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,

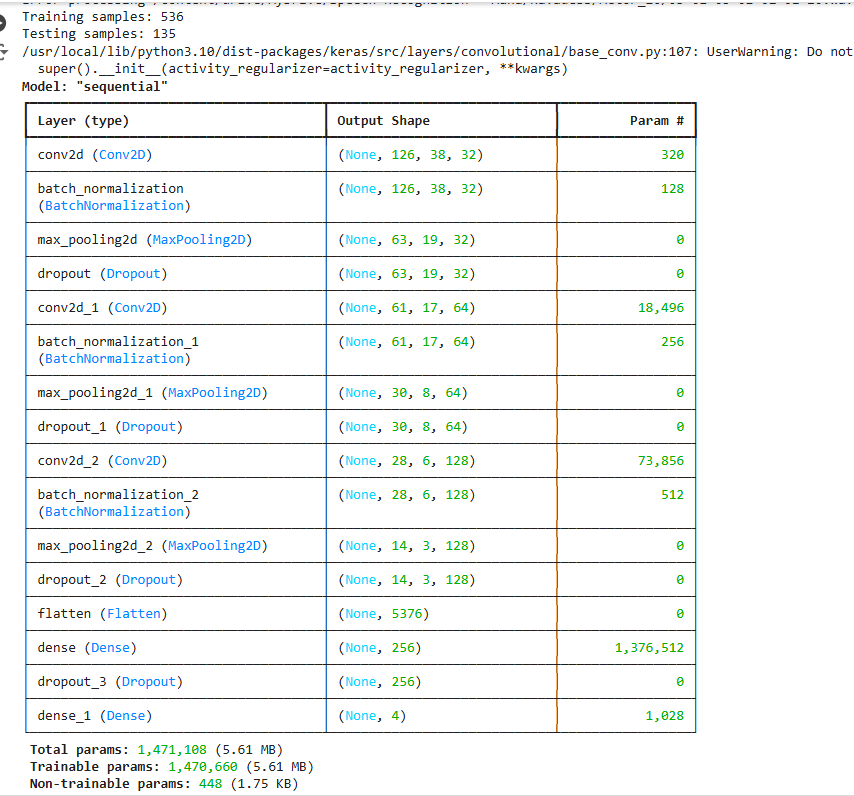
yticklabels=class\_names)

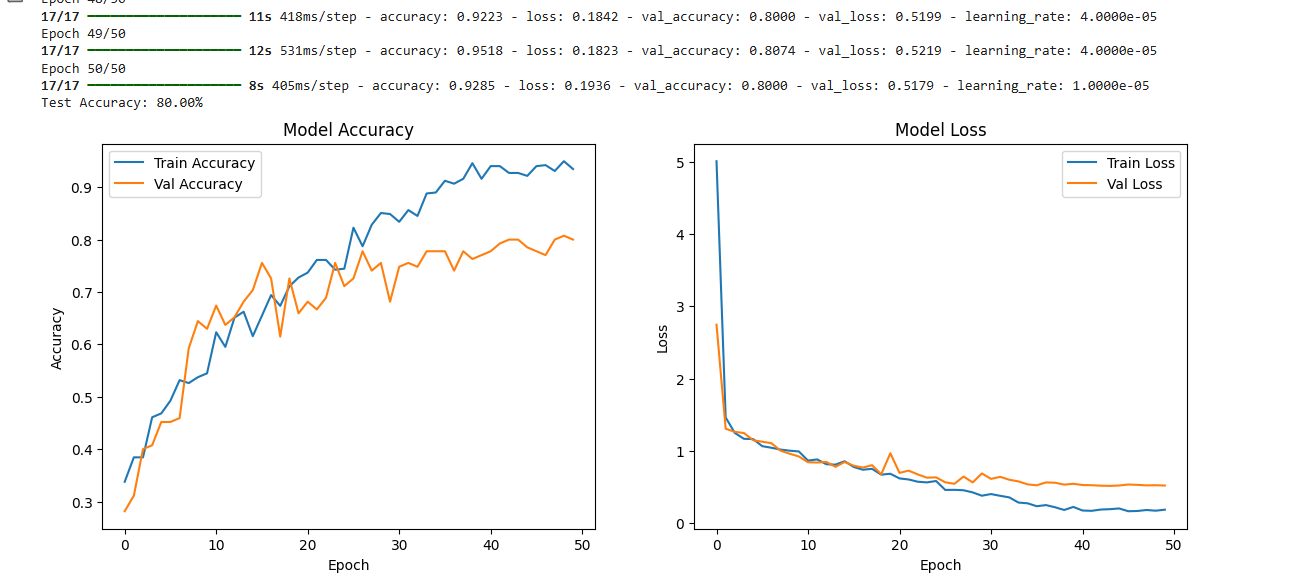
plt.title('Confusion Matrix')

