import sys

import argparse

import cv2

import numpy as np

from collections import OrderedDict

from datetime import datetime, timedelta

from faces import FaceDetector

from data import FaceData

from gabor import GaborBank

from emotions import EmotionsDetector

from gensim.summarization.summarizer import summarize

from bs4 import BeautifulSoup

count=0

val\_0=0

val\_1=0

val\_2=0

val\_3=0

val\_4=0

val\_5=0

val\_6=0

#---------------------------------------------

class VideoData:

def \_\_init\_\_(self):

self.\_faceDet = FaceDetector()

self.\_bank = GaborBank()

self.\_emotionsDet = EmotionsDetector()

self.\_face = FaceData()

self.\_emotions = OrderedDict()

#-----------------------------------------

def detect(self, frame):

ret, face = self.\_faceDet.detect(frame)

if ret:

self.\_face = face

# Crop just the face region

frame, face = face.crop(frame)

# Filter it with the Gabor bank

responses = self.\_bank.filter(frame)

# Detect the prototypic emotions based on the filter responses

self.\_emotions = self.\_emotionsDet.detect(face, responses)

return True

else:

self.\_face = None

return False

#-----------------------------------------

def draw(self, frame):

# Font settings

font = cv2.FONT\_HERSHEY\_SIMPLEX

scale = 0.5

thick = 1

glow = 3 \* thick

# Color settings

black = (0, 0, 0)

white = (255, 255, 255)

yellow = (0, 255, 255)

red = (0, 0, 255)

empty = True

# Plot the face landmarks and face distance

x = 5

y = 0

w = int(frame.shape[1]\* 0.2)

try:

face = self.\_face

empty = face.isEmpty()

face.draw(frame)

except:

pass

# Plot the emotion probabilities

global count

global val\_0

global val\_1

global val\_2

global val\_3

global val\_4

global val\_5

global val\_6

try:

emotions = self.\_emotions

if empty:

labels = []

values = []

#count=0

else:

labels = list(emotions.keys())

values = list(emotions.values())

bigger = labels[values.index(max(values))]

#d1[bigger]=max(values)

count=count+1

val\_0=val\_0+values[0]

val\_1=val\_1+values[1]

val\_2=val\_2+values[2]

val\_3=val\_3+values[3]

val\_4=val\_4+values[4]

val\_5=val\_5+values[5]

val\_6=val\_6+values[6]

print(count)

print("Neutral ",val\_0)

print("Happiness ",val\_1)

print("Sadness ",val\_2)

print("Anger ",val\_3)

print("Fear ",val\_4)

print("Surprise ",val\_5)

print("Disgust ",val\_6)

#print (bigger)

#print (max(values))

# Draw the header

text = 'emotions'

size, \_ = cv2.getTextSize(text, font, scale, thick)

y += size[1] + 20

cv2.putText(frame, text, (x, y), font, scale, black, glow)

cv2.putText(frame, text, (x, y), font, scale, yellow, thick)

y += 5

cv2.line(frame, (x,y), (x+w,y), black, 1)

size, \_ = cv2.getTextSize('happiness', font, scale, thick)

t = size[0] + 20

w = 150

h = size[1]

for l, v in zip(labels, values):

lab = '{}:'.format(l)

val = '{:.2f}'.format(v)

size, \_ = cv2.getTextSize(l, font, scale, thick)

# Set a red color for the emotion with bigger probability

color = red if l == bigger else yellow

y += size[1] + 15

p1 = (x+t, y-size[1]-5)

p2 = (x+t+w, y-size[1]+h+5)

cv2.rectangle(frame, p1, p2, black, 1)

# Draw the filled rectangle proportional to the probability

p2 = (p1[0] + int((p2[0] - p1[0]) \* v), p2[1])

cv2.rectangle(frame, p1, p2, color, -1)

cv2.rectangle(frame, p1, p2, black, 1)

# Draw the emotion label

cv2.putText(frame, lab, (x, y), font, scale, black, glow)

cv2.putText(frame, lab, (x, y), font, scale, color, thick)

# Draw the value of the emotion probability

cv2.putText(frame, val, (x+t+5, y), font, scale, black, glow)

cv2.putText(frame, val, (x+t+5, y), font, scale, white, thick)

except Exception as e:

print(e)

pass

#---------------------------------------------

def main(argv):

from googletrans import Translator

translator = Translator(service\_urls=['translate.google.com','translate.google.co.kr',])

filename=input('Enter File Name:-')

f=open(filename,'r')

f1=open('trans.txt','w')

f2=open('summ.txt','w')

lines =list()

for line in f:

lines.append(line)

translations = translator.translate(list(lines), dest='kn')

for translation in translations:

f1.write(translation.text)

f1.close()

print('Translated file saved in main folder')

with open('abc.txt', 'r') as myfile:

text2=myfile.read()

x=summarize(text2, word\_count=100)

f2.write(x)

