

# Paper 2: Computer Vision: Face Recognition and Object Detection

Submission To: Dr. Mohsen Behesthi  
CSC500-Research Methods  
Muhammad Zaid Kamil & Siddharda Kavuri

**Abstract**— The goal of CV is to use systems to interpret and understand visual complex problems. By using digital media, we want the machine to accurately identify and classify objects. Some computer vision problems are trivial for human beings, such as object and face detection. In this collaborative paper will give an ideal approach for different techniques involved in computer vision. We will first present a general introduction of computer vision and impact on our lives. Then the report will be split into two sections (section 2 and 3), where section 2 will dive into the Research paper 1 will explore Face recognition, which will give a discussion on the methodology of RCNN algorithm and compare two libraries. In section 3 we will explore the Research paper 2, which brief overview of how the object recognition works and how YOLO proposes to tackle in real time. In the final section, based on the knowledge gained from the papers, we will compare two detection models and implement few face recognition and detection models using our digital images/video. Finally, we will conclude the paper.

**Keywords:** Computer Vision, Object Detection, Face Recognition, YOLO, RCNN, OpenCV, Dlib, Implementation, Approaches

## I. INTRODUCTION

Computer vision is a field of Artificial Intelligence and Machine Learning. We want computers to sense media (digital images/videos) similar to human perception. As humans, we are trained a lifetime to tell objects apart, and recognize face from incoming people. We use our retina and optic neurons to discern objects and recognize face. This is where computer vision comes, where cv trains system to perform this function (face detection and face recognition) with less time and with the limit of cameras, data and algorithms. Although thinking and behaving like human might not be the pinnacle of vision that humans have for a machine, teaching computers how to see is the frontier that could not be ignored. The paper 1 will explore another topic in the field of Computer vision: Face recognition, which we will dive into the methodology of Face recognition algorithm. The Paper 2 will explore the methods that are proposed and used before YOLO are decent at recognizing objects, but they came short of recognizing them in the real time. Paper 1 which was retrieved from the IEEE Explore Journal which was written from researchers in Lviv Polytechnic National University, Ukraine. The paper 2 which was retrieved from pjrredie which was written from researchers from University of Washington, Facebook AI research team.

## II. SECTION: RESEARCH PAPER 1

### A. Introduction

The research paper 1 topic is the evaluation and comparison of software for Face recognition. This paper will investigate into two main computer vision libraries: OpenCV and dlib. Which will explore the algorithms used for face recognition based on the library. Explore the features, time complexity of the two libraries. The paper also explores two applications for face recognition from the two libraries.

#### 1) Computer Vision applications

One of the popular computer vision problems is phone recognition/biometrics measurement. Initially biometrics was limited to the uniqueness of finger scan to classify individuals. For example, if we talk about smartphone devices, initially users used pin code to unlock their personal devices, then smart phones included an additional unlocking system which uses finger scanner. Now in recent smart phone products, users use face recognition to unlock, and pay from their personal devices. From the figure below we can represent the progress of biometric system in phone.



Fig. 1. Progress of phone biometric system for screen lock

#### 2) Computer Vision Libraries

The result of computer vision applications resulted in formation of different types of computer vision libraries and APIs which developed solution in different computer vision areas in this case for face recognition. The paper will compare the performance using these two libraries (OpenCV and dlib) and build two simple recognition systems. The libraries are mainly written in C++ and python programming language.

The best way to get acquainted with a particular library is to look at the documentation. For OpenCV the documentation contains modules and examples from face analysis to texture mapping. Each module gives a description of the functions used. And the example program presents the C++ code and screenshot of the program. Documentation of other computer vision libraries such as DeepFace (written in python) are in presented in open source libraries like GitHub.

## B. Methodology of Algorithm

This section will go in depth of the four steps for the face recognition algorithm: From searching the face to recognizing and identifying the face.

### 1) Step 1: Search the face in digital images/video.

This step is crucial because if the system skips a face and regard another object as a face, then the recognition will give false results. Using the HOG Algorithm. The system can detect the face.

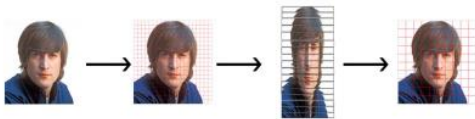


Fig. 2. Visualization of using the HOG algorithm for Face detection

The basic idea of HOG is dividing the image into small connected cells. Then for each cell it computes histogram. Then it combines all histograms to form a final feature vector. Where the vector is unique for each face. The disadvantage of HOG it doesn't work on faces at different angles. Advantage of HOG is to search for faces in official documents: Driving license, passport photo.

