NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

Department of Electronics and Communication Engineering

LEARNING ACTIVITY - 2

Subject – ANN

Title – using CNN and supporting flow chart implement the face recognition

Date of submission: 25/11/2020

Submitted By:

Adithya V Amberker (1NT18EC007)

Omkar Singh (1NT18EC108)

Sunil Kumar Behera (1NT18EC169)

Vikash Kumar (1NT18EC181)

5th Semester ECE

Submitted To:

Dr. Jayavrinda Vrindavanam
Associate Professor,
NMIT Bangalore.

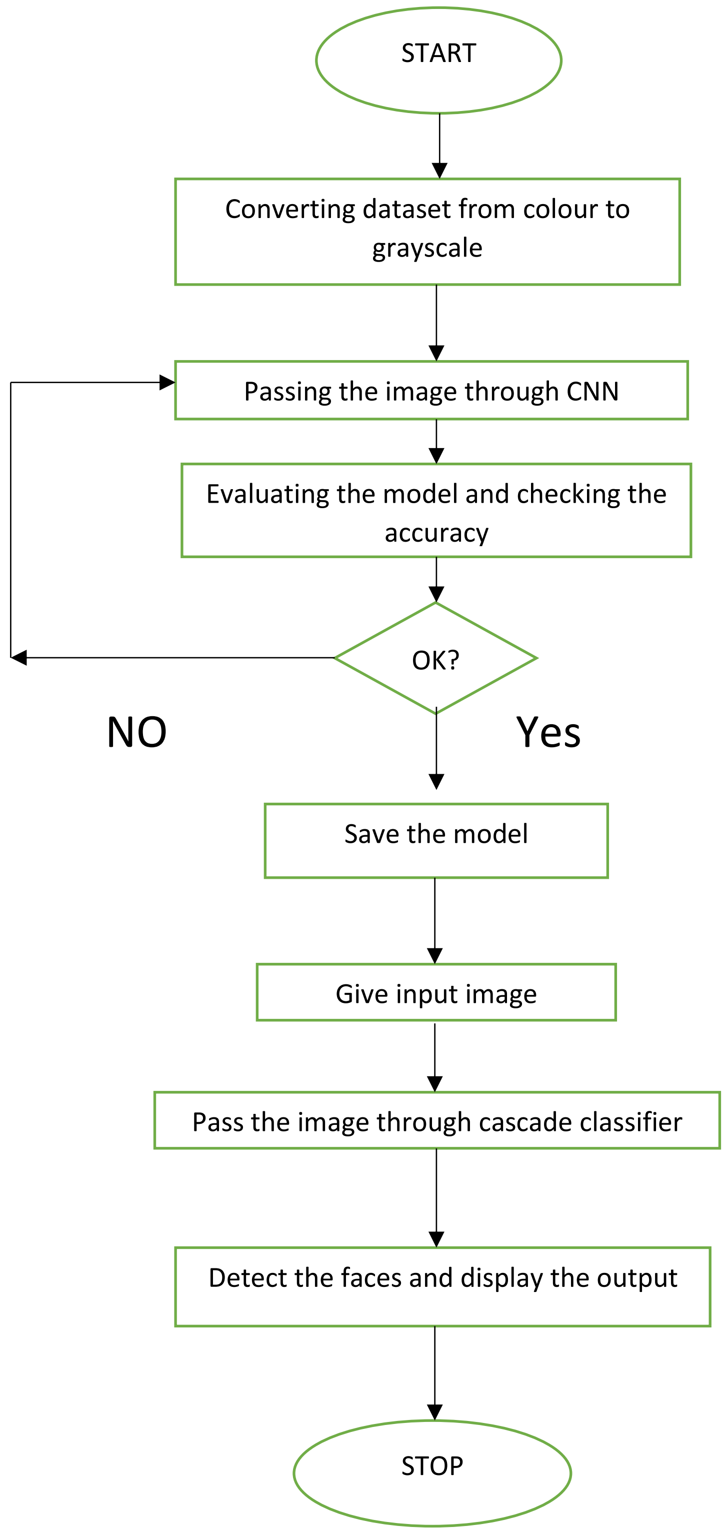
Introduction

In this modern world, the information technology has grown rapidly providing lots of information to the people. With this advancement the security of information has become increasingly serious. There are many application which are in need of identification technology to protect the user’s identity. Face recognition is one of such identification technology that can be used for recognizing faces. Several major industries have benefitted from the rapid advancements that have been made in Facial Recognition technology over the past 60 years and these include law enforcement, border control, retail, mobile technology and banking and finance.

