - val_loss: 6.3801 - val_accuracy: 0.4645
Epoch 14/100
31/31 [==============================] - 30s 959ms/step - loss: 0.0175 - accuracy: 0.9929 - val_loss: 0.9753 - val_accuracy: 0.8957
Epoch 15/100
31/31 [==============================] - 31s 997ms/step - loss: 0.0225 - accuracy: 0.9878 - val_loss: 0.3801 - val_accuracy: 0.8910
Epoch 16/100
31/31 [==============================] - 30s 958ms/step - loss: 0.0083 - accuracy: 0.9980 - val_loss: 2.7242 - val_accuracy: 0.7488
Epoch 17/100
31/31 [==============================] - 30s 974ms/step - loss: 0.0159 - accuracy: 0.9929 - val_loss: 0.7403 - val_accuracy: 0.9147
Epoch 18/100
31/31 [==============================] - 30s 966ms/step - loss: 0.0213 - accuracy: 0.9919 - val_loss: 4.6235 - val_accuracy: 0.6540
Epoch 19/100
31/31 [==============================] - 30s 965ms/step - loss: 0.1270 - accuracy: 0.9736 - val_loss: 12.3557 - val_accuracy: 0.4550
Epoch 20/100
31/31 [==============================] - 31s 967ms/step - loss: 0.0953 - accuracy: 0.9736 - val_loss: 208.3306 - val_accuracy: 0.4550
Epoch 21/100
31/31 [==============================] - 30s 958ms/step - loss: 0.0603 - accuracy: 0.9797 - val_loss: 25.5094 - val_accuracy: 0.4550
Epoch 22/100
31/31 [==============================] - 30s 955ms/step - loss: 0.0366 - accuracy: 0.9909 - val_loss: 19.0090 - val_accuracy: 0.4550
Epoch 23/100
31/31 [==============================] - 30s 961ms/step - loss: 0.0156 - accuracy: 0.9929 - val_loss: 10.4897 - val_accuracy: 0.4550
Epoch 24/100
31/31 [==============================] - 30s 965ms/step - loss: 0.0252 - accuracy: 0.9899 - val_loss: 0.3215 - val_accuracy: 0.9621
Epoch 25/100
31/31 [==============================] - 30s 954ms/step - loss: 0.0113 - accuracy: 0.9970 - val_loss: 0.0226 - val_accuracy: 0.9905
Epoch 26/100
```

```
from PIL import Image
import os

def resize_images(input_dir, output_dir, new_size):
    # Ensure the output directory exists
    os.makedirs(output_dir, exist_ok=True)

    # Loop through each class folder in the input directory
    for class_folder in os.listdir(input_dir):
        if os.path.isdir(os.path.join(input_dir, class_folder)):
            class_input_dir = os.path.join(input_dir, class_folder)
            class_output_dir = os.path.join(output_dir, class_folder)
            os.makedirs(class_output_dir, exist_ok=True)

            # Loop through each image file in the class folder
            for filename in os.listdir(class_input_dir):
                if filename.endswith(".jpg") or filename.endswith(".png"):  # Add more formats if needed
                    # Open the image
                    image_path = os.path.join(class_input_dir, filename)
                    with Image.open(image_path) as img:
                        # Resize the image
                        resized_img = img.resize(new_size)

                        # Save the resized image
                        output_path = os.path.join(class_output_dir, filename)
                        resized_img.save(output_path)

    print("Image resizing complete.")

# Set the directory containing the images
input_dir = "/content/drive/MyDrive/DATASET"
output_dir = "/content/drive/MyDrive/DATA224"
new_size = (244, 244)  # Define the new size of the images

# Resize the images
resize_images(input_dir, output_dir, new_size)
```

```
Image resizing complete.
Epoch 61/100
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
31/31 [==============================] - 32s 1s/step - loss: 0.0072 - accuracy: 0.9959 - val_loss: 1.4215 - val_accuracy: 0.8540
```

```
!pip install split-folders
```

```
Collecting split-folders
  Downloading split_folders-0.5.1-py3-none-any.whl (8.4 kB)
Installing collected packages: split-folders
Successfully installed split-folders-0.5.1
31/31 [==============================] - 31s 994ms/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.4155 - val_accuracy: 0.9431
```

