face detection recognition DNN video-190919-adrian1

September 19, 2019

[ ]:
```
from os import listdir
from os.path import isfile, join
import os
import cv2
import dlib
import numpy as np

# Define Image Path Here


def draw_label(image, point, label, font=cv2.FONT_HERSHEY_SIMPLEX,
               font_scale=0.8, thickness=1):
    size = cv2.getTextSize(label, font, font_scale, thickness)[0]
    x, y = point
    cv2.rectangle(image, (x, y - size[1]), (x + size[0], y), (255, 0, 0), cv2.
 ↪FILLED)
    cv2.putText(image, label, point, font, font_scale, (255, 255, 255),␣
 ↪thickness, lineType=cv2.LINE_AA)

detector1 = dlib.get_frontal_face_detector()

# Initialize Webcam
cap = cv2.VideoCapture('sridevi1.mp4')
#cap = cv2.VideoCapture(0)
img_size = 64
margin = 0.2
frame_count = 0

_videodir = "./dataset"

        # create dynamic name, like "D:\Current Download\Attachment82673"
_videodir = os.path.join(_videodir, 'sridevinew')

        # create 'dynamic' dir, if it does not exist
if not os.path.exists(_videodir):
        os.makedirs(_videodir)
```

```
i=1;
while True:
    ret, frame = cap.read()
    frame_count += 1
    print(frame_count)
    if frame_count>40 and frame_count<60:
        file_name = _videodir + "/"+ str(frame_count)+ "_"+str(i)+".jpg"
        i+=1
        cv2.imshow("Face Detector", frame)
        cv2.imwrite(file_name, frame)
    elif cv2.waitKey(1) == 13:
        break
    else:
        continue
cap.release()
cv2.destroyAllWindows()
```

[ ]:
```
from imutils import paths
import numpy as np
import argparse
import imutils
import pickle
import dlib
import face_recognition
import cv2
import os
```

[ ]:
```
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images("./dataset"))


# initialize the total number of faces processed
total = 0

#grab the paths to the input images in our dataset, then initialize
# out data list (which we'll soon populate)

#imagePaths = list(paths.list_images("./face-clustering/friendsvideo"))
#imagePaths = list(paths.list_images(args["dataset"]))
data = []
```

[ ]:
```
print(imagePaths)
```

[ ]:
```
# loop over the image paths
total=0
knownEncodings=[]
knownNames=[]
for (i, imagePath) in enumerate(imagePaths):
```

```
        # extract the person name from the image path
        print("[INFO] processing image {}/{}".format(i + 1,
                len(imagePaths)))
        name = imagePath.split(os.path.sep)[-2]

        # load the image, resize it to have a width of 600 pixels (while
        # maintaining the aspect ratio), and then grab the image
        # dimensions
        image = cv2.imread(imagePath)
        image = cv2.imread(imagePath)
        image = imutils.resize(image, width=600)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            # detect the (x, y)-coordinates of the bounding boxes
        # corresponding to each face in the input image
        boxes = face_recognition.face_locations(rgb,model="hog")
        # compute the facial embedding for the face
        encodings = face_recognition.face_encodings(rgb, boxes)

        for encoding in encodings:
                # add each encoding + name to our set of known names and
                # encodings
                knownEncodings.append(encoding)
                knownNames.append(name)
        total += 1
    # build a dictionary of the image path, bounding box location,
        # and facial encodings for the current image


# dump the facial encodings data to disk
```

[ ]:
```
# # dump the facial embeddings + names to disk
# print("[INFO] serializing {} encodings...".format(total))

# dump the facial encodings + names to disk
print("[INFO] serializing {} encodings...".format(total))
data = {"encodings": knownEncodings, "names": knownNames}
f = open("./encodingsclass", "wb")
f.write(pickle.dumps(data))
f.close()
```