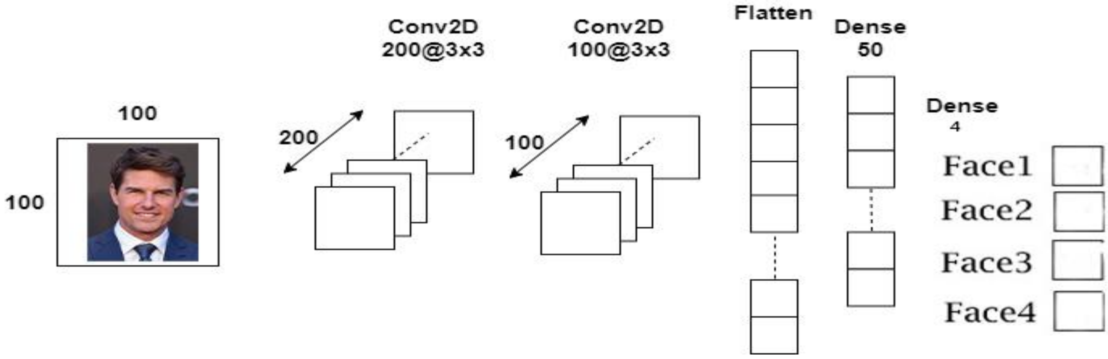
Face recognition is the process of identifying people from images. It has a high practical value for the detection and recognition of specific sensitive characters. Face detection is the pre-step for face recognition that is performed using Haar-like features. It uses Deep learning’s sub-field that is Convolutional Neural Network (CNN) to achieve face recognition. This project gives us an idea of some simple methods and algorithms that can be used to detect faces in images and how they can be identified or matched with a given face database.

Flow Chart

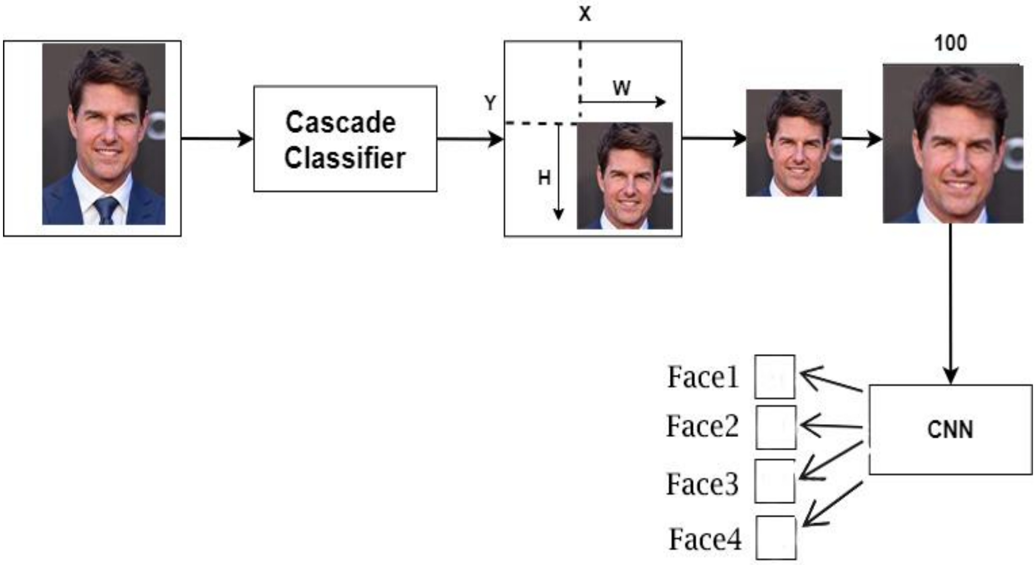
The project follows a step by step approach to detect the faces in the images



Design Flow



CNN Architecture



Dataset:

The dataset used is provided by the Kaggle website. The data consists of 959 cropped images of celebrities . The images are classified and saved in folders based on the celebrity.