```
import splitfolders

# Split with a ratio.
# To only split into training and validation set, set a tuple to ratio, i.e, (.8, .2).
splitfolders.ratio("/content/drive/MyDrive/DATASET", output="/content/drive/MyDrive/DATA224",
    seed=1337, ratio=(.7, .15, .15), group_prefix=None, move=False) # default values
```

```
Copying files: 1409 files [00:28, 49.61 files/s]
31/31 [==============================] - 30s 952ms/step - loss: 0.0031 - accuracy: 0.9990 - val_loss: 1.4404 - val_accuracy: 0.8009
```

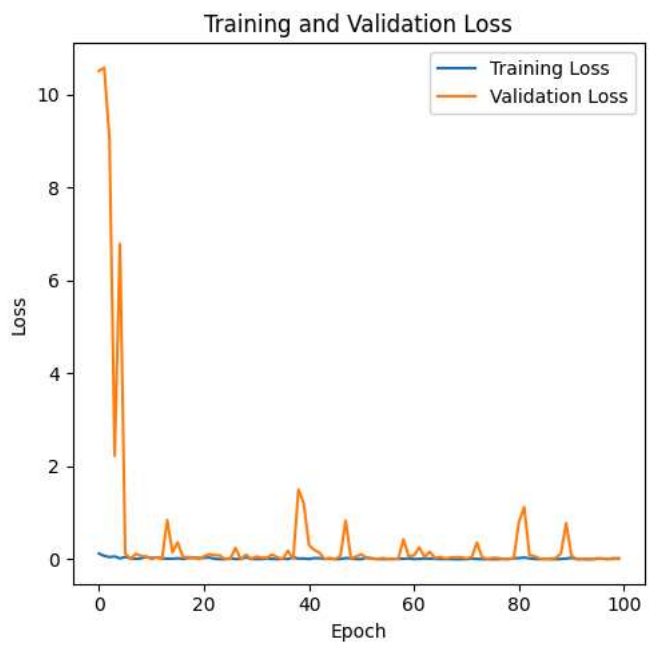
```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
from itertools import cycle

# Define the dataset directories
dataset_dirs = ['/content/drive/MyDrive/DATA224SPLIT']
dataset_names = ['Image_Data']

# Define common hyperparameters
img_width, img_height = 244, 244  #MobileNet requires input shape of 299x299
input_shape = (img_width, img_height, 3)
batch_size = 32

# Data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=20,
    horizontal_flip=True,
    fill_mode='nearest')

# Data augmentation for validation and testing (only rescale)
val_test_datagen = ImageDataGenerator(rescale=1.0 / 255)

# Define a list of colors for plotting ROC curves
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'green'])

# Loop over datasets
for dataset_dir, dataset_name in zip(dataset_dirs, dataset_names):
    print(f"Training and evaluating on {dataset_name}...")

    # Load and preprocess data for the current dataset
    train_data_dir ='/content/drive/MyDrive/DATA224SPLIT/train'
    val_data_dir = '/content/drive/MyDrive/DATA224SPLIT/val'
    test_data_dir = '/content/drive/MyDrive/DATA224SPLIT/test'

    train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')  # Set class_mode to 'binary'

    val_generator = val_test_datagen.flow_from_directory(
        val_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')  # Set class_mode to 'binary'

    test_generator = val_test_datagen.flow_from_directory(
        test_data_dir,
        target_size=(img_width, img_height),
        batch_size=batch_size,
        class_mode='binary')  # Set class_mode to 'binary'

    # Build and train the model
    base_model = MobileNet(input_shape=input_shape, include_top=False, weights='imagenet')

    custom_model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')  # Change the number of output units to 1 for binary classification
    ])

    custom_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Change loss to 'binary_crossentropy'

    history = custom_model.fit(train_generator, epochs=100, validation_data=val_generator)
    validation_accuracy = history.history['val_accuracy'][-1]

    # Plot training and validation loss
    plt.figure(figsize=(10, 5))
```

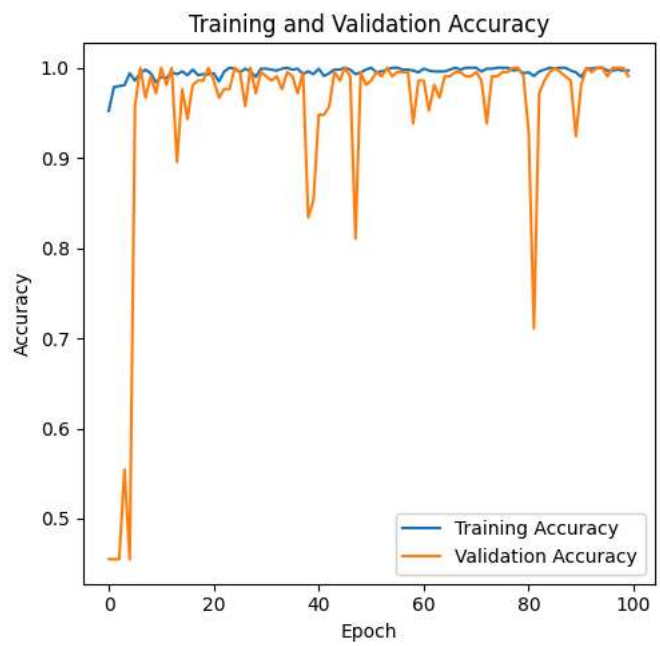
```
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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

plt.tight_layout()
plt.show()

# Evaluate the model on testing data
loss, testing_accuracy = custom_model.evaluate(test_generator)

# Make predictions using the model on the testing set
preds = custom_model.predict(test_generator)
class_preds = np.round(preds).flatten().astype(int)  # Round predictions for binary classification

# Print classification report
print("Classification Report:")
print(classification_report(test_generator.classes, class_preds))

# Compute the confusion matrix
cm = confusion_matrix(test_generator.classes, class_preds)
print("Confusion Matrix:")
print(cm)

# Calculate ROC curve and AUC for the binary class
fpr, tpr, _ = roc_curve(test_generator.classes, preds)
roc_auc = auc(fpr, tpr)
print("ROC and AUC for Binary Class:")
print(f"FPR: {fpr}")
print(f"TPR: {tpr}")
print(f"AUC: {roc_auc}")

# Plot ROC curve for the binary class
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f' MobileNet (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title(f'ROC Curve -  MobileNet - {dataset_name}')
plt.legend(loc='lower right')
plt.show()
```

