```
#!pip install tensorflow
import tensorflow
#!pip install keras
import keras

import numpy as np
import cv2
import keras

from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D
from tensorflow.keras.optimizers import Adam
from keras.layers import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

```
from google.colab import drive
drive.mount('/content/drive/')
import os
os.chdir("/content/drive/My Drive/")
!ls
```

```
train_dir = "archive (1)/train"
val_dir = "archive (1)/test"
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
      train_dir,
      target_size=(48,48),
      batch_size=64,
      color_mode="grayscale",
      class_mode= "categorical")

validation_generator = val_datagen.flow_from_directory(
      val_dir,
      target_size=(48,48),
      batch_size=64,
      color_mode="grayscale",
      class_mode= "categorical")
```

```
Found 28719 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

```
for i in os.listdir("train/"):
      print(str(len(os.listdir("train/"+i))) +" "+ i +" images")
```

```
3995 angry images
436 disgust images
4097 fear images
7215 happy images
4965 neutral images
4830 sad images
3171 surprise images
```

```
for i in os.listdir("test/"):
      print(str(len(os.listdir("test/"+i))) +" "+ i +" images")
```

```
958 angry images
111 disgust images
1024 fear images
1774 happy images
1233 neutral images
1247 sad images
831 surprise images
```

```
emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48,48,1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))

emotion_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)            (None, 46, 46, 32)        320

 conv2d_5 (Conv2D)            (None, 44, 44, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 22, 22, 64)        0
 2D)

 dropout_3 (Dropout)          (None, 22, 22, 64)        0

 conv2d_6 (Conv2D)            (None, 20, 20, 128)       73856

 max_pooling2d_4 (MaxPooling  (None, 10, 10, 128)       0
 2D)

 conv2d_7 (Conv2D)            (None, 8, 8, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)         0
 2D)

 dropout_4 (Dropout)          (None, 4, 4, 128)         0

 flatten_1 (Flatten)          (None, 2048)              0
```

```
 dense_2 (Dense)              (None, 1024)              2098176

 dropout_5 (Dropout)          (None, 1024)              0

 dense_3 (Dense)              (None, 7)                 7175

=================================================================
Total params: 2,345,607
Trainable params: 2,345,607
Non-trainable params: 0
_________________________________________________________________
```

```
emotion_model.compile(loss='categorical_crossentropy',optimizer=Adam(learning_
rate=0.0001, decay=1e-6),metrics=['accuracy'])

emotion_model_info = emotion_model.fit(
      train_generator,
      steps_per_epoch=28709 // 64,
      epochs=50,
      validation_data=validation_generator,
      validation_steps=7178 // 64)
```

```
Epoch 1/50
448/448 [==============================] - 11568s 26s/step - loss:
1.7982 - accuracy: 0.2627 - val_loss: 1.7023 - val_accuracy: 0.3324
Epoch 2/50
448/448 [==============================] - 441s 984ms/step - loss:
1.6223 - accuracy: 0.3672 - val_loss: 1.5393 - val_accuracy: 0.4083
Epoch 3/50
448/448 [==============================] - 439s 980ms/step - loss:
1.5224 - accuracy: 0.4160 - val_loss: 1.4542 - val_accuracy: 0.4488
Epoch 4/50
448/448 [==============================] - 447s 997ms/step - loss:
1.4517 - accuracy: 0.4464 - val_loss: 1.3943 - val_accuracy: 0.4714
Epoch 5/50
448/448 [==============================] - 457s 1s/step - loss: 1.3933
- accuracy: 0.4709 - val_loss: 1.3512 - val_accuracy: 0.4870
Epoch 6/50
448/448 [==============================] - 449s 1s/step - loss: 1.3457
- accuracy: 0.4902 - val_loss: 1.3092 - val_accuracy: 0.5008
Epoch 7/50
448/448 [==============================] - 441s 984ms/step - loss:
1.2992 - accuracy: 0.5057 - val_loss: 1.2722 - val_accuracy: 0.5183
Epoch 8/50
448/448 [==============================] - 441s 984ms/step - loss:
1.2637 - accuracy: 0.5217 - val_loss: 1.2808 - val_accuracy: 0.5047
Epoch 9/50
448/448 [==============================] - 442s 986ms/step - loss:
1.2316 - accuracy: 0.5399 - val_loss: 1.2348 - val_accuracy: 0.5339
Epoch 10/50
448/448 [==============================] - 430s 960ms/step - loss:
1.2017 - accuracy: 0.5518 - val_loss: 1.2158 - val_accuracy: 0.5340
Epoch 11/50
448/448 [==============================] - 433s 965ms/step - loss:
1.1744 - accuracy: 0.5628 - val_loss: 1.1820 - val_accuracy: 0.5516
```

```
Epoch 12/50
448/448 [==============================] - 431s 962ms/step - loss:
1.1485 - accuracy: 0.5726 - val_loss: 1.1646 - val_accuracy: 0.5534
Epoch 13/50
448/448 [==============================] - 432s 963ms/step - loss:
1.1245 - accuracy: 0.5783 - val_loss: 1.1483 - val_accuracy: 0.5643
Epoch 14/50