Algorithms and libraries used

Convolutional Neural Network (CNN):

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign learnable weights and biases to various aspects/objects in the image and be able to differentiate one from the other.

The CNN architecture comprises of the input layer where the face images of same dimensions are fed. This is followed by a convolutional layer which consists of a kernel or a filter of a fixed size that slides in a window fashion to perform convolution operation on the windowed image to extract features. To overcome the uneven mapping with filter size padding is applied onto the size of input image. Activation function RELU is used which assigns zero value to hidden units. POOL refers to pooling layer, which is responsible for down sampling and dimensionality reduction. Max pooling and Average pooling are the two common function used. Flatten layers are introduced to convert the 2-D features into 1-D.

Dense layer is a fully connected layer where each neuron in the input is connected to each neuron in the output. This layer computes the score of a particular class giving N outputs. To prevent overfitting of CNN a fraction of inputs are dropped out by setting their values to 0 at each update during training. This is done using a Dropout layer. Softmax classifier is used to classify faces in the fully connected layer.

Haar Cascade:

Haar cascade is a machine learning object detection algorithm used to identify object in a image or video. It is based on the concept of features proposed by Paul Viola and Michael Jones in their paper “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001.

Initially, the algorithm needs a lot of positive images of faces and negative images without faces to train the classifier. Based on this training it is used to detect the faces in the other images.

Keras:

Keras is a neural networks library written in Python that is high-level in nature and runs on top of TensorFlow. It is extremely simple and intuitive to use. It acts as an interface for the TensorFlow library.

Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers etc. In addition to standard neural networks, it also supports convolutional neural networks. Utility layers like dropout, batch normalization and pooling are also supported.

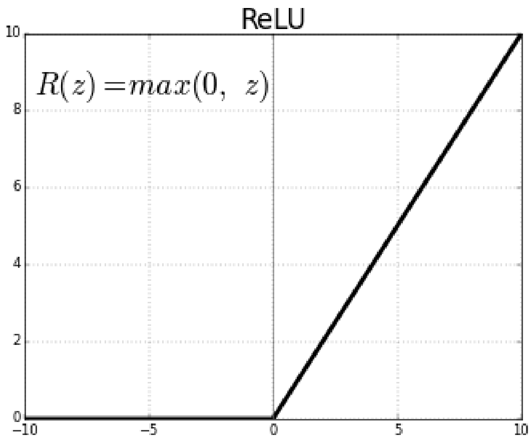
OpenCV:

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. It was built to provide a common infrastructure for computer vision applications. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects etc.

Activation Function(Relu):

The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

This means that the neurons will only be deactivated if the output of the linear transformation is less than 0



Model Evaluation

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 200)       2000
_________________________________________________________________
activation (Activation)      (None, 98, 98, 200)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 200)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 100)       180100
_________________________________________________________________
activation_1 (Activation)    (None, 47, 47, 100)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 100)       0
_________________________________________________________________
flatten (Flatten)            (None, 52900)             0
_________________________________________________________________
dropout (Dropout)            (None, 52900)             0
_________________________________________________________________
dense (Dense)                (None, 50)                2645050
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 204
=================================================================
Total params: 2,827,354
Trainable params: 2,827,354
Non-trainable params: 0
_________________________________________________________________
None
(None, 100, 100, 1)
```

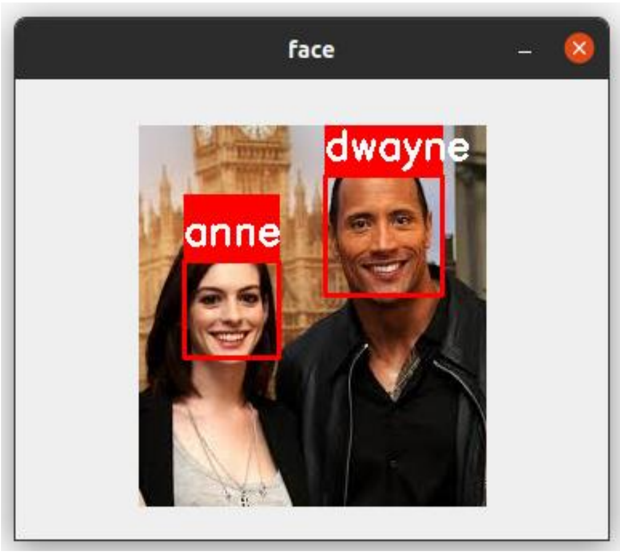
Loss= 0.5392

Accuracy=0.8854

Output:

Input given:

Output:



Conclusion & Future Improvements

We conclude from our project that the faces were recognized for the given input image and based on the model chosen.

For future improvements we can use it to safeguard user's information, to identify and find missing persons and help in Law enforcement.

CODE

In [1]:
```python
# Importing Libraries
from keras.models import Sequential
from keras.layers import Dense,Activation,Flatten,Dropout
from keras.layers import Conv2D,GlobalAveragePooling2D,MaxPooling2D
from keras.layers.normalization import BatchNormalization
from keras import optimizers
from keras.callbacks import ModelCheckpoint, History
from matplotlib import pyplot as plt
import cv2,os
import numpy as np
```

In [2]:
```python
# loading the dataset
data_path='dataset'
categories=os.listdir(data_path)
labels=[i for i in range(len(categories))] #empty dictionary

