Face Clustering DNN

September 17, 2019

[1]:
```
from imutils import paths
import face_recognition
import argparse
import pickle
import cv2
import os
```

[ ]:
```
# grab the paths to the input images in our dataset, then initialize
# out data list (which we'll soon populate)
print("[INFO] quantifying faces...")
imagePaths = list(paths.list_images("./face-clustering/dataset"))
#imagePaths = list(paths.list_images(args["dataset"]))
data = []
```

[ ]:
```
# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
        # load the input image and convert it from RGB (OpenCV ordering)
        # to dlib ordering (RGB)
        print("[INFO] processing image {}/{}".format(i + 1,len(imagePaths)))
        print(imagePath)
        image = cv2.imread(imagePath)
        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)

        # build a dictionary of the image path, bounding box location,
        # and facial encodings for the current image
        d = [{"imagePath": imagePath, "loc": box, "encoding": enc}
                for (box, enc) in zip(boxes, encodings)]
        data.extend(d)

# dump the facial encodings data to disk
print("[INFO] serializing encodings...")
```

```
f = open("encodings", "wb")
f.write(pickle.dumps(data))
f.close()
```

[2]:
```
# import the necessary packages
from sklearn.cluster import DBSCAN
from imutils import build_montages
import numpy as np
import pickle
import cv2
```

[3]:
```
# load the serialized face encodings + bounding box locations from
# disk, then extract the set of encodings to so we can cluster on
# them
print("[INFO] loading encodings...")
data = pickle.loads(open("encodings", "rb").read())
data = np.array(data)
encodings = [d["encoding"] for d in data]
```

```
[INFO] loading encodings...
```

[4]:
```
# cluster the embeddings
print("[INFO] clustering...")
clt = DBSCAN(metric="euclidean")
clt.fit(encodings)

# determine the total number of unique faces found in the dataset
labelIDs = np.unique(clt.labels_)
numUniqueFaces = len(np.where(labelIDs > -1)[0])
print("[INFO] # unique faces: {}".format(numUniqueFaces))
```

```
[INFO] clustering...
[INFO] # unique faces: 4
```

[7]:
```
# loop over the unique face integers
for labelID in labelIDs:
        # find all indexes into the `data` array that belong to the
        # current label ID, then randomly sample a maximum of 25 indexes
        # from the set
        print("[INFO] faces for face ID: {}".format(labelID))
        idxs = np.where(clt.labels_ == labelID)[0]
        idxs = np.random.choice(idxs, size=min(25, len(idxs)),
                replace=False)

        # initialize the list of faces to include in the montage
        faces = []
            # loop over the sampled indexes
        for i in idxs:
```

```
                # load the input image and extract the face ROI
                image = cv2.imread(data[i]["imagePath"])
                (top, right, bottom, left) = data[i]["loc"]
                face = image[top:bottom, left:right]

                # force resize the face ROI to 96x96 and then add it to the
                # faces montage list
                face = cv2.resize(face, (96, 96))
                faces.append(face)        # create a montage using 96x96 "tiles"␣
↪with 5 rows and 5 columns
        montage = build_montages(faces, (96, 96), (5, 5))[0]
        # show the output montage
        title = "Face ID #{}".format(labelID)
        title = "Unknown Faces" if labelID == -1 else title
        cv2.imshow(title, montage)
        k=cv2.waitKey(20)
        if k== 27:
                break
        #cv2.destroyAllWindows()
```

```
[INFO] faces for face ID: -1
[INFO] faces for face ID: 0
[INFO] faces for face ID: 1
[INFO] faces for face ID: 2
[INFO] faces for face ID: 3
```

[8]:
```
cv2.destroyAllWindows()
```

[ ]:
```

```