Face recognition project

Note-: Please run this in Colab.

Facial recognition is a technology that is capable of recognizing a person based on their face. It employs machine learning algorithms which find, capture, store and analyze facial features in order to match them with images of individuals in a pre-existing database. Human face recognition system has two phases- one is to detect face, and another is to recognize face. Detection of face means to identify if the images contain any face or not and recognition of face means to identify the name of the detecting face.

This project is based on TensorFlow and TFlearn i.e., a model is created using these libraries. To create model, dataset (labelled data) is required which is generated using OpenCV and for visualization of output, another dataset (without labelling) is generated and this time, a model predicted the label (i.e., name) of that images.

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in
/usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in
/usr/local/lib/python3.7/dist-packages (from opencv-python) (1.21.6)
```

```
from IPython.display import display, Javascript, Image
from google.colab.output import eval_js
from base64 import b64decode, b64encode
import cv2
import numpy as np
import PIL
import io
import html
import time
```

```
# function to convert the JavaScript object into an OpenCV image
def js_to_image(js_reply):
  """
  Params:
          js_reply: JavaScript object containing image from webcam
  Returns:
          img: OpenCV BGR image
  """
  # decode base64 image
  image_bytes = b64decode(js_reply.split(',')[1])
  # convert bytes to numpy array
  jpg_as_np = np.frombuffer(image_bytes, dtype=np.uint8)
  # decode numpy array into OpenCV BGR image
  img = cv2.imdecode(jpg_as_np, flags=1)

  return img
```

```
# JavaScript to properly create our live video stream using our webcam
as input
def video_stream():
  js = Javascript('''
    var video;
    var div = null;
    var stream;
    var captureCanvas;
    var imgElement;
    var labelElement;

    var pendingResolve = null;
    var shutdown = false;

    function removeDom() {
       stream.getVideoTracks()[0].stop();
       video.remove();
       div.remove();
       video = null;
       div = null;
       stream = null;
       imgElement = null;
       captureCanvas = null;
       labelElement = null;
    }

    function onAnimationFrame() {
      if (!shutdown) {
        window.requestAnimationFrame(onAnimationFrame);
      }
      if (pendingResolve) {
        var result = "";
        if (!shutdown) {
          captureCanvas.getContext('2d').drawImage(video, 0, 0, 640,
480);
          result = captureCanvas.toDataURL('image/jpeg', 0.8)
        }
        var lp = pendingResolve;
        pendingResolve = null;
        lp(result);
      }
    }

    async function createDom() {
      if (div !== null) {
        return stream;
      }

      div = document.createElement('div');
      div.style.border = '2px solid black';
```

```
      div.style.padding = '3px';
      div.style.width = '100%';
      div.style.maxWidth = '600px';
      document.body.appendChild(div);

      const modelOut = document.createElement('div');
      modelOut.innerHTML = "<span>Status:</span>";
      labelElement = document.createElement('span');
      labelElement.innerText = 'No data';
      labelElement.style.fontWeight = 'bold';
      modelOut.appendChild(labelElement);
      div.appendChild(modelOut);

      video = document.createElement('video');
      video.style.display = 'block';
      video.width = div.clientWidth - 6;
      video.setAttribute('playsinline', '');
      video.onclick = () => { shutdown = true; };
      stream = await navigator.mediaDevices.getUserMedia(
          {video: { facingMode: "environment"}});
      div.appendChild(video);

      imgElement = document.createElement('img');
      imgElement.style.position = 'absolute';
      imgElement.style.zIndex = 1;
      imgElement.onclick = () => { shutdown = true; };
      div.appendChild(imgElement);

      const instruction = document.createElement('div');
      instruction.innerHTML =
          '<span style="color: red; font-weight: bold;">' +
          'When finished, click here or on the video to stop this
demo</span>';
      div.appendChild(instruction);
      instruction.onclick = () => { shutdown = true; };

      video.srcObject = stream;
      await video.play();

      captureCanvas = document.createElement('canvas');
      captureCanvas.width = 640; //video.videoWidth;
      captureCanvas.height = 480; //video.videoHeight;
      window.requestAnimationFrame(onAnimationFrame);

      return stream;
    }
    async function stream_frame(label, imgData) {
      if (shutdown) {
        removeDom();
        shutdown = false;
```