label_dict=dict(zip(categories,labels))

print(label_dict)
print(categories)
print(labels)
```

```
{'bill_gates': 0, 'dwayne_johnson': 1, 'anne_hathaway': 2, 'pins_Rihanna': 3}
['bill_gates', 'dwayne_johnson', 'anne_hathaway', 'pins_Rihanna']
[0, 1, 2, 3]
```

In [3]:
```
img_size=100
data=[]
target=[]


for category in categories:
    folder_path=os.path.join(data_path,category)
    img_names=os.listdir(folder_path)

    for img_name in img_names:
        img_path=os.path.join(folder_path,img_name)
        img=cv2.imread(img_path)

        try:
            # #Coverting the image into gray scale
            gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
            #resizing the gray scale into 100x100
            resized=cv2.resize(gray,(img_size,img_size))
            #appending the image and the label(categorized) into the list (dataset)
            data.append(resized)
            target.append(label_dict[category])


        except Exception as e:
            print('Exception:',e)
            #if any exception rasied, the exception will be printed here. And pass to the next image
```

In [4]:
```
# normalize the imaages
data=np.array(data)/255.0

# reshaping to 4D array
data=np.reshape(data,(data.shape[0],img_size,img_size,1))

target=np.array(target)

from keras.utils import np_utils

new_target=np_utils.to_categorical(target)
```

In [5]:
```
# saving the data and target
np.save('data',data)
np.save('target',new_target)
```

In [6]:
```
# loading the save numpy arrays
data=np.load('data.npy')
target=np.load('target.npy')
```

In [7]:
```python
# neural network architecture

model=Sequential()

model.add(Conv2D(200,(3,3),input_shape=data.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#The first CNN layer followed by Relu and MaxPooling layers

model.add(Conv2D(100,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
#The second convolution layer followed by Relu and MaxPooling layers

model.add(Flatten())
model.add(Dropout(0.5))
#Flatten layer to stack the output convolutions from second convolution layer
model.add(Dense(50,activation='relu'))
#Dense layer of 64 neurons
model.add(Dense(4,activation='softmax'))
#The Final layer with two outputs for two categories

print(model.summary())
print(model.input.shape)

model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 200)       2000
_________________________________________________________________
activation (Activation)      (None, 98, 98, 200)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 200)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 100)       180100
_________________________________________________________________
activation_1 (Activation)    (None, 47, 47, 100)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 100)       0
_________________________________________________________________
flatten (Flatten)            (None, 52900)             0
_________________________________________________________________
dropout (Dropout)            (None, 52900)             0
_________________________________________________________________
dense (Dense)                (None, 50)                2645050
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 204
=================================================================
Total params: 2,827,354
Trainable params: 2,827,354
Non-trainable params: 0
_________________________________________________________________
None
(None, 100, 100, 1)
```

In [8]:
```
# splitting the dataset into 90% training and 10% testing
from sklearn.model_selection import train_test_split

