```
import numpy as np
import streamlit as st
import pandas as pd
import cv2
from collections import Counter
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
# Read csv file containing data.
df = pd.read_csv('muse_v3.csv')
# Renaming column of dataframe.
df['link']  = df['lastfm_url']
df['name'] = df['track']
df['emotional']  =  df['number_of_emotion_tags']
df['pleasant'] = df['valence_tags']
# Taking out useful column.
df = df[['name','emotional','pleasant','link','artist']]
# Sort column based on emotional & pleasant column value.
# Pleasant = degree of pleasant in that particular song.
# Emotional = emotional word used in that song.
df = df.sort_values(by=["emotional", "pleasant"])
df.reset_index()
# Diving dataframe based on emotional & pleasant value in sorted order.
df_sad = df[:18000]
df_fear = df[18000:36000]
```

```
df_angry = df[36000:54000]
df_neutral = df[54000:72000]
df_happy = df[72000:]
# Task of function 'fun' is to take list of unique emotions & return dataframe of 30 rows.
def fun(list):
 # Creating Empty Dataframe
 data = pd.DataFrame()
# If list of emotion's contain only 1 emotion
 if len(list) == 1:
  # Emotion name
  v = list[0]
  # Number of rows for this emotion
  t = 30
  if v == 'Neutral':
   # Adding  rows  to  data
   data  = data.append(df_neutral.sample(n=t))
  elif v == 'Angry': # Adding rows to data
   data = data.append(df_angry.sample(n=t))
  elif v == 'fear':
   # Adding  rows  to  data
   data  = data.append(df_fear.sample(n=t))
  elif v == 'happy':
   # Adding  rows  to  data
   data  = data.append(df_happy.sample(n=t))
  else:
   # Adding rows to data
```

```
  data = data.append(df_sad.sample(n=t))
elif len(list) == 2:
 # Row's count per emotion
 times  = [20,10]
 for i in range(len(list)):
  # Emotion name
  v = list[i]
  # Number of rows for this emotion
  t = times[i]
  if v == 'Neutral':
   # Adding  rows  to  data
   data = data.append(df_neutral.sample(n=t))
  elif v == 'Angry':
   # Adding rows to data
   data = data.append(df_angry.sample(n=t))
  elif v == 'fear':
   # Adding  rows  to  data
   data = data.append(df_fear.sample(n=t))
  elif v == 'happy':
   # Adding  rows  to  data
   data = data.append(df_happy.sample(n=t))
  else:
   # Adding rows to data
   data = data.append(df_sad.sample(n=t))
elif len(list) == 3:
 # Row's count per emotion
```

```
 times = [15,10,5]
 for i in range(len(list)):
  # Emotion name
  v = list[i]
  # Number of rows for this emotion
  t = times[i]
  if v == 'Neutral':
   # Adding  rows  to  data
   data = data.append(df_neutral.sample(n=t))
  elif v == 'Angry':
   # Adding rows to data
   data = data.append(df_angry.sample(n=t))
  elif v == 'fear':
   # Adding  rows  to  data
   data  = data.append(df_fear.sample(n=t))
  elif v == 'happy':
   # Adding  rows  to  data
   data = data.append(df_happy.sample(n=t))
  else:
   # Adding rows to data
   data = data.append(df_sad.sample(n=t))
 elif len(list) == 4:
  # Row's count per emotion
  times = [10,9,8,3]
  for i in range(len(list)):
   # Emotion name
```

```
 v = list[i]
 # Number of rows for this emotion
 t = times[i]
 if v == 'Neutral':
  # Adding  rows  to  data
  data = data.append(df_neutral.sample(n=t))
 elif v == 'Angry':
  # Adding rows to data
  data = data.append(df_angry.sample(n=t))
 elif v == 'fear':
  # Adding  rows  to  data
  data = data.append(df_fear.sample(n=t))
 elif v == 'happy':
  # Adding  rows  to  data
  data = data.append(df_happy.sample(n=t))
 else:
  # Adding rows to data
  data = data.append(df_sad.sample(n=t))
else:
 # Row's count per emotion
 times = [10,7,6,5,2]
 for i in range(len(list)):
  # Emotion name
  v = list[i]
  # Number of rows for this emotion
  t = times[i]
```

```
 if v == 'Neutral':
  # Adding  rows  to  data
  data = data.append(df_neutral.sample(n=t))
 elif v == 'Angry':
  # Adding rows to data
  data = data.append(df_angry.sample(n=t))
 elif v == 'fear':
  # Adding  rows  to  data
  data = data.append(df_fear.sample(n=t))
 elif v == 'happy':
  # Adding  rows  to  data
  data = data.append(df_happy.sample(n=t))
else:
 # Adding rows to data
 data = data.append(df_sad.sample(n=t))
 return data
# Task of function 'pre' is to take list of emotions (containing duplicate also) &
#return unique list of emotion in sorted order based on count. def pre(l):
# result contain sorted emotion's(duplicate present if any)
result = [item for items, c in Counter(l).most_common() for
item in [items] * c]
# Creating empty unique list
ul = []
for x in result:
 if x not in ul:
  ul.append(x)
```

```
 return ul

# Creating model

# kerel_size = specifying the height and width of the 2D convolution

window. model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',input_shape=(48,48,1)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(128,  kernel_size=(3,  3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(7, activation='softmax'))

# Loading weights from pretrained model model.load_weights('model.h5')

emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy", 4: "Neutral", 5: "Sad", 6: "Surprised"}

# Required syntax

cv2.ocl.setUseOpenCL(False)

cap = cv2.VideoCapture(0)

# Text or heading's

st.markdown("<h2 style='text-align: center; color: white;'><b>Emotion based music recommendation</b></h2>", unsafe_allow_html=True)
```

```
  st.markdown("<h5 style='text-align: center; color: grey;'><b>Click on the name of recommended song to reach website</b></h5>", unsafe_allow_html=True)

  # Just for indentation

  col1,col2,col3  = st.columns(3)

  list = []

  with col1: pass

  with col2:

   if st.button('SCAN EMOTION(Click here)'):

    # Clearing  values

    count = 0

    list.clear()

    while True: ret, frame = cap.read()

    if not ret:

     break

    face  =  cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = face.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

    # Counter

    count = count + 1

    for (x, y, w, h) in faces:

     # Creating rectangle around face cv2.rectangle(frame, (x, y - 50), (x + w, y + h + 10), (255, 0, 0), 2) roi_gray = gray[y:y + h, x:x + w]

     # Taking image out

     cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray, (48, 48)), 1), 0)

     # Predicting model on cropped image

     prediction  = model.predict(cropped_img)

     # Appending emotion to list
```

```
    max_index = int(np.argmax(prediction))
    list.append(emotion_dict[max_index])
    # Putting text of emotion on top of rectangle
    cv2.putText(frame, emotion_dict[max_index], (x + 20, y - 60),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.imshow('Video', cv2.resize(frame, (1000, 700), interpolation=cv2.INTER_CUBIC))
    # For emergency close window
    if cv2.waitKey(1) & 0xFF == ord('x'):
     break
    # To get time for window to stay, so that we take input
    if count >= 20:
     break
    # Destroy cv2 window
    cap.release()
    cv2.destroyAllWindows()
    # Preprocessing list of emotion's
    list = pre(list)
    with col3: pass
    # Calling 'fun()' or creating dataframe
    new_df = fun(list)
    # Just for separation
    st.write("")
    # Normal text
    st.markdown("<h5 style='text-align: center; color: grey;'><b>Recommended song's with
artist names</b></h5>", unsafe_allow_html=True)
    # Just for separation
```

```
    st.write("----------------------------------------------------------------------------------------------------
--  ----------------")

    try:

     # l = iterator over link column in dataframe

     # a = iterator over artist column in dataframe

     # i = iterator from (0 to 30)

     # n = iterator over name column in dataframe

     for l,a,n,i in zip(new_df["link"],new_df['artist'],new_df['name'],range(30)):

      # Recommended song name

      st.markdown("""<h4  style='text-align: center;'><a  href={}>{}
{}</a></h4>""".format(l,i+1,n),unsafe_allow_html=True)

      # Artist name

      st.markdown("<h5 style='text-align: center; color:
grey;'><i>{}</i></h5>".format(a),unsafe_allow_html=True)

      # Just for separation

      st.write("----------------------------------------------------------------------------------------------------  --
----------")

    except:

      pass
```