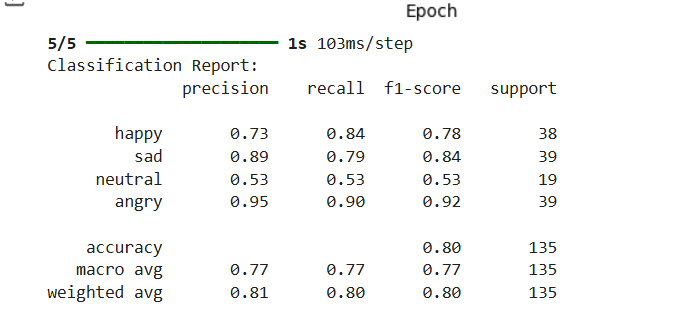
plt.xlabel('Predicted')

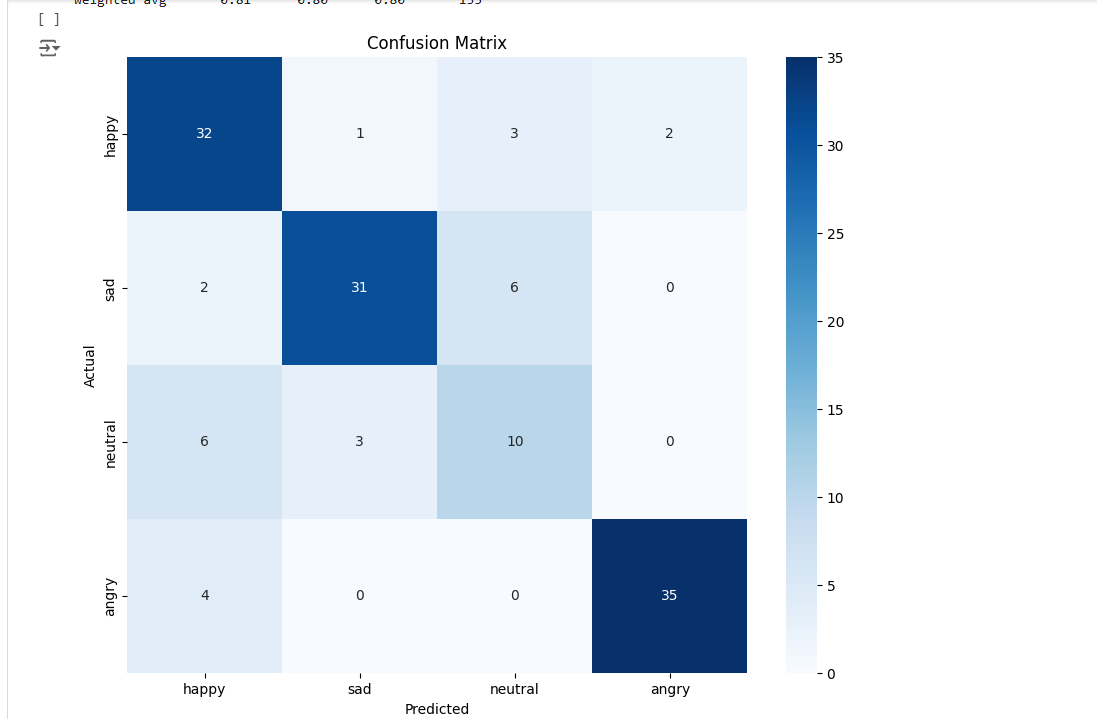
plt.ylabel('Actual')

plt.show()