[ ]:
```
# FACE classification using euclidean classes
# python recognize_faces_image.py --encodings encodings.pickle --image examples/
 ↪example_01.png

# import the necessary packages
import face_recognition
import argparse
import pickle
```

```
import cv2



# load the known faces and embeddings
print("[INFO] loading encodings...")
data = pickle.loads(open("./encodingsclass", "rb").read())

# load the input image and convert it from BGR to RGB
image = cv2.imread("./images/50237.jpg")
rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# detect the (x, y)-coordinates of the bounding boxes corresponding
# to each face in the input image, then compute the facial embeddings
# for each face
print("[INFO] recognizing faces...")
boxes = face_recognition.face_locations(rgb, model="hog")
encodings = face_recognition.face_encodings(rgb, boxes)

# initialize the list of names for each face detected
names = []

# loop over the facial embeddings
for encoding in encodings:
        # attempt to match each face in the input image to our known
        # encodings
        matches = face_recognition.compare_faces(data["encodings"],encoding)
        name = "Unknown"

        # check to see if we have found a match
        if True in matches:
                # find the indexes of all matched faces then initialize a
                # dictionary to count the total number of times each face
                # was matched
                matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                counts = {}

                # loop over the matched indexes and maintain a count for
                # each recognized face face
                for i in matchedIdxs:
                        name = data["names"][i]
                        counts[name] = counts.get(name, 0) + 1

                # determine the recognized face with the largest number of
                # votes (note: in the event of an unlikely tie Python will
                # select first entry in the dictionary)
                name = max(counts, key=counts.get)
```

```
        # update the list of names
        names.append(name)


# loop over the recognized faces
for ((top, right, bottom, left), name) in zip(boxes, names):
        # draw the predicted face name on the image
        cv2.rectangle(image, (left, top), (right, bottom), (0, 255, 0), 2)
        y = top - 15 if top - 15 > 15 else top + 15
        cv2.putText(image, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (0, 255, 0), 2)

# show the output image
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[ ]:
```
# FACE DETECTION RECOGNITION USING DLIB WITH COMPARISON IN VIDEO FILE
# python recognize_faces_video_file.py --encodings encodings.pickle --input␣
 ↪videos/lunch_scene.mp4
# python recognize_faces_video_file.py --encodings encodings.pickle --input␣
 ↪videos/lunch_scene.mp4 --output output/lunch_scene_output.avi --display 0

# import the necessary packages
import face_recognition
import argparse
import imutils
import pickle
import time
import cv2
from imutils.video import FPS

# load the known faces and embeddings
print("[INFO] loading encodings...")
data = pickle.loads(open("./encodingsclass", "rb").read())

# initialize the pointer to the video file and the video writer
print("[INFO] processing video...")
stream = cv2.VideoCapture("adrian.mp4")
fps = FPS().start()
writer = None
display=1
# loop over frames from the video file stream
while True:
        # grab the next frame
        (grabbed, frame) = stream.read()
```

```
# if the frame was not grabbed, then we have reached the
# end of the stream
if not grabbed:
        break

# convert the input frame from BGR to RGB then resize it to have
# a width of 750px (to speedup processing)
rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
rgb = imutils.resize(frame, width=750)
r = frame.shape[1] / float(rgb.shape[1])

# detect the (x, y)-coordinates of the bounding boxes
# corresponding to each face in the input frame, then compute
# the facial embeddings for each face
boxes = face_recognition.face_locations(rgb,
        model="hog")
encodings = face_recognition.face_encodings(rgb, boxes)
names = []

# loop over the facial embeddings
for encoding in encodings:
        # attempt to match each face in the input image to our known
        # encodings
        matches = face_recognition.compare_faces(data["encodings"],
                encoding)
        name = "Unknown"

        # check to see if we have found a match
        if True in matches:
                # find the indexes of all matched faces then initialize a
                # dictionary to count the total number of times each face
                # was matched
                matchedIdxs = [i for (i, b) in enumerate(matches) if b]
                counts = {}

                # loop over the matched indexes and maintain a count for
                # each recognized face face
                for i in matchedIdxs:
                        name = data["names"][i]
                        counts[name] = counts.get(name, 0) + 1

                # determine the recognized face with the largest number
                # of votes (note: in the event of an unlikely tie Python
                # will select first entry in the dictionary)
                name = max(counts, key=counts.get)
```

```
                # update the list of names
                names.append(name)

