Rule_based_3 - Test

December 15, 2021

[7]:
```
import cv2

from PIL import Image
import numpy as np
import sys
import os
import csv
import dlib
import scipy
from scipy import ndimage
from skimage import measure, io, img_as_ubyte
import matplotlib.pyplot as plt
from skimage.color import label2rgb, rgb2gray
import numpy as np
import glob
import pandas as pd
from sklearn.cluster import KMeans #for clusterin
from os import listdir
from os.path import isfile, join
from imblearn.over_sampling import SMOTE
```

[ ]:
```
import numpy as np
import cv2
import dlib
import matplotlib.pyplot as plt
import pathlib
from pathlib import Path
import os
import imutils
import math

def get_norm(image,x1,y1,x2,y2):
    x = (int(image[x1][y1][0])-int(image[x2][y2][0]))**2
    y = (int(image[x1][y1][1])-int(image[x2][y2][1]))**2
    z = (int(image[x1][y1][2])-int(image[x2][y2][2]))**2
    norm = x + y + z
    return norm
```

```
def get_min(image,x1,y1):
    x = int(image[x1][y1][0])
    y = int(image[x1][y1][1])
    z = int(image[x1][y1][2])
    return np.min([x,y,z])

def get_color(image,x,y,gray):
    if gray == 1: image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return np.min(image[y,x])

def get_lum(image,x,y,w,h,k,gray):
    if gray == 1:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    i1 = range(int(-w/2),int(w/2))
    j1 = range(0,h)
    lumar = np.zeros((len(i1),len(j1)))
    for i in i1:
        for j in j1:
            lum = np.min(image[y+k*h,x+i])
            lumar[i][j] = lum

    return np.min(lumar)

def get_ave_down(image,x,y,h,w):
    ave = np.min(image[x-w:x+w,y-h:y])
    return int(ave)
def get_ave_up(image,x,y,h,w):
    ave = np.max(image[x-w:x+w,y:y+h])
    return int(ave)

def d(landmarks,index1,index2):
#get distance between i1 and i2
    x1 = landmarks[int(index1)][0]
    y1 = landmarks[int(index1)][1]
    x2 = landmarks[int(index2)][0]
    y2 = landmarks[int(index2)][1]
    x_diff = (x1 - x2)**2
    y_diff = (y1 - y2)**2
    dist = math.sqrt(x_diff + y_diff)
    return dist

def q(landmarks,index1,index2):
#get angle between a i1 and i2
    x1 = landmarks[int(index1)][0]
    y1 = landmarks[int(index1)][1]
    x2 = landmarks[int(index2)][0]
    y2 = landmarks[int(index2)][1]
```

```
    x_diff = float(x1 - x2)
    if (y1 == y2): y_diff = 0.1
    if (y1 < y2): y_diff = float(np.absolute(y1 - y2))
    if (y1 > y2):
        y_diff = 0.1

        print("Error: Facial feature located below chin.")

    return np.absolute(math.atan(x_diff/y_diff))
```

[ ]:
```
b=[]
def features(image, faceCascade):
    global data
    faces = faceCascade.detectMultiScale(image,
                                         scaleFactor = 1.1, #1.1
                                         minNeighbors = 9,
                                         minSize = (30, 30),
                                         flags = cv2.CASCADE_SCALE_IMAGE  )
    for (x, y, w, h) in faces:
        y=int(0.95*y)
        h=int(1.05*h)
        cv2.rectangle(image, (x, y), (x + w, y + h), (255, 255, 255), 2)
        dlib_rect = dlib.rectangle(int(x), int(0.95*y), int(x + w), int(y + 1.
↪05*h))
        detected_landmarks = predictor(image, dlib_rect).parts()
        landmarks = np.matrix([[p.x, p.y] for p in detected_landmarks])
        image_copy = image.copy()
        for idx, point in enumerate(landmarks):
            pos = (point[0, 0], point[0, 1])
            cv2.circle(image_copy, pos, 3, color=(255, 153, 0))
            p27 = (landmarks[27][0,0],landmarks[27][0,1])
            x = p27[0]
            y1 = p27[1]
            gray = 0
            diff = get_lum(image,x,y1,8,1,-1,gray)
                #print(diff)
            limit = diff-5

            while (diff > limit):
                    y1 = int(y1 - 1)
                    diff = get_lum(image,x,y1,6,1,-1,gray)
            cv2.circle(image_copy, (x,y1), 3, color=(255, 153, 0))
            plt.imshow(cv2.cvtColor(image_copy, cv2.COLOR_BGR2RGB))
            cv2.imwrite("agreene.jpg", image_copy)
                #plt.show()
            cv2.waitKey(0)
```

```
            lmark = landmarks.tolist()
            p68 = ((x,y1))
            lmark.append(p68)
            g=[]
            h=[]

            fwidth = d(lmark,0,16)
            fheight = d(lmark,8,68)
            c=(fheight/fwidth)
            jwidth = d(lmark,4,12)
            j=(jwidth/fwidth)

            hchinmouth = d(lmark,57,8)
            e=(hchinmouth/fwidth)
            ref = q(lmark,27,8)
            f=ref
            for k in range(0,17):
              if k != 8:
               theta = q(lmark,k,8)
               g.append(theta)
            x=np.array(g)
            for k in range(1,8):
              dist = d(lmark,k,16-k)
              h.append(dist/fwidth)
            y=np.array(h)
            z=np.concatenate([x,y])
            #k=[label_2,c,j,e,f]

            #feature=np.concatenate([k,z])


        feature=[diff,fwidth, fheight,c,jwidth,j,hchinmouth,e,ref]    # dfmaster
↪= pd.concat( df_t,    data)
        data_1=np.concatenate([feature,z])
        data=data_1.tolist()
        #print(data)
    return data
```

[ ]:
```
train_data ='E:/projects/Face shape classification/spyder/data/crop img/*'
faceCascade =  cv2.CascadeClassifier('C:/Users/2109902/My program/
↪face_feature_Randomforest/haarcascade_frontalface_default.xml')
predictor_path= 'C:/Users/2109902/My program/face_feature_Randomforest/
↪shape_predictor_68_face_landmarks.dat'

predictor = dlib.shape_predictor(predictor_path)
```

[ ]:
```
SIZE=224

train_images=[]
train_labels=[]
feature_train1=[]
for directory_path in glob.glob(train_data):
    label_1=directory_path.split("\\")[-1]
    print(label_1)
    for img_path in glob.glob(os.path.join(directory_path, "*.jpg")):


        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        img = cv2.resize(img, (SIZE, SIZE))
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        plt.imshow(img)
        features_1 = features(img ,faceCascade)
        print(label_1)
        face= faceCascade.detectMultiScale(img, 1.01,␣
 ↪9,minSize=(30,30),flags=cv2.CASCADE_SCALE_IMAGE)

        for (x,y,w,h) in face:
          y=int(0.95*y)
          h=int(1.05*h)
        images=img[x:x+w, y:y+h]
        images = cv2.resize(images, (SIZE, SIZE))
        train_images.append(images)
        feature_train1.append(features_1)       #Train _Image _features
        train_labels.append( label_1)
        # Train labels
        #images_1=images_1.tolist()
             # Train Images


train_label=np.array(train_labels)
train_images_1=np.array(train_images)
feature_train1=np.array(feature_train1)
```

[ ]:
```

```