```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam


# Define the number of classes
num_classes = 10

# Define data generators for training and validation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)


train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='training')

validation_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/New Plant Diseases Dataset(Augmented)/New Plant Diseases Dataset(Augmented)/valid',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset='validation')
```

```
Found 13420 images belonging to 10 classes.
Found 914 images belonging to 10 classes.
```

```
# Load the InceptionV3 model pre-trained on ImageNet
base_model = InceptionV3(weights='imagenet', include_top=False)

# Add a global average pooling layer and a dense layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(num_classes, activation='softmax')(x)

# Combine the base model with our custom layers
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_
87910968/87910968 [==============================] - 0s 0us/step
```

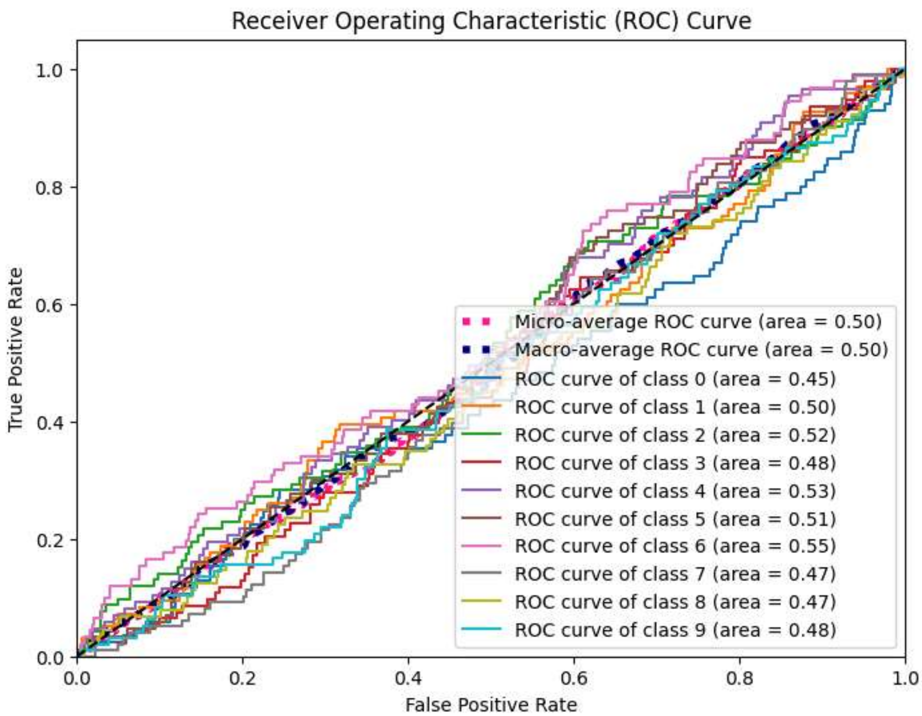
```
# Train the model with a fixed number of steps per epoch and validation steps
model.fit(
    train_generator,
    steps_per_epoch=50,  # Set a fixed number of steps per epoch
    validation_data=validation_generator,
    validation_steps=50,  # Set a fixed number of validation steps
    epochs=10)

# Save the model
model.save('model_inception.h5')
```

```
Epoch 1/10
50/50 [==============================] - ETA: 0s - loss: 1.2926 - accuracy: 0.5375WARNING:tensorflow:Your input ran out of data; interru
50/50 [==============================] - 559s 11s/step - loss: 1.2926 - accuracy: 0.5375 - val_loss: 1.4544 - val_accuracy: 0.5602
Epoch 2/10
50/50 [==============================] - 335s 7s/step - loss: 0.9449 - accuracy: 0.6734
Epoch 3/10
50/50 [==============================] - 308s 6s/step - loss: 0.8754 - accuracy: 0.7044
Epoch 4/10
50/50 [==============================] - 254s 5s/step - loss: 0.7125 - accuracy: 0.7556
Epoch 5/10
50/50 [==============================] - 209s 4s/step - loss: 0.7234 - accuracy: 0.7468
Epoch 6/10
50/50 [==============================] - 206s 4s/step - loss: 0.6955 - accuracy: 0.7594
Epoch 7/10
50/50 [==============================] - 188s 4s/step - loss: 0.6563 - accuracy: 0.7663
Epoch 8/10
50/50 [==============================] - 148s 3s/step - loss: 0.5933 - accuracy: 0.7949
Epoch 9/10
50/50 [==============================] - 151s 3s/step - loss: 0.6175 - accuracy: 0.7956
Epoch 10/10
50/50 [==============================] - 159s 3s/step - loss: 0.6008 - accuracy: 0.7850
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
```

```
import numpy as np
from sklearn.metrics import classification_report, roc_curve, auc, confusion_matrix
import matplotlib.pyplot as plt

# Evaluate the model on the validation set
val_loss, val_accuracy = model.evaluate(validation_generator)

# Predict probabilities for the validation set
y_pred_prob = model.predict(validation_generator)
y_true = validation_generator.classes

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(num_classes):
    fpr[i], tpr[i], _ = roc_curve(y_true == i, y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr_micro, tpr_micro, _ = roc_curve(np.eye(num_classes)[y_true].ravel(), y_pred_prob.ravel())
roc_auc_micro = auc(fpr_micro, tpr_micro)

# Compute macro-average ROC curve and ROC area
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(num_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(num_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= num_classes
fpr_macro = all_fpr
tpr_macro = mean_tpr
roc_auc_macro = auc(fpr_macro, tpr_macro)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_micro, tpr_micro, label='Micro-average ROC curve (area = {0:0.2f})'.format(roc_auc_micro), color='deeppink', linestyle=':', linew
plt.plot(fpr_macro, tpr_macro, label='Macro-average ROC curve (area = {0:0.2f})'.format(roc_auc_macro), color='navy', linestyle=':', linewidtl
for i in range(num_classes):
    plt.plot(fpr[i], tpr[i], label='ROC curve of class {0} (area = {1:0.2f})'.format(i, roc_auc[i]))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Compute confusion matrix
y_pred = np.argmax(y_pred_prob, axis=1)
conf_mat = confusion_matrix(y_true, y_pred)

# Display classification report and confusion matrix
```

```
print("Classification Report:")
print(classification_report(y_true, y_pred))
print("Confusion Matrix:")
print(conf_mat)
```

```
29/29 [==============================] - 20s 677ms/step - loss: 0.8122 - accuracy: 0.7079
29/29 [==============================] - 16s 497ms/step
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.09      0.02      0.04        85
           1       0.10      0.15      0.12        96
           2       0.14      0.11      0.12        92
           3       0.06      0.06      0.06        93
           4       0.12      0.14      0.13        87
           5       0.09      0.10      0.09        87
           6       0.17      0.16      0.17        91
           7       0.05      0.05      0.05        98
           8       0.08      0.08      0.08        89
           9       0.09      0.08      0.08        96

    accuracy                           0.10       914
   macro avg       0.10      0.10      0.09       914
weighted avg       0.10      0.10      0.09       914

Confusion Matrix:
[[ 2 10  5  7  6 11  8 16  9 11]
 [ 1 14  7 13  9  9  6 11 10 16]
 [ 2 13 10  9 13 12  6 10  8  9]
 [ 1 18  7  6 12 13 12  9  9  6]
 [ 4  5 10  8 12 10 13  8  9  8]
 [ 3 15  3 16 10  9  7  7  7 10]
 [ 6 13  4  9 13  9 15  7  6  9]
 [ 1 22 10 18  7  9  8  5 11  7]
 [ 2 16  5 12 12  7  7 11  7 10]
 [ 1 16  9 11  7 14  8  9 13  8]]
```