        # loop over the recognized faces
        for ((top, right, bottom, left), name) in zip(boxes, names):
                # rescale the face coordinates
                top = int(top * r)
                right = int(right * r)
                bottom = int(bottom * r)
                left = int(left * r)

                # draw the predicted face name on the image
                cv2.rectangle(frame, (left, top), (right, bottom),
                        (0, 255, 0), 2)
                y = top - 15 if top - 15 > 15 else top + 15
                cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
                        0.75, (0, 255, 0), 2)

#         # if the video writer is None *AND* we are supposed to write
#         # the output video to disk initialize the writer
#         if writer is None and args["output"] is not None:
#                 fourcc = cv2.VideoWriter_fourcc(*"MJPG")
#                 writer = cv2.VideoWriter(args["output"], fourcc, 24,
#                         (frame.shape[1], frame.shape[0]), True)

#         # if the writer is not None, write the frame with recognized
#         # faces t odisk
#         if writer is not None:
#                 writer.write(frame)

#         # check to see if we are supposed to display the output frame to
#         # the screen
        if display > 0:
                cv2.imshow("Frame", frame)
                fps.update()
                key = cv2.waitKey(1) & 0xFF

                # if the `q` key was pressed, break from the loop
                if key == ord("q"):
                        break

# stop the timer and display FPS information
fps.stop()
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# do a bit of cleanup
```

```
stream.release()
cv2.destroyAllWindows()

# check to see if the video writer point needs to be released
if writer is not None:
        writer.release()
```

[2]:
```
# Face matching in video
# python recognize_faces_video.py --encodings encodings.pickle
# python recognize_faces_video.py --encodings encodings.pickle --output output/
 ↪jurassic_park_trailer_output.avi --display 0

# import the necessary packages
from imutils.video import VideoStream
import face_recognition
import argparse
import imutils
import pickle
import time
import cv2
import os

# load the known faces and embeddings
print("[INFO] loading encodings...")
data = pickle.loads(open("./encodingsclass", "rb").read())

# initialize the video stream and pointer to output video file, then
# allow the camera sensor to warm up
print("[INFO] starting video stream...")
_videodir = "./dataset"
#vs = cv2.VideoCapture(0)
vs = VideoStream(src=0).start()
writer = None
time.sleep(2.0)
display=1
count=0
saveimage='n'
# loop over frames from the video file stream
while True:
        # grab the frame from the threaded video stream
        frame = vs.read()

        # convert the input frame from BGR to RGB then resize it to have
        # a width of 750px (to speedup processing)
        rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        rgb = imutils.resize(frame, width=750)
        r = frame.shape[1] / float(rgb.shape[1])
```

```
	# detect the (x, y)-coordinates of the bounding boxes
	# corresponding to each face in the input frame, then compute
	# the facial embeddings for each face
	boxes = face_recognition.face_locations(rgb,
		model="hog")
	encodings = face_recognition.face_encodings(rgb, boxes)
	names = []

	# loop over the facial embeddings
	for encoding in encodings:
		# attempt to match each face in the input image to our known
		# encodings
		matches = face_recognition.compare_faces(data["encodings"],
			encoding)
		name = "Unknown"

		# check to see if we have found a match
		if True in matches:
			# find the indexes of all matched faces then initialize a
			# dictionary to count the total number of times each face
			# was matched
			matchedIdxs = [i for (i, b) in enumerate(matches) if b]
			counts = {}

			# loop over the matched indexes and maintain a count for
			# each recognized face face
			for i in matchedIdxs:
				name = data["names"][i]
				counts[name] = counts.get(name, 0) + 1

			# determine the recognized face with the largest number
			# of votes (note: in the event of an unlikely tie Python
			# will select first entry in the dictionary)
			name = max(counts, key=counts.get)

		# update the list of names
		names.append(name)

	# loop over the recognized faces
	for ((top, right, bottom, left), name) in zip(boxes, names):
		# rescale the face coordinates
		top = int(top * r)
		right = int(right * r)
		bottom = int(bottom * r)
		left = int(left * r)
```

