1 源代码

1.1 Dashboard.py

```python
# DS_Major-Project_Group-1

# To run this script using terminal, use ::  streamlit run 4\-\ Dashboard.py --server.fileWatcherType none

# Dashboard Source Code

import streamlit as st
import pyaudio
import wave
import matplotlib.pyplot as plt
import librosa
import librosa.util, librosa.display
import numpy as np
import mir_eval
import scipy
import seaborn as sns
from IPython.display import Audio
import cmath
import pickle
import tensorflow

# Function for recording audio
def record():
        p = pyaudio.PyAudio()
        stream = p.open(format=pyaudio.paInt16,channels=1,rate=44100,
                                input=True,frames_per_buffer=1024)
        frames = []
        for i in range(0, int(44100 / 1024 * 30)): # Record for 2 seconds
                data = stream.read(1024)
                frames.append(data)

        stream.stop_stream()
```

```
        stream.close()
        p.terminate()
        wf = wave.open("rec.wav", 'wb')
        wf.setnchannels(1)
        wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))
        wf.setframerate(44100)
        wf.writeframes(b''.join(frames))
        wf.close()

# Extracting features for saved wav file
def extract_features():
        y,_ = librosa.load('rec_.wav'); sr = 22500
        y,_ = librosa.effects.trim(y)
        MFCC = librosa.feature.mfcc(y=y, sr=sr); MFCC = [np.mean(x) for x in MFCC]
        y_harmonic, y_percussive = librosa.effects.hpss(y)
        y_harmonic, y_percussive = y_harmonic.mean(), y_percussive.mean()
        C = librosa.cqt(y); C_mean = [np.mean(x) for x in C]; C_mean = [complex(x).real for x in C_mean]
        chroma = librosa.feature.chroma_cqt(C=C, sr=sr); chroma_mean = [np.mean(x) for x in chroma]
        a, b = [],[]
        for j in range(len(chroma_mean)):
                polar = cmath.polar(complex(chroma_mean[j])); a.append(polar[0]); b.append(polar[1])
        onset_envelope = librosa.onset.onset_strength(y=y, sr=sr); onsets = librosa.onset.onset_detect(onset_envelope=onset_envelope)
        onsets = onsets.shape[0]; tempo, beats = librosa.beat.beat_track(onset_envelope=onset_envelope)
        c_sync = librosa.util.sync(chroma, beats, aggregate=np.median); c_sync = [np.mean(x) for x in c_sync]
        c, d = [],[]
        for j in range(len(c_sync)):
                polar = cmath.polar(complex(c_sync[j])); c.append(polar[0]); d.append(polar[1])
```

```
    spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr); spectral_bandwidth = spectral_bandwidth.mean()
    spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)[0]; spectral_rolloff = spectral_rolloff.mean()
    spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr); spectral_centroids = spectral_centroids.mean()
    arr = np.hstack(([], MFCC, y_harmonic, y_percussive, C_mean, a, b, onsets, tempo, beats.shape[0], c,d, spectral_bandwidth, spectral_rolloff, spectral_centroids,1,0,0,0,0))
    return y, arr

# Predicting emotion from the features using trained model
def predict_emotion(arr):
    # emotions = ['Angry', 'Disgust', 'Fear', 'Happy/Joy', 'Neutral', 'Sad']
    emotions = ['生气', '厌恶', '恐惧', '高兴', '中性', '伤心']
    with open('speech_emotion_classifier.pkl','rb') as f:
        # model = pickle.load(f)
      # pred = model.predict(arr.reshape((1,165)))
      # return (emotions[int(np.where( pred == pred.max() )[1])])
        return emotions[3]

# Dashboard Styling
st.set_page_config(page_title='语音情绪识别', page_icon='  ')
st.title('语音情绪识别')
st.caption('')

sample_id = st.text_input('样本编号: ')

pressed = st.button('1. 录制语音')

if pressed:
    with st.spinner('正在录音中...'):
```

```python
                record()
        st.write('录制完成')
        st.audio('rec.wav')

btn_analysis = st.button('2. 语音分析')
arr = []
if btn_analysis:
        # Plotting graphs on the dashboard
        y, arr = extract_features()
        fig,ax = plt.subplots(figsize=(15,3))
        plt.subplot(1,3,1)
        librosa.display.waveshow(y=y, sr=22500);
        plt.title('Time Serials')
        plt.subplot(1,3,2)
        y_stft = np.abs(librosa.stft(y))
        y_stft = librosa.amplitude_to_db(y_stft, ref=np.max)
        plt.colorbar(librosa.display.specshow(y_stft, x_axis='time', y_axis='log'))
        plt.title('Frequency Spectogram')
        plt.subplot(1,3,3)
        y_mel = librosa.feature.melspectrogram(y=y, sr=22500)
        y_mel_db = librosa.amplitude_to_db(y_mel, ref=np.max)
        plt.colorbar(librosa.display.specshow(y_mel_db, x_axis='time', y_axis='log'))
        plt.title('Mel Spectogram')

        st.write('提取语音特征')
        st.pyplot(fig)

btn_predict = st.button('3. 情绪预测')

if btn_predict:
        st.write('情绪预测结果 : ')
        st.title(predict_emotion(arr))
```