```
Training and evaluating on Image_Data...
Found 986 images belonging to 2 classes.
Found 211 images belonging to 2 classes.
Found 212 images belonging to 2 classes.
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224,
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet/mobilenet_1_0_224_tf_no_top.h5
17225924/17225924 [==============================] - 0s 0us/step
Epoch 1/100
31/31 [==============================] - 253s 7s/step - loss: 0.1214 - accuracy: 0.9523 - val_loss: 10.5088 - val_accuracy: 0.4550
Epoch 2/100
31/31 [==============================] - 19s 604ms/step - loss: 0.0722 - accuracy: 0.9787 - val_loss: 10.5760 - val_accuracy: 0.4550
Epoch 3/100
31/31 [==============================] - 18s 592ms/step - loss: 0.0458 - accuracy: 0.9797 - val_loss: 9.0358 - val_accuracy: 0.4550
Epoch 4/100
31/31 [==============================] - 19s 600ms/step - loss: 0.0584 - accuracy: 0.9807 - val_loss: 2.2207 - val_accuracy: 0.5545
Epoch 5/100
31/31 [==============================] - 19s 605ms/step - loss: 0.0175 - accuracy: 0.9939 - val_loss: 6.7853 - val_accuracy: 0.4550
Epoch 6/100
31/31 [==============================] - 19s 597ms/step - loss: 0.0479 - accuracy: 0.9858 - val_loss: 0.1248 - val_accuracy: 0.9573
Epoch 7/100
31/31 [==============================] - 20s 638ms/step - loss: 0.0209 - accuracy: 0.9939 - val_loss: 9.1939e-04 - val_accuracy: 1.00
Epoch 8/100
31/31 [==============================] - 19s 597ms/step - loss: 0.0071 - accuracy: 0.9980 - val_loss: 0.1174 - val_accuracy: 0.9668
Epoch 9/100
31/31 [==============================] - 19s 627ms/step - loss: 0.0147 - accuracy: 0.9939 - val_loss: 0.0675 - val_accuracy: 0.9905
Epoch 10/100
31/31 [==============================] - 18s 586ms/step - loss: 0.0482 - accuracy: 0.9838 - val_loss: 0.0584 - val_accuracy: 0.9716
Epoch 11/100
31/31 [==============================] - 18s 581ms/step - loss: 0.0256 - accuracy: 0.9899 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 12/100
31/31 [==============================] - 19s 604ms/step - loss: 0.0359 - accuracy: 0.9888 - val_loss: 0.0408 - val_accuracy: 0.9810
Epoch 13/100
31/31 [==============================] - 18s 579ms/step - loss: 0.0180 - accuracy: 0.9949 - val_loss: 0.0018 - val_accuracy: 1.0000
Epoch 14/100
31/31 [==============================] - 19s 606ms/step - loss: 0.0124 - accuracy: 0.9929 - val_loss: 0.8420 - val_accuracy: 0.8957
Epoch 15/100
31/31 [==============================] - 18s 583ms/step - loss: 0.0093 - accuracy: 0.9959 - val_loss: 0.1482 - val_accuracy: 0.9763
Epoch 16/100
31/31 [==============================] - 19s 627ms/step - loss: 0.0207 - accuracy: 0.9919 - val_loss: 0.3645 - val_accuracy: 0.9431
Epoch 17/100
31/31 [==============================] - 18s 591ms/step - loss: 0.0066 - accuracy: 0.9980 - val_loss: 0.0466 - val_accuracy: 0.9810
Epoch 18/100
31/31 [==============================] - 20s 634ms/step - loss: 0.0243 - accuracy: 0.9919 - val_loss: 0.0463 - val_accuracy: 0.9858
Epoch 19/100
31/31 [==============================] - 18s 593ms/step - loss: 0.0299 - accuracy: 0.9929 - val_loss: 0.0370 - val_accuracy: 0.9858
Epoch 20/100
31/31 [==============================] - 20s 640ms/step - loss: 0.0266 - accuracy: 0.9929 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 21/100
31/31 [==============================] - 18s 590ms/step - loss: 0.0277 - accuracy: 0.9939 - val_loss: 0.0627 - val_accuracy: 0.9858
Epoch 22/100
31/31 [==============================] - 19s 607ms/step - loss: 0.0366 - accuracy: 0.9848 - val_loss: 0.1074 - val_accuracy: 0.9668
Epoch 23/100
31/31 [==============================] - 18s 591ms/step - loss: 0.0090 - accuracy: 0.9959 - val_loss: 0.0876 - val_accuracy: 0.9763
Epoch 24/100
31/31 [==============================] - 19s 597ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.0823 - val_accuracy: 0.9763
Epoch 25/100
31/31 [==============================] - 19s 601ms/step - loss: 0.0017 - accuracy: 0.9990 - val_loss: 0.0017 - val_accuracy: 1.0000
Epoch 26/100
31/31 [==============================] - 18s 594ms/step - loss: 0.0202 - accuracy: 0.9949 - val_loss: 0.0085 - val_accuracy: 0.9953
Epoch 27/100
31/31 [==============================] - 19s 594ms/step - loss: 0.0040 - accuracy: 0.9990 - val_loss: 0.2427 - val_accuracy: 0.9573
Epoch 28/100
31/31 [==============================] - 19s 596ms/step - loss: 0.0067 - accuracy: 0.9959 - val_loss: 1.4793e-04 - val_accuracy: 1.00
Epoch 29/100
31/31 [==============================] - 19s 609ms/step - loss: 0.0376 - accuracy: 0.9899 - val_loss: 0.0956 - val_accuracy: 0.9716
Epoch 30/100
31/31 [==============================] - 19s 596ms/step - loss: 0.0063 - accuracy: 0.9990 - val_loss: 0.0122 - val_accuracy: 0.9953
Epoch 31/100
31/31 [==============================] - 19s 622ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 0.0572 - val_accuracy: 0.9905
Epoch 32/100
31/31 [==============================] - 18s 588ms/step - loss: 0.0037 - accuracy: 0.9980 - val_loss: 0.0325 - val_accuracy: 0.9858
Epoch 33/100
31/31 [==============================] - 20s 630ms/step - loss: 0.0170 - accuracy: 0.9970 - val_loss: 0.0346 - val_accuracy: 0.9905
Epoch 34/100
31/31 [==============================] - 18s 599ms/step - loss: 0.0058 - accuracy: 0.9990 - val_loss: 0.1009 - val_accuracy: 0.9763
Epoch 35/100
31/31 [==============================] - 19s 616ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.0380 - val_accuracy: 0.9953
Epoch 36/100
31/31 [==============================] - 18s 586ms/step - loss: 0.0112 - accuracy: 0.9980 - val_loss: 0.0095 - val_accuracy: 0.9905
Epoch 37/100
31/31 [==============================] - 19s 624ms/step - loss: 0.0033 - accuracy: 0.9990 - val_loss: 0.1804 - val_accuracy: 0.9716
Epoch 38/100
31/31 [==============================] - 18s 585ms/step - loss: 0.0416 - accuracy: 0.9929 - val_loss: 0.0066 - val_accuracy: 0.9953
Epoch 39/100
```

