**Realtime Custom Object Detection & Recognition**

Implementation

***Submitted By:***

***Arul Raj (1001704417)***

***Divya Saxena (1001773376)***

Table of Contents

[1 Implementation Process Flow 3](#_Toc36162129)

[2 Python Libraries used 4](#_Toc36162130)

[3 Celebrity Face Recognition Implementation 4](#_Toc36162131)

[3.1 Steps followed 4](#_Toc36162132)

[3.2 Why Convolutional Neural Networks? 4](#_Toc36162133)

[3.3 Keras 5](#_Toc36162134)

[3.4 VGG-Face Model 5](#_Toc36162135)

[3.5 Nn4.small2.v1 Model – Face Alignment 5](#_Toc36162136)

[3.6 Loss – Squared L2 Distance 6](#_Toc36162137)

[3.7 What should be the Distance Threshold? 7](#_Toc36162138)

[3.8 Face Recognition – Measuring the Accuracy 8](#_Toc36162139)

[3.9 Dataset Visualization 9](#_Toc36162140)

[3.10 References decided on 10](#_Toc36162141)

[3.11 What we have implemented from the reference? 10](#_Toc36162142)

[3.12 What different we have done from the reference? 11](#_Toc36162143)

[4 COCO Realtime Multiple Object Detection using YOLO 11](#_Toc36162144)

[4.1 Steps followed 11](#_Toc36162145)

[4.2 References decided on 12](#_Toc36162146)

[4.3 What we have implemented from the reference? 13](#_Toc36162147)

[4.4 What different we have done from the reference? 13](#_Toc36162148)

[5 Realtime Single Object Detection 13](#_Toc36162149)

[6 Project technical Requirements 17](#_Toc36162150)

[7 Project Structure Outline 17](#_Toc36162151)

[8 Summary 19](#_Toc36162152)

# **Implementation Process Flow**

In Implementation, we have developed 3 parts of the Realtime object detection and recognition,

1. *Celebrity Face Recognition in real-time videos*

* Input will be a video for a celebrity
* The network (CNN) is trained with this celebrity
* Output will be the video with bounding boxes and labels around the celebrity images.​

1. *COCO real-time Multiple object detection using YOLO*

* Input will be an image or a video of common objects (COCO)
* YOLO is trained with the COCO dataset.
* Output will be the image or video with bounding boxes and labels around multiple objects detected.​

1. *Realtime Single object detection of input from webcam*

* Input will be an image, or a video of an object
* Object will be captured using webcam by drawing bounding boxes
* The network YOLO would have been trained with that in the process.
* Output will be the image or video with confidence levels.

The technologies/libraries used in the implementation are outlined in the next section of the document.

The process flow is outlined as,

* Supervised Training is executed, that is the training data is labeled using the YAT Annotation tool (as per the previous submission 2)
* The input to the model will be an image or a video not from the training dataset.
* The output will be the image or video having the bounding boxes with the labels around the corresponding object recognized as a part of this process.

# **Python Libraries used**

* OpenCV
* Dlib
* Matplotlib
* NumPy
* Sklearn
* Keras
* Tensorflow
* Imutils

# **Celebrity Face Recognition Implementation**

The celebrity Face Recognition is implemented using Python libraries Keras, Dlib and OpenCV.

## **Steps followed**

The whole process for face recognition using Keras can be divided in four major steps:

1. Detect/ Identify faces in an image using Dlib and OpenCV
2. Convert image into grayscale and crop into 200X200 pixels
3. Design convolutional neural network using Keras
4. Train the model on the test model on testing data

ConvNets derive their name from the “convolution” operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. It preserves the spatial relationship between pixels by learning image features using small squares of input data.

## **Why Convolutional Neural Networks?**

Convolutional Neural Networks (ConvNets or CNNs) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification.

There are four main operations in the ConvNet:

1. Convolution
2. Non-Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

These operations are the basic building blocks of every Convolutional Neural Network.

Some of the important concepts are underlined,

***What is an Image?***  
An Image is a matrix of pixel values. Essentially, every image can be represented as a matrix of pixel values.

