CNN Model:

```
import os
import glob
import soundfile
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras import layers, models, Input
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

########################################
# Configuration
########################################
DATA_PATH = "/content/drive/MyDrive/Speech Recognition -
Maha/Ravadess/Actor_*/*.wav"
N_MFCC = 40
MAX_FRAMES = 128
SAMPLE_RATE = 22050
AVAILABLE_EMOTIONS = {"angry", "sad", "neutral", "happy"}

label_map = {'happy':0, 'sad':1, 'neutral':2, 'angry':3}
int2emotion = {
    "01": "neutral",
    "02": "calm",
    "03": "happy",
    "04": "sad",
    "05": "angry",
    "06": "fearful",
    "07": "disgust",
    "08": "surprised"
}

########################################
# Feature Extraction as Sequence
########################################
def extract_mfcc_2d(file_path, n_mfcc=40, max_frames=128, sr=22050):
    try:
        with soundfile.SoundFile(file_path) as sf:
```

```
            audio = sf.read(dtype='float32')
            file_sr = sf.samplerate

        # If too short, skip
        if len(audio) < 2048:
            return None

        # Resample if needed
        if file_sr != sr:
            audio = librosa.resample(audio, orig_sr=file_sr,
target_sr=sr)

        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)  #
shape: [n_mfcc, frames]
        mfcc = mfcc.T  # now shape: [frames, n_mfcc]

        if mfcc.shape[0] == 0:
            return None

        # Pad or truncate
        if mfcc.shape[0] < max_frames:
            pad_width = max_frames - mfcc.shape[0]
            mfcc = np.pad(mfcc, ((0,pad_width),(0,0)), mode='constant')
        else:
            mfcc = mfcc[:max_frames, :]

        return mfcc
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
        return None

def load_data(test_size=0.2):
    X, y = [], []
    for file in glob.glob(DATA_PATH):
        basename = os.path.basename(file)
        emotion_code = basename.split("-")[2]
        if emotion_code not in int2emotion:
            continue

        emotion = int2emotion[emotion_code]
        if emotion not in AVAILABLE_EMOTIONS:
            continue
```

```
        mfcc = extract_mfcc_2d(file_path=file, n_mfcc=N_MFCC,
max_frames=MAX_FRAMES, sr=SAMPLE_RATE)
        if mfcc is None:
            continue

        X.append(mfcc)
        y.append(label_map[emotion])

    X = np.array(X) # shape: [samples, MAX_FRAMES, N_MFCC]
    y = np.array(y)
    return train_test_split(X, y, test_size=test_size, random_state=42,
stratify=y)

#######################################
# Load Data
#######################################
X_train, X_test, y_train, y_test = load_data(test_size=0.2)
print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])

num_classes = len(label_map)
y_train_oh = to_categorical(y_train, num_classes=num_classes)
y_test_oh = to_categorical(y_test, num_classes=num_classes)

#######################################
# Transformer Encoder Block
#######################################
def transformer_encoder(x, num_heads, ff_dim, dropout=0.1, d_model=64):
    attn_output = layers.MultiHeadAttention(num_heads=num_heads,
key_dim=d_model, dropout=dropout)(x, x)
    attn_output = layers.Dropout(dropout)(attn_output)
    x = layers.LayerNormalization(epsilon=1e-6)(x + attn_output)

    ffn = models.Sequential([
        layers.Dense(ff_dim, activation='relu'),
        layers.Dense(d_model)
    ])
    ffn_output = ffn(x)
    ffn_output = layers.Dropout(dropout)(ffn_output)
    x = layers.LayerNormalization(epsilon=1e-6)(x + ffn_output)
    return x

#######################################
```

```
# Build Transformer Model
#######################################
def build_transformer_model(sequence_length, feature_dim, num_classes,
d_model=64, num_heads=4, ff_dim=128, num_layers=2, dropout=0.1):
    inputs = Input(shape=(sequence_length, feature_dim))  # (batch,
MAX_FRAMES, N_MFCC)

    # Create a trainable positional embedding
    # Positions: [0 ... sequence_length-1]
    pos_embedding_layer = layers.Embedding(input_dim=sequence_length,
output_dim=d_model)
    positions = tf.range(start=0, limit=sequence_length, delta=1)
    pos_emb = pos_embedding_layer(positions)  # (sequence_length,
d_model)
    pos_emb = tf.expand_dims(pos_emb, axis=0)  # (1, sequence_length,
d_model)

    # Project input features to d_model dimension
    x = layers.Dense(d_model)(inputs)
    # Add positional encoding by broadcasting pos_emb to match batch
size at runtime.
    x = x + pos_emb

    # Stacking multiple transformer encoder layers
    for _ in range(num_layers):
        x = transformer_encoder(x, num_heads=num_heads, ff_dim=ff_dim,
dropout=dropout, d_model=d_model)

    # Global Average Pooling
    x = layers.GlobalAveragePooling1D()(x)
    x = layers.Dropout(dropout)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

model = build_transformer_model(sequence_length=MAX_FRAMES,
feature_dim=N_MFCC, num_classes=num_classes, d_model=64, num_heads=4,
ff_dim=128, num_layers=2, dropout=0.1)
model.summary()
```

```
########################################
# Train the Model
########################################
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=5, min_lr=1e-5)

history = model.fit(
    X_train, y_train_oh,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test_oh),
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

########################################
# Evaluate the Model
########################################
loss, acc = model.evaluate(X_test, y_test_oh, verbose=0)
print(f"Test Accuracy: {acc*100:.2f}%")

# Plot training history
plt.figure(figsize=(14,5))
# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

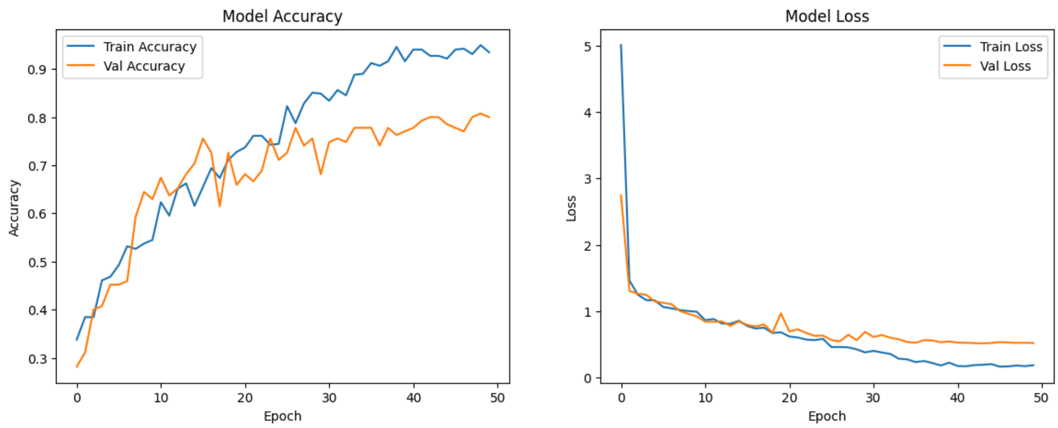
```
plt.show()