448/448 [==============================] - 431s 962ms/step - loss:
1.1004 - accuracy: 0.5898 - val_loss: 1.1406 - val_accuracy: 0.5699
Epoch 15/50
448/448 [==============================] - 435s 971ms/step - loss:
1.0754 - accuracy: 0.5982 - val_loss: 1.1322 - val_accuracy: 0.5695
Epoch 16/50
448/448 [==============================] - 431s 961ms/step - loss:
1.0482 - accuracy: 0.6118 - val_loss: 1.1183 - val_accuracy: 0.5798
Epoch 17/50
448/448 [==============================] - 432s 965ms/step - loss:
1.0298 - accuracy: 0.6159 - val_loss: 1.1163 - val_accuracy: 0.5778
Epoch 18/50
448/448 [==============================] - 433s 966ms/step - loss:
1.0058 - accuracy: 0.6266 - val_loss: 1.1011 - val_accuracy: 0.5865
Epoch 19/50
448/448 [==============================] - 431s 963ms/step - loss:
0.9859 - accuracy: 0.6354 - val_loss: 1.0941 - val_accuracy: 0.5837
Epoch 20/50
448/448 [==============================] - 430s 959ms/step - loss:
0.9699 - accuracy: 0.6394 - val_loss: 1.0866 - val_accuracy: 0.5935
Epoch 21/50
448/448 [==============================] - 431s 961ms/step - loss:
0.9433 - accuracy: 0.6514 - val_loss: 1.0894 - val_accuracy: 0.5931
Epoch 22/50
448/448 [==============================] - 431s 962ms/step - loss:
0.9186 - accuracy: 0.6631 - val_loss: 1.0761 - val_accuracy: 0.6010
Epoch 23/50
448/448 [==============================] - 441s 985ms/step - loss:
0.8915 - accuracy: 0.6704 - val_loss: 1.0681 - val_accuracy: 0.6014
Epoch 24/50
448/448 [==============================] - 436s 972ms/step - loss:
0.8772 - accuracy: 0.6790 - val_loss: 1.0696 - val_accuracy: 0.6021
Epoch 25/50
448/448 [==============================] - ETA: 0s - loss: 0.8457 -
accuracy: 0.6898
```

```
emotion_model.save_weights('model.h5')
```

```
cv2.ocl.setUseOpenCL(False)

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

cap = cv2.VideoCapture(0)

while True:
      ret, frame = cap.read()
      if not ret:
```

```
            break
      bounding_box=cv2.CascadeClassifier('haarcascade_frontalface_default.xml
')
      gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      num_faces=bounding_box.detectMultiScale(
            gray_frame,
            scaleFactor=1.3,
            minNeighbors=5)

      for (x, y, w, h) in num_faces:
            cv2.rectangle(frame, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)
            roi_gray_frame = gray_frame[y:y + h, x:x + w]
            cropped_img = np.expand_dims(
                  np.expand_dims(
                  cv2.resize(roi_gray_frame, (48, 48)), -1), 0)
            emotion_prediction = emotion_model.predict(cropped_img)
            maxindex = int(np.argmax(emotion_prediction))
            cv2.putText(frame, emotion_dict[maxindex], (x+20, y-60),
                  cv2.FONT_HERSHEY_SIMPLEX, 1,
                  (255, 255, 255), 2, cv2.LINE_AA)

      cv2.imshow('Video', cv2.resize(
            frame,(1200,860),interpolation = cv2.INTER_CUBIC))

      if cv2.waitKey(1) & 0xFF == ord('q'):
            cap.release()
            cv2.destroyAllWindows()
            break
```

```
import tkinter as tk
from tkinter import *
import cv2
from PIL import Image, ImageTk
import os
import numpy as np
import cv2
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D
from keras.optimizers import Adam
from keras.layers import MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
```

```
emotion_model = Sequential()
```

```
emotion_model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48,48,1)))
emotion_model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
emotion_model.add(MaxPooling2D(pool_size=(2, 2)))
emotion_model.add(Dropout(0.25))

emotion_model.add(Flatten())
emotion_model.add(Dense(1024, activation='relu'))
emotion_model.add(Dropout(0.5))
emotion_model.add(Dense(7, activation='softmax'))
emotion_model.load_weights('model.h5')

emotion_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 46, 46, 32)        320

conv2d_17 (Conv2D)           (None, 44, 44, 64)        18496

max_pooling2d_12 (MaxPoolin  (None, 22, 22, 64)        0
g2D)

dropout_12 (Dropout)         (None, 22, 22, 64)        0

conv2d_18 (Conv2D)           (None, 20, 20, 128)       73856

max_pooling2d_13 (MaxPoolin  (None, 10, 10, 128)       0
g2D)

conv2d_19 (Conv2D)           (None, 8, 8, 128)         147584

max_pooling2d_14 (MaxPoolin  (None, 4, 4, 128)         0
g2D)

dropout_13 (Dropout)         (None, 4, 4, 128)         0

flatten_4 (Flatten)          (None, 2048)              0

dense_8 (Dense)              (None, 1024)              2098176

dropout_14 (Dropout)         (None, 1024)              0

dense_9 (Dense)              (None, 7)                 7175

=================================================================
Total params: 2,345,607
```

```
Trainable params: 2,345,607
Non-trainable params: 0
_________________________________________________________________
```