```
31/31 [==============================] - 19s 615ms/step - loss: 0.0119 - accuracy: 0.9959 - val_loss: 1.5014 - val_accuracy: 0.8341
Epoch 40/100
31/31 [==============================] - 18s 591ms/step - loss: 0.0153 - accuracy: 0.9929 - val_loss: 1.2029 - val_accuracy: 0.8531
Epoch 41/100
31/31 [==============================] - 19s 599ms/step - loss: 0.0051 - accuracy: 0.9990 - val_loss: 0.3041 - val_accuracy: 0.9479
Epoch 42/100
31/31 [==============================] - 19s 608ms/step - loss: 0.0226 - accuracy: 0.9909 - val_loss: 0.1983 - val_accuracy: 0.9479
Epoch 43/100
31/31 [==============================] - 18s 593ms/step - loss: 0.0223 - accuracy: 0.9939 - val_loss: 0.1411 - val_accuracy: 0.9573
Epoch 44/100
31/31 [==============================] - 20s 632ms/step - loss: 0.0061 - accuracy: 0.9980 - val_loss: 0.0055 - val_accuracy: 0.9953
Epoch 45/100
31/31 [==============================] - 19s 624ms/step - loss: 0.0120 - accuracy: 0.9980 - val_loss: 0.0251 - val_accuracy: 0.9858
Epoch 46/100
31/31 [==============================] - 21s 689ms/step - loss: 0.0032 - accuracy: 1.0000 - val_loss: 1.8036e-04 - val_accuracy: 1.00
Epoch 47/100
31/31 [==============================] - 19s 603ms/step - loss: 0.0046 - accuracy: 0.9980 - val_loss: 0.0761 - val_accuracy: 0.9905
Epoch 48/100
31/31 [==============================] - 20s 629ms/step - loss: 0.0248 - accuracy: 0.9929 - val_loss: 0.8332 - val_accuracy: 0.8104
Epoch 49/100
31/31 [==============================] - 19s 598ms/step - loss: 0.0158 - accuracy: 0.9949 - val_loss: 0.0166 - val_accuracy: 0.9953
Epoch 50/100
31/31 [==============================] - 19s 624ms/step - loss: 0.0027 - accuracy: 0.9980 - val_loss: 0.0604 - val_accuracy: 0.9810
Epoch 51/100
31/31 [==============================] - 18s 585ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.1070 - val_accuracy: 0.9858
Epoch 52/100
31/31 [==============================] - 19s 609ms/step - loss: 0.0361 - accuracy: 0.9949 - val_loss: 0.0214 - val_accuracy: 0.9953
Epoch 53/100
31/31 [==============================] - 19s 591ms/step - loss: 0.0117 - accuracy: 0.9959 - val_loss: 0.0202 - val_accuracy: 0.9905
Epoch 54/100
31/31 [==============================] - 18s 589ms/step - loss: 0.0047 - accuracy: 0.9980 - val_loss: 6.7149e-04 - val_accuracy: 1.00
Epoch 55/100
31/31 [==============================] - 19s 623ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.0267 - val_accuracy: 0.9905
Epoch 56/100
31/31 [==============================] - 18s 577ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.0058 - val_accuracy: 0.9953
Epoch 57/100
31/31 [==============================] - 19s 606ms/step - loss: 0.0054 - accuracy: 0.9980 - val_loss: 0.0061 - val_accuracy: 0.9953
Epoch 58/100
31/31 [==============================] - 18s 587ms/step - loss: 0.0075 - accuracy: 0.9980 - val_loss: 0.0072 - val_accuracy: 0.9953
Epoch 59/100
31/31 [==============================] - 18s 574ms/step - loss: 0.0083 - accuracy: 0.9970 - val_loss: 0.4282 - val_accuracy: 0.9384
Epoch 60/100
31/31 [==============================] - 19s 605ms/step - loss: 0.0171 - accuracy: 0.9949 - val_loss: 0.0599 - val_accuracy: 0.9858