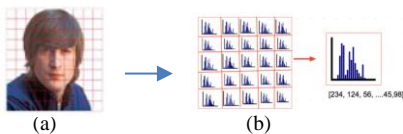


Fig. 3. (a) grid cell for image[1] (b) computes histogram for each cell[2]

The above Figure 3 (b) brings all the histograms together to form feature vector, which is unique for every face, in the above example the vector = [234, 124, 56, ...45, 98][2]. This vector is unique to this particular image and face.

### 2) Step 2: Determine the face position

After finding the face, the task of the system is to position it. Most of the media the face is angular to the lens (side face position), and not center to the lens. Most efficient way to solve this to recognize face is to use the face landmark estimation

algorithm. Where we have to identify key points/landmarks in the face and represent with a numeric number.



Fig. 4. 68 landmark points in face using Landmark estimation algorithm[1]

From the figure above shows the total 68 landmarks, detecting 68 points in the face which include: mouth, eyes, nose and face structure. Where each landmark is represented by a numeric number from 1-68.

### 3) Step 3: Determine the unique facial features

#### 3.1) Method 1: Database for face recognition

To distinguish among faces, need to determine unique features. There are two methods to distinguish between faces. The first method to recognize faces is to compare new face with all the other faces stored in the pre-defined database of face images. In each recognition compares to the set of images in database, hoping to find the same face. If there is a large amount of data in the database would be a better option to store the database in the cloud rather than the system. However, it can cause delays between request and replies from the server refer Fig 5(b). This method is beneficial if the task involves identifying a small group of people in the database.

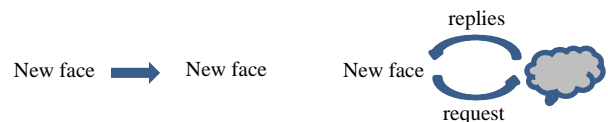


Fig. 5. (a) grid cell for image (b) computes histogram for each cell

#### 3.1) Method 2: Training the D-CNN Neural Network

The second method uses training method such as neural networks DCNN, which will be trained to identify 128 unique facial features.

Step 1: In order to train the DCNN neural network, we first need to upload three images:



Figure 6: Three representation of Images uploaded to the network[1]

In the above Figure 6, The two images uploaded is of the same person (Picture No 1 and Picture No 2) and the third image (Picture No 3) is another person's face.

Step 2: The neural network will adjust the results get 128 characteristics of Image No1 and No2 are as close as possible and the Image No3 will be different.

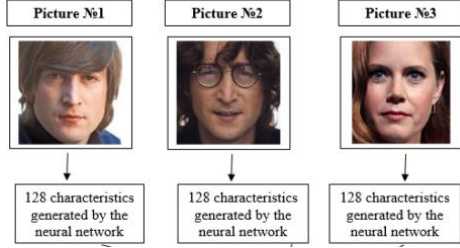


Figure 7: Neural network getting 128 characteristics of Image [1]

Step 3: The process of training neural network to generate numerical features of the face is complex process and requires huge database of faces. We use existing features to train the data set. As soon as neural network is trained, it will be able to generate unique characteristics for any face person. The neural network only needs to be trained once.

Step 4: Compare the existing data from previous step, 128 features of the uploaded faces with the data about the people. The Support Vector Machine (SVM) is used to check if the data coincide[1]. If the data is homogeneous, the more accurate to determine the face. The linear SVM uses a regression analysis model.

$$(\vec{x}_1, y_1), \dots, (\vec{x}_n, y_n)$$

Where y is 1 or -1, which indicates a class xi which the the point belongs.

The example given from the paper is as follows:

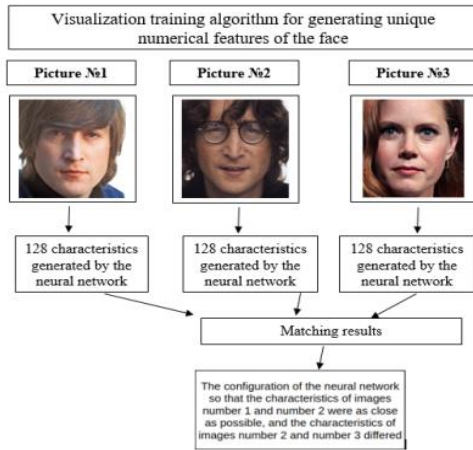




















Figure 8: Visualization of the neural learning algorithm to check face match[1]

Table 1. Comparisons of the pseudocode, results and time for detection using OpenCV and dlib libraries [1]

Libraries	Pseudocode	Results												
OpenCV	<pre>Face_detector_opencv.py: # Import the system and OpenCV libraries import sys import cv2 from skimage import io # Call the basic function library to find faces lbp_face_cascade = cv2.CascadeClassifier('data/ lbpcascade_frontalface.xml') # Go through all entered images for images in sys.argv[1:]:     print("Processing image: {}".format(images))     image = io.imread(images)     faces_detected_img = detect_faces(lbp_face_cascade, image, scaleFactor=1.2)     io.imshow('face_recognition.jpg', image)     print("Number of faces detected: {}".format ( len(faces) ) ) # Print the result plt.imshow(convertToRGB(faces_detected_img)) #Wait for "Enter" button to continue cv2.waitKey(0)</pre>	<table><tr><th></th><th>Experiment №1</th><th>Experiment №2</th><th>Experiment №3</th></tr><tr><td>OpenCV</td><td></td><td></td><td></td></tr><tr><td>Recognition on time (sec.)</td><td>0.5652</td><td>0.0954</td><td>0.6121</td></tr></table>		Experiment №1	Experiment №2	Experiment №3	OpenCV				Recognition on time (sec.)	0.5652	0.0954	0.6121
	Experiment №1	Experiment №2	Experiment №3											
OpenCV														
Recognition on time (sec.)	0.5652	0.0954	0.6121											
Dlib	<pre>Face_detector_dlib.py: # Import the system and dlib libraries import sys import dlib from skimage import io # Call the basic function library to find faces face_detector = dlib.get_frontal_face_detector() # Go through all entered images for images in sys.argv[1:]:     print("Processing image: {}".format(images))     image = io.imread(images)     faces_detected_image= face_detector (image, 1)     io.imshow('face_recognition.jpg', faces_detected_image)     print("Number of faces detected: {}".format ( len( faces_detected_image) ) ) # Print the result win = dlib.image_window() win.clear_overlay() win.set_image(image) win.add_overlay(dets) #Wait for "Enter" button to continue dlib.hit_enter_to_continue()</pre>	<table><tr><th></th><th>Experiment №1</th><th>Experiment №2</th><th>Experiment №3</th></tr><tr><td>dlib</td><td></td><td></td><td></td></tr><tr><td>Recognition on time (sec.)</td><td>1.6584</td><td>0.2163</td><td>1.6889</td></tr></table>		Experiment №1	Experiment №2	Experiment №3	dlib				Recognition on time (sec.)	1.6584	0.2163	1.6889
	Experiment №1	Experiment №2	Experiment №3											
dlib														
Recognition on time (sec.)	1.6584	0.2163	1.6889											

The above table compares the pseudocode, and time spent to search for the face using two face detection systems, one is from the OpenCV library and second from the dlib libraries. The following pseudocode was written in python for both libraries. As from the results it can be seen the OpenCV was the more effective library since the recognition time is almost three times faster compared to the algorithm from dlib library.

### III. SECTION: RESEARCH PAPER 2

#### A. Introduction

YOLO is short for You Only Look Once. There are multiple versions of YOLO algorithm that are released after the first paper, written by Joseph Redmon et al., later named v1. After v3, its development and optimization are being done by others after Joseph Redmon stopped working on YOLO, citing its potential misuse [4].

YOLO, as the name suggests, gets the output of the image with only a single convolution network [3]. This output

produces multiple bounding boxes along with the class probability the object belongs to

Real-Time Detectors	Train	mAP	FPS
100Hz DPM	2007	16.0	100
30Hz DPM	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
Less Than Real-Time			
Fastest DPM	2007	30.4	15
R-CNN Minus R	2007	53.5	6
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG-16	2007+2012	73.2	7
Faster R-CNN ZF	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Fig. 1. Progress of phone biometric system for screen lock [3]

Popular systems before YOLO used various approaches, like the use of sliding window as in the case of Deformable parts model (DPM) [5]. Other popular system, RCNN estimates the potential bounding boxes using region proposal methods at the start. Then these potential grids are used to localize the search for the object. The various bounding boxes that are then obtained are then cleared to finally localize the object [4]. Many of the systems split the recognition process into various sub processes, trained separately, which in reality are complex to implement and slow to run [3].

Author claims that this algorithm produced results at the speed of 45 FPS, with mAP of 63.4, when trained on Pascal VOC 2007 dataset with a Titan X GPU [3]. This is astounding when you compare the speed of recognition with the accuracy it produces as shown in the figure Fig. 1.a

### B. Unified Detection

The image is divided into  $S \times S$  grids. These grids are used to denote the center of the object, in the case of the prediction. Each grid can have  $B$  bounding boxes in its model, for predicting multiple objects of the same class or of different classes.

Each of these bounding boxes has 4 variable predictions ( $x$ ,  $y$ ,  $w$ ,  $h$ ) and an object confidence  $C$ . The variables ( $x$ ,  $y$ ) represent the coordinates of object's mid-point relative to the bounds of the grid cell,  $w$  is the width of object and  $h$  is the height of the object relative to the image.

$$\Pr(\text{Class}_i | \text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

Each cell has its own confidence  $C$ , that there is probability that there is an object present in that given cell, given by  $\Pr(\text{Class}_i | \text{Object})$ . The class probability of the grid is predicted by multiplying the conditional class probabilities and the individual box confidence predictions. As each grid is responsible for  $B$  bounding boxes, that would result in lot of overlapping bounding boxes as shown in the figure

#### 1) Non-Max suppressions

The naïve way to clear the bounding boxes would be to take the bounding box with the highest class probability, but if there are multiple objects of the same class in the image, it would clean up the bounding box of the other object. To avoid this, overlapping bounding boxes can be cleaned by using non-max suppression. In non-max suppression the value called Intersection of Union is calculated by taking bounding box, given by the training data, and the overlapping bounding boxes as shown in fig 2.

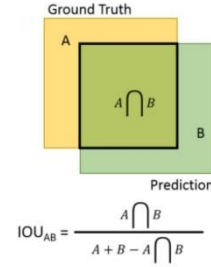


Fig. 2. Overlapping bounding boxes IOU [3]

The bounding boxes with the overlap area greater than some threshold, then all the bounding boxes are replaced with the bounding box that has the greatest class probability in all the bounding boxes under consideration. This process is shown in fig3.

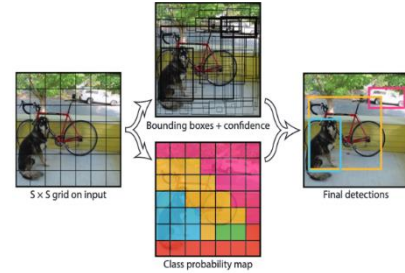


Fig. 3. Process of class probability using YOLO[3]

### C. Network design and training

For evaluation of YOLO on PASCAL VOC dataset, we take convolutional matrix of  $7 \times 7$ ,  $B = 2$ , and the dataset has labelled class size  $C = 20$ , which would output a tensor of size  $7 \times 7 \times 30$  [1], as shown in fig 4.

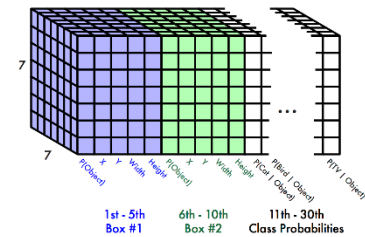




Fig. 4[3]

A neural network with 24 convolutional layers and 2 fully connected layers. The image resolution is doubled from 224 x 224 to 448 x 448. The network is shown in the image below.

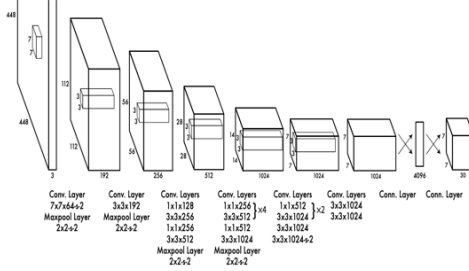


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating  $1 \times 1$  convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution ( $224 \times 224$  input image) and then double the resolution for detection.

Fig. 5. Detection network[3]

The convolutional layers extract the features in the image whereas the fully connected layers will give us the coordinates and the probability of the object by relying on the convolutional layer output.

To get the non-linearity in the network after every layer, a leaky rectified linear activation function is used.

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

An image is fed into the network and an output is generated. To optimize the output against the training data, a loss function used is as follows

$$\begin{aligned} & \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\ & + \lambda_{nobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{ij}^{nobj} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \sum_{c \in \text{classes}} \mathbb{I}_{ij}^{obj} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

Where, the sum squared error of corresponding feature is taken for the loss function. To reduce the effect of grids without objects and increase the effect of grids with objects on the network,  $\lambda_{coord}$  of value 5 and  $\lambda_{nobj}$  of value 0.5 is used. To reduce the effect large images would have on the network than the small images, square root of both parameters  $w, h$  is taken.

The dataset chosen, PASCAL VOC, is ran for 120 epochs with a batch size of 64, momentum 0.9 and decay of 0.0005 [3]. The

system learning rate is set to increase from  $10^{-3}$  to  $10^{-2}$  for the first 75 epoch to keep the diverge rate low in the start, then increased to  $10^{-3}$  for the next 30 epochs and then to  $10^{-4}$  for the next 30 epochs [3]. Data set is increased by scaling images, rotating, and translating images [3].

#### IV. IMPLEMENTATION AND COMPARISON OF OBJECT DETECTION AND FACE RECOGNITION

##### 1) Comparison of detection tools: YOLO and R-CNN

Based on the knowledge gained from the face recognition and object detection tools from the two research papers, we can make a comparison on the type of algorithm used, advantages and disadvantages between the face detection and face recognition.

When we want to make a comparison, we can compare between the tools used for Object detection and face recognition: YOLO and RCNN. The paper 2 provides a section for comparison of YOLO to Other real time computer vision systems. We can represent the comparison on Table 2 as shown below. Real time detectors are detectors which can perform object/face detection in real live video.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM	2007	16.0	100
30Hz DPM	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
Less Than Real-Time			
Fastest DPM	2007	30.4	15
R-CNN Minus R	2007	53.5	6
Fast R-CNN	2007+2012	70.0	0.5
Faster R-CNN VGG-16	2007+2012	73.2	7
Faster R-CNN ZF	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Figure 1: YOLO compared to other real time systems [3]

Based on the above Figure 4, the borderline for the fps between real time and less than real time is 45 FPS. From the Table in Figure 4 we identify that the fps of YOLO is 90 times faster compared to the Fast R-CNN detection system. The mAP is the mean average precision which is a popular metric for accuracy for object detection. It is calculated based on the following equation:  $mAP = TP / (TP + FP)$ , where  $TP$  = True Positive and  $FP$  = False positive.

Table 1: Comparison of YOLO and RCNN

	YOLO	R-CNN
Approach	Detects a variety of objects simultaneously.	Extracts regions of interest and features from object
Similar	Forms each grid cell and proposes bounding boxes using convoluted features	
Speed (FPS)	45 [3]	0.5 (Fast R-CNN) [3]
Accuracy (mAP)	63.4 [3]	70 (Fast R-CNN) [3]
Advantages	Fast object detection methods. Can provide real time detection	Can verify the identity of face is same or different

Disadvantages	Has spatial constraints on bounding box. Each bounding box can predict one object type.	Use computational power to train the neural network. Some face recognition model may be inaccurate when verifying face.
---------------	---	---

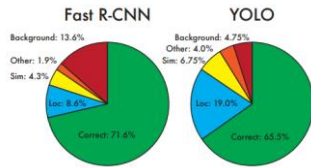


Figure 2: Error Analysis Fast R-CNN vs YOLO

The above Figure compares the Error analysis between Fast R CNN and YOLO. YOLO struggles to localize objects correctly. Fast R-CNN makes much fewer localization errors (8.6%) but far more background errors (13.6%) compared to YOLO. Fast R-CNN is almost 3x (13.6%) more likely to predict background error compared to than YOLO (3.6%). We can make combination of Fast RCNN with YOLO, to give significant performance boost. The mAP of Fast RCNN when combined increases by 3.2% from 70% to 75.0%.

## 2) Implementation of Face recognition and Face Detection tools

### 2.1) Implementation of Face recognition

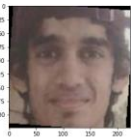
We were able to implement the code to test our own images for face recognition. For the Face recognition we used DeepFace library written on python on jupyter notebook and for Face Detection we used OpenCV written in C++. The source for the projects is mentioned in the references below.

```
In [38]: from deepface import DeepFace
import matplotlib.pyplot as plt #face recognition task using DeepFace
import pandas as pd


In [40]: img1_path = 'deepface/deepface/tests/dataset/img1.jpeg'
img2_path = 'deepface/deepface/tests/dataset/img7.jpg'

In [41]: img1 = DeepFace.detectFace(img1_path)
img7 = DeepFace.detectFace(img2_path)

In [42]: plt.imshow(img1)
Out[42]: <matplotlib.image.AxesImage at 0x211f032da0>
```



```
In [43]: plt.imshow(img7)
Out[43]: <matplotlib.image.AxesImage at 0x211e409240>
```



```
In [44]: model_name = 'Facenet' #using Facenet model

In [45]: resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name)
1/i [=====] - 0s 80ms/step
1/i [=====] - 0s 71ms/step

In [46]: resp #true image pair is same person. Facial recognition task
Out[46]: {'verified': False,
'distance': 1.03030801028072,
'threshold': 0.4,
'model': 'Facenet',
'detector_backend': 'opencv',
'similarity_metric': 'cosine'}
```

```
In [47]: model_name = 'Arcface' #using ArcFace model


In [48]: resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name) #image pair as input
1/i [=====] - 2s 2s/step
1/i [=====] - 0s 127ms/step

In [49]: resp #true image pair is same person. Facial verify function have 01 time complexity
Out[49]: {'verified': False,
'distance': 1.026108064780356,
'threshold': 0.68,
'model': 'Arcface',
'detector_backend': 'opencv',
'similarity_metric': 'cosine'}
```


Figure 3: Comparison of Face recognition model using Jupyter notebook integrated python kernel

The above Figure 3 shows the pseudocode which was obtained from DeepFace repository [6]. The objective was to determine to verify two face images are the same or different. We first import the DeepFace model from the library. For each of the face image we input the exact image paths stored in the PC database. Plt.imshow is a function that uses OpenCV to extract the face image 1 and 2 from the picture and removes the background. We use the function DeepFace.verify checks if the two face images are same or different. We compared the efficiency of two face recognition models using Facenet and ArcFace. We can see there is a lower coefficient when using the Facenet model therefore it is more efficient.

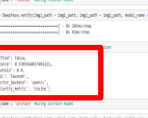
```
In [40]: plt.imshow(img1)
Out[40]: <matplotlib.image.AxesImage at 0x27280f50f0>
```



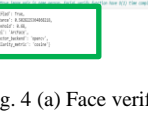
```
In [40]: plt.imshow(img2)
Out[40]: <matplotlib.image.AxesImage at 0x27280f4250>
```



```
In [41]: plt.imshow(img1)
Out[41]: <matplotlib.image.AxesImage at 0x27280f4250>
```



```
In [41]: plt.imshow(img2)
Out[41]: <matplotlib.image.AxesImage at 0x27280f4250>
```



```
In [42]: resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name)
1/i [=====] - 0s 80ms/step
1/i [=====] - 0s 71ms/step

In [43]: resp #true image pair is same person. Facial recognition task
Out[43]: {'verified': False,
'distance': 1.03030801028072,
'threshold': 0.4,
'model': 'Facenet',
'detector_backend': 'opencv',
'similarity_metric': 'cosine'}
```

```
In [44]: resp = DeepFace.verify(img1_path = img1_path, img2_path = img2_path, model_name = model_name)
1/i [=====] - 2s 2s/step
1/i [=====] - 0s 127ms/step

In [45]: resp #true image pair is same person. Facial verify function have 01 time complexity
Out[45]: {'verified': False,
'distance': 1.026108064780356,
'threshold': 0.68,
'model': 'Arcface',
'detector_backend': 'opencv',
'similarity_metric': 'cosine'}
```

Fig. 4 (a) Face verify for Image 1 (b) Face verify for Image 2

In the above Figure a and b, when we input two images of the same face. The FaceNet model shows the face verification of the two same person face images are not similar which is contradictory results. However, the ArcFace model shows the face verification is similar for both the images of the same person. This shows that ArcFace is the more accurate model when detecting similarity of faces. The incorrect face recognition could be due to the image lighting, and blur which can affect the system to verify the face image is the same. The best approach is to check the effectiveness of a face recognition model is to test the comparison with multiple different images.

## 2.2) Implementation of Face detection

To implement face detection, we used the face cascade from Haar Cascade algorithm from OpenCV library code. The code was written in C++ using Visual Studio IDE. The source for the projects is mentioned in the references below [7].

```

1  #include <opencv2/imgcodecs.hpp>
2  #include <opencv2/highgui.hpp>
3  #include <opencv2/imgproc.hpp>
4  #include <opencv2/objdetect.hpp>
5  #include <iostream>
6  using namespace cv;
7  using namespace std;
8
9  void main() {
10     string path = "Resources/CSC500Team4.png";
11     Mat img = imread(path);
12     CascadeClassifier faceCascade;
13     faceCascade.load("Resources/haarcascade_frontalface_default.xml");
14
15     if (faceCascade.empty()) { cout << "No File" << endl; }
16
17     vector<Rect> faces;
18     faceCascade.detectMultiScale(img, faces, 1.1, 10);
19
20     for (int i = 0; i < faces.size(); i++)
21     {
22         rectangle(img, faces[i].tl(), faces[i].br(), Scalar(255, 0, 255), 3);
23     }
24     imshow("Image", img);
25     waitKey(0);
26 }

```

Fig. 5 Implementation of Face Detection using Haar Cascade Algorithm

From the above Figure 5, we first imported the opencv library. Under the main function. We have to input the image path from the directory folder. We use the faceCascade algorithm from the CascadeClassifier. To form the bounding boxes, we use a for loop. If Face is detected, a rectangular bounding box is formed. Then the image is showed using imshow with a wait time of infinity since waitKey(0).

The results of the code when run in Visual studio is as follow:

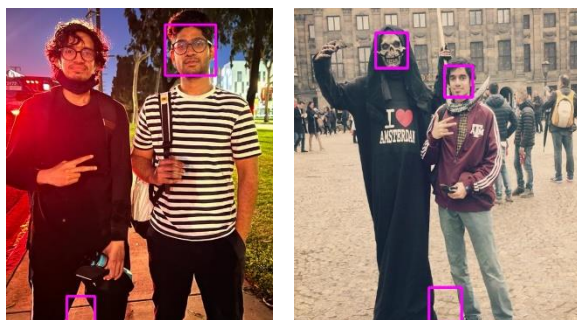


Fig. 6 (a) Face detection for Image 1 (b) Face detection for Image 2

As we can observe from Fig 6 (a) and 6 (b). The Cascade algorithm was not able to detect one of the faces in the image. However, was able to detect my colleague face. In Fig 6(b), the cascade algorithm was more efficient as was able to detect both faces in the image.

## V. CONCLUSION

In In this paper, we have presented two research papers that is the main subset of computer vision: Face recognition and Object detection. Then we compared the two detection models: YOLO and R-CNN and we did some of our own implementation of Face recognition and Face detection. One of the main companies that have initiated and invested into computer vision research is IBM. As we discussed in the paper, there can be mistakes from the system when detecting objects, faces and verifying similarity of faces. The system cannot be as perfect as human beings vision, which we take for granted. However, with emerging libraries with greater accurate models we can make it as close as possible. Each of the detection model is good for its use case: the YOLO detection is good for real time object detection and the R-CNN approach is good for face recognition if trained the model well. The Haar Cascade does the job for face detection. Each of the algorithm is specific for its own use case. The best approach would be to combine the detection methods which will provide a faster detection time with a more precise detection rate.

## REFERENCES

- Research Paper 1:
- [1] N. Boyko, O. Basystiuk and N. Shakhovska, "Performance Evaluation and Comparison of Software for Face Recognition, based on Dlib and Opencv Library", IEEE, 2018. Available: <https://ieeexplore.ieee.org/document/8478556>. [Accessed 28 June 2022]. Website 1:
  - [2] S. Jack, "Face detection using dlib HOG", *Medium*, 2022. [Online]. Available: <https://medium.com/mlcrunch/face-detection-using-dlib-hog-198414837945#> [Accessed: 28- Jun- 2022].
- Research Paper 2:
- [3] Redmon, S. Divvala and R. Girshick, "You Only Look Once: Unified, Real-Time Object Detection", Allen Institute of AI, 2022. Available: <https://arxiv.org/pdf/1506.02640v5.pdf>. [Accessed 28 June 2022].
  - [4] F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ra- manan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.
  - [5] Girshick, J. Donahue, T. Darrell, and J. Malik. Rich fea- ture hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR)*, 2014 IEEE Conference on, pages 580–587. IEEE, 2014.
- Face Recognition and Face Detection code retrieved from:
- [6] <https://github.com/serengil/deepface>
  - [7] <https://www.computervision.zone/courses/opencv-cv/>