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


emoji_dist={0:"C:/Users/Jothy Natarajan/Downloads/Angry
emoji.png",1:"C:/Users/Jothy Natarajan/Downloads/Disgusted
emoji.png",2:"C:/Users/Jothy Natarajan/Downloads/Fear
emoji.png",3:"C:/Users/Jothy Natarajan/Downloads/Happy
emoji.png",4:"C:/Users/Jothy Natarajan/Downloads/Neutral
emoji.png",5:"C:/Users/Jothy Natarajan/Downloads/Sad
emoji.png",6:"C:/Users/Jothy Natarajan/Downloads/Surprised emoji.png"}

global last_frame1
last_frame1 = np.zeros((480, 640, 3), dtype=np.uint8)
global cap1
show_text=[0]
def show_vid():
      cap1 = cv2.VideoCapture(0)
      if not cap1.isOpened():
            print("cant open the camera1")

      flag1, frame1 = cap1.read()
      frame1 = cv2.resize(frame1,(600,500))
      bounding_box=cv2.CascadeClassifier('haarcascade_frontalface_default.xml
')
      gray_frame = cv2.cvtColor(frame1, cv2.COLOR_BGR2GRAY)
      num_faces=bounding_box.detectMultiScale(
            gray_frame,
            scaleFactor=1.3,
            minNeighbors=5)

      for (x, y, w, h) in num_faces:
            cv2.rectangle(frame1, (x, y-50), (x+w, y+h+10), (255, 0, 0), 2)
            roi_gray_frame = gray_frame[y:y + h, x:x + w]
            cropped_img=np.expand_dims(
                  np.expand_dims(
                  cv2.resize(roi_gray_frame,(48,48)),-1),0)
            prediction = emotion_model.predict(cropped_img)
            maxindex = int(np.argmax(prediction))
            cv2.putText(frame1, emotion_dict[maxindex], (x+20, y-60),
                  cv2.FONT_HERSHEY_SIMPLEX, 1,(255, 255, 255), 2, cv2.LINE_AA)
            show_text[0]=maxindex
```

```
      if flag1 is None:
            print ("Major error!")


      elif flag1:
            global last_frame1
            last_frame1 = frame1.copy()
            pic = cv2.cvtColor(last_frame1, cv2.COLOR_BGR2RGB)
            img = Image.fromarray(pic)
            imgtk = ImageTk.PhotoImage(image=img)
            lmain.imgtk = imgtk
            lmain.configure(image=imgtk)
            lmain.after(10, show_vid)
      if (cv2.waitKey(1) & 0xFF == ord('q')):
            exit()


def show_vid2():
      frame2=cv2.imread(emoji_dist[show_text[0]])
      pic2=cv2.cvtColor(frame2,cv2.COLOR_BGR2RGB)
      img2=Image.fromarray(frame2)
      imgtk2=ImageTk.PhotoImage(image=img2)
      lmain2.imgtk2=imgtk2
      lmain3.configure(text=emotion_dict[show_text[0]],font=('arial',45,'bold'
))
      lmain2.configure(image=imgtk2)
      lmain2.after(10, show_vid2)

if __name__ == '__main__':
      root=tk.Tk()
      img = ImageTk.PhotoImage(Image.open("Logo.png"))

      heading = Label(root,image=img,bg='black')
      heading.pack()
      heading2= Label(
            root,text="Photo to Emoji",
            pady=20,
            font=('arial',45,'bold'),
            bg='black',fg='#CDCDCD')
      heading2.pack()

      lmain = tk.Label(master=root,padx=50,bd=10)
      lmain2 = tk.Label(master=root,bd=10)
      lmain3=tk.Label(master=root,bd=10,fg="#CDCDCD",bg='black')
      lmain.pack(side=LEFT)
      lmain.place(x=50,y=250)
      lmain3.pack()
```

```
      lmain3.place(x=960,y=250)
      lmain2.pack(side=RIGHT)
      lmain2.place(x=900,y=350)
      root.title("Photo To Emoji")
      root.geometry("1400x900+100+10")
      root['bg']='black'
      exitbutton = Button(root,
text='Quit',fg="red",command=root.destroy,font=('arial',25,'bold')).pack(side
= BOTTOM)
      show_vid()
      show_vid2()
      root.mainloop()
```