Epoch 61/100
31/31 [==============================] - 18s 577ms/step - loss: 0.0029 - accuracy: 0.9990 - val_loss: 0.0837 - val_accuracy: 0.9858
Epoch 62/100
31/31 [==============================] - 19s 599ms/step - loss: 0.0064 - accuracy: 0.9970 - val_loss: 0.2543 - val_accuracy: 0.9526
Epoch 63/100
31/31 [==============================] - 18s 583ms/step - loss: 0.0147 - accuracy: 0.9959 - val_loss: 0.0528 - val_accuracy: 0.9810
Epoch 64/100
31/31 [==============================] - 19s 604ms/step - loss: 0.0140 - accuracy: 0.9959 - val_loss: 0.1612 - val_accuracy: 0.9668
Epoch 65/100
31/31 [==============================] - 18s 584ms/step - loss: 0.0096 - accuracy: 0.9959 - val_loss: 0.0286 - val_accuracy: 0.9905
Epoch 66/100
31/31 [==============================] - 18s 583ms/step - loss: 0.0048 - accuracy: 0.9980 - val_loss: 0.0498 - val_accuracy: 0.9905
Epoch 67/100
31/31 [==============================] - 18s 587ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.0159 - val_accuracy: 0.9953
Epoch 68/100
31/31 [==============================] - 18s 592ms/step - loss: 0.0022 - accuracy: 0.9980 - val_loss: 0.0364 - val_accuracy: 0.9953
Epoch 69/100
31/31 [==============================] - 18s 584ms/step - loss: 1.2013e-04 - accuracy: 1.0000 - val_loss: 0.0453 - val_accuracy: 0.99
Epoch 70/100
31/31 [==============================] - 18s 588ms/step - loss: 1.2746e-04 - accuracy: 1.0000 - val_loss: 0.0429 - val_accuracy: 0.99
Epoch 71/100
31/31 [==============================] - 18s 585ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.0185 - val_accuracy: 0.9953
Epoch 72/100
31/31 [==============================] - 18s 575ms/step - loss: 0.0141 - accuracy: 0.9959 - val_loss: 0.0426 - val_accuracy: 0.9858
Epoch 73/100
31/31 [==============================] - 19s 592ms/step - loss: 0.0053 - accuracy: 0.9990 - val_loss: 0.3567 - val_accuracy: 0.9384
Epoch 74/100
31/31 [==============================] - 18s 585ms/step - loss: 0.0015 - accuracy: 0.9990 - val_loss: 0.0329 - val_accuracy: 0.9905
Epoch 75/100
31/31 [==============================] - 19s 605ms/step - loss: 2.3933e-04 - accuracy: 1.0000 - val_loss: 0.0114 - val_accuracy: 0.99
Epoch 76/100
31/31 [==============================] - 18s 575ms/step - loss: 2.8193e-04 - accuracy: 1.0000 - val_loss: 0.0306 - val_accuracy: 0.99
Epoch 77/100
31/31 [==============================] - 19s 618ms/step - loss: 1.8485e-04 - accuracy: 1.0000 - val_loss: 0.0257 - val_accuracy: 0.99
Epoch 78/100
31/31 [==============================] - 18s 586ms/step - loss: 0.0055 - accuracy: 0.9970 - val_loss: 0.0056 - val_accuracy: 1.0000
Epoch 79/100
31/31 [==============================] - 19s 616ms/step - loss: 0.0053 - accuracy: 0.9980 - val_loss: 0.0021 - val_accuracy: 1.0000
Epoch 80/100
31/31 [==============================] - 19s 599ms/step - loss: 0.0211 - accuracy: 0.9939 - val_loss: 0.0198 - val_accuracy: 0.9905
Epoch 81/100
31/31 [==============================] - 19s 605ms/step - loss: 0.0227 - accuracy: 0.9949 - val_loss: 0.8045 - val_accuracy: 0.9289
```