########################################
# Classification Report
########################################
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

reverse_label_map = {v:k for k,v in label_map.items()}
class_names = [reverse_label_map[i] for i in range(num_classes)]
print("Classification Report:")
print(classification_report(y_test, y_pred_classes,
target_names=class_names))

cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Training samples: 536
Testing samples: 135
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
```

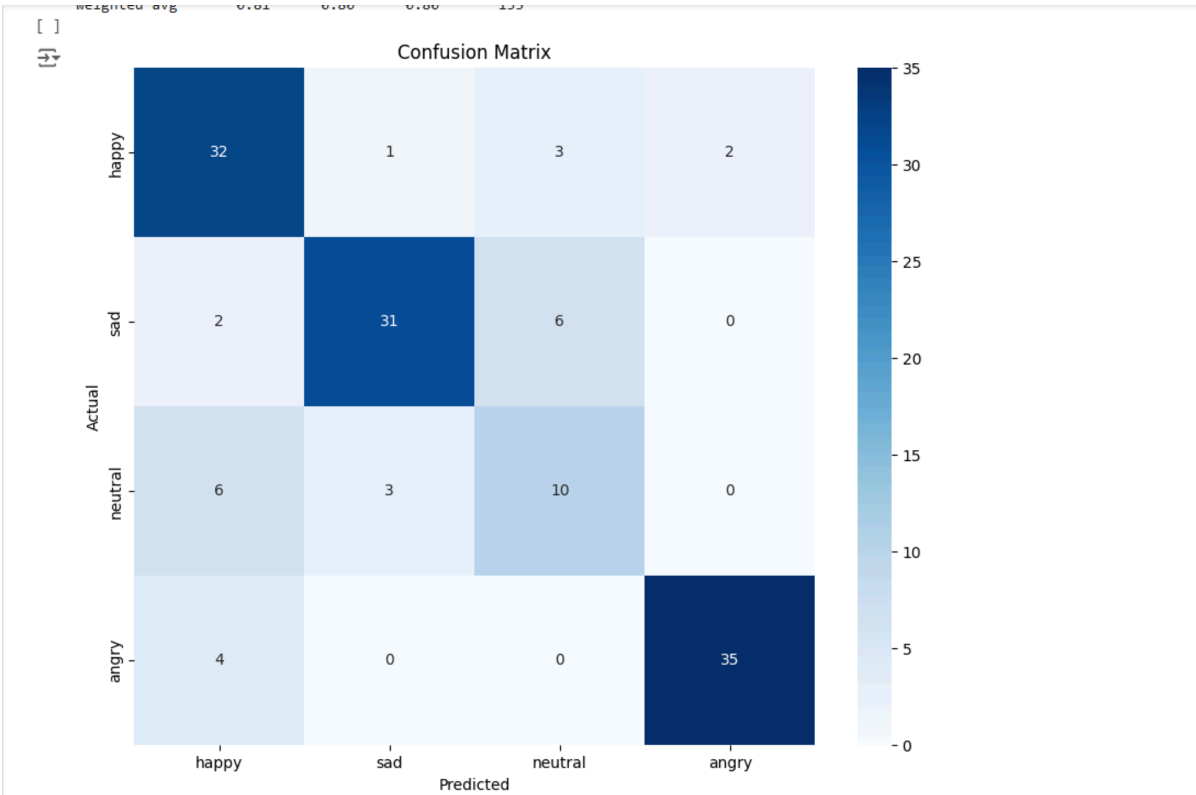
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 38, 32)	320
batch_normalization (BatchNormalization)	(None, 126, 38, 32)	128
max_pooling2d (MaxPooling2D)	(None, 63, 19, 32)	0
dropout (Dropout)	(None, 63, 19, 32)	0
conv2d_1 (Conv2D)	(None, 61, 17, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 61, 17, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 30, 8, 64)	0
dropout_1 (Dropout)	(None, 30, 8, 64)	0
conv2d_2 (Conv2D)	(None, 28, 6, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 28, 6, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 14, 3, 128)	0
dropout_2 (Dropout)	(None, 14, 3, 128)	0
flatten (Flatten)	(None, 5376)	0
dense (Dense)	(None, 256)	1,376,512
dropout_3 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 4)	1,028