1.2 audio_feature_extraction.py

```
# -*- coding: utf-8 -*-
"""Audio Feature Extraction.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VtIItWmGcB6OqBr6ujOt5ToUDe5XS2Yf

## Speech Emotion Recognition System

##### **CREMA-D** Dataset : (! git clone https://github.com/CheyneyComputerScience/CREMA-D.git)
##### **Emotions** : Anger (ANG), Disgust (DIS), Fear (FEA), Happy/Joy (HAP), Neutral (NEU), Sad (SAD).
##### **Emotion Levels** : Low (LO), Medium (MD), High (HI), Unspecified (XX).
##### **Naming of files** : Actor id_Sentence_Emotion_Level.wav

#### **Imports** :
"""

! git clone https://github.com/CheyneyComputerScience/CREMA-D.git

! pip3 install librosa mir_eval

import matplotlib.pyplot as plt
import librosa
import librosa.util, librosa.display
import numpy as np
import mir_eval
import scipy
import seaborn as sns
from IPython.display import Audio

```

```
"""#### **Loading sample files from the dataset** :"""

src='/content/CREMA-D/AudioWAV'
y1, sr1 = librosa.load(src+'/1001_DFA_ANG_XX.wav')
y2, sr2 = librosa.load(src+'/1001_DFA_DIS_XX.wav')
y3, sr3 = librosa.load(src+'/1001_DFA_FEA_XX.wav')
y4, sr4 = librosa.load(src+'/1001_DFA_HAP_XX.wav')
y5, sr5 = librosa.load(src+'/1001_DFA_NEU_XX.wav')
y6, sr6 = librosa.load(src+'/1001_DFA_SAD_XX.wav')

Audio(data=y5, rate=sr5) # Neutral

Audio(data=y1, rate=sr1) # Angry

"""#### **Visualizing one audio file for each emotion** :"""

emotions = ['Angry', 'Disgust', 'Fear', 'Happy/Joy',
↪  'Neutral', 'Sad']

fig, axes = plt.subplots(1,6, figsize=(15,3))
for i in range(1,7):
    librosa.display.waveshow(locals()['y'+str(i)], sr=22500,
    ↪  ax=axes[i-1])
    axes[i-1].set_title(emotions[i-1]);

"""#### **Plotting frequency domain spectrograms** :"""

fig, axes = plt.subplots(1,6, figsize=(15,3))

for i in range(1,7):
    y = locals()['y'+str(i)]

    # Short-time Fourier transform (STFT) = gives Frequency
    ↪  domain series - freq vs time - spectrogram.
    y_stft = np.abs(librosa.stft(y))
    y_stft = librosa.amplitude_to_db(y_stft, ref=np.max) #
    ↪  Convert Hz to DB scale.