```
      return '';
    }

    var preCreate = Date.now();
    stream = await createDom();

    var preShow = Date.now();
    if (label != "") {
      labelElement.innerHTML = label;
    }

    if (imgData != "") {
      var videoRect = video.getClientRects()[0];
      imgElement.style.top = videoRect.top + "px";
      imgElement.style.left = videoRect.left + "px";
      imgElement.style.width = videoRect.width + "px";
      imgElement.style.height = videoRect.height + "px";
      imgElement.src = imgData;
    }

    var preCapture = Date.now();
    var result = await new Promise(function(resolve, reject) {
      pendingResolve = resolve;
    });
    shutdown = false;

    return {'create': preShow - preCreate,
            'show': preCapture - preShow,
            'capture': Date.now() - preCapture,
            'img': result};
  }
  ''')

  display(js)

def video_frame(label, bbox):
  data = eval_js('stream_frame("{}", "{}")'.format(label, bbox))
  return data
```

We are generating training and test data and storing them in google drive at locations-:

training data-:/content/drive/MyDrive/fr/data

test data-: /content/drive/MyDrive/fr/test

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
dir_name='/content/drive/MyDrive/fr'
dir_exists= os.path.isdir(dir_name)
if not dir_exists:
  os.mkdir(dir_name)
  print("making directory %s" %dir_name)
dir_name='/content/drive/MyDrive/fr/data'
dir_exists= os.path.isdir(dir_name)
if not dir_exists:
  os.mkdir(dir_name)
  print("making directory %s" %dir_name)
dir_name='/content/drive/MyDrive/fr/test'
dir_exists= os.path.isdir(dir_name)
if not dir_exists:
  os.mkdir(dir_name)
  print("making directory %s" %dir_name)
```

```
making directory /content/drive/MyDrive/fr/test
```

We will run a simple object detection algorithm called Haar Cascade on our video fetched from our webcam. OpenCV has a pre-trained Haar Cascade face detection model.

Please upload haarcascade_frontalface_default.xml alongside with this notebook at location /content/sample_data/

```
# initialize the Haar Cascade face detection model
face_cascade =
cv2.CascadeClassifier(cv2.samples.findFile(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml'))
```

In this project we are training model to successfully recognize user1 and user2.

```
users=[1,2]
```

Code to create training data. We will be generating 200 samples images of each person. Take note of how image names are being stored in the folder. It is "user.userid.sampleid.jpg". For example User 1 sample 1 image will be stored at location with name like "user.1.1.jpg".

Note-: It will wait for 20 seconds before starting video capturing for second user

```
from google.colab.patches import cv2_imshow
def generate_dataset():
  #face_classifier =
cv2.CascadeClassifier("/content/sample_data/haarcascade_frontalface_de
fault.xml")
  def face_cropped(img):
    #img= cv2.imread(img)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)

    if faces is ():
```

```
      return None
    for (x,y,w,h) in faces:
      cropped_face = img[y:y+h,x:x+w]
    return cropped_face

  for id in users:
    video_stream()
    # label for video
    label_html = 'Capturing...'
    # initialze bounding box to empty
    bbox = ''
    img_id=0
    while True:
      frame = video_frame(label_html, bbox)
      if not frame:
        break
      img = js_to_image(frame["img"])
      #print(img)
      if face_cropped(img) is not None:
        img_id +=1
        face = cv2.resize(face_cropped(img), (200,200))
        face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
        file_name_path =
"/content/drive/MyDrive/fr/data/user."+str(id)+"."+str(img_id)+".jpg"
        #file_name_path =
"/content/drive/MyDrive/fr/test/user"+str(img_id)+'.jpg'
        cv2.imwrite(file_name_path, face)
        cv2.putText(face, str(img_id), (50,50),
cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2 )

        cv2_imshow(face)
        print(img_id)
        if cv2.waitKey(1)==13 or int(img_id)==200:
          break
    time.sleep(20)

  #cap.release()
  #cv2.destroyAllWindows()
  print("Collecting samples is completed !!!")

generate_dataset()
```