```
Epoch 82/100
31/31 [==============================] - 18s 589ms/step - loss: 0.0355 - accuracy: 0.9909 - val_loss: 1.1230 - val_accuracy: 0.7109
Epoch 83/100
31/31 [==============================] - 19s 627ms/step - loss: 0.0178 - accuracy: 0.9959 - val_loss: 0.0914 - val_accuracy: 0.9716
Epoch 84/100
31/31 [==============================] - 18s 591ms/step - loss: 0.0057 - accuracy: 0.9980 - val_loss: 0.0627 - val_accuracy: 0.9858
Epoch 85/100
31/31 [==============================] - 19s 603ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.0136 - val_accuracy: 0.9953
Epoch 86/100
31/31 [==============================] - 18s 587ms/step - loss: 0.0084 - accuracy: 0.9990 - val_loss: 9.0613e-04 - val_accuracy: 1.00
Epoch 87/100
31/31 [==============================] - 19s 608ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0108 - val_accuracy: 0.9953
Epoch 88/100
31/31 [==============================] - 19s 606ms/step - loss: 5.4175e-04 - accuracy: 1.0000 - val_loss: 0.0237 - val_accuracy: 0.99
Epoch 89/100
31/31 [==============================] - 18s 585ms/step - loss: 0.0056 - accuracy: 0.9970 - val_loss: 0.1258 - val_accuracy: 0.9858
Epoch 90/100
31/31 [==============================] - 20s 641ms/step - loss: 0.0133 - accuracy: 0.9949 - val_loss: 0.7731 - val_accuracy: 0.9242
Epoch 91/100
31/31 [==============================] - 18s 585ms/step - loss: 0.0297 - accuracy: 0.9899 - val_loss: 0.0782 - val_accuracy: 0.9810
Epoch 92/100
31/31 [==============================] - 20s 623ms/step - loss: 0.0032 - accuracy: 0.9990 - val_loss: 5.4350e-05 - val_accuracy: 1.00
Epoch 93/100
31/31 [==============================] - 19s 602ms/step - loss: 0.0024 - accuracy: 0.9990 - val_loss: 0.0071 - val_accuracy: 0.9953
Epoch 94/100
31/31 [==============================] - 20s 632ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 95/100
31/31 [==============================] - 18s 592ms/step - loss: 2.4822e-04 - accuracy: 1.0000 - val_loss: 0.0014 - val_accuracy: 1.00
Epoch 96/100
31/31 [==============================] - 20s 633ms/step - loss: 0.0119 - accuracy: 0.9970 - val_loss: 0.0214 - val_accuracy: 0.9905
Epoch 97/100
31/31 [==============================] - 18s 582ms/step - loss: 0.0123 - accuracy: 0.9970 - val_loss: 0.0058 - val_accuracy: 1.0000
Epoch 98/100
31/31 [==============================] - 19s 615ms/step - loss: 0.0035 - accuracy: 0.9980 - val_loss: 0.0037 - val_accuracy: 1.0000
Epoch 99/100
31/31 [==============================] - 19s 591ms/step - loss: 0.0192 - accuracy: 0.9970 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 100/100
31/31 [==============================] - 19s 602ms/step - loss: 0.0109 - accuracy: 0.9970 - val_loss: 0.0239 - val_accuracy: 0.9905
```






```
7/7 [==============================] - 50s 8s/step - loss: 0.0857 - accuracy: 0.9575
7/7 [==============================] - 1s 120ms/step
Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.60      0.58       116
           1       0.47      0.43      0.45        96

    accuracy                           0.52       212
   macro avg       0.52      0.52      0.51       212
weighted avg       0.52      0.52      0.52       212

Confusion Matrix:
[[70 46]
 [55 41]]
ROC and AUC for Binary Class:
FPR: [0.         0.12931034 0.13793103 0.18965517 0.19827586 0.20689655
 0.21551724 0.21551724 0.22413793 0.24137931 0.25       0.25
 0.25862069 0.25862069 0.30172414 0.30172414 0.31034483 0.31034483
```