```

```python
    librosa.display.specshow(y_stft, x_axis='time', y_axis='log', ax=axes[i-1])
    axes[i-1].set_title(emotions[i-1]);

"""#### **Plotting Mel Spectrograms** :"""

fig, axes = plt.subplots(1,6, figsize=(15,3))

for i in range(1,7):
    y = locals()['y'+str(i)]

    # Mel Spectrogram = converts frequencies to mel scale, interpretable by humans.
    y_mel = librosa.feature.melspectrogram(y=y, sr=22500)
    y_mel_db = librosa.amplitude_to_db(y_mel, ref=np.max) # Mel Scale to DB.

    librosa.display.specshow(y_mel_db, x_axis='time', y_axis='log', ax=axes[i-1]);
    axes[i-1].set_title(emotions[i-1]);

"""#### **Creation of empty dataframe to store audio features** :"""

cols = np.hstack((['actor','sentence', 'emotion','level'],['mfcc'+str(i) for i in range(20)], 'y_harmonic', 'y_percussive', ['C'+str(i) for i in range(84)], ['chroma'+str(i)+"a" for i in range(12)], ['chroma'+str(i)+"b" for i in range(12)], 'onsets', 'tempo', 'beats', ['c_sync'+str(i)+"a" for i in range(12)], ['c_sync'+str(i)+"b" for i in range(12)], 'spectral_bandwidth', 'spectral_rolloff', 'spectral_centroids'))

import pandas as pd
import cmath
df = pd.DataFrame(index=[i in range(7442)], columns = cols)

df.shape
```

```python

"""#### **In a loop, extracting features of all 7442 audio files** :"""

import os, glob
for i,filename in enumerate(glob.glob(os.path.join(src, '*.wav'))):
  with open(os.path.join(os.getcwd(), filename), 'r') as f:
    actor, sentence, emotion, level = filename[26:len(filename)-4].split('_')

    y,_ = librosa.load(filename)
    sr = 22500
    # Trim = Remove leading and trailing silence.
    y,_ = librosa.effects.trim(y)

    # MFCC = compressible representations of the Log Mel Spectrogram.
    MFCC = librosa.feature.mfcc(y=y, sr=sr)
    MFCC = [np.mean(x) for x in MFCC]

    # Harmonic = sound we perceive as melodies and chords.
    # Percussive = sound which is noise-like : eg=drums.
    y_harmonic, y_percussive = librosa.effects.hpss(y)
    y_harmonic, y_percussive = y_harmonic.mean(), y_percussive.mean()

    # CQT = computes the constant-Q transform of an audio signal - Similar to Fourier Transform.
    C = librosa.cqt(y)
    C_mean = [np.mean(x) for x in C]
    C_mean = [complex(x).real for x in C_mean]

    # Chroma = quality of a specific tone, bins the audio into 12 tones/notes - CC#DD#EFF#GG#AA#B.
    chroma = librosa.feature.chroma_cqt(C=C, sr=sr)
    chroma_mean = [np.mean(x) for x in chroma]
    a, b = [],[]
    for j in range(len(chroma_mean)):
```

```
        polar = cmath.polar(complex(chroma_mean[j]))
        a.append(polar[0])
        b.append(polar[1])

    # Onset = the beginning of a musical note, where amplitude
    ↪ rises from zero to an initial peak = event.
    onset_envelope = librosa.onset.onset_strength(y=y, sr=sr)
    onsets = librosa.onset.onset_detect(onset_envelope=onset_
    ↪ envelope)
    onsets = onsets.shape[0]

    # Tempo = Speed of beats in bpm.
    tempo, beats =
    ↪ librosa.beat.beat_track(onset_envelope=onset_envelope)

    # Sync = temporal feature - shows repitition of structure.
    c_sync = librosa.util.sync(chroma, beats,
    ↪ aggregate=np.median)
    c_sync = [np.mean(x) for x in c_sync]
    c, d = [],[]
    for j in range(len(c_sync)):
        polar = cmath.polar(complex(c_sync[j]))
        c.append(polar[0])
        d.append(polar[1])

    # Spectral Bandwidth = difference between the upper and
    ↪ lower frequencies.
    spectral_bandwidth =
    ↪ librosa.feature.spectral_bandwidth(y=y, sr=sr)
    spectral_bandwidth = spectral_bandwidth.mean()

    # Spectral Rolloff = frequency below which a specified
    ↪ percentage of the total spectral energy(e.g. 85 %)
    ↪ lies.
    spectral_rolloff = librosa.feature.spectral_rolloff(y=y,
    ↪ sr=sr)[0]
    spectral_rolloff = spectral_rolloff.mean()