Code to create test data. We will be generating 20 samples of each person for testing purpose. Take note of how image names are being stored in the folder. It is "user.sampleid.jpg". For example User 1 sample 1 image will be stored at location with name like "user.1.jpg".

Note-: It will wait for 20 seconds before starting video capturing for second user

```
from google.colab.patches import cv2_imshow
def generate_dataset():
```

```
  #face_classifier =
cv2.CascadeClassifier("/content/sample_data/haarcascade_frontalface_de
fault.xml")
  def face_cropped(img):
    #img= cv2.imread(img)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray)

    if faces is ():
      return None
    for (x,y,w,h) in faces:
      cropped_face = img[y:y+h,x:x+w]
    return cropped_face
  users=[1,2]

  for id in users:
    video_stream()
    # label for video
    label_html = 'Capturing...'
    # initialze bounding box to empty
    bbox = ''
    img_id=0
    while True:
      frame = video_frame(label_html, bbox)
      if not frame:
        break
      img = js_to_image(frame["img"])
      #print(img)
      if face_cropped(img) is not None:
        img_id +=1
        face = cv2.resize(face_cropped(img), (200,200))
        face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
        #file_name_path =
"/content/drive/MyDrive/fr/data/user."+str(id)+"."+str(img_id)+".jpg"
        file_name_path =
"/content/drive/MyDrive/fr/test/user"+str(img_id)+'.jpg'
        cv2.imwrite(file_name_path, face)
        cv2.putText(face, str(img_id), (50,50),
cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2 )

        cv2_imshow(face)
        print(img_id)
        if cv2.waitKey(1)==13 or int(img_id)==20:
          break
    time.sleep(20)

  #cap.release()
  #cv2.destroyAllWindows()
  print("Collecting samples is completed !!!")
```

```
generate_dataset()
```

```
import numpy as np # pip install numpy
```

Now we are creating target variable y. If it is user 1, y=[1,0] and if it is user 2, y=[0,1].

```
def my_label(image_name):
    name = image_name.split('.')[-3]
    l= len(users)
    #print(l)
    res= np.zeros(l)
    for i in range(l):
      if name==str(users[i]):
        res[i]=1
        #print(res)
        return res
```

```
import os
from random import shuffle
from tqdm import tqdm
```

Now we are reading and resizing images in data folder using cv2 libraries and appending image matrix and its lable in "data" list.

```
def my_data():
  data = []
  p="/content/drive/MyDrive/fr/data/"
  #print(os.listdir(path))
  for img in os.listdir(p):
    path=os.path.join(p,img)
    img_data = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    #print(img_data)
    img_data = cv2.resize(img_data, (50,50))
    data.append([np.array(img_data), my_label(img)])
  shuffle(data)
  return data
```

```
data = my_data()
```

We have total of 400 images. Out of which we are using 340 for training purpose and rest for validation purpose.

```
train = data[:340]
```

```
test = data[340:]
X_train = np.array([i[0] for i in train]).reshape(-1,50,50,1)
print(X_train.shape)
y_train = [i[1] for i in train]
X_test = np.array([i[0] for i in test]).reshape(-1,50,50,1)
print(X_test.shape)
y_test = [i[1] for i in test]
```

```
(340, 50, 50, 1)
(60, 50, 50, 1)
```

tflearn is not automatically installed for colab session and hence we need to first install tflearn libraries.

```
!pip install tflearn
```