train_data,test_data,train_target,test_target=train_test_split(data,target,test_size=0.1)
```

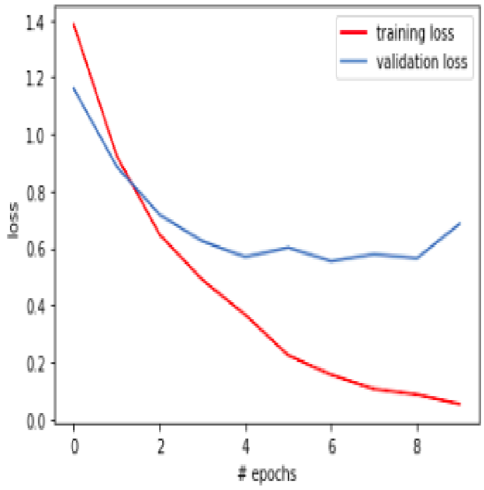
In [9]:
```
checkpoint = ModelCheckpoint('model-{epoch:03d}.model',monitor='val_loss',verbose=0,save_best_only=True,mode='auto'

# Train the neural network
history=model.fit(train_data,train_target,epochs=10,callbacks=[checkpoint],validation_split=0.2)
```

```
Epoch 1/10
22/22 [==============================] - ETA: 0s - loss: 1.3832 - accuracy: 0.3449WARNING:tensorflow:From /home/sun
il/anaconda3/lib/python3.8/site-packages/tensorflow/python/training/tracking/tracking.py:111: Model.state_updates
(from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
WARNING:tensorflow:From /home/sunil/anaconda3/lib/python3.8/site-packages/tensorflow/python/training/tracking/track
ing.py:111: Layer.updates (from tensorflow.python.keras.engine.base_layer) is deprecated and will be removed in a f
uture version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied automatically.
INFO:tensorflow:Assets written to: model-001.model/assets
22/22 [==============================] - 22s 1s/step - loss: 1.3832 - accuracy: 0.3449 - val_loss: 1.1592 - val_acc
uracy: 0.5665
Epoch 2/10
22/22 [==============================] - ETA: 0s - loss: 0.9234 - accuracy: 0.6290INFO:tensorflow:Assets written t
o: model-002.model/assets
22/22 [==============================] - 22s 1s/step - loss: 0.9234 - accuracy: 0.6290 - val_loss: 0.8861 - val_acc
uracy: 0.6301
Epoch 3/10
22/22 [==============================] - ETA: 0s - loss: 0.6473 - accuracy: 0.7493INFO:tensorflow:Assets written t
o: model-003.model/assets
22/22 [==============================] - 23s 1s/step - loss: 0.6473 - accuracy: 0.7493 - val_loss: 0.7171 - val_acc
uracy: 0.6994
Epoch 4/10
22/22 [==============================] - ETA: 0s - loss: 0.4880 - accuracy: 0.8261INFO:tensorflow:Assets written t
o: model-004.model/assets
22/22 [==============================] - 23s 1s/step - loss: 0.4880 - accuracy: 0.8261 - val_loss: 0.6239 - val_acc
uracy: 0.7283
Epoch 5/10
22/22 [==============================] - ETA: 0s - loss: 0.3652 - accuracy: 0.8638INFO:tensorflow:Assets written t
o: model-005.model/assets
22/22 [==============================] - 23s 1s/step - loss: 0.3652 - accuracy: 0.8638 - val_loss: 0.5690 - val_acc
uracy: 0.7977
Epoch 6/10
22/22 [==============================] - 22s 994ms/step - loss: 0.2234 - accuracy: 0.9246 - val_loss: 0.6019 - val_
accuracy: 0.7803
Epoch 7/10
22/22 [==============================] - ETA: 0s - loss: 0.1547 - accuracy: 0.9609INFO:tensorflow:Assets written t
o: model-007.model/assets
22/22 [==============================] - 23s 1s/step - loss: 0.1547 - accuracy: 0.9609 - val_loss: 0.5534 - val_acc
uracy: 0.7919
Epoch 8/10
22/22 [==============================] - 22s 997ms/step - loss: 0.1051 - accuracy: 0.9696 - val_loss: 0.5782 - val_
accuracy: 0.8439
Epoch 9/10
22/22 [==============================] - 22s 994ms/step - loss: 0.0849 - accuracy: 0.9739 - val_loss: 0.5639 - val_
accuracy: 0.8266
Epoch 10/10
22/22 [==============================] - 22s 1s/step - loss: 0.0527 - accuracy: 0.9884 - val_loss: 0.6858 - val_acc
uracy: 0.8266
```

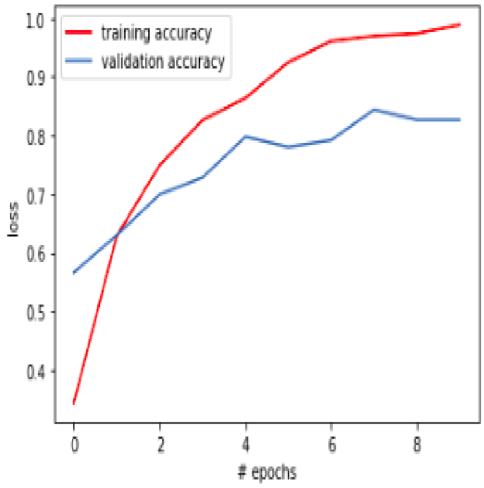
In [10]:
```
model.save('model-009.model')
```