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

#######################################

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,

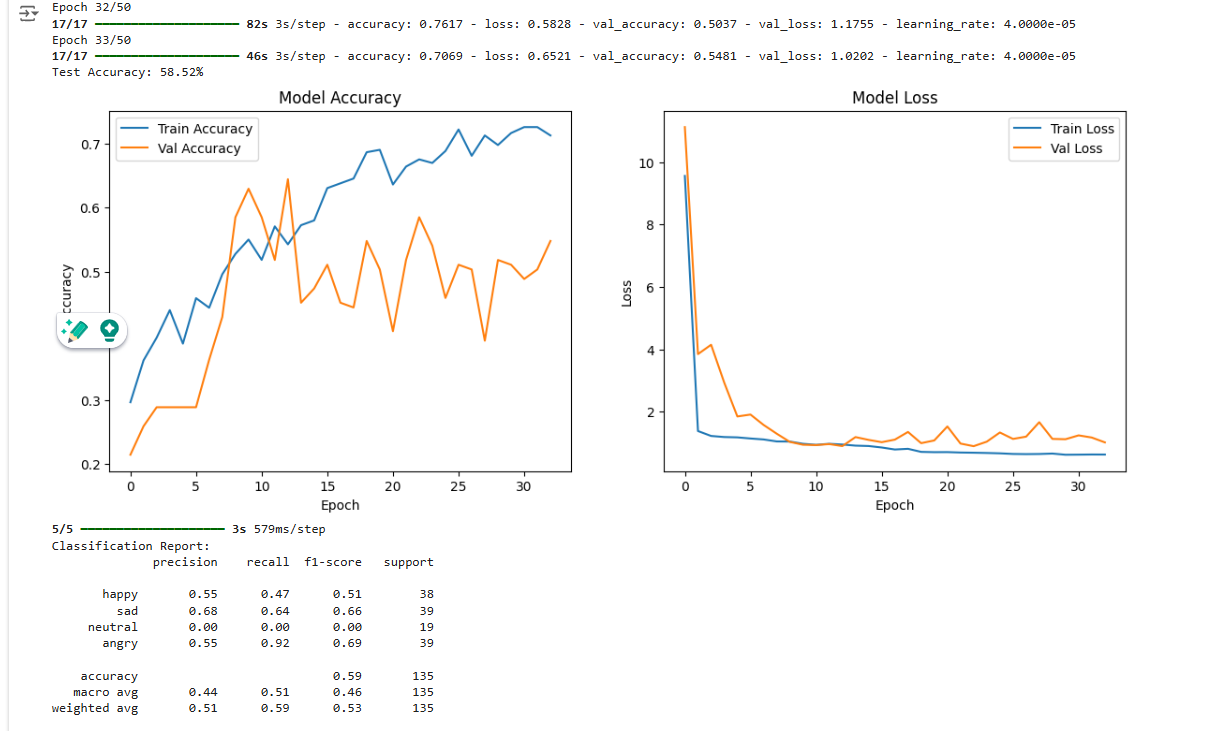
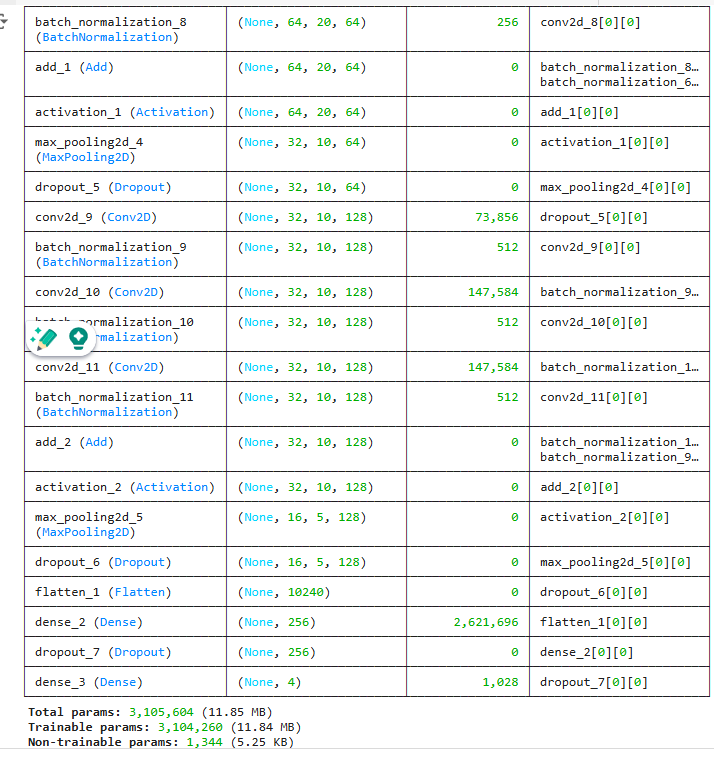
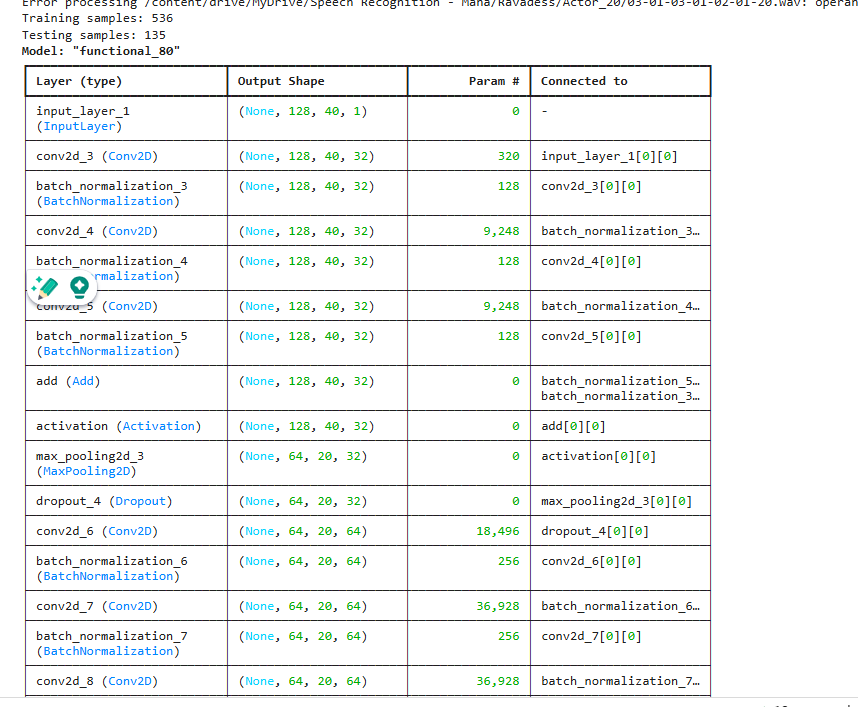
yticklabels=class\_names)