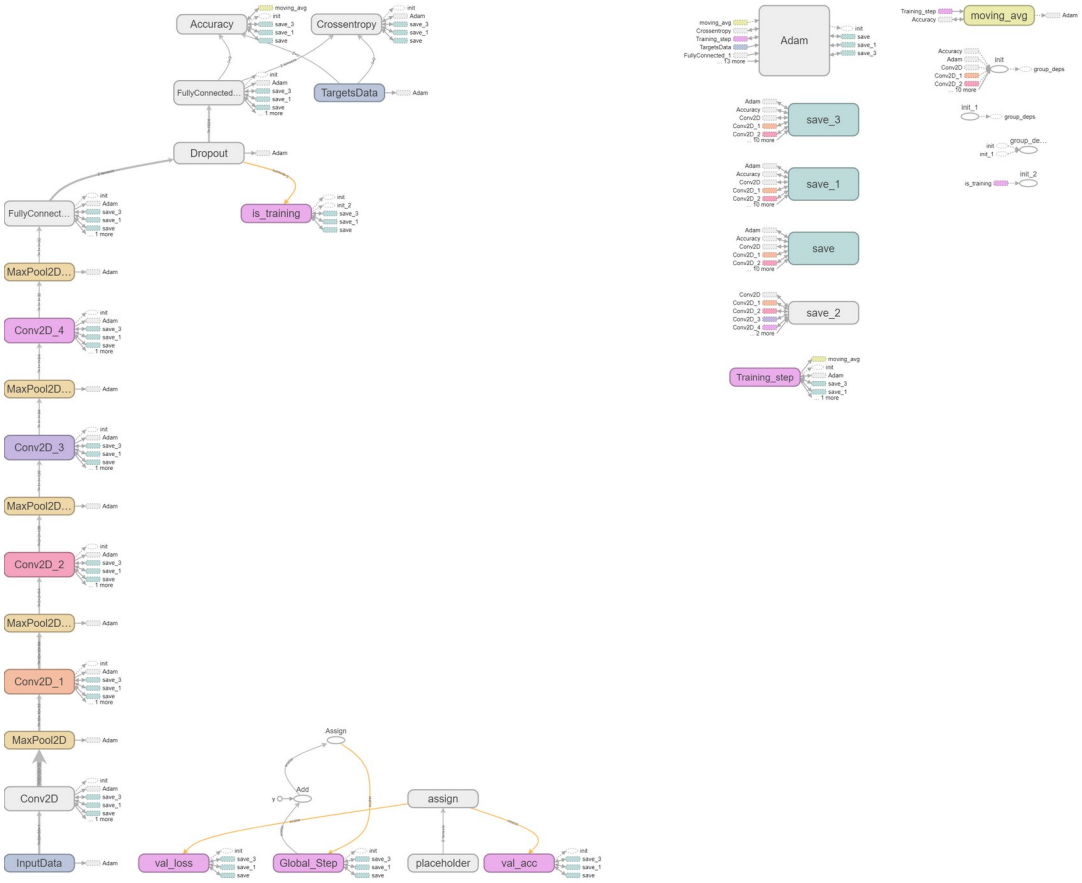
```
Collecting tflearn
  Downloading tflearn-0.5.0.tar.gz (107 kB)
ent already satisfied: numpy in /usr/local/lib/python3.7/dist-packages
(from tflearn) (1.21.6)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from tflearn) (1.15.0)
Requirement already satisfied: Pillow in
/usr/local/lib/python3.7/dist-packages (from tflearn) (7.1.2)
Building wheels for collected packages: tflearn
  Building wheel for tflearn (setup.py) ... e=tflearn-0.5.0-py3-none-
any.whl size=127299
sha256=5d50e3fba20b4c99c5e7c706140a85485c8c9c28cad3c1f7bba8377994c1044
1
  Stored in directory:
/root/.cache/pip/wheels/5f/14/2e/1d8e28cc47a5a931a2fb82438c9e37ef9246c
c6a3774520271
Successfully built tflearn
Installing collected packages: tflearn
Successfully installed tflearn-0.5.0
```

```
# import warnings
# warnings.filterwarnings('ignore')

import tensorflow as tf
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
```

```
WARNING:tensorflow:From
/usr/local/lib/python3.7/dist-packages/tensorflow/python/compat/v2_com
pat.py:107: disable_resource_variables (from
tensorflow.python.ops.variable_scope) is deprecated and will be
removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

We will be creating below model using tflearn libraries.




```
tf.compat.v1.reset_default_graph()
convnet = input_data(shape=[50,50,1])
convnet = conv_2d(convnet, 32, 5, activation='relu')
# 32 filters and stride=5 so that the filter will move 5 pixel or unit
at a time
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 64, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 128, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 64, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 32, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = fully_connected(convnet, 1024, activation='relu')
convnet = dropout(convnet, 0.8)
convnet = fully_connected(convnet, 2, activation='softmax')
convnet = regression(convnet, optimizer='adam', learning_rate = 0.001,
loss='categorical_crossentropy')
```

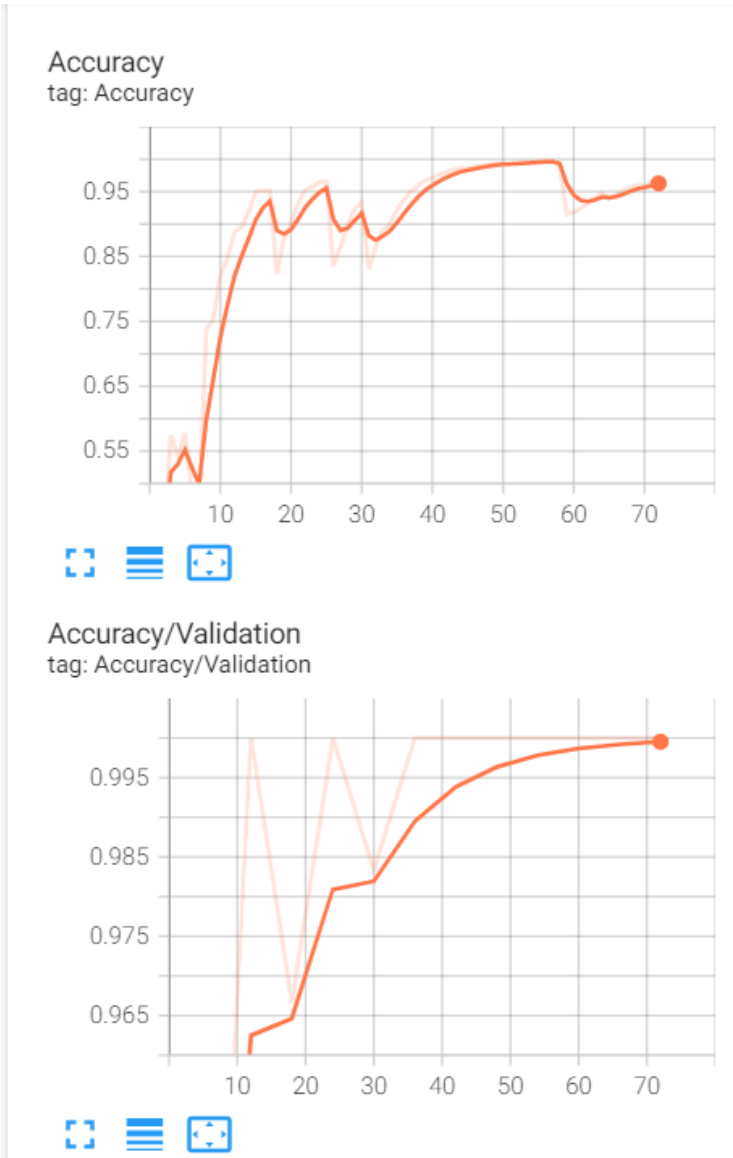
```
model = tflearn.DNN(convnet, tensorboard_verbose=0,
tensorboard_dir='/content/sample_data/tflearn_logs/')
```

We are now training the model on our data.