```
Total params: 1,471,108 (5.61 MB)
Trainable params: 1,470,660 (5.61 MB)
Non-trainable params: 448 (1.75 KB)
```

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 11s 418ms/step - accuracy: 0.9223 - loss: 0.1842 - val_accuracy: 0.8000 - val_loss: 0.5199 - learning_rate: 4.0000e-05
Epoch 49/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 531ms/step - accuracy: 0.9518 - loss: 0.1823 - val_accuracy: 0.8074 - val_loss: 0.5219 - learning_rate: 4.0000e-05
Epoch 50/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 8s 405ms/step - accuracy: 0.9285 - loss: 0.1936 - val_accuracy: 0.8000 - val_loss: 0.5179 - learning_rate: 1.0000e-05
Test Accuracy: 80.00%
```



```
                                                        Epoch
     5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 103ms/step
     Classification Report:
                   precision    recall  f1-score   support

            happy       0.73      0.84      0.78        38
              sad       0.89      0.79      0.84        39
          neutral       0.53      0.53      0.53        19
            angry       0.95      0.90      0.92        39

         accuracy                           0.80       135
        macro avg       0.77      0.77      0.77       135
     weighted avg       0.81      0.80      0.80       135
```



ResNet Model:

```
import os
import glob
import soundfile
import numpy as np
import librosa
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import layers, models, Input
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

########################################
# Configuration
########################################
DATA_PATH = "/content/drive/MyDrive/Speech Recognition -
Maha/Ravadess/Actor_*/*.wav"
N_MFCC = 40        # number of MFCC coefficients
MAX_FRAMES = 128   # max number of frames we'll keep for each sample
SAMPLE_RATE = 22050
AVAILABLE_EMOTIONS = {"angry", "sad", "neutral", "happy"}

label_map = {'happy':0, 'sad':1, 'neutral':2, 'angry':3}
int2emotion = {
    "01": "neutral",
    "02": "calm",
    "03": "happy",
    "04": "sad",
    "05": "angry",
    "06": "fearful",
    "07": "disgust",
    "08": "surprised"
}

########################################
# Feature Extraction as 2D Input
########################################
def extract_mfcc_2d(file_path, n_mfcc=40, max_frames=128, sr=22050):
    try:
        with soundfile.SoundFile(file_path) as sf:
            audio = sf.read(dtype='float32')
            file_sr = sf.samplerate

        # If too short, skip
        if len(audio) < 2048:
            return None

        # Resample if needed
        if file_sr != sr:
            audio = librosa.resample(audio, orig_sr=file_sr,
target_sr=sr)
```

```
        # Extract MFCCs
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)  #
shape: [n_mfcc, frames]
        mfcc = mfcc.T  # shape: [frames, n_mfcc]

        if mfcc.shape[0] == 0:
            return None

        # Pad or truncate
        if mfcc.shape[0] < max_frames:
            pad_width = max_frames - mfcc.shape[0]
            mfcc = np.pad(mfcc, ((0,pad_width),(0,0)), mode='constant')
        else:
            mfcc = mfcc[:max_frames, :]

        return mfcc
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
        return None

def load_data(test_size=0.2):
    X, y = [], []
    for file in glob.glob(DATA_PATH):
        basename = os.path.basename(file)
        emotion_code = basename.split("-")[2]
        if emotion_code not in int2emotion:
            continue

        emotion = int2emotion[emotion_code]
        if emotion not in AVAILABLE_EMOTIONS:
            continue

        mfcc = extract_mfcc_2d(file_path=file, n_mfcc=N_MFCC,
max_frames=MAX_FRAMES, sr=SAMPLE_RATE)
        if mfcc is None:
            continue

        X.append(mfcc)
        y.append(label_map[emotion])

    X = np.array(X) # shape: [samples, frames, n_mfcc]
    y = np.array(y)
```

```
    return train_test_split(X, y, test_size=test_size, random_state=42,
stratify=y)

#######################################
# Load the Data
#######################################
X_train, X_test, y_train, y_test = load_data(test_size=0.2)
print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])

# Add channel dimension for CNN/ResNet: (samples, frames, n_mfcc, 1)
X_train = X_train[..., np.newaxis]
X_test = X_test[..., np.newaxis]

num_classes = len(label_map)