***What is a Channel?***  
Channel is a conventional term used to refer to a certain component of an image.

An image from a standard digital camera will have three channels – red, green and blue – one can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

***Why we are using the grayscale image?***  
A grayscale image, on the other hand, has just one channel.

We will only consider grayscale images, so we will have a single 2d matrix representing an image. The value of each pixel in the matrix will range from 0 to 255 – zero indicating black and 255 indicating white.

## **Keras**

Keras is a high-level library, used specially for building neural network models. It is written in Python and is compatible with both Python – 2.7 & 3.5.

Keras was specifically developed for fast execution of ideas. It has a simple and highly modular interface, which makes it easier to create even complex neural network models. This library abstracts low level libraries, namely Theano and TensorFlow so that, the user is free from “implementation details” of these libraries.

## **VGG-Face Model**

We have used the VGG-Face model. Briefly, the VGG-Face model is the same Neural Net architecture as the VGG16 model used to identity 1000 classes of object in the ImageNet competition.

The VGG16 name simply states the model originated from the Visual Geometry Group and that it was 16 trainable layers.

The main difference between the VGG16-ImageNet and VGG-Face model is the set of calibrated weights as the training sets were different.

The model architecture is a linear sequence of layer transformations of the following types:

* Convolution + ReLU activations
* MaxPooling
* SoftMax

## **Nn4.small2.v1 Model – Face Alignment**

The nn4.small2.v1 model was trained with aligned face images, therefore, the face images from the custom dataset must be aligned too.

Here, we have used Dlib for face detection and OpenCV for image transformation and cropping to produce aligned 96x96 RGB face images.

That is,

A person smiling for the camera

Description automatically generated

## **Loss – Squared L2 Distance**

We calculate the squared L2 distance and get the result as,

A close up of a person

Description automatically generated

Here, the distance between the two images of Jacques Chirac is smaller than the distance between an image of Jacques Chirac and an image of Gerhard Schröder (0.30 < 1.12). But we still do not know what distance threshold τ is the best boundary for making a decision between *same identity* and *different identity*.

## **What should be the Distance Threshold?**

To find the optimal value for τ, the face verification performance needs to be evaluated on a range of distance threshold values.

At a given threshold, all possible embedding vector pairs are classified as either *same identity* or *different identity* and compared to the ground truth. Since we’re dealing with skewed classes (much more negative pairs than positive pairs), we use the ***F1 score as evaluation metric instead of accuracy***.

A close up of a map

Description automatically generated

The face verification accuracy at τ = 0.56 is 95.7% which is not bad given a baseline of 89% for a classifier that always predicts *different identity* but since nn4.small2.v1 is a relatively small model it is still less than what can be achieved by state-of-the-art models (> 99%).

A screenshot of a cell phone

Description automatically generated

## **Face Recognition – Measuring the Accuracy**

Now if we have an estimate of the distance threshold τ, face recognition is as simple as calculating the distances between an input embedding vector and all embedding vectors in a database.

* The input is assigned the label (i.e. identity) of the database entry with the smallest distance if it is less than τ or label *unknown* otherwise.
* It also supports one-shot learning, as adding only a single entry of a new identity might be sufficient to recognize new examples of that identity.]

But we have used a more robust approach for labelling the input using KNN classification with a Euclidean distance metric and a linear support vector machine (SVM) trained with the database entries and used to classify i.e. identify new inputs.

Note:- For training these classifiers we use 50% of the dataset, for evaluation the other 50%.

***The accuracy for celebrity recognition using 1. K-NN 2. Support Vector Machines (SVM) is 96% and 98% respectively***, as shown in the screenshot below

A screenshot of a computer

Description automatically generated

## **Dataset Visualization**

To embed the dataset into 2D space for displaying identity clusters, t-distributed Stochastic Neighbor Embedding (t-SNE) is applied to the 128-dimensional embedding vectors.

Except from a few outliers, identity clusters are well separated.

A picture containing photo, large, group, people

Description automatically generated

## **References decided on**

The reference we have followed for general ideas is about “Person Finder”. The link has been outlined below.

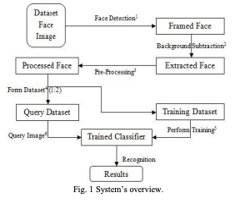
<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&cad=rja&uact=8&ved=2ahUKEwjXi_L1jbvnAhVGIqwKHTJlDNQQFjABegQIAhAB&url=https%3A%2F%2Farxiv.org%2Fpdf%2F1302.6379&usg=AOvVaw3T9G28b-P1ECXauRR-dskp>

we have also used the below references for guidelines in the implementation of this.

