

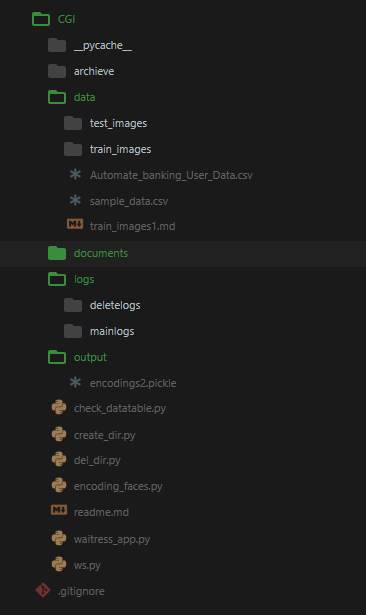
Face Recognition

Documentation

Akshay Nevrekar | Senior Developer - AI | 09-01-2019

# Project Structure

The Machine Learning/AI part of this project is encapsulated in CGI folder.



### archieve

It stores the files which were used while developing the project but no longer necessary to run it.

### data

It contains Data used to train and test model over the time to develop the project. Now we are using only ***train\_images*** to store the images captured while employee registration.

### Documents

It contains all the documents related to this project.

### Logs

As name suggests it saves the logs. **mainlog** directory records logs for webservices which are part of ***ws.py***

### Output

It stores the face-recognition model obtained from ***encoding\_faces.py*** in pickle format.

# Code Files:

# **Filename** : *encoding\_faces.py*

This scripts is backbone of this project. It trains the model on existing data (images captured during registration) and stores encoding into pickle file.

1. ***Packages Required***:

* imutils
* face\_recognition
* pickle
* cv2
* os

1. ***Code Blocks***

.

args = {'dataset': 'data/train\_images',

'encodings': 'output/encodings2.pickle',

'detection\_method': 'hog'}

imagePaths = list(paths.list\_images(args["dataset"]))

Dictionary **args** stores the path to the dataset, Path to store pickle file and detection method to be used in face\_recognition.

**for** **(**i**,** imagePath**)** **in** enumerate**(**imagePaths**):**

**print(**"[INFO] processing image {}/{}"**.**format**(**i **+** 1**,** len**(**imagePaths**)))**

name **=** imagePath**.**split**(**os**.**path**.**sep**)[-**2**]**

image **=** cv2**.**imread**(**imagePath**)**

rgb **=** cv2**.**cvtColor**(**image**,** cv2**.**COLOR\_BGR2RGB**)**

**try:**

boxes **=** face\_recognition**.**face\_locations**(**rgb**,** model**=**args**[**"detection\_method"**])**

encodings **=** face\_recognition**.**face\_encodings**(**rgb**,** boxes**)**

**del** boxes

**for** encoding **in** encodings**:**

knownEncodings**.**append**(**encoding**)**

knownNames**.**append**(**name**)**

**del** encodings**,** name

**except:**

**pass**

The above code iterate over imagepaths obtained from codeblock1.

First it fetches the label (which is directory name).

Then it reads the image and convert it into RGB (For OpenCV it is BGR by default).

In try block, 1st it fetches the locations of faces using detection method passed in args dictionary, Then it encodes the faces using face\_encoding.

The label and the encoding are stored in lists knownEncodings and knownEncodings.

**with** open**(**args**[**"encodings"**],** "wb"**)** **as** f**:**

f**.**write**(**pickle**.**dumps**(**data**))**

The above code creates a pickle file to write. The filename is passed in args dictionary and it writes data into it.

# **Filename** : *ws.py*

This script contains the Flask Web Services to be called by Android App.

1. ***Modules Required***

* flask
* face\_recognition
* pickle
* cv2
* pymongo
* pandas
* numpy
* json
* imutils
* os
* sys
* time
* requests
* skimage
* shutil
* datetime
* logging
* pyodbc
* logging

1. ***Code Blocks***

data **=** pickle**.**loads**(**open**(**"output/encodings2.pickle"**,** "rb"**).**read**())**

The above line loads the pretrained model using pickle.

## **Web Service 1**: To register the employee

**Link :** <http://fr.go4automate.com:8080/register>

**Input:**

* 4 imagefiles
* Id :- unique userid
* Authtoken: unique authentication token for that session
* Type: device from which service is called. Android=1, iOS=2, webapp=3

**Output:**

* {

'ReconStatus': True/False,

'Msg': ‘Message to be displayed’

}

**Objective**:

Register the new employee using images received for Face-Recognition and update the status in database.

**Code Blocks:**

client **=** pymongo**.**MongoClient**(**hostname**)**

db **=** client**.**FaceRecognition

coll **=** db**[**"Registration"**]**

coll**.**insert\_one**({**'userid:'**:** id\_**,** 'authtoken'**:** auth\_token**,** 'type'**:** type\_**,** 'timestamp'**:** timestamp**})**

It creates a connection with MongoDB and inserts the input data into collection called *Registration*.

**if** os**.**path**.**exists**(**new\_dir**):**

shutil**.**rmtree**(**new\_dir**)**

time**.**sleep**(**0.05**)**

os**.**makedirs**(**new\_dir**)**

Above code checks if directory already exists and if it exists then removes the directory and creates a new one.

**for** i**,**img **in** enumerate**(**img\_list**):**

offset **=** int**(**time**.**time**())**

save\_dir **=** new\_dir**+**"/imagefile"**+**str**(**i**)+**str**(**offset**)+**".jpg"

img**.**save**(**save\_dir**)**

It stores all input images into the directory created in previous block.

**for** i **in** imagePaths**:**

temp **=** cv2**.**imread**(**i**)**

rgb **=** cv2**.**cvtColor**(**temp**,** cv2**.**COLOR\_BGR2RGB**)**

boxes **=** face\_recognition**.**face\_locations**(**rgb**,** model**=**"hog"**)**

# print("Faces in image {}: {}".format(i, boxes))

**if** **not** boxes**:**

records **=** **{**'ReconStatus'**:** **False,** 'Msg'**:** 'Can not find face in the image'**}**

logger**.**debug**(**"[Face Detection]: Not able to find a face in the image."**)**

**return** json**.**dumps**(**records**)**

**elif** len**(**boxes**)** **>** 1**:**

records **=** **{**'ReconStatus'**:** **False,** 'Msg'**:** 'Multiple Faces detected in the image'**}**

logger**.**debug**(**"[Face Detection]: Multiple faces detected in the image."**)**

**return** json**.**dumps**(**records**)**

This code block finds the faces in the image using *face\_locations* using HOG(Histogram of Oriented Gradients) and the co-ordinates are stored in *boxes* variable.

If the boxes is empty it gives the messages as **Not able to find a face in the image**.

If the boxes contains more than one value, it returns **Multiple faces detected in the image**.

response **=** requests**.**post**(**"http://wsbanking.go4automate.com/MService.svc/UpdateFaceReconStatus"**,** json**={**"ReconStatus"**:** **True,** "authToken"**:** auth\_token**})**

response1 **=** json**.**loads**(**response**.**text**)**

**if** response1**[**'Response'**][**'Type'**]** **==** 'SUCCESS'**:**

records **=** **{**'ReconStatus'**:** **True,** 'Msg'**:** response1**[**'Response'**][**'Message'**]}**

**else:**

records **=** **{**'ReconStatus'**:** **False,** 'Msg'**:** response1**[**'Response'**][**'Message'**]}**

It calls the Web-service to update the FaceReconStatus in the Database.

And it stores the response received from web-service.

## **Web Service 2**: To Check-in

**URL**: <http://fr.go4automate.com:8080/faceparams>

**Input**: {

"path": "path to the image",

"id": unique user id(int)

"date": "12-27-2018" (date in mm-dd-yyyy)

}

**Output**: {Status: True/False}

**Objective**:

This web-service check if photo taken while checking-in matches with photos taken at registration.

**Code** **Block**:

image **=** io**.**imread**(**path**)**

image **=** cv2**.**resize**(**image**,** **(**512**,** 512**))**

rgb **=** cv2**.**cvtColor**(**image**,** cv2**.**COLOR\_BGR2RGB**)**)

First read the image from given path.

Resize the image into 512\*512 pixels and the convert it into RGB format.(By default cv2 uses BGR)

boxes **=** face\_recognition**.**face\_locations**(**rgb**,** model**=**'cnn'**)**

encodings **=** face\_recognition**.**face\_encodings**(**rgb**,** boxes**)**

Find the location of the faces using *cnn* (alternative is *hog*) and conver that into vectors using *face\_encodings.*

**for** encoding **in** encodings**:**

matches **=** face\_recognition**.**compare\_faces**(**data**[**'encodings'**],** encoding**,** tolerance**=**0.6**)**

name **=** "Unknown"

**if** **True** **in** matches**:**

matchedIdxs **=** **[**i **for** **(**i**,** b**)** **in** enumerate**(**matches**)** **if** b**]**

counts **=** **{}**

**for** i **in** matchedIdxs**:**

name **=** data**[**"names"**][**i**]**

counts**[**name**]** **=** counts**.**get**(**name**,** 0**)** **+** 1

name **=** max**(**counts**,** key**=**counts**.**get**)**

names**.**append**(**name**)**

Iterate over encodings extracted from previous block.

*compare\_faces* tries to find matching faces for the image and appends the names in the list.