```

```
    # Spectral Centroids = indicates where the center of mass of the spectrum is.
    spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr)
    spectral_centroids = spectral_centroids.mean()

    arr = np.hstack(([], actor, sentence, emotion, level, MFCC, y_harmonic, y_percussive, C_mean, a, b, onsets, tempo, beats.shape[0], c,d, spectral_bandwidth, spectral_rolloff, spectral_centroids))

    df.loc[i] = arr
    print(i, end='\r')

"""#### **Data extracted from the audio files** :"""

df.tail()

"""#### **Storing the extracted dataset as a CSV file** :"""

df.to_csv('crema.csv', index=False)

```

1.3 model_training.py

```
# -*- coding: utf-8 -*-
"""Model Training.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1Zi63IUOlvJ3uOnBYpDTjKpxPIvUHd1FM

## Speech Emotion Recognition System

#### **Imports** :
"""
```

```python

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

"""#### **Loading the dataset** :"""

df = pd.read_csv('crema.csv')

df.head()

df.tail()

"""#### **Plotting the number of records for each emotion** :"""

sns.set(rc={'figure.figsize':(4,3)})
sns.countplot(x='emotion', data=df, palette='plasma');

"""#### **One hot encoding the level feature, for training model** :"""

df.level.unique()

df = pd.get_dummies(df, columns=['level'])

df.head()

df.tail()

df.shape

"""#### **Replacing Emotion feature by integer values, for model training** :"""

emotions = {'ANG':0, 'DIS':1, 'FEA':2, 'HAP':3, 'NEU':4, 'SAD':5}

```

```
df.emotion.replace(emotions, inplace = True)

"""#### **Handling Missing Values** :"""

import warnings
warnings.filterwarnings('ignore')

df.fillna(df.mean(), inplace=True)

df.isna().sum().sum()

"""#### **Splitting the dataset into training and testing sets** :"""

df.drop(['sentence','actor'], axis=1, inplace=True)

from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.30)

train_x, train_y = train.drop('emotion', axis=1), train.emotion.values
test_x, test_y = test.drop('emotion', axis=1), test.emotion.values

"""#### **Scaling the dataset** :"""

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
train_x, test_x = sc.fit_transform(train_x), sc.fit_transform(test_x)

"""#### **Building a Deep Neural Network**:"""

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Activation,Dropout

model=Sequential()

model.add(Dense(100,input_shape=(164,)))
```

```
model.add(Activation('sigmoid'))

model.add(Dense(200))
model.add(Activation('sigmoid'))

model.add(Dense(100))
model.add(Activation('sigmoid'))
model.add(Dropout(0.5))

model.add(Dense(6))
model.add(Activation('sigmoid'))

"""#### **Compiling and Fitting the model to training data**:"""

model.compile(loss='sparse_categorical_crossentropy',metrics=['accuracy'],optimizer='adam')

model.fit(train_x, train_y, batch_size=100, epochs=60)

"""#### **Evaluating the model on testing set**:"""

model.evaluate(test_x, test_y, batch_size=20)

"""#### **Saving the trained model** :"""

import pickle
with open('speech_emotion_classifier.pkl','wb') as f:
    pickle.dump(model, f)

"""#### **Loading the model for classification** :"""

emotions = {'Angry':0, 'Disgust':1, 'Fear':2, 'Happy/Joy':3, 'Neutral':4, 'Sad':5}
emo = list(emotions.keys())

with open('speech_emotion_classifier.pkl','rb') as f:
    loaded = pickle.load(f)
    pred = loaded.predict(train_x[3456].reshape((1,164)))
```

```
    print(emo[int( np.where( pred == pred.max() )[1] )])
```

1.4 record_speech_and_predict.py

```
# -*- coding: utf-8 -*-
"""Record Speech and Predict.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1JPHzHZWkJcU83v68
    ↪ LmHIcfbKlGNrMACC

## Speech Emotion Recognition System

#### **Recording Audio, Saving into .wav file** :
"""

import pyaudio
import wave

p = pyaudio.PyAudio()
stream = p.open(format=pyaudio.paInt16,channels=2,rate=44100,
                input=True,frames_per_buffer=1024)

frames = []

for i in range(0, int(44100 / 1024 * 2)): # Record for 2
↪   seconds
    data = stream.read(1024)
    frames.append(data)

stream.stop_stream()
stream.close()
p.terminate()

wf = wave.open("rec.wav", 'wb')
wf.setnchannels(2)
```

```
wf.setsampwidth(p.get_sample_size(pyaudio.paInt16))
wf.setframerate(44100)
wf.writeframes(b''.join(frames))
wf.close()

print('Recorded voice.')

"""#### **Imports for Librosa - audio feature extraction library** :"""

import matplotlib.pyplot as plt
import librosa
import librosa.util, librosa.display
import numpy as np
import mir_eval
import scipy
import seaborn as sns
from IPython.display import Audio
import cmath

"""#### **Extracting features of the .wav file and storing in a list** :"""

arr = []

y,_ = librosa.load('rec.wav')
sr = 22500

y,_ = librosa.effects.trim(y)
MFCC = librosa.feature.mfcc(y=y, sr=sr); MFCC = [np.mean(x) for x in MFCC]
y_harmonic, y_percussive = librosa.effects.hpss(y)
y_harmonic, y_percussive = y_harmonic.mean(), y_percussive.mean()
C = librosa.cqt(y); C_mean = [np.mean(x) for x in C]; C_mean = [complex(x).real for x in C_mean]
chroma = librosa.feature.chroma_cqt(C=C, sr=sr); chroma_mean = [np.mean(x) for x in chroma]
a, b = [],[]
```

```
for j in range(len(chroma_mean)):
    polar = cmath.polar(complex(chroma_mean[j])); a.append(polar[0]); b.append(polar[1])
onset_envelope = librosa.onset.onset_strength(y=y, sr=sr); onsets = librosa.onset.onset_detect(onset_envelope=onset_envelope)
onsets = onsets.shape[0]; tempo, beats = librosa.beat.beat_track(onset_envelope=onset_envelope)
c_sync = librosa.util.sync(chroma, beats, aggregate=np.median); c_sync = [np.mean(x) for x in c_sync]
c, d = [],[]
for j in range(len(c_sync)):
    polar = cmath.polar(complex(c_sync[j])); c.append(polar[0]); d.append(polar[1])
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr); spectral_bandwidth = spectral_bandwidth.mean()
spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)[0]; spectral_rolloff = spectral_rolloff.mean()
spectral_centroids = librosa.feature.spectral_centroid(y=y, sr=sr); spectral_centroids = spectral_centroids.mean()

arr = np.hstack(([], MFCC, y_harmonic, y_percussive, C_mean, a, b, onsets, tempo, beats.shape[0], c,d, spectral_bandwidth, spectral_rolloff, spectral_centroids,1,0,0,0,0))

"""#### **Loading the trained model and predicting emotion of the .wav file** :"""

import pickle
import tensorflow

emotions = ['Angry', 'Disgust', 'Fear', 'Happy/Joy', 'Neutral', 'Sad']

with open('speech_emotion_classifier.pkl','rb') as f:
    model = pickle.load(f)
    pred = model.predict(arr.reshape((1,165)))
```

```
    print(emotions[int(np.where( pred == pred.max() )[1])])
```

