```
import os
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Directories and Parameters
base_dir = "/content/drive/MyDrive/spectrograms_Images"
IMG_HEIGHT, IMG_WIDTH = 128, 128

# Data Loading and Preprocessing
datagen = ImageDataGenerator(rescale=1.0 / 255)
data = datagen.flow_from_directory(
    base_dir,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=1,
    class_mode='categorical',
    shuffle=False
)

# Convert images and labels to arrays
images = []
labels = []

for i in range(len(data)):
    img, label = data[i]
    images.append(img[0])
    labels.append(label[0])

images = np.array(images).reshape(len(images), -1)  # Flatten images
labels = np.argmax(labels, axis=1)  # Convert one-hot labels to integers

# Standardize Features
scaler = StandardScaler()
images_scaled = scaler.fit_transform(images)

# One-Hot Encode Labels for Neural Network
encoder = OneHotEncoder(sparse_output=False)
labels_onehot = encoder.fit_transform(labels.reshape(-1, 1))

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(images_scaled, labels,
test_size=0.2, random_state=42)
X_train_nn, X_test_nn, y_train_nn, y_test_nn = train_test_split(images_scaled,
labels_onehot, test_size=0.2, random_state=42)

# Random Forest Classifier
print("Training Random Forest Classifier...")
rf_model = RandomForestClassifier(n_estimators=200, max_depth=20,
random_state=42)
rf_model.fit(X_train, y_train)

# Evaluate Random Forest
y_pred_rf = rf_model.predict(X_test)
rf_acc = np.mean(y_pred_rf == y_test) * 100
print(f"\nRandom Forest Test Accuracy: {rf_acc:.2f}%")
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))

# Confusion Matrix for Random Forest
cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=data.
class_indices.keys(), yticklabels=data.class_indices.keys())
```

```
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Neural Network Model
print("Training Neural Network...")
nn_model = Sequential([
    Dense(1024, activation='relu', input_shape=(X_train_nn.shape[1],)),
    Dropout(0.4),
    Dense(512, activation='relu'),
    Dropout(0.3),
    Dense(256, activation='relu'),
    Dropout(0.2),
    Dense(y_train_nn.shape[1], activation='softmax')  # Output layer
])

# Compile the model
nn_model.compile(optimizer=Adam(learning_rate=5e-4),
loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = nn_model.fit(X_train_nn, y_train_nn, validation_data=(X_test_nn,
y_test_nn), epochs=30, batch_size=64)

# Evaluate the Neural Network
nn_loss, nn_acc = nn_model.evaluate(X_test_nn, y_test_nn)
print(f"\nNeural Network Test Accuracy: {nn_acc * 100:.2f}%")

# Plot Accuracy and Loss for Neural Network
def plot_nn_history(history):
    plt.figure(figsize=(14, 6), dpi=100)
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy',
    color='blue')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy',
    color='orange')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

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

    plt.tight_layout()
    plt.show()

plot_nn_history(history)

# Confusion Matrix for Neural Network
y_pred_nn_probs = nn_model.predict(X_test_nn)
y_pred_nn = np.argmax(y_pred_nn_probs, axis=1)
y_true_nn = np.argmax(y_test_nn, axis=1)

cm_nn = confusion_matrix(y_true_nn, y_pred_nn)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nn, annot=True, fmt='d', cmap='Blues', xticklabels=data.
class_indices.keys(), yticklabels=data.class_indices.keys())
plt.title("Confusion Matrix - Neural Network")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

print("\nNeural Network Classification Report:")
print(classification_report(y_true_nn, y_pred_nn, target_names=data.
class_indices.keys()))
```

```
class_indices.keys())))
```