<paste the reference links here>

## **What we have implemented from the reference?**

* From the reference, we implemented the below flowchart signifying the process of doing the face detection and recognition.



* We used the concept of Support Vector Machine (SVM) classifier.

## **What different we have done from the reference?**

We have extended the implementation for the celebrity faces recognition using the neural network nn4.small2.v1 with Keras.

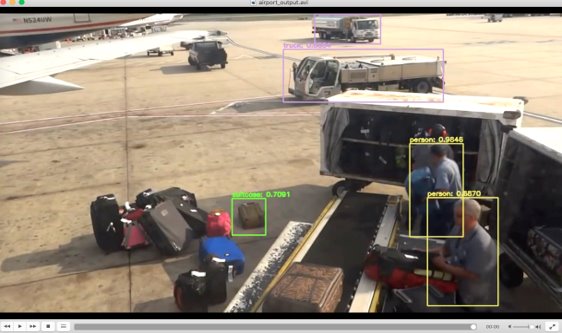
# **COCO Realtime Multiple Object Detection using YOLO**

## **Steps followed**

The steps followed for implementation:

* The COCO dataset is already pre-processed, therefore no need for pre-processing.
* The dataset images or videos will be given as input to YOLO network that has been trained on COCO dataset.
* The videos for input will be taken from the /videos folder in the project directory.
* The output video with the bounding boxes with labels will be stored in the folder /output in the project directory.





## **References decided on**

The reference we have followed for general ideas is about “Object detection using YOLO”. The link has been outlined below.

<https://pjreddie.com/media/files/papers/YOLOv3.pdf>

we have also used the below references for guidelines in the implementation of this.

<paste the reference links here>

## **What we have implemented from the reference?**

We have used the below concepts in our implementation,

* Bounding box prediction
* Class Prediction
* Feature extractor (CNN)

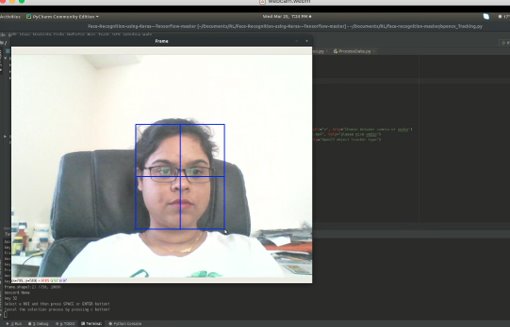
## **What different we have done from the reference?**

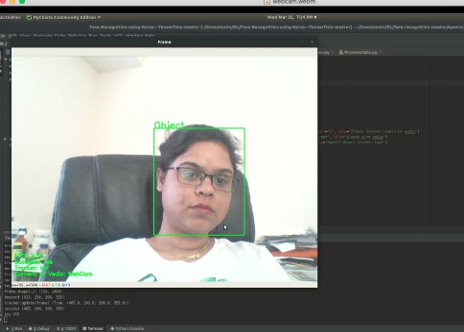
As an extension, we implemented it for the single object detection by drawing the bounding boxes (using bounding box predictions) over the inputs captured directly from the webcam.

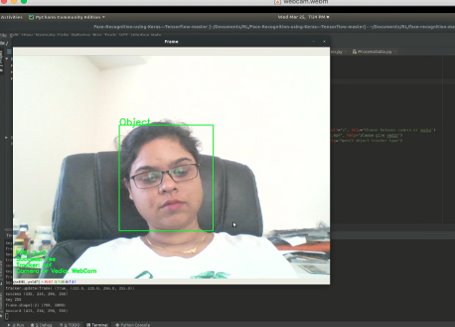
# **Realtime Single Object Detection**

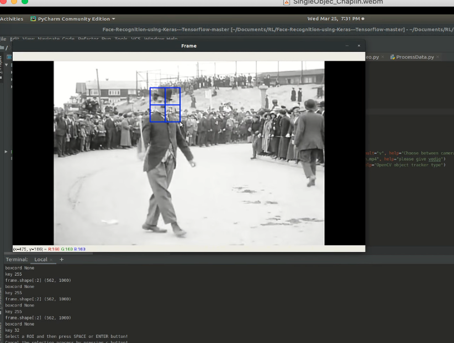
The steps followed for implementation:

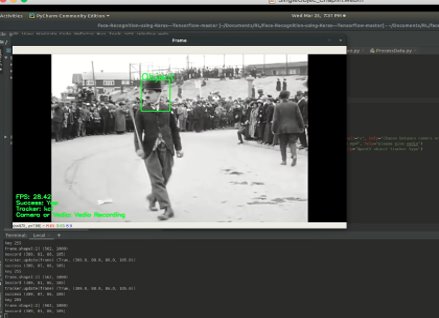
* The input will be captured by drawing the bounding boxes around the object to detect in real-time captured by the webcam.
* The input will be the video or webcam (camera) whichever argument we supply at runtime.
* The videos for input will be taken from the /videos folder in the project directory.
* We can also change the trackers by supplying the relevant tracker IDs as outlined below,
  + csrt": cv2.TrackerCSRT\_create
  + kcf": cv2.TrackerKCF\_create
  + boosting": cv2.TrackerBoosting\_create
  + mil": cv2.TrackerMIL\_create,
  + tld": cv2.TrackerTLD\_create
  + medianflow": cv2.TrackerMedianFlow\_create
  + mosse": cv2.TrackerMOSSE\_create









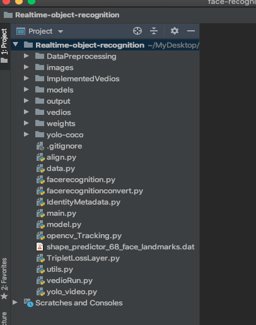


# **Project technical Requirements**

The modules/libraries listed under the “Requirements.txt” needs to be fulfilled for running the code. Please refer the project directory.

# **Project Structure Outline**

The Realtime Object Recognition project structure is as follows:



The folders and their usage is outlined here:

* ***DataPreprocessing***:
  + Last submission folder
  + Contains the annotated images of the celebrities.
* ***Images***:
  + Contains 10 images of each of the 10 celebrities.
  + Celebrities are:
    - Ariel Sharon
    - Arnold Schwarzenegger
    - Colin Powell
    - Donald Rumsfeld
    - George W Bush
    - Gerhard Schroeder
    - Hugo Chavez
    - Jacques Chirac
    - Tony Blair
    - Vladimir Putin
* ***Weights***:
  + Contains the weights for the network nn4.small2.v1.
* ***Videos***:
  + The input videos are placed in this folder.
* ***Output***:
  + It will have the output videos.
* ***Implemented Videos***:
  + It just has the outputs for the videos created.
* ***Yolo-coco:***
  + It has the yolo weights, coco names and conf file.

The pseudocode structure of the implementation for ***celebrity face recognition*** is outlined below:

* Driver code – main.py
  + facerecognition.py
    - model.py (initializing and creating the model)
      * utils.py (activation functions)
    - TripletLossLayer.py (calculating the squared L2 distance loss)
    - IdentifyMetadata.py (collecting metadata)
    - Data.py (generating triplet data “anchor, pos, neg”)
    - align.py (Face alignment)

To run the code:

Python main.py <argument- a number for the image we want to predict>

E.g. Python main.py 2

The pseudocode structure of the implementation for ***real-time multiple object detection*** is outlined below:

* Driver code – yolo\_video.py
  + VideoRun.py

To run the code:-

Python yolo\_video.py –input videos/airport.mp4 –output output/airport\_output.avi –yolo yolo-coco

This takes 3 arguments namely, input, output and the yolo-coco folder.

The pseudocode structure of the implementation for ***real-time single object detection*** using webcam is outlined below:

* Driver code – opencv\_Tracking.py

To run the code:-

Python opencv\_Tracking.py –camera\_or\_video <argument c or v> --video <path to video> --tracker <tracker id>

This takes 3 arguments namely, input type, input path if video and the tracker id.

*Note: the code zip has been submitted along with this document.*

# **Summary**

In the end, briefing up the work done so far, (in the month of Feb and March)

* Project scope evaluation
* Framing project description
* Timeline setup
* Clubbing References
* Data Preparation
* Implementation
* Project Report creation started