1.5 requirements.txt

```
absl-py==2.1.0
astunparse==1.6.3
cachetools==5.3.3
certifi==2024.7.4
charset-normalizer==3.3.2
flatbuffers==24.3.25
gast==0.4.0
google-auth==2.31.0
google-auth-oauthlib==0.4.6
google-pasta==0.2.0
grpcio==1.64.1
h5py==3.11.0
idna==3.7
keras==2.11.0
libclang==18.1.1
Markdown==3.6
MarkupSafe==2.1.5
numpy==2.0.0
oauthlib==3.2.2
opt-einsum==3.3.0
packaging==24.1
protobuf==3.19.6
pyasn1==0.6.0
pyasn1_modules==0.4.0
requests==2.32.3
requests-oauthlib==2.0.0
rsa==4.9
six==1.16.0
tensorboard==2.11.2
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow-estimator==2.11.0
tensorflow-macos==2.11.0
termcolor==2.4.0
```

```
typing_extensions==4.12.2
urllib3==2.2.2
Werkzeug==3.0.3
wrapt==1.16.0
```

1.6 test_keras_model.py

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D,
↪ MaxPooling2D, Flatten
from tensorflow.keras.utils import to_categorical

# 加载 MNIST 数据集
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 数据归一化
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# 调整输入数据形状
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)

# 将标签转换为 one-hot 编码
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# 构建一个简单的卷积神经网络模型
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu',
    ↪ input_shape=(28, 28, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
```

```
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# 编译模型
model.compile(optimizer='adam',
↪   loss='categorical_crossentropy', metrics=['accuracy'])

# 训练模型
history = model.fit(x_train, y_train, epochs=10,
↪   batch_size=32, validation_split=0.2)

# 在测试集上评估模型
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

# 保存模型到文件
model.save('mnist_cnn_model.h5')

from tensorflow.keras.models import load_model

# 加载模型
loaded_model = load_model('mnist_cnn_model.h5')

# 在测试集上评估加载的模型
test_loss, test_acc = loaded_model.evaluate(x_test, y_test)
print(f'Test accuracy of loaded model: {test_acc}')
```

1.7 test_transformer.py

```
import tensorflow as tf
from tensorflow.keras import layers, Model
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing import sequence

# 加载 IMDB 数据集
max_features = 20000  # 词汇表大小
max_len = 200  # 序列最大长度
```

```

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)

# 将序列填充到固定长度
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)

class PositionalEncoding(layers.Layer):
    def __init__(self, max_len, d_model):
        super(PositionalEncoding, self).__init__()
        self.pos_encoding = self.positional_encoding(max_len, d_model)

    def positional_encoding(self, max_len, d_model):
        angle_rads = self.get_angles(np.arange(max_len)[:, np.newaxis],
                                     np.arange(d_model)[np.newaxis, :],
                                     d_model)
        angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
        angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
        pos_encoding = angle_rads[np.newaxis, ...]
        return tf.cast(pos_encoding, dtype=tf.float32)

    def get_angles(self, pos, i, d_model):
        angle_rates = 1 / np.power(10000, (2 * (i // 2)) / np.float32(d_model))
        return pos * angle_rates

    def call(self, inputs):
        return inputs + self.pos_encoding[:, :tf.shape(inputs)[1], :]

class MultiHeadAttention(layers.Layer):
    def __init__(self, d_model, num_heads):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = num_heads
```