#print(x)

#print('abc')

f2.close()

# Parse the command line

args = parseCommandLine(argv)

# Loads the video or starts the webcam

irceName = 'Webcam #{}'.format(args.id)

video = cv2.VideoCapture(args.file)

filekanaam=args.file

if not video.isOpened():

print('Error opening video file {}'.format(args.file))

sys.exit(-1)

fps = int(video.get(cv2.CAP\_PROP\_FPS))

frameCount = int(video.get(cv2.CAP\_PROP\_FRAME\_COUNT))

sourceName = args.file

video.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 1280);

video.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 720);

# Create the helper class

data = VideoData()

# Text settings

font = cv2.FONT\_HERSHEY\_SIMPLEX

scale = 1

thick = 1

glow = 3 \* thick

# Color settings

color = (255, 255, 255)

paused = False

frameNum = 0

# Process the video input

while True:

if not paused:

start = datetime.now()

ret, img = video.read()

if ret:

frame = img.copy()

else:

paused = True

drawInfo(frame, frameNum, frameCount, paused, fps, args.source)

data.detect(frame)

data.draw(frame)

cv2.imshow(sourceName, frame)

if paused:

key = cv2.waitKey(0)

else:

end = datetime.now()

delta = (end - start)

if fps != 0:

delay = int(max(1, ((1 / fps) - delta.total\_seconds()) \* 1000))

else:

delay = 1

key = cv2.waitKey(delay)

if key == ord('q') or key == ord('Q') or key == 27:

break

elif key == ord('p') or key == ord('P'):

paused = not paused

elif args.source == 'video' and (key == ord('r') or key == ord('R')):

frameNum = 0

video.set(cv2.CAP\_PROP\_POS\_FRAMES, frameNum)

elif args.source == 'video' and paused and key == 2424832: # Left key

frameNum -= 1

if frameNum < 0:

frameNum = 0

video.set(cv2.CAP\_PROP\_POS\_FRAMES, frameNum)

elif args.source == 'video' and paused and key == 2555904: # Right key

frameNum += 1

if frameNum >= frameCount:

frameNum = frameCount - 1

elif args.source == 'video' and key == 2162688: # Pageup key

frameNum -= (fps \* 10)

if frameNum < 0:

frameNum = 0

video.set(cv2.CAP\_PROP\_POS\_FRAMES, frameNum)

elif args.source == 'video' and key == 2228224: # Pagedown key

frameNum += (fps \* 10)

if frameNum >= frameCount:

frameNum = frameCount - 1

video.set(cv2.CAP\_PROP\_POS\_FRAMES, frameNum)

elif key == 7340032: # F1

showHelp(sourceName, frame.shape)

if not paused:

frameNum += 1

video.release()

cv2.destroyAllWindows()

print("\n")

print("Average\_Neutral ",(val\_0/count))

print("Average\_Happiness ",(val\_1/count))

print("Average\_Sadness ",(val\_2/count))

print("Average\_Anger ",(val\_3/count))

print("Average\_Fear ",(val\_4/count))

print("Average\_Surprise ",(val\_5/count))

print("Average\_Disgust ",(val\_6/count))

best\_emotion=max(val\_0/count,val\_1/count,val\_2/count,val\_3/count,val\_4/count,val\_5/count,val\_6/count)

if best\_emotion==(val\_0/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Neutral','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Neutral with optimality of ",best\_emotion)

if best\_emotion==(val\_1/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Happiness','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Happiness with optimality of ",best\_emotion)

if best\_emotion==(val\_2/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Sadness','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Sadness with optimality of ",best\_emotion)

if best\_emotion==(val\_3/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Anger','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Anger with optimality of ",best\_emotion)

if best\_emotion==(val\_4/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Fear','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Fear with optimality of ",best\_emotion)

if best\_emotion==(val\_5/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Surprise','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Surprise with optimality of ",best\_emotion)

if best\_emotion==(val\_6/count):

import pandas as pd

df=pd.read\_excel("../output.xlsx", header=None)

numrows=len(df)

df.loc[numrows+1]=[filekanaam,'Disgust','','']

df.to\_excel("../output.xlsx",header=None,index=None)

print("Best average emotion is Disgust with optimality of ",best\_emotion)