y_train_oh = to_categorical(y_train, num_classes=num_classes)
y_test_oh = to_categorical(y_test, num_classes=num_classes)

#######################################
# Residual Block Definition
#######################################
def residual_block(x, filters, kernel_size=(3,3)):
    # First conv layer
    shortcut = x
    x = layers.Conv2D(filters, kernel_size, padding='same',
activation='relu')(x)
    x = layers.BatchNormalization()(x)
    # Second conv layer
    x = layers.Conv2D(filters, kernel_size, padding='same',
activation=None)(x)
    x = layers.BatchNormalization()(x)

    # Add the shortcut (input) back
    x = layers.Add()([x, shortcut])
    x = layers.Activation('relu')(x)
    return x

#######################################
# Build a Simple ResNet-like Model
#######################################
def build_resnet(input_shape, num_classes):
    inputs = Input(shape=input_shape)
```

```
    # Initial Conv layer
    x = layers.Conv2D(32, (3,3), activation='relu',
padding='same')(inputs)
    x = layers.BatchNormalization()(x)

    # First residual block
    x = residual_block(x, 32)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Dropout(0.25)(x)

    # Second residual block
    x = layers.Conv2D(64, (3,3), padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = residual_block(x, 64)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Dropout(0.25)(x)

    # Third residual block
    x = layers.Conv2D(128, (3,3), padding='same', activation='relu')(x)
    x = layers.BatchNormalization()(x)
    x = residual_block(x, 128)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Dropout(0.25)(x)

    x = layers.Flatten()(x)
    x = layers.Dense(256, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs, outputs)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

model = build_resnet((MAX_FRAMES, N_MFCC, 1), num_classes)
model.summary()

#######################################
# Train the Model
#######################################
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=5, min_lr=1e-5)

history = model.fit(
    X_train, y_train_oh,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test_oh),
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

#######################################
# Evaluate the Model
#######################################
loss, acc = model.evaluate(X_test, y_test_oh, verbose=0)
print(f"Test Accuracy: {acc*100:.2f}%")

# Plot training history
plt.figure(figsize=(14,5))
# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()

#######################################
# Classification Report
#######################################
```

```python
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

reverse_label_map = {v:k for k,v in label_map.items()}
class_names = [reverse_label_map[i] for i in range(num_classes)]
print("Classification Report:")
print(classification_report(y_test, y_pred_classes,
target_names=class_names))

cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Error processing /content/drive/MyDrive/Speech Recognition - Mana/Ravadess/Actor_20/03-01-03-01-02-01-20.wav: operan
Training samples: 536
Testing samples: 135
```

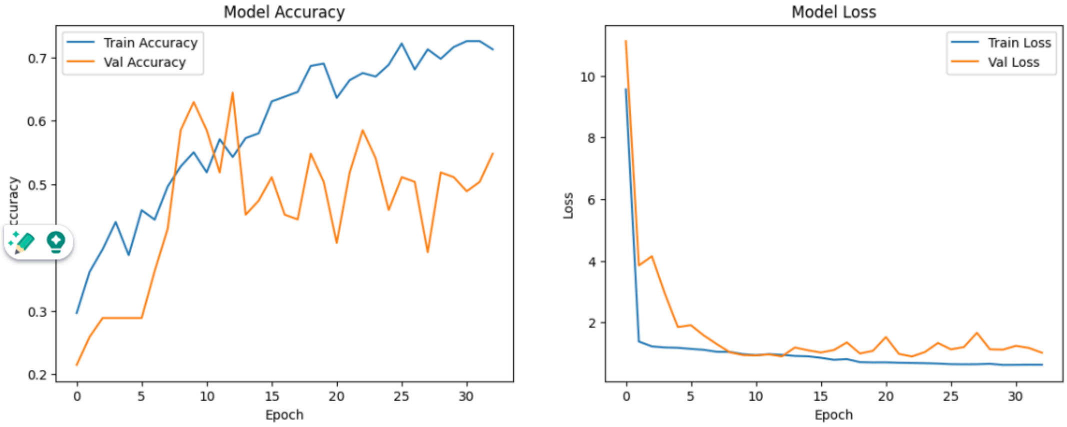
Model: "functional_80"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 128, 40, 1)	0	-
conv2d_3 (Conv2D)	(None, 128, 40, 32)	320	input_layer_1[0][0]
batch_normalization_3 (BatchNormalization)	(None, 128, 40, 32)	128	conv2d_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 40, 32)	9,248	batch_normalization_3…
batch_normalization_4 (BatchNormalization)	(None, 128, 40, 32)	128	conv2d_4[0][0]
conv2d_5 (Conv2D)	(None, 128, 40, 32)	9,248	batch_normalization_4…
batch_normalization_5 (BatchNormalization)	(None, 128, 40, 32)	128	conv2d_5[0][0]
add (Add)	(None, 128, 40, 32)	0	batch_normalization_5… batch_normalization_3…
activation (Activation)	(None, 128, 40, 32)	0	add[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 64, 20, 32)	0	activation[0][0]
dropout_4 (Dropout)	(None, 64, 20, 32)	0	max_pooling2d_3[0][0]
conv2d_6 (Conv2D)	(None, 64, 20, 64)	18,496	dropout_4[0][0]
batch_normalization_6 (BatchNormalization)	(None, 64, 20, 64)	256	conv2d_6[0][0]
conv2d_7 (Conv2D)	(None, 64, 20, 64)	36,928	batch_normalization_6…
batch_normalization_7 (BatchNormalization)	(None, 64, 20, 64)	256	conv2d_7[0][0]
conv2d_8 (Conv2D)	(None, 64, 20, 64)	36,928	batch_normalization_7…

batch_normalization_8 (BatchNormalization)	(None, 64, 20, 64)	256	conv2d_8[0][0]
add_1 (Add)	(None, 64, 20, 64)	0	batch_normalization_8… batch_normalization_6…
activation_1 (Activation)	(None, 64, 20, 64)	0	add_1[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 32, 10, 64)	0	activation_1[0][0]
dropout_5 (Dropout)	(None, 32, 10, 64)	0	max_pooling2d_4[0][0]
conv2d_9 (Conv2D)	(None, 32, 10, 128)	73,856	dropout_5[0][0]
batch_normalization_9 (BatchNormalization)	(None, 32, 10, 128)	512	conv2d_9[0][0]
conv2d_10 (Conv2D)	(None, 32, 10, 128)	147,584	batch_normalization_9…
batch_normalization_10 (BatchNormalization)	(None, 32, 10, 128)	512	conv2d_10[0][0]
conv2d_11 (Conv2D)	(None, 32, 10, 128)	147,584	batch_normalization_1…
batch_normalization_11 (BatchNormalization)	(None, 32, 10, 128)	512	conv2d_11[0][0]
add_2 (Add)	(None, 32, 10, 128)	0	batch_normalization_1… batch_normalization_9…
activation_2 (Activation)	(None, 32, 10, 128)	0	add_2[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 16, 5, 128)	0	activation_2[0][0]
dropout_6 (Dropout)	(None, 16, 5, 128)	0	max_pooling2d_5[0][0]
flatten_1 (Flatten)	(None, 10240)	0	dropout_6[0][0]
dense_2 (Dense)	(None, 256)	2,621,696	flatten_1[0][0]
dropout_7 (Dropout)	(None, 256)	0	dense_2[0][0]
dense_3 (Dense)	(None, 4)	1,028	dropout_7[0][0]

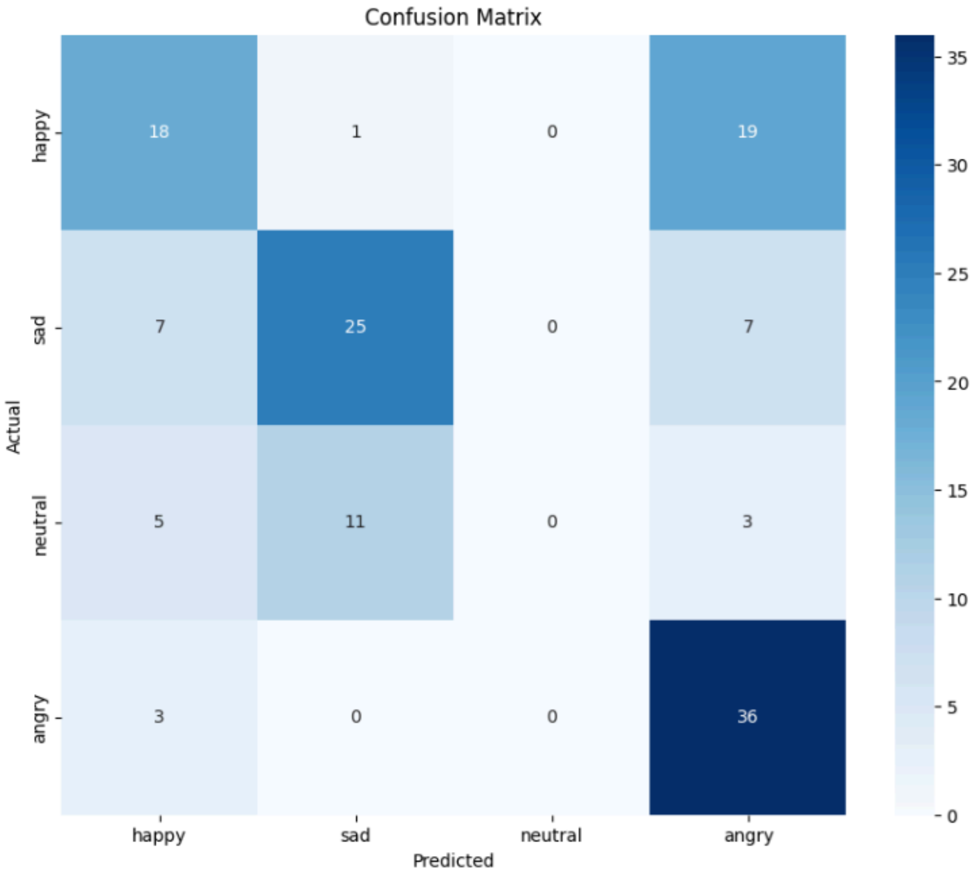
Total params: 3,105,604 (11.85 MB)
Trainable params: 3,104,260 (11.84 MB)
Non-trainable params: 1,344 (5.25 KB)

```
Epoch 32/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 82s 3s/step - accuracy: 0.7617 - loss: 0.5828 - val_accuracy: 0.5037 - val_loss: 1.1755 - learning_rate: 4.0000e-05
Epoch 33/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 46s 3s/step - accuracy: 0.7069 - loss: 0.6521 - val_accuracy: 0.5481 - val_loss: 1.0202 - learning_rate: 4.0000e-05
Test Accuracy: 58.52%
```




```
5/5 ━━━━━━━━━━━━━━━━━━━━ 3s 579ms/step
Classification Report:
              precision    recall  f1-score   support

       happy       0.55      0.47      0.51        38
         sad       0.68      0.64      0.66        39
     neutral       0.00      0.00      0.00        19
       angry       0.55      0.92      0.69        39

    accuracy                           0.59       135
   macro avg       0.44      0.51      0.46       135
weighted avg       0.51      0.59      0.53       135
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



Transformer based Model:
```
import os
import glob
import soundfile
import numpy as np
import librosa
import tensorflow as tf
from tensorflow.keras import layers, models, Input
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