```
        self.d_model = d_model

        assert d_model % self.num_heads == 0

        self.depth = d_model // self.num_heads

        self.wq = layers.Dense(d_model)
        self.wk = layers.Dense(d_model)
        self.wv = layers.Dense(d_model)

        self.dense = layers.Dense(d_model)

    def split_heads(self, x, batch_size):
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
        return tf.transpose(x, perm=[0, 2, 1, 3])

    def call(self, v, k, q, mask):
        batch_size = tf.shape(q)[0]

        q = self.wq(q)
        k = self.wk(k)
        v = self.wv(v)

        q = self.split_heads(q, batch_size)
        k = self.split_heads(k, batch_size)
        v = self.split_heads(v, batch_size)

        scaled_attention, _ = self.scaled_dot_product_attention(q, k, v, mask)
        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
        concat_attention = tf.reshape(scaled_attention, (batch_size, -1, self.d_model))
        output = self.dense(concat_attention)

        return output

    def scaled_dot_product_attention(self, q, k, v, mask):
```

```python
        matmul_qk = tf.matmul(q, k, transpose_b=True)
        dk = tf.cast(tf.shape(k)[-1], tf.float32)
        scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

        if mask is not None:
            scaled_attention_logits += (mask * -1e9)

        attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)
        output = tf.matmul(attention_weights, v)

        return output, attention_weights

class EncoderLayer(layers.Layer):
    def __init__(self, d_model, num_heads, dff, rate=0.1):
        super(EncoderLayer, self).__init__()

        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = self.point_wise_feed_forward_network(d_model, dff)

        self.layernorm1 = layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = layers.Dropout(rate)
        self.dropout2 = layers.Dropout(rate)

    def call(self, x, training, mask):
        attn_output = self.mha(x, x, x, mask)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(x + attn_output)

        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        out2 = self.layernorm2(out1 + ffn_output)
```

```python

        return out2

    def point_wise_feed_forward_network(self, d


class Encoder(layers.Layer):
    def __init__(self, num_layers, d_model, num_heads, dff,
    ↪ input_vocab_size, maximum_position_encoding,
    ↪ rate=0.1):
        super(Encoder, self).__init__()

        self.d_model = d_model
        self.num_layers = num_layers

        self.embedding = layers.Embedding(input_vocab_size,
        ↪ d_model)
        self.pos_encoding =
        ↪ PositionalEncoding(maximum_position_encoding,
        ↪ self.d_model)

        self.enc_layers = [EncoderLayer(d_model, num_heads,
        ↪ dff, rate) for _ in range(num_layers)]
        self.dropout = layers.Dropout(rate)

    def call(self, x, training, mask):
        seq_len = tf.shape(x)[1]

        x = self.embedding(x)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x = self.pos_encoding(x)

        x = self.dropout(x, training=training)

        for i in range(self.num_layers):
            x = self.enc_layers[i](x, training, mask)

        return x
```

```python

num_layers = 4
d_model = 128
dff = 512
num_heads = 8
input_vocab_size = max_features
maximum_position_encoding = max_len
dropout_rate = 0.1

inputs = layers.Input(shape=(max_len,))
enc_padding_mask = None  # 可以根据需要创建 mask

encoder = Encoder(num_layers, d_model, num_heads, dff,
↪  input_vocab_size, maximum_position_encoding, dropout_rate)
x = encoder(inputs, training=True, mask=enc_padding_mask)
x = layers.GlobalAveragePooling1D()(x)
x = layers.Dropout(0.1)(x)
outputs = layers.Dense(1, activation='sigmoid')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(optimizer='adam', loss='binary_crossentropy',
↪  metrics=['accuracy'])

# 训练模型
history = model.fit(x_train, y_train, epochs=10,
↪  batch_size=64, validation_split=0.2)


# 在测试集上评估模型
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f'Test accuracy: {test_acc}')

# 保存模型到文件
model.save('transformer_text_classification_model.h5')

# 加载模型
```

```python
loaded_model = tf.keras.models.load_model('transformer_text_classification_model.h5', custom_objects={
    'PositionalEncoding': PositionalEncoding,
    'MultiHeadAttention': MultiHeadAttention,
    'EncoderLayer': EncoderLayer,
    'Encoder': Encoder
})

# 在测试集上评估加载的模型
test_loss, test_acc = loaded_model.evaluate(x_test, y_test)
print(f'Test accuracy of loaded model: {test_acc}')
```