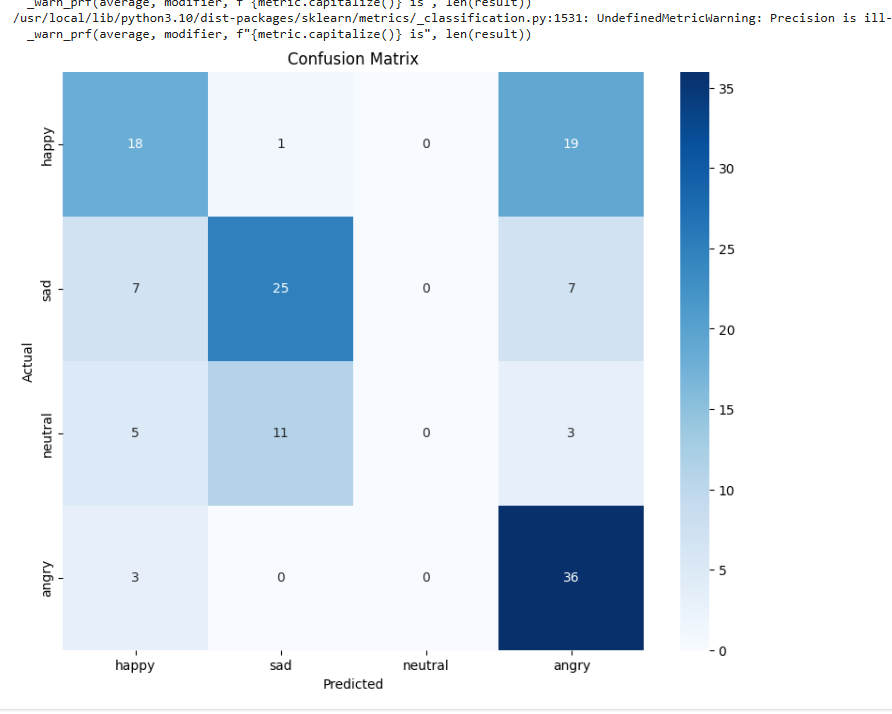
plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()





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()