```
model.fit(X_train, y_train, n_epoch=12, validation_set=(X_test,
y_test), show_metric = True, run_id="FRS" )
```

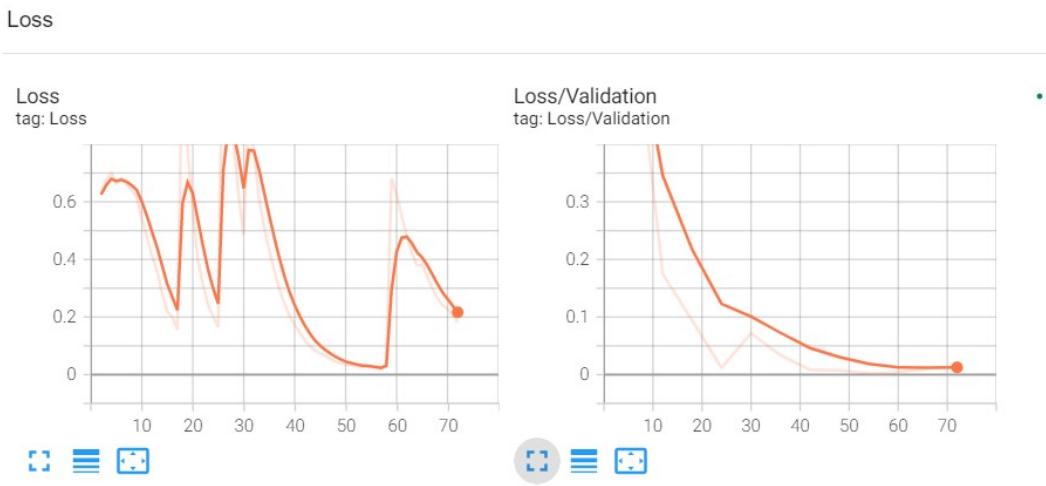
```
Training Step: 71  | total loss: 0.20464 | time: 0.769s
| Adam | epoch: 012 | loss: 0.20464 - acc: 0.9638 -- iter: 320/340
Training Step: 72  | total loss: 0.18223 | time: 1.989s
| Adam | epoch: 012 | loss: 0.18223 - acc: 0.9678 | val_loss: 0.01354
- val_acc: 1.0000 -- iter: 340/340
--
```

We can see below, loss on validation data converges and we are getting close to 99% accuarcy for our model.



Accuracy-:

Loss-:




```
%load_ext tensorboard
%tensorboard --logdir='/content/sample_data/tflearn_logs'
```

```
<IPython.core.display.Javascript object>
```

Now we test our model on data in test folder.

```
def data_for_visualization():
    Vdata = []
    p="/content/drive/MyDrive/fr/test/"
    #print(os.listdir(p))
    for img in tqdm(os.listdir(p)):
        path = os.path.join(p, img)
        img_num = img.split('.')[0]
        img_data = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img_data = cv2.resize(img_data, (50,50))
        Vdata.append([np.array(img_data), img_num])
    shuffle(Vdata)
    return Vdata

Vdata = data_for_visualization()
```

```
100%|██████████| 40/40 [00:08<00:00,  4.62it/s]
```

You can see below, our model successfully recognize sample picturres in test folder.

```
import matplotlib.pyplot as plt   # pip install matplotlib

fig = plt.figure(figsize=(20,20))
for num, data in enumerate(Vdata[:20]):
    img_data = data[0]
    y = fig.add_subplot(5,5, num+1)
    image = img_data
    data = img_data.reshape(50,50,1)
    model_out = model.predict([data])[0]
```

```
    if np.argmax(model_out) == 0:
        my_label = 'Divya'
    elif np.argmax(model_out) == 1:
        my_label = 'Falguni'

    y.imshow(image, cmap='gray')
    plt.title(my_label)

    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
plt.show()
```