#---------------------------------------------

def drawInfo(frame, frameNum, frameCount, paused, fps, source):

# Font settings

font = cv2.FONT\_HERSHEY\_SIMPLEX

scale = 0.5

thick = 1

glow = 3 \* thick

# Color settings

black = (0, 0, 0)

yellow = (0, 255, 255)

# Print the current frame number and timestamp

if source == 'video':

text = 'Frame: {:d}/{:d} {}'.format(frameNum, frameCount - 1,

'(paused)' if paused else '')

else:

text = 'Frame: {:d} {}'.format(frameNum, '(paused)' if paused else '')

size, \_ = cv2.getTextSize(text, font, scale, thick)

x = 5

y = frame.shape[0] - 2 \* size[1]

cv2.putText(frame, text, (x, y), font, scale, black, glow)

cv2.putText(frame, text, (x, y), font, scale, yellow, thick)

if source == 'video':

timestamp = datetime.min + timedelta(seconds=(frameNum / fps))

elapsedTime = datetime.strftime(timestamp, '%H:%M:%S')

timestamp = datetime.min + timedelta(seconds=(frameCount / fps))

totalTime = datetime.strftime(timestamp, '%H:%M:%S')

text = 'Time: {}/{}'.format(elapsedTime, totalTime)

size, \_ = cv2.getTextSize(text, font, scale, thick)

y = frame.shape[0] - 5

cv2.putText(frame, text, (x, y), font, scale, black, glow)

cv2.putText(frame, text, (x, y), font, scale, yellow, thick)

# Print the help message

text = 'Press F1 for help'

size, \_ = cv2.getTextSize(text, font, scale, thick)

x = frame.shape[1] - size[0] - 5

y = frame.shape[0] - size[1] + 5

cv2.putText(frame, text, (x, y), font, scale, black, glow)

cv2.putText(frame, text, (x, y), font, scale, yellow, thick)

#---------------------------------------------

def showHelp(windowTitle, shape):

# Font settings

font = cv2.FONT\_HERSHEY\_SIMPLEX

scale = 1.0

thick = 1

# Color settings

black = (0, 0, 0)

red = (0, 0, 255)

# Create the background image

image = np.ones((shape[0], shape[1], 3)) \* 255

# The help text is printed in one line per item in this list

helpText = [

'Controls:',

'-----------------------------------------------',

'[q] or [ESC]: quits from the application.',

'[p]: toggles paused/playing the video/webcam input.',

'[r]: restarts the video playback (video input only).',

'[left/right arrow]: displays the previous/next frame (video input only).',

'[page-up/down]: rewinds/fast forwards by 10 seconds (video input only).',

' ',

' ',

'Press any key to close this window...'

]

# Print the controls help text

xCenter = image.shape[1] // 2

yCenter = image.shape[0] // 2

margin = 20 # between-lines margin in pixels

textWidth = 0

textHeight = margin \* (len(helpText) - 1)

lineHeight = 0

for line in helpText:

size, \_ = cv2.getTextSize(line, font, scale, thick)

textHeight += size[1]

textWidth = size[0] if size[0] > textWidth else textWidth

lineHeight = size[1] if size[1] > lineHeight else lineHeight

x = xCenter - textWidth // 2

y = yCenter - textHeight // 2

for line in helpText:

cv2.putText(image, line, (x, y), font, scale, black, thick \* 3)

cv2.putText(image, line, (x, y), font, scale, red, thick)

y += margin + lineHeight

# Show the image and wait for a key press

cv2.imshow(windowTitle, image)

cv2.waitKey(0)

#---------------------------------------------

def parseCommandLine(argv):

parser = argparse.ArgumentParser(description='Tests the face and emotion '

'detector on a video file input.')

parser.add\_argument('source', nargs='?', const='Yes',

choices=['video', 'cam'], default='cam',

help='Indicate the source of the input images for '

'the detectors: "video" for a video file or '

'"cam" for a webcam. The default is "cam".')

parser.add\_argument('-f', '--file', metavar='<name>',

help='Name of the video file to use, if the source is '

'"video". The supported formats depend on the codecs '

'installed in the operating system.')

parser.add\_argument('-i', '--id', metavar='<number>', default=0, type=int,

help='Numerical id of the webcam to use, if the source '

'is "cam". The default is 0.')

args = parser.parse\_args()

if args.source == 'video' and args.file is None:

parser.error('-f is required when source is "video"')

return args

#---------------------------------------------

# namespace verification for invoking main

#---------------------------------------------

if \_\_name\_\_ == '\_\_main\_\_':

main(sys.argv[1:])