########################################
# Configuration
########################################
DATA_PATH = "/content/drive/MyDrive/Speech Recognition -
Maha/Ravadess/Actor_*/*.wav"
N_MFCC = 40
MAX_FRAMES = 128
SAMPLE_RATE = 22050
AVAILABLE_EMOTIONS = {"angry", "sad", "neutral", "happy"}

label_map = {'happy':0, 'sad':1, 'neutral':2, 'angry':3}
int2emotion = {
    "01": "neutral",
    "02": "calm",
    "03": "happy",
    "04": "sad",
    "05": "angry",
    "06": "fearful",
    "07": "disgust",
    "08": "surprised"
}

########################################
# Feature Extraction as Sequence
########################################
def extract_mfcc_2d(file_path, n_mfcc=40, max_frames=128, sr=22050):
    try:
        with soundfile.SoundFile(file_path) as sf:
```

```
            audio = sf.read(dtype='float32')
            file_sr = sf.samplerate

        # If too short, skip
        if len(audio) < 2048:
            return None

        # Resample if needed
        if file_sr != sr:
            audio = librosa.resample(audio, orig_sr=file_sr,
target_sr=sr)

        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)  #
shape: [n_mfcc, frames]
        mfcc = mfcc.T  # now shape: [frames, n_mfcc]

        if mfcc.shape[0] == 0:
            return None

        # Pad or truncate
        if mfcc.shape[0] < max_frames:
            pad_width = max_frames - mfcc.shape[0]
            mfcc = np.pad(mfcc, ((0,pad_width),(0,0)), mode='constant')
        else:
            mfcc = mfcc[:max_frames, :]

        return mfcc
    except Exception as e:
        print(f"Error processing {file_path}: {e}")
        return None

def load_data(test_size=0.2):
    X, y = [], []
    for file in glob.glob(DATA_PATH):
        basename = os.path.basename(file)
        emotion_code = basename.split("-")[2]
        if emotion_code not in int2emotion:
            continue

        emotion = int2emotion[emotion_code]
        if emotion not in AVAILABLE_EMOTIONS:
            continue
```

```
        mfcc = extract_mfcc_2d(file_path=file, n_mfcc=N_MFCC,
max_frames=MAX_FRAMES, sr=SAMPLE_RATE)
        if mfcc is None:
            continue

        X.append(mfcc)
        y.append(label_map[emotion])

    X = np.array(X) # shape: [samples, MAX_FRAMES, N_MFCC]
    y = np.array(y)
    return train_test_split(X, y, test_size=test_size, random_state=42,
stratify=y)

#######################################
# Load Data
#######################################
X_train, X_test, y_train, y_test = load_data(test_size=0.2)
print("Training samples:", X_train.shape[0])
print("Testing samples:", X_test.shape[0])

num_classes = len(label_map)
y_train_oh = to_categorical(y_train, num_classes=num_classes)
y_test_oh = to_categorical(y_test, num_classes=num_classes)

#######################################
# Transformer Encoder Block
#######################################
def transformer_encoder(x, num_heads, ff_dim, dropout=0.1, d_model=64):
    attn_output = layers.MultiHeadAttention(num_heads=num_heads,
key_dim=d_model, dropout=dropout)(x, x)
    attn_output = layers.Dropout(dropout)(attn_output)
    x = layers.LayerNormalization(epsilon=1e-6)(x + attn_output)

    ffn = models.Sequential([
        layers.Dense(ff_dim, activation='relu'),
        layers.Dense(d_model)
    ])
    ffn_output = ffn(x)
    ffn_output = layers.Dropout(dropout)(ffn_output)
    x = layers.LayerNormalization(epsilon=1e-6)(x + ffn_output)
    return x

#######################################
```

```
# Build Transformer Model
#######################################
def build_transformer_model(sequence_length, feature_dim, num_classes,
d_model=64, num_heads=4, ff_dim=128, num_layers=2, dropout=0.1):
    inputs = Input(shape=(sequence_length, feature_dim))  # (batch,
MAX_FRAMES, N_MFCC)

    # Create a trainable positional embedding
    # Positions: [0 ... sequence_length-1]
    pos_embedding_layer = layers.Embedding(input_dim=sequence_length,
output_dim=d_model)
    positions = tf.range(start=0, limit=sequence_length, delta=1)
    pos_emb = pos_embedding_layer(positions)  # (sequence_length,
d_model)
    pos_emb = tf.expand_dims(pos_emb, axis=0)  # (1, sequence_length,
d_model)

    # Project input features to d_model dimension
    x = layers.Dense(d_model)(inputs)
    # Add positional encoding by broadcasting pos_emb to match batch
size at runtime.
    x = x + pos_emb

    # Stacking multiple transformer encoder layers
    for _ in range(num_layers):
        x = transformer_encoder(x, num_heads=num_heads, ff_dim=ff_dim,
dropout=dropout, d_model=d_model)

    # Global Average Pooling
    x = layers.GlobalAveragePooling1D()(x)
    x = layers.Dropout(dropout)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

model = build_transformer_model(sequence_length=MAX_FRAMES,
feature_dim=N_MFCC, num_classes=num_classes, d_model=64, num_heads=4,
ff_dim=128, num_layers=2, dropout=0.1)
model.summary()
```

```
########################################
# Train the Model
########################################
early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=5, min_lr=1e-5)

history = model.fit(
    X_train, y_train_oh,
    epochs=50,
    batch_size=32,
    validation_data=(X_test, y_test_oh),
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

########################################
# Evaluate the Model
########################################
loss, acc = model.evaluate(X_test, y_test_oh, verbose=0)
print(f"Test Accuracy: {acc*100:.2f}%")

# Plot training history
plt.figure(figsize=(14,5))
# Accuracy
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

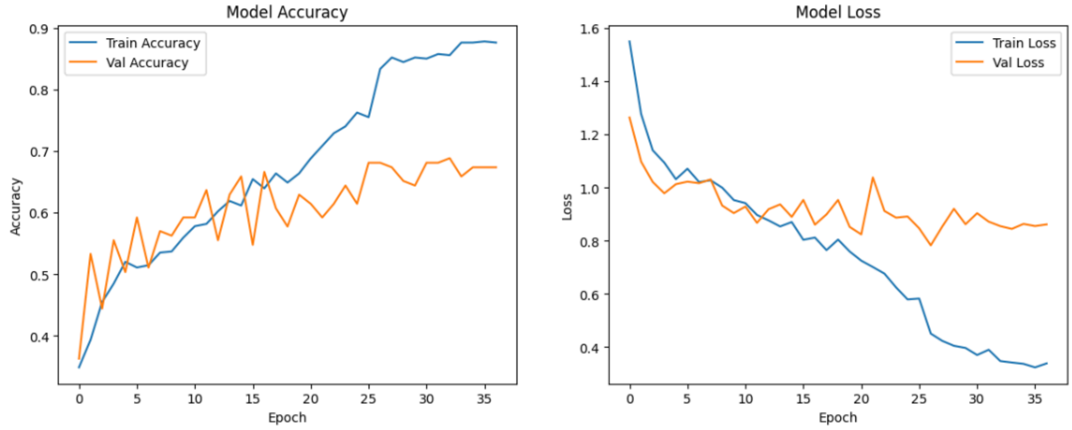
```
plt.show()

#######################################
# Classification Report
#######################################
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

reverse_label_map = {v:k for k,v in label_map.items()}
class_names = [reverse_label_map[i] for i in range(num_classes)]
print("Classification Report:")
print(classification_report(y_test, y_pred_classes,
target_names=class_names))

cm = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names,
            yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Training samples: 536
Testing samples: 135
Model: "functional_83"

Layer (type)	Output Shape	Param #	Connected to
input_layer_4 (InputLayer)	(None, 128, 40)	0	-
dense_4 (Dense)	(None, 128, 64)	2,624	input_layer_4[0][0]
add_3 (Add)	(None, 128, 64)	0	dense_4[0][0]
multi_head_attention (MultiHeadAttention)	(None, 128, 64)	66,368	add_3[0][0], add_3[0][0]
dropout_9 (Dropout)	(None, 128, 64)	0	multi_head_attention[…
add_4 (Add)	(None, 128, 64)	0	add_3[0][0], dropout_9[0][0]
layer_normalization (LayerNormalization)	(None, 128, 64)	128	add_4[0][0]
sequential_1 (Sequential)	(None, 128, 64)	16,576	layer_normalization[0…
dropout_10 (Dropout)	(None, 128, 64)	0	sequential_1[0][0]
add_5 (Add)	(None, 128, 64)	0	layer_normalization[0… dropout_10[0][0]
layer_normalization_1 (LayerNormalization)	(None, 128, 64)	128	add_5[0][0]
multi_head_attention_1 (MultiHeadAttention)	(None, 128, 64)	66,368	layer_normalization_1… layer_normalization_1…
dropout_12 (Dropout)	(None, 128, 64)	0	multi_head_attention_…
add_6 (Add)	(None, 128, 64)	0	layer_normalization_1… dropout_12[0][0]
layer_normalization_2 (LayerNormalization)	(None, 128, 64)	128	add_6[0][0]
sequential_2 (Sequential)	(None, 128, 64)	16,576	layer_normalization_2…
dropout_13 (Dropout)	(None, 128, 64)	0	sequential_2[0][0]
add_7 (Add)	(None, 128, 64)	0	layer_normalization_2… dropout_13[0][0]
layer_normalization_3 (LayerNormalization)	(None, 128, 64)	128	add_7[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 64)	0	layer_normalization_3…
dropout_14 (Dropout)	(None, 64)	0	global_average_poolin…
dense_9 (Dense)	(None, 4)	260	dropout_14[0][0]

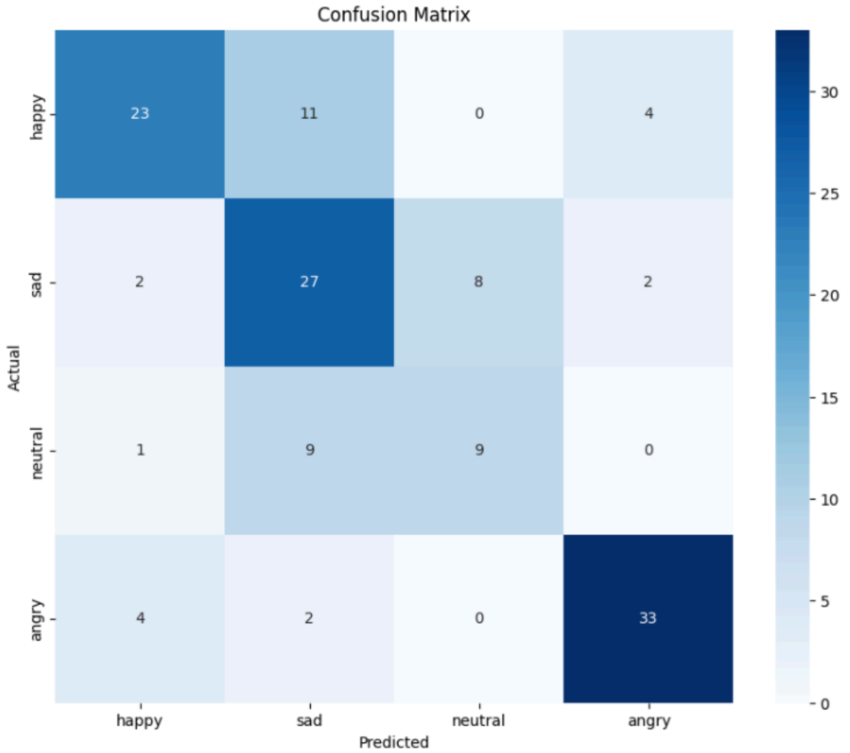
Total params: 169,284 (661.27 KB)
Trainable params: 169,284 (661.27 KB)
Non-trainable params: 0 (0.00 B)
Epoch 1/50

```
17/17 ━━━━━━━━━━━━━━━━━━━━ 12s 462ms/step - accuracy: 0.8795 - loss: 0.3287 - val_accuracy: 0.6741 - val_loss: 0.8560 - learning_rate: 4.0000e-05
Epoch 37/50
17/17 ━━━━━━━━━━━━━━━━━━━━ 8s 475ms/step - accuracy: 0.8866 - loss: 0.3413 - val_accuracy: 0.6741 - val_loss: 0.8621 - learning_rate: 4.0000e-05
Test Accuracy: 68.15%
```




```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 173ms/step
Classification Report:
              precision    recall  f1-score   support

       happy       0.77      0.61      0.68        38
         sad       0.55      0.69      0.61        39
     neutral       0.53      0.47      0.50        19
       angry       0.85      0.85      0.85        39

    accuracy                           0.68       135
   macro avg       0.67      0.65      0.66       135
weighted avg       0.69      0.68      0.68       135
```


weighted avg 0.69 0.68 0.68 135




```
[ ] # saving model
```