```
INFO:tensorflow:Assets written to: model-009.model/assets
```

In [11]:
```
# plotting training and validation loss
plt.plot(history.history['loss'],'r',label='training loss')
plt.plot(history.history['val_loss'],label='validation loss')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



In [12]:
```
# plotting training and validation accuracy
plt.plot(history.history['accuracy'],'r',label='training accuracy')
plt.plot(history.history['val_accuracy'],label='validation accuracy')
plt.xlabel('# epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



In [13]:
```
print(model.evaluate(test_data,test_target))
```

```
3/3 [==============================] - 1s 179ms/step - loss: 0.5392 - accuracy: 0.8854
[0.539158284664154, 0.8854166865348816]
```

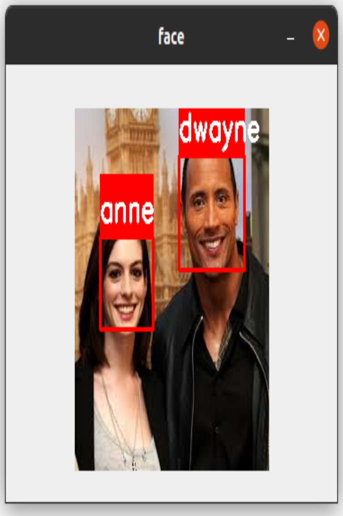
In [1]:

```python
# importing libraries
import cv2
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
import time
from keras.models import load_model
```

In [2]:

```python
# Load the saved model
model = load_model('model-009.model')

# Create the haar cascade
face_clsfr=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

img = cv2.imread('download.jpeg') # Read the image

labels_dict={0: 'bill',1: 'dwayne',2: 'anne',3:'Rihanna'}
color_dict={0:(0,0,255),1:(0,0,255),2:(0,0,255),3:(0,0,255)}
```

```python
In [*]: #Converting the image to gray scale
gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

# Detect faces in the image
faces=face_clsfr.detectMultiScale(gray,1.3,5)

# Draw a rectangle around the faces
for (x,y,w,h) in faces:


        face_img=gray[y:y+w,x:x+w]
        resized=cv2.resize(face_img,(100,100))
        normalized=resized/255.0
        reshaped=np.reshape(normalized,(1,100,100,1))
        result=model.predict(reshaped)

        label=np.argmax(result,axis=1)[0]

        cv2.rectangle(img,(x,y),(x+w,y+h),color_dict[label],2)
        cv2.rectangle(img,(x,y-40),(x+w,y),color_dict[label],-1)
        cv2.putText(img, labels_dict[label], (x, y-10),cv2.FONT_HERSHEY_SIMPLEX,0.8,(255,255,255),2)

# display the image
cv2.imshow('face',img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



Reference:

https://github.com

www.kaggle.com

www.youtube.com

www.realpython.com