```
                # draw the predicted face name on the image
                cv2.rectangle(frame, (left, top), (right, bottom),
                        (0, 255, 0), 2)
                y = top - 15 if top - 15 > 15 else top + 15
                cv2.putText(frame, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX,
                        0.75, (0, 255, 0), 2)
                cv2.imwrite("./dataset/unknown1/frame%d.jpg" %count,frame)
                count+=1
#         # if the video writer is None *AND* we are supposed to write
#         # the output video to disk initialize the writer
#         if writer is None and output is not None:
#                 fourcc = cv2.VideoWriter_fourcc(*"MJPG")
#                 writer = cv2.VideoWriter(output, fourcc, 20,
#                         (frame.shape[1], frame.shape[0]), True)

#         # if the writer is not None, write the frame with recognized
#         # faces t odisk
#         if writer is not None:
#                 writer.write(frame)

# #         # check to see if we are supposed to display the output frame to
# #         # the screen
        if display > 0:
                cv2.imshow("Frame", frame)

                key = cv2.waitKey(1) & 0xFF

                # if the `q` key was pressed, break from the loop
                if key == ord("q"):
                        break

# do a bit of cleanup
# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()

# check to see if the video writer point needs to be released
if writer is not None:
        writer.release()
```

```
[INFO] loading encodings...
[INFO] starting video stream...
```

[3]:
```
#store new person
from imutils.video import VideoStream
import face_recognition
```

```
import argparse
import imutils
import dlib
import pickle
import time
import cv2

_videodir= "./dataset/"
_videodir1 = "./dataset/unknown1/"
if name=='Unknown':
        saveimage=input("Do you want to Authenticate this person ( Press y/n)?")
        if saveimage=='y':
                personname=input("Enter person name ")

# create dynamic name, like "D:\Current Download\Attachment82673"
                _videodir2= os.path.join(_videodir, personname)

        # create 'dynamic' dir, if it does not exist
                if not os.path.exists(_videodir2):
                        os.makedirs(_videodir2)
        listing = os.listdir(_videodir1)
        for file in listing:
                img = cv2.imread(_videodir1+file)
                outfile= _videodir2+"/"+file
                cv2.imwrite(outfile,img)
```

[4]:
```
filelist = [ f for f in os.listdir(_videodir1) if f.endswith(".jpg") ]
for f in filelist:
    os.remove(os.path.join(_videodir1, f))
```

[5]:
```
from os import listdir
from os.path import isfile, join
from imutils import paths
#store face encodings of new person
# loop over the image paths
total=0
knownEncodings1=[]
knownNames1=[]
imagePaths = list(paths.list_images(_videodir2))

for (i, imagePath) in enumerate(imagePaths):
        # extract the person name from the image path
        print("[INFO] processing image {}/{}".format(i + 1,
                len(imagePaths)))
        name = imagePath.split(os.path.sep)[-2]
```

```
        # load the image, resize it to have a width of 600 pixels (while
        # maintaining the aspect ratio), and then grab the image
        # dimensions
        image = cv2.imread(imagePath)
        image = cv2.imread(imagePath)
        image = imutils.resize(image, width=600)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            # detect the (x, y)-coordinates of the bounding boxes
        # corresponding to each face in the input image
        boxes = face_recognition.face_locations(rgb,model="hog")
        # compute the facial embedding for the face
        encodings = face_recognition.face_encodings(rgb, boxes)

        for encoding in encodings:
                # add each encoding + name to our set of known names and
                # encodings
                knownEncodings1.append(encoding)
                knownNames1.append(name)
        total += 1
    # build a dictionary of the image path, bounding box location,
        # and facial encodings for the current image


# dump the facial encodings data to disk
```

```
        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-5-27a32ed8ff66> in <module>
          7 knownEncodings1=[]
          8 knownNames1=[]
    ----> 9 imagePaths = list(paths.list_images(_videodir2))
         10
         11 for (i, imagePath) in enumerate(imagePaths):


        NameError: name '_videodir2' is not defined
```

[ ]:
```
# dump the facial encodings + names of new person to disk
print("[INFO] serializing {} encodings...".format(total))
data1 = {"encodings": knownEncodings1, "names": knownNames1}
f = open("./encodingsclass", "wb")
f.write(pickle.dumps(data1))
```

```
f.close()
```

[ ]: