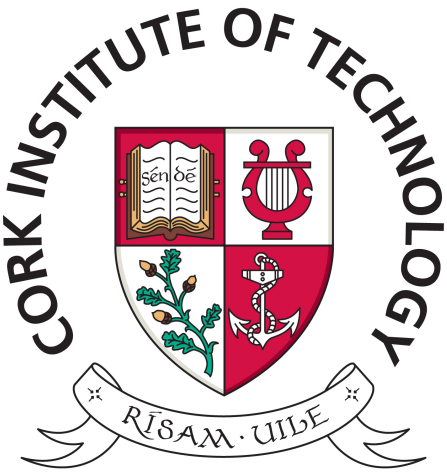

Face Detection & Recognition using YOLO, Facenet & Classification

by

Phalguni Rathod

This thesis has been submitted in partial fulfillment for the degree of Master of Science in Artificial Intelligence

in the
Faculty of Engineering and Science
Department of Computer Science

May 2020

Declaration of Authorship

This report, Face Detection & Recognition using YOLO, Facenet & Classification, is submitted in partial fulfillment of the requirements of Master of Science in Artificial Intelligence at Cork Institute of Technology. I, Phalguni Rathod, declare that this thesis titled, Face Detection & Recognition using YOLO, Facenet & Classificationand the work represents substantially the result of my own work except where explicitly indicated in the text. This report may be freely copied and distributed provided the source is explicitly acknowledged. I confirm that:

- This work was done wholly or mainly while in candidature Master of Science in Artificial Intelligence at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Phalguni Rathod

Date: 17th May, 2020

CORK INSTITUTE OF TECHNOLOGY

Abstract

Faculty of Engineering and Science
Department of Computer Science

Master of Science

by Phalguni Rathod

Face detection and recognition has been one of the most complex AI tasks to achieve. There have been quite a few algorithms to achieve this, yet there remains a great scope for exploration. This research offers collaboration of two state-of-the-art algorithms - YOLO & FaceNet with a classifier to detect and recognize faces in images and real-time live streaming from webcam. It proposes a 3-stage architecture built and implemented from scratch and takes an effort to create a custom data set from scratch.

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this research, and without whom I would not have made it through my masters degree!

My foremost gratitude to my supervisor **Prof. Triona McSweeny** for consistent support, showing right direction and constant engagement and motivation to make this thesis a success. I thank my course coordinator **Dr. Ted Scully** for constant help and encouragement.

CIT has played a crucial part in supporting in difficult times of pandemic & lockdown and I thank them for that.

I'm grateful to **my family** whom I dedicate this thesis to, without their efforts, scarifies, support and constant motivation this MS wasn't possible.

Thanks to all the friends I made here, who became family away from home. Thank you **Karan Thakkar**, **Shivani Goyal**, **Vishakha Oram** & **Dhanashri Lad** for proof-reading and helping me shape this thesis.

Thanks!

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelliegnce
YOLO	**Y**ou **O**nly **L**ook **O**nce
DL	**D**eep **L**earning
P-Net	**P**roposal **Net**work
R-Net	**R**efine **Net**work
O-Net	**O**utput **Net**work
MTCNN	**M**ulti **T**ask **C**onvolutional **N**eural **N**etwork
KNN	**K** **N**earest **N**eighbour
DT	**D**ecision **T**ree
RF	**R**andom **F**orest
NN	**N**eural **N**etwork
IPL	**I**nformation **P**rocessing **L**anguage
FGCP	**F**ifth **G**eneration **C**omputer **P**rogram
ML	**M**achine **L**earning
CV	**C**omputer **V**ision
OCR	**O**ptical **C**haracter **R**ecognition
RCNN	**R**egional **C**onvolutional **N**eural **N**etwork
SSD	**S**ingle **S**hot Multibox **D**etection
RPN	**R**egion **P**roposal **N**etwork
SGD	**S**tochastic **G**radient **D**escent
FDDB	**F**ace **D**etection **D**atabase & **B**enchmark
DNN	**D**eep **N**eural **N**etwork

Dedicated to my

Parents

Mr. Suhas Rathod and Mrs. Vandana Rathod

Mr. Vinay Rathod and Mrs. Suvarna Rathod

Granny - Mrs. Lata Rathod

Siblings - Simran & Atharva Rathod

Thanks for all the Efforts, Sacrifices, Motivation & Support

—— Swami Samarth——

Chapter 1

Introduction

Did you get your letter to *Hogwarts*?

If not, then I regret to tell you that you won't have *poly-juice potion* to get away from getting your face-recognized!

1.1 Motivation

Recognition is basically understanding "what is it?" If an object then, "what object is it?" If it is human then "who is it?" This capability is natural to humans, but even as we were born we didn't know anything, and with time we started understanding differences between things, living - non-living, human - animals, "who that human is?"- starting with our parents. AI technology as a whole is similar. It starts with nothing (rather something random) and starts learning from that point by doing multiple iterations of the same thing while learning something every time. This analogy of AI with humans motivated me to study AI.

One of the most fascinating replication of human traits of ability to see, understand, analyze and derive meaning as well as taking action accordingly, is done through a branch of AI, called Machine/Computer Vision [Section 2.3]. Within computer vision, Face Detection and Recognition is considered as one of the most complex problems, yet in the recent times, it has progressed a lot. I was motivated to study how machine interprets a face, detects and recognizes too because all my life I have been on the recognizing end, now I'd like to re-engineer and see it from the other side.

Due to my interest in computer vision, I started reading and researching about it and came across amazing state-of-the-art papers and technologies like, YOLO [Section

2.4.1.2] and FaceNet [Section 2.9]. These papers motivated me to do some experimentation with them.

1.2 Contribution

1.2.1 Research Question

- How image data is collected for building face related dataset?
- How YOLO can be used to detect faces?
- How FaceNet is implemented?
- How we can integrate YOLO based face detection with FaceNet?
- Which algorithm to choose for classifying the faces?
- How to do face recognition in real-time with web-camera stream?

1.2.2 Research Objectives

- Use state-of-the-art algorithm of YOLO to detect faces
- Implement & use state-of-the-art framework of FaceNet
- Understand which classification algorithm to use for the best results
- Build and implement an architecture to integrate state-of-the-art algorithms into a single framework
- Gather and organize custom dataset for the thesis
- This system should be able to process both images and real-time live streaming via webcam

1.3 Structure of This Document

This document reports the thesis done in partial completion of the MSc AI course. It holds all the efforts made to successfully achieve the expected outcomes. It has been structured chapter wise.

Chapter 1 giving an overview of the project, tells the motivation that drives the project, contribution this thesis has made, broken down into research questions and objective and describes the structure of project.

Chapter 2 tells about the background research undertaken from scratch i.e starting from how AI came into existence and grew over the span of 50 years. Following the various sub-fields of AI, digging them down a bit, and moving forward to computer vision, object detection, Face Detection, State-of-the-art YOLO and FaceNet.

Chapter 3 describes the problem statement, also outlines the main objectives to be achieved for the project, then breaks down the requirements for the project into functional and non-functional, also discusses the technology used to implement this project, and narrate the architecture as built from the scratch.

Chapter 4 here the actual implementation and groundwork is explained, starting with how data is collected and structured, moving to how data is loaded, then how state-of-the-art models are loaded, explains implementation of three stages of the proposed architecture, in each dedicated section as Face Detection, Face Recognition and Face Classification. Also, describes the test architecture. Explains how exactly training and testing takes place. Ending the section with what all data is stored and how.

Chapter 5 here the testing and evaluation of the system is done. And an example of output are also shared to get an intuition of expected outcome.

Chapter 6 gives a closure to the thesis holding a discussion, a final conclusion and telling possible future work on the project.

Chapter 2

Literature Review

2.1 Artificial Intelligence

Artificial Intelligence (AI) has become a buzz word in the entire world. People having infinitesimal knowledge of what AI actually is, also get excited just by hearing the word. Companies around the world have started marketing their products as "Artificially Intelligent" and people are all going crazy for that! It started by making our phones smart and now even the cameras of our phone have become intelligent 'artificially'! Take a pause here! Think about the time we had limited access to even a telephone,then we got wireless phones for each home/room, then moved to our personal phones-mobile and all of a sudden it became smart! Now, think of the time vs transition each of these technology, it's exponentially fast! Technology became easily accessible to each and everyone, plus it's also getting smarter & better day by day.

But how did it all start? How did our phones and other techonology become smart and intelligent?

It all started with one thought!

2.1.1 Can Machine Think?

Alan Turing, a very respected and reputed mathematician, computer scientist, logician, cryptanalyst, philosopher, and theoretical biologist[13], have asked this question for the first time in 1950 in his paper titled 'Computing Machinery and Intelligence'[14]. In this paper he reframed the above question into a game called 'Imitation Game', briefly it says - if a human judge interacts with 2 participants without knowing anything about them, but only one of them is real human and other is a machine, then could the human

judge make out the difference between them or not? There is a lot more in this game, but it is out of the scope of this thesis.

2.1.2 Logic Theorist

Even though the question and the reasoning of machine thinking was proposed in 1950, but the actual term 'Artifical Intelligence' was coined in 1956 by John McCarthy when he held first academic conference on this topic - Dartmouth Summer Research Project on Artificial Intelligence. This conference didn't meet the expectations of John McCarthy, nonetheless it was this conference where Allen Newell, Cliff Shaw, and Herbert Simon Figure2.1 showed the world a proof of concept of AI - Logic Theorist[15]. Logic Theorist was first known to imitate the skills similar to human to solve a problem, the program was coded in IPL(Information Processing Language) by Cliff Shaw and executed on RAND's computer at Santa Monica's Research facility[15]. Attendees of this conference were later known as founging fathers on AI (Figure 2.1)

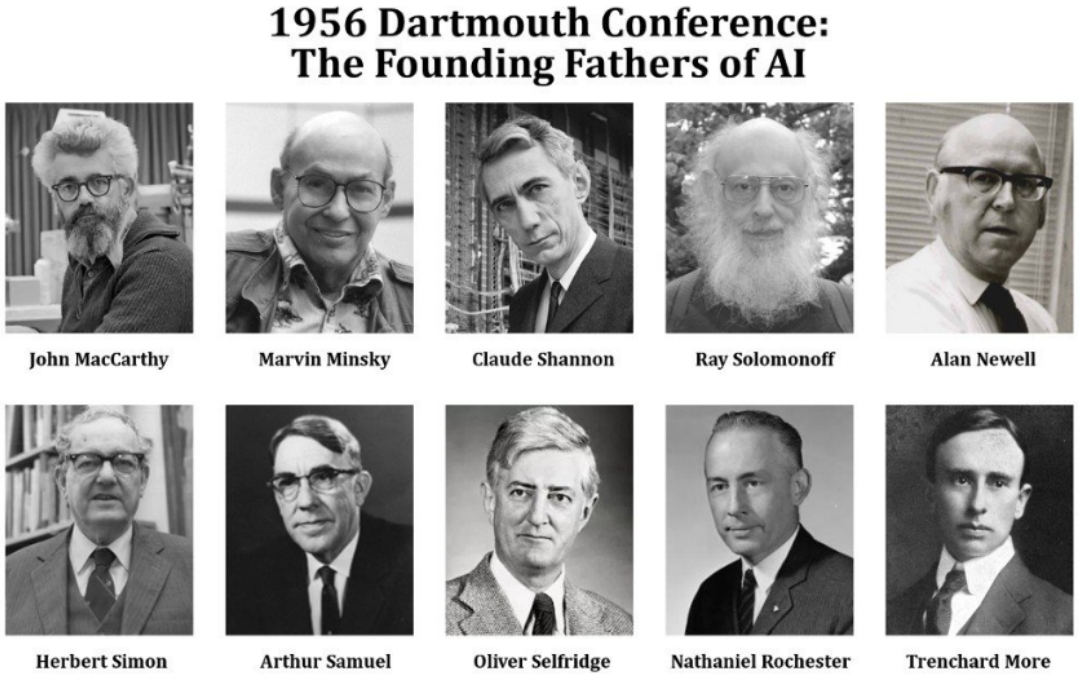

FIGURE 2.1: Fathers of AI
[16]

2.1.3 Ups & Downs

The journey of AI as we know it today isn't simple, it took more than half of a century for a thought to become reality (Figure 2.2). There were many obstacles starting from

the lack of computing resources, during the time of Alan Turing computers lacked storage/memory. Computers made calculations and executed commands but couldn't store them, plus computer were expensive back then, only eminent universities and research facilities could afford them. From 1956 to 1974, AI grew and did well. Govt and their subsidiaries invested in AI hoping that machines could do transcription and translation for them. But all this was very ambitious. The major hurdle was the computational power to process and store such a huge data. Japan in 1982 decided to fund AI under their 'Fifth Generation Computer Program' (FGCP), they invested 400USD million, unfortunately thier expectations were not met but FGCP gave rise to generation of AI researchers. But with this, AI went out of focus. Yet in 1990s-2000s AI prospered, in 1997 AI Chess by IBM - DeepBlue defeated then grand master Gary Kasparov[1]. During the same time frame, another firm Dragon Systems implemented speech recognition software on Windows. Since then, AI never stopped! And, here we are today - surrounded with everything Artificially Intelligent around us!

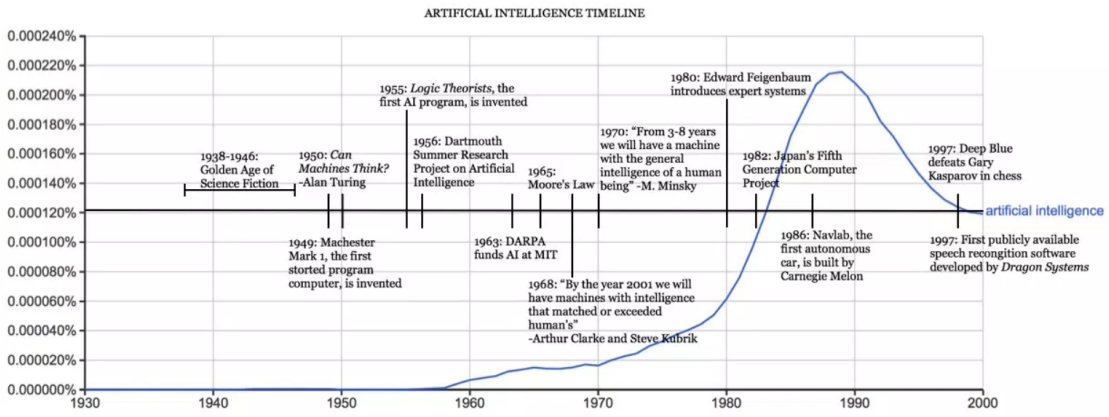

FIGURE 2.2: The Journey of AI [1]

2.2 AI Sub-fields

AI is an umbrella term referring to a number of sub fields under AI in layman terms. In reality, AI is humongous.

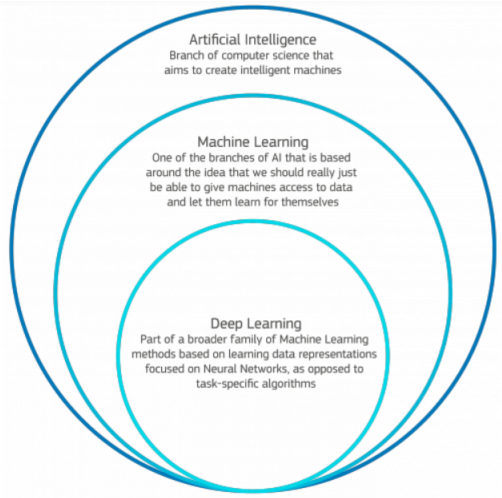

FIGURE 2.3: SubFields of AI [2]

2.2.1 Machine Learning

One of the major sub field of AI is Machine Learning (ML). Data is fed into the ML Algorithm and its main focus is to do the prediction. The data fed to the algorithm is called training data, generally having instances and target labels(not always). Then the algorithm is executed multiple times so, the algorithm learns from it and gives prediction when untrained data is given into the system. Basically, ML provides a means by which programs can infer new knowledge from observational data.

ML further has 3 types:

1. Supervised Learning: Feature and Target data both are provided in the training set for the algorithm to learn. Once learned or trained as we generally call it, we input a previously unseen feature data and expect the algorithm to predict the expected output based on its earlier learning. Example: House price prediction, Breast Cancer Prediction, etc

2. Unsupervised Learning: Here, we only feed features to the algorithm and not what is expected from it. We let the algorithm understand the data and find patterns, groups or similarities.

3. Reinforcement Learning: This is a very big area in itself. Basically, here we let the system learn from itself, just like self learning systems. It employs trial and error to come up with a solution to the problem. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward[17].

2.2.2 Deep Learning

Deep learning is a branch of machine learning where artificial neural networks — algorithms inspired by the way neurons work in the brain — find patterns in raw data by combining multiple layers of artificial neurons. As the layers increase, so does the neural network's ability to learn increasingly abstract concepts.

For example, neural networks can learn how to recognize human faces. How? The first layer of neurons takes pixels from example images, the next layers learn the concept of how pixels form an edge, then that layer passes that knowledge to other layers, combining that knowledge of edges to learn the concept of a face. This process of layering knowledge continues until BAM! — the neural network algorithms recognize specific features, and ultimately specific faces[18].

Applications of Deep Learning isn't limited here, it can be applied to sounds, images, videos, text, etc Figure-2.4.

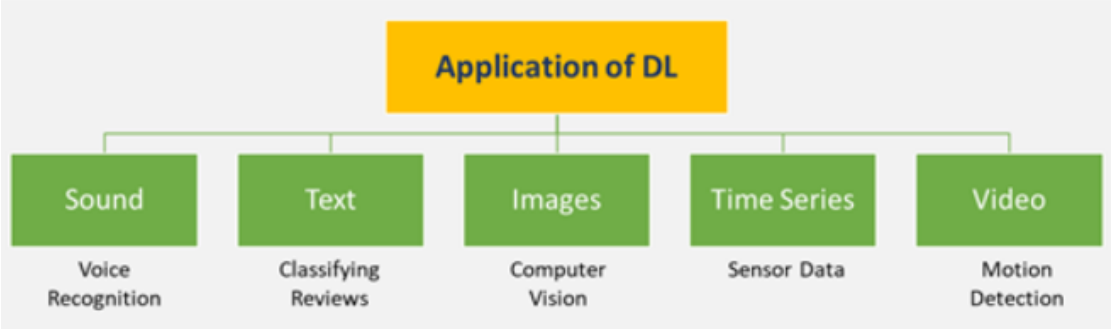

FIGURE 2.4: Applications of DL [3]

2.3 Computer Vision

Computer Vision, by name itself one can get an intuition of what to expect in it. It is popularly known as CV, is a domain that build algorithms that will help computers "see" and understand the content of digital images such as photographs and videos. It's major sub-field of AI where the cameras are treated as eyes of the machine. Just like humans have eyes to look around, observe the environment, researchers have leveraged the cameras to capture images & videos which can be fed to a system for learning & analyzing purpose.

Video cameras are extraordinarily affordable and easily integrated into today's mobile and static units such as surveillance cameras, auto-dashcams, police body cameras, laptops, smartphones, GoPro, and GoogleGlass. This leads to generation of huge image & video data[11]. CV leverages on this humongous data to create number of applications.

It has many applications like object detections, OCR, Image classification, Image segmentation, and it's also used in critical fields like medical services for detecting cancerous tumors by analyzing the X-rays for various parts of body. Most of the CV applications can be generalized by their implementations on 'Objects', few of the are as follows:

1. Object Classification: What broad category of object is in this frame?
2. Object Identification: Which type of a given object is in this frame?
3. Object Verification: Is the object in the frame?
4. Object Detection: Where are the objects in the frame?
5. Object Localization: Exact positioning of the object in the frame?
6. Object Landmark Detection: What are the key points for the object in the frame?
7. Object Segmentation: What pixels belong to the object in the frame?
8. Object Recognition: What objects are in this frame and where are they?

Even in the time of pandemic, COVID-19, researchers are trying their best to utilize CV's potential to contribute in the fight of controlling COVID-19. Modelling and analysis is being done on CT Scans, X-rays to understand more about the virus. Temperature screening is also been done to check possibility of the virus[19]. This shows how critical CV can get when used in the right way.

2.4 Object Detection

Object detection in layman terms is an ability to detect the presence of an object in the camera frame. This is a natural trait in humans and one to the most critical ones. There have been efforts made for years by researchers to impart this property to machines by leveraging the power of computer vision figure 2.5[4]. Object detection have massive applications and many of these applications could be critical, for instance a driverless car is moving on a street, it should be able to detect traffic around it, roads to drive on, people on the streets and make sure not to hurt them. It should be able to detect traffic signals and follow them, etc. Another heavily used application of it is, Optical Character Recognition commonly known as OCR, this is used in detecting handwritten words/numbers, car number plate detection. It has use case in sports too, like player tracking, activity recognition, ball tracking etc.

Object detection when done on certain type of objects dataset like cats/dogs, faces, cars, etc becomes specialized on that.

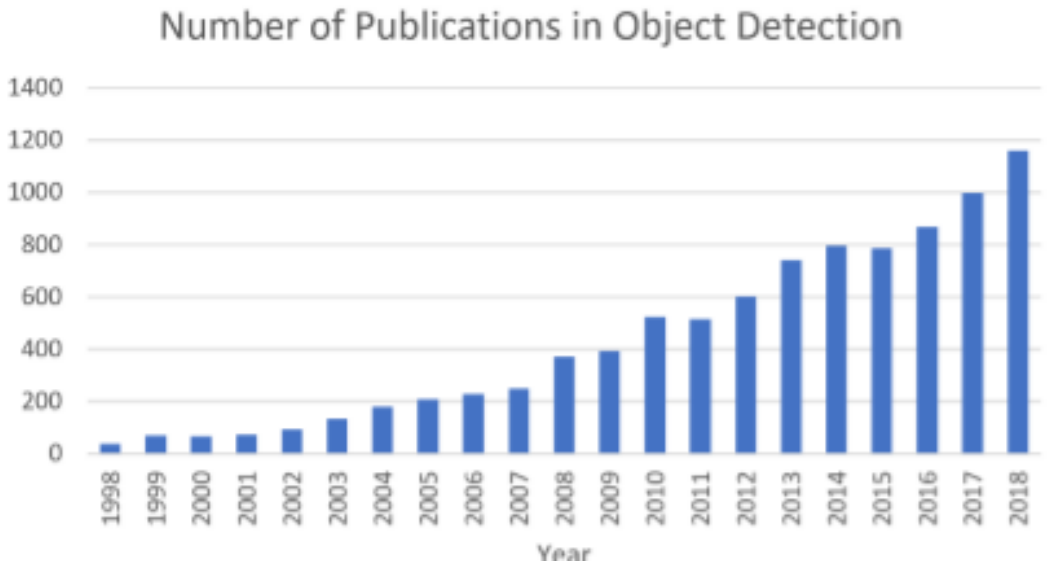

FIGURE 2.5: The increasing number of publications in object detection from 1998 to 2018 [4]

2.4.1 Deep Learning Based Object Detection

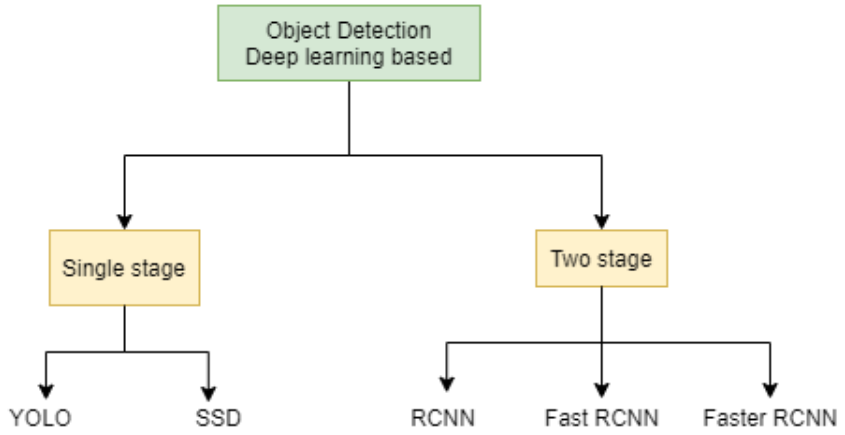

FIGURE 2.6: DL Based Object Detector Types

When you look for DL based Object Detection algorithms they are of 2 types - 1 Stage or 2 Stage(Figure 2.6) and the first 3 primary detectors that will pop up are:

1. RCNN Its variants - RCNN Fast RCNN, Faster-RCNN
2. SSD: Single Shot Multibox Detector

3. YOLO: You Only Look Once

2.4.1.1 RCNN

Region based-CNNs, popularly known as RCNNs, is one of the initial models incorporating CNNs to detect objects. It works in 2 stages. Where in stage 1 it finds the bounding box and in second stage it does classification on that. In [20], it put forward an object detector which employs Selective Search[21] or similar algorithm to propose suitable bounding boxes that might be having objects. One we get the candidate bounding boxes, they were passed through CNN for classification. The only drawback of this basic RCNN was painfully slow and not a fully end-to-end object detector.

Another paper by same author was published in 2015 with claims of making RCNN "Fast" and was also called the same - Fast-RCNN - [22]. Remarkable improvements in terms of increased accuracy and decreased time taken for forwardpass were made by Fast R-CNN. Yet it had to use selective search like algorithm to find bounding box.

It was in 2015 paper called Faster R-CNN, when R-CNNs dropped selective search and introduced a region proposal network (RPN) and finally became a real end-to-end deep learning object detector[23]. RPN is a fully convolutional network and now have the capability to predict the object bounding boxes and "objectness" scores that by what chances each bounding box holds an object. Similar to earlier versions, output of stage 1 is passed to stage to for classification.

Even after so much growth, being accurate, the consistent issue with the R-CNN-variants was - speed — they were really slow, could reach only 5 FPS on GPU

2.4.1.2 YOLO!

To overcome the speed issue, single stage detectors came into picture like SSD - Single Shot-Multibox Detector and YOLO! The only drawback of single stage detector over 2 stage detector is accuracy, they are relatively less accurate but significantly faster. We will be discussing YOLO in detail.

YOLO deals with object detection in a different way. In all the earlier work before yolo, object detection was treated as classification problem, while in YOLO, they redefined it as regression problem to spatially separated bounding boxes and associated class probabilities[5]. Only one evaluation of single neural network is needed to predict bounding boxes and class probabilities directly from full images. End-to-end optimization of detection performance is possible as the whole detection pipeline is a single network[5].

YOLO have many variants and all of them are fast. The basic model itself processes about 45 frames per second in real time. While, another smaller version of the network claims to reach astonishing 155 frames per second and benchmarking double the mAP when compared to other detectors[5]. Like all the systems, YOLO have its own drawback, it makes more localization errors but is less likely to predict false positives on background[5].

Input to the model is an image, which the system divides into an SxS grid.'B' bounding boxes are predicted in each grid, along with confidence score for each box. Confidence score indicates how sure the model is, that an object is contained in the box as well as how accurately the box predicts what object it is. Confidence score is zero if no object is detected. 5 predictions were made for each bounding box: x, y, w, h and confidence. Here, (x,y) tells the centre of the box with respect to grid. The grid where the centre lies, is responsible for detecting the object[5] (Figrure 2.7).

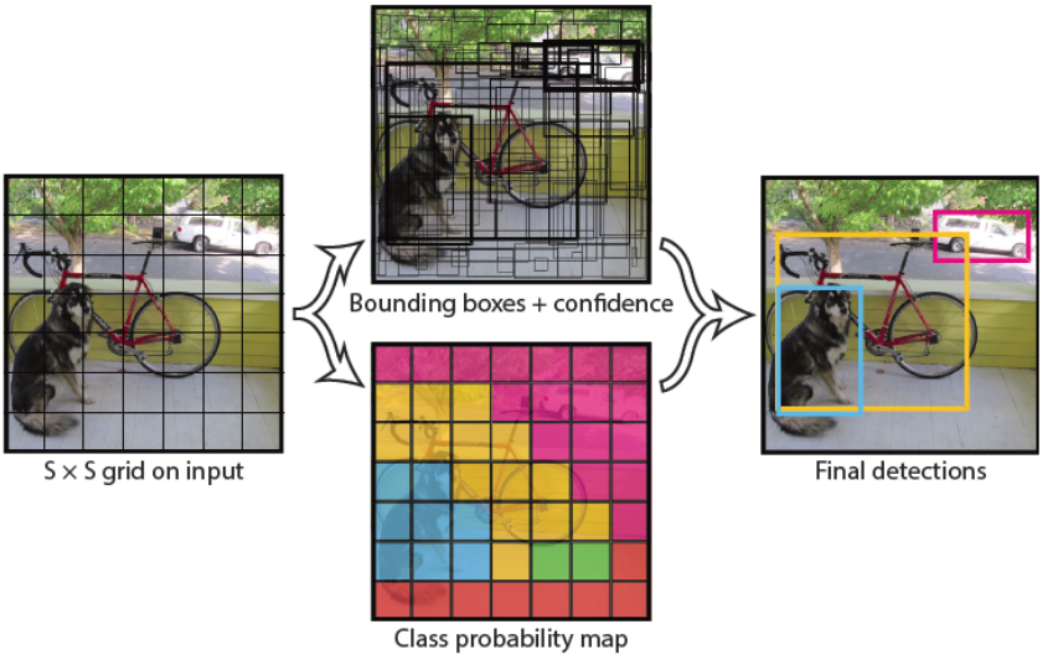

FIGURE 2.7: YOLO bounding boxes and object detection [5]

2.5 Face Detection

Face detection is a special case of object detection with the model trained specifically on faces to be detected. Face detection and recognition is one of the many application of object detection[4]. At various points in our daily life we come across the applications of face detection and/or recognition, and so many times it goes unnoticed. For an instance

any smartphone in today's time have cameras and these cameras use face-detection while clicking pictures and do auto-focus on face to produce good quality pictures. Many of the mid-budget to high-end smartphones have face unlocks like apple have FaceId, Samsung have their Face & Iris Unlock and other giants like OnePlus, Xiaomi, etc have their own tech for the same purpose. There are reports suggesting that 1 Billion smartphones could have face recognition by 2020 figure 2.8 [6]

Another common use case is the google photos, it stores all the images in cloud and does processing on it. One can simply search for all the photos related to a person just by click on their face in search bar. That's done by first detecting the face and then clustering similar faces together[24].

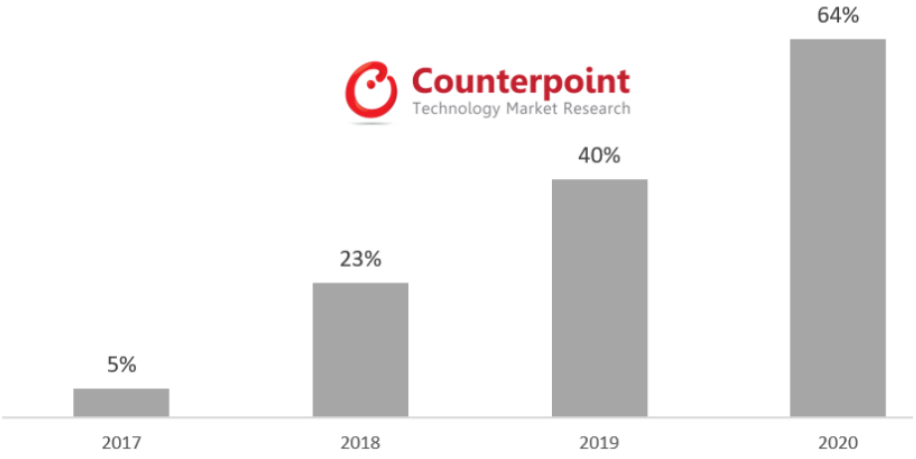

FIGURE 2.8: Smartphone Facial Recognition Feature Penetration of Smartphone Shipments Forecast [6]

2.5.1 Viola Jones Algorithm

It was first introduced in 2001 by Paul Viola and Michael Jones. It is a framework popularly used for detecting objects in still images as well as in real-time. The algorithm is used for last 20 years with tweaks[25]. This algorithm have 4 stages(Figure 2.9):

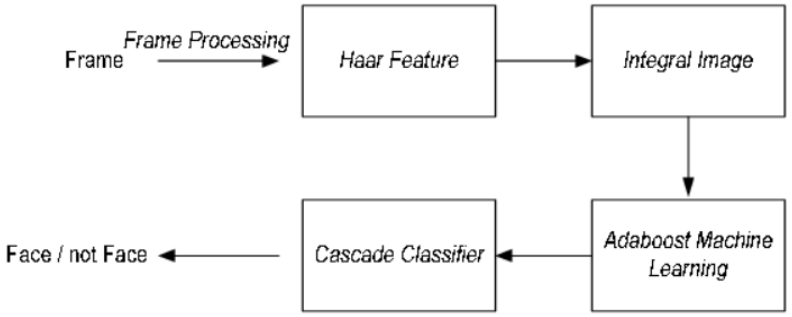

FIGURE 2.9: Flow of Viola Jones Detector [7]

1. Haar Features: The haar-like algorithm is also used for feature selection or feature extraction for an object in an image, with the help of edge detection, line detection, centre detection[26]. Haar like wavelets are used to calculate features like for eyes(eye region darker than nose bridge), nose, mouth, etc (Figure 2.10)[8]. To implement this, VJ uses 24*24 pixel window and takes single pixel at a time and compares with entire window and calculates value by subtracting white region from black one[27]. But doing this for each pixel in each window for all the windows across the frame is big computation. That's why we have Integral Images.

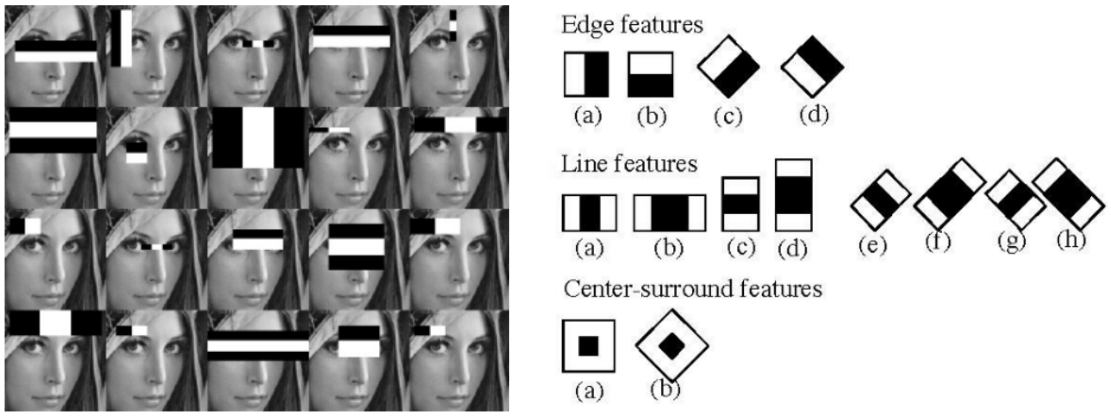

FIGURE 2.10: Flow of Viola Jones Detector [8]

2. Integral Image: It is used to speed up the process of detecting features using Haar. It is created by calculating the rectangles side-by-side to the rectangle present at a given pixel position (x,y) into a single image representation and speeding up the procedure.

3. AdaBoosting ML: AdaBoosting is a classifier and used for classifying the image. It can be seen as a feature extractor, where not so important features like background noise, clothes etc which are not part of face are removed. This drops the features from 160k+ to within few hundreds[27].

4. Cascade Classifier: Laying out series of classifiers to remove more and more unrelated features. A window/feature is pass through these classifiers and if any one of the all classifiers rejects the window, the feature is dropped.

2.5.2 Deep Learning Based Face Detector

There have been studies involving deep learning architecture to demonstrate face detection. One approach which reached benchmark results is shown by Kaipeng Zhang et al, in their paper. They called this approach Multi-Task Cascaded Convolution Neural

Network (MTCNN). Along with detection of face, MTCNN is also capable of locating eyes, nose and other facial features generally referred as landmarks[9].

The architecture of MTCNN consisted of one rescaling layer and 3 stages of cascaded framework. It consist of 3 networks called: P-Net, R- net and O-Net(Figure 2.11).

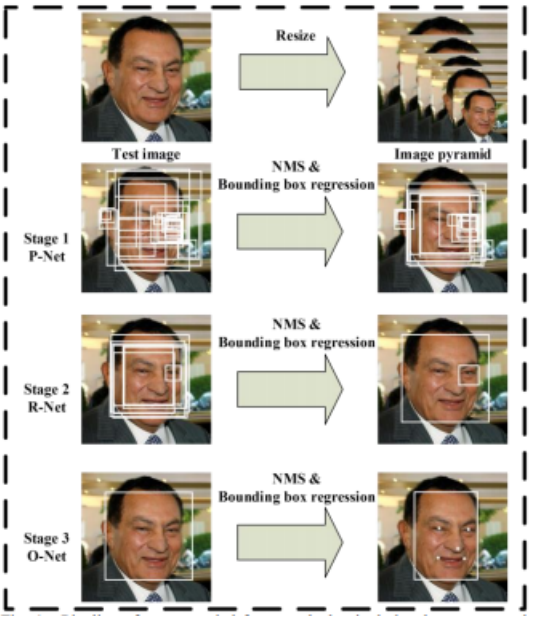

FIGURE 2.11: Three Stage Multitask P-R-O-Net structure of MTCNN [9]

1. P-Net: Stands for Proposal Net, here we get candidate bounding boxes and apply NMS on it to remove highly overlapped bounding boxes.
2. R-Net: Stands for Refine Net. Boxes from P-Net is passed to this and then it further filters out false candidates as bounding box repression is performed.
3. O-Net: Stands for Output Net. Here detection of position of five facial landmarks is outputted.

2.6 YOLO for Face Detection

As we have overall idea of working of YOLO [Section 2.4.1.2] as well as face detection [Section 2.5]. Now we will look at how we can detect face using YOLO. The attempt for this has been made in [28]. [10] and [29]. The datasets used for training the YOLO network are different for all three.

[28] uses WIDER Face dataset[30] for training and Celeb Faces[31], FDDB[32], WIDER FACE[30] for testing. It employs SGD for training with learning rate of 0.001 over 30,200 iterations.

While [10] uses Face Detection Dataset and Benchmark (FDDB) Dataset[32], which consist on 5171 faces in total of 2845 images. The dataset is bifurcated into train and test sets in the 70%-30% ratio. The uses graident descent optimiser and trains for 25 epochs with learning rate 0.0001(Figure 2.12). It gives consistent and noteworthy result post 20th epoch with accuracy of 92.2%.

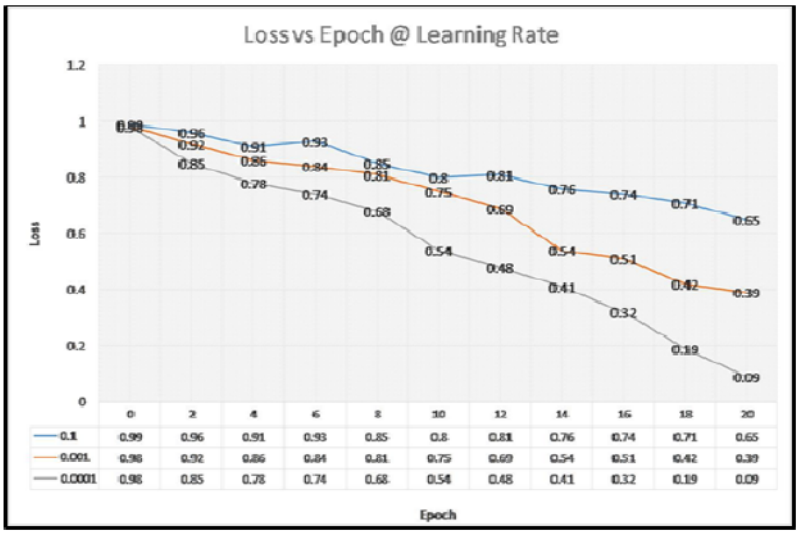

FIGURE 2.12: Loss vs Epoch for range of learning rate[10]

While doing the research for past work on YOLO for face detection, an interesting open source github repository[33] came into light. Here, the author have already trained the YOLO of FDDB dataset and share the config and model weights. These files are used in making this system. It can be loaded in dnn and used, this helps avoid reinventing the wheel.

2.7 Face Recognition

Face Recognition the term explains itself quite well. It simply mean given one or more people in an image/video, it should be able identify who the each person is. To achieve this the system should first be able to detect faces in the frame, hence face detection is step-1 for face recognition. It was in 1970s that the research on auto-recognition of face using machine was started[34]. And half century later we have the technology practically available with us, as well as flourishing by the day. Face Recognition can be further broken into Face Identification and Face Verification (Figure 2.13).

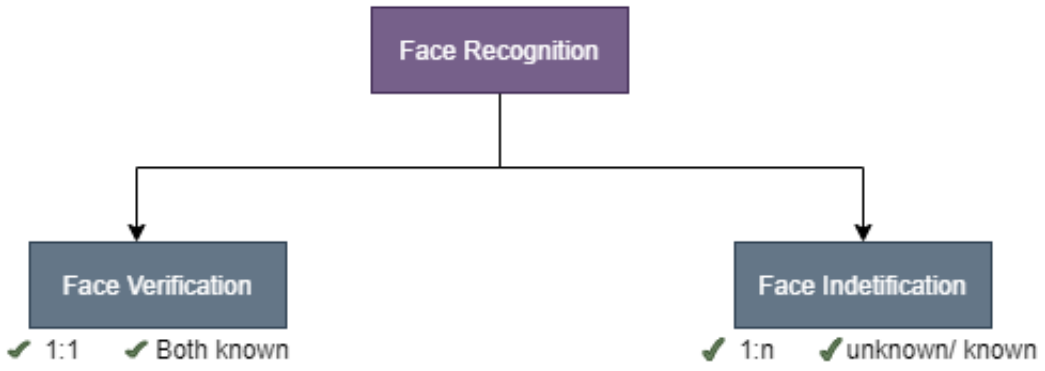

FIGURE 2.13: Face Recognition Type

2.8 Face Recognition

There are number approaches to implement face recognition broadly they can be classified into using deep learning and not using deep learning - such approaches are called shallow approaches[35]. From number of deep approaches, this thesis will work on Facenet.

2.9 FaceNet

Earlier deep learning based face recognition algorithm had an intermediate layer which used to create bottleneck, while facenet uses deep convolutional network to directly optimize the embeddings. Facenet directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity[12].

2.9.1 Workflow

The core of Facenet is calculation of euclidean embedding for image in 128-dimension by utilizing the power of deep convolutional networks[36]. Face similarity is directly proportional to squared L2 distances[37] of embedding space on which the network is trained. Faces of the same person have small L2 distances and faces of distinct people have large L2 distances. As soon as we get the 128-d embedding we can perform number of face related tasks such as,

1. Face Verification: Can be done by thresholding the distance between two embedding

2. Face Recognition: Applying classification algorithms on embeddingg

3. Face Clustering: Can be implemented using clutering techniques like k-means or agglomerative clustering

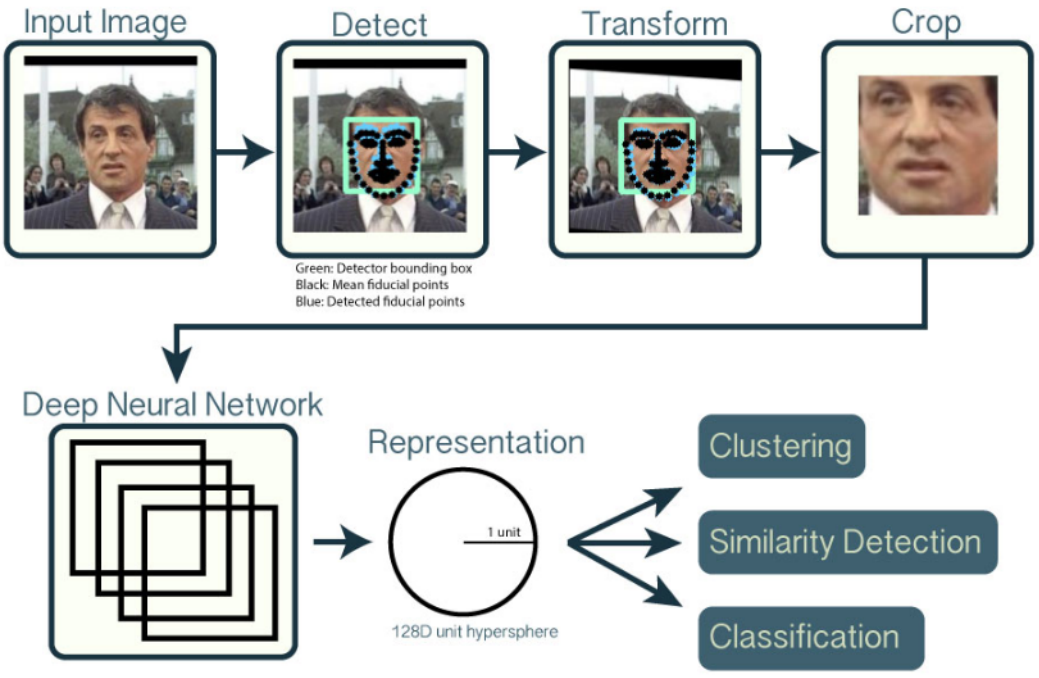

FIGURE 2.14: Flow of single image through Deep Face Recognition[11]

2.9.2 Architecture

FaceNet employs two categories of deep convolutional networks, first is The Zeiler & Fergus Style networks[38] (Table 2.1) and second beinf Google's Inception type network[39] (Table 2.2). For training these networks SStochastic Gradient Descent (SGD) alongwith standard backpropagation and AdaGrad is used[12]. Learning rate for the experiments generally started with 0.05 and lowered towards finalizing the models. The models took 1000-2000 hours to train and showed signs of consistence in accuracy from 500th epoch, yet training it more increased the performance[12].

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

TABLE 2.1: NN1 - FaceNet[12]

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

TABLE 2.2: NN2 - FaceNet[12]

2.9.3 Triplet Loss

It is a loss function which is used along with an optimiser to update the weights in a neural network for learning the parameters effectively. Its used in face-recognition,

verification and clustering related NN. We take into to account 3 images for calculating the triplet loss, an 'Anchor' image (denoted by A) which is like a base image, a 'Positive' image (denoted by P) which belongs to the same class/person as A and a 'Negative' image (denoted by N) which be belonging to some other class/person, figure2.15. Triplet Loss basically calculates the distance between d(A,P) and d(A,N) and tries to minimize the distance between and A,P while maximize between A,N, it updates the weights accordingly and back-propagates it, figure 2.16.

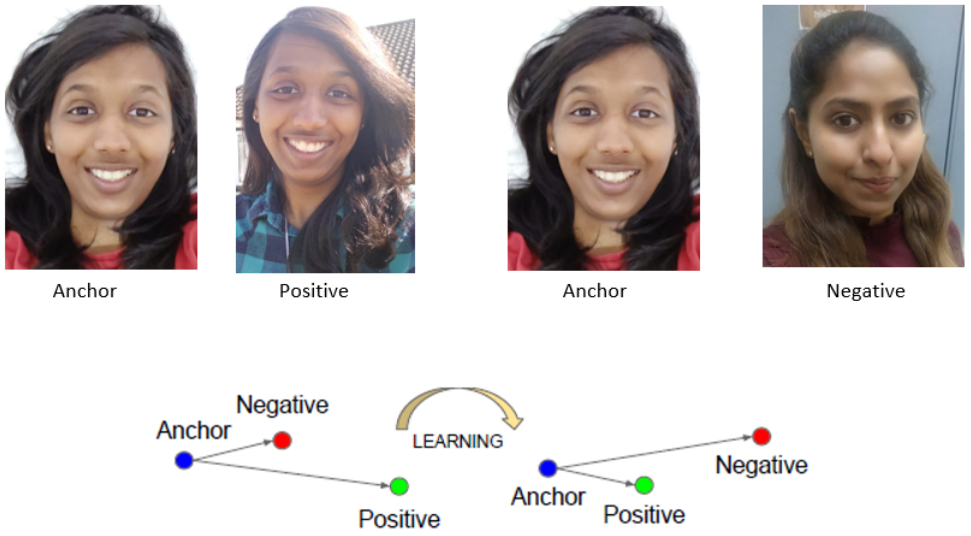

FIGURE 2.15: Triplet loss: Anchor, Positive & Negative Images, Distances before after learning from the triplet loss [12]

The main idea behind it is, images belonging to same classes should be nearer to each other in terms of embedding calculated, while embedding calculated for different classes should be far from each other which will create a distinction between images of different classes/personalities.

2.9.4 Illumination & Pose Variance

Humans eyes have natural ability to adjust itself with respect to lighting, when we are in dark the iris opening is wide, while in bright light it squeezes rapidly to allow only the amount to light required. Our machines are becoming smarter day by day but haven't become so smart to impart this fascinating property of human eye, hence variation in lighting or illumination as we call it here, effects the face recognition system.

Face recognition can be easily pulled off in a set-up environment where everything can be controlled and tuned according to the need like, lighting, positioning, angles, etc.

But to get the best result in natural environment is real is big challenge for most of the face recognition techniques[40].

Even if there is a subtle change in the direction of illumination could lead to key changes in face image, shades of gradient, due to this even the face of the same person appears different.

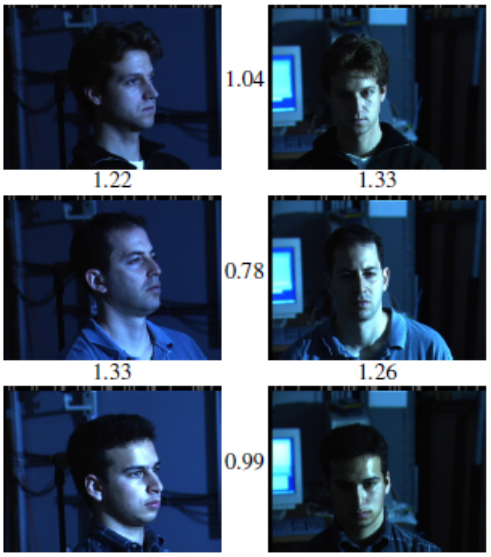

FIGURE 2.16: Illumination and Pose invariance[12]

Face detection algorithm generally outputs the coordinates around the face detected that when joined forms a bounding box. The face detected in the frame can be in any direction, looking down/sideways/tilted head, etc but it is expected to be in specific template, hence needs to be aligned to compensate the variations. The process of aligning the face aims to get higher accuracy in localizing and normalizing face image as the face detection gets us the crude estimation. We can overcome the problem of pose variation, by expanding the data set and using face images captured from various position to get the we use multiple views of same like frontal, side profiles.

FaceNet's approach being driven by data,learning from pixels of images. It avoids using engineered features for learning about the illumination & pose variance, it simply uses large data set of balled faces to achieve the required invariance[12].

Chapter 3

Design

3.1 Problem Definition

As a part of this dissertation, this project proposes to give a new approach to face recognition problem. The project is taking up 2 state-of-the-art algorithms namely YOLO[5] and FaceNet[12] along with a classification algorithm and make a 3 staged face recognition system. Its also proposed to make this system work on still images as well as live streaming from webcam. And the data used to train this network will be custom and gathered from scratch.

3.2 Objectives

- Collect and Organize data to build custom dataset
- Use YOLO to detect faces
- Use FaceNet to get embeddings
- Integrate YOLO for Face detection and FaceNet
- Choose from variety of classification algorithm, the one that suits the problem.
- The system should be able to recognize faces in images and streaming from webcam
- The output image/video should have bounding boxes, along with confidence score and name

3.3 Requirements

3.3.1 Functional Requirements

- Load the dataset properly
- Detect faces using YOLOv3 and return bounding boxes
- Extract face patch from each detected face
- Save the face patch with its coordinates
- Load all the face patches to calculate embedding using FaceNet
- Save the embeddings
- Load embedding for classification
- Use a classifier to fit the model on Training data
- Take test image from user or default or webcam
- Detect & extract faces from test and Calculate embedding.
- Predict the class of each face and also find probability, mark it as unknown if doesn't fit the criteria
- Draw bounding box with confidence and name of person around the face
- Display number of detection, number of recognition and update in real time when streaming through webcam.

3.3.2 Non-functional Requirements

These are the requirement which acts more like a constraint on functionality rather than actual functionality. It controls the actual functionalities.

- While training the model, if no face is detected from the model, it should skip it
- System shouldn't crash even if no face is detected while live streaming the webcam.
- If probability of predicted face in classification is less than 95%, mark it as unknown

3.4 Design

3.4.1 Technologies Used

1. Python
    - Version 3.7
    - All the programming for this project is coded in Python[41]
    - Chosen because of the huge python community which supports usage for data science, analysis, etc
    - A lot of open source ML, DL, CV and many more packages available
2. Numpy
    - Version 1.18.1
    - Python package used for easy array calculation[42]
    - In this project we will be converting images into Numpy Array and store load them
3. Pillow (PIL)
    - Version 6.1.0
    - It is forked version of Python Image Library (PIL)[43]
    - We are using it to read Images and manipulate them
4. OpenCV
    - Version 4.2.0.32
    - OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library[44].
    - We are using it to read DNNs,removes overlapping bounding boxes - NMS, creating BLOBs, showing images, reading images from web-cam, etc
5. Web Camera:
    - This project will use the default web camera of your device this can integrated web cam of laptop
    - Used for running the system on real-time live streaming.

3.4.2 Solution Architecture

To build the proposed system, break down the problem statement into 3 major parts: Face detection [Section 4.4], Face Recognition [Section 4.6] -(Embedding Calculation) and Face Classification [Section 4.6]. The data passed will go through these 3 stages before outputing it to the end user. The Full architecture of the proposed system is drawn in Figure 3.1 The architecture in Figure 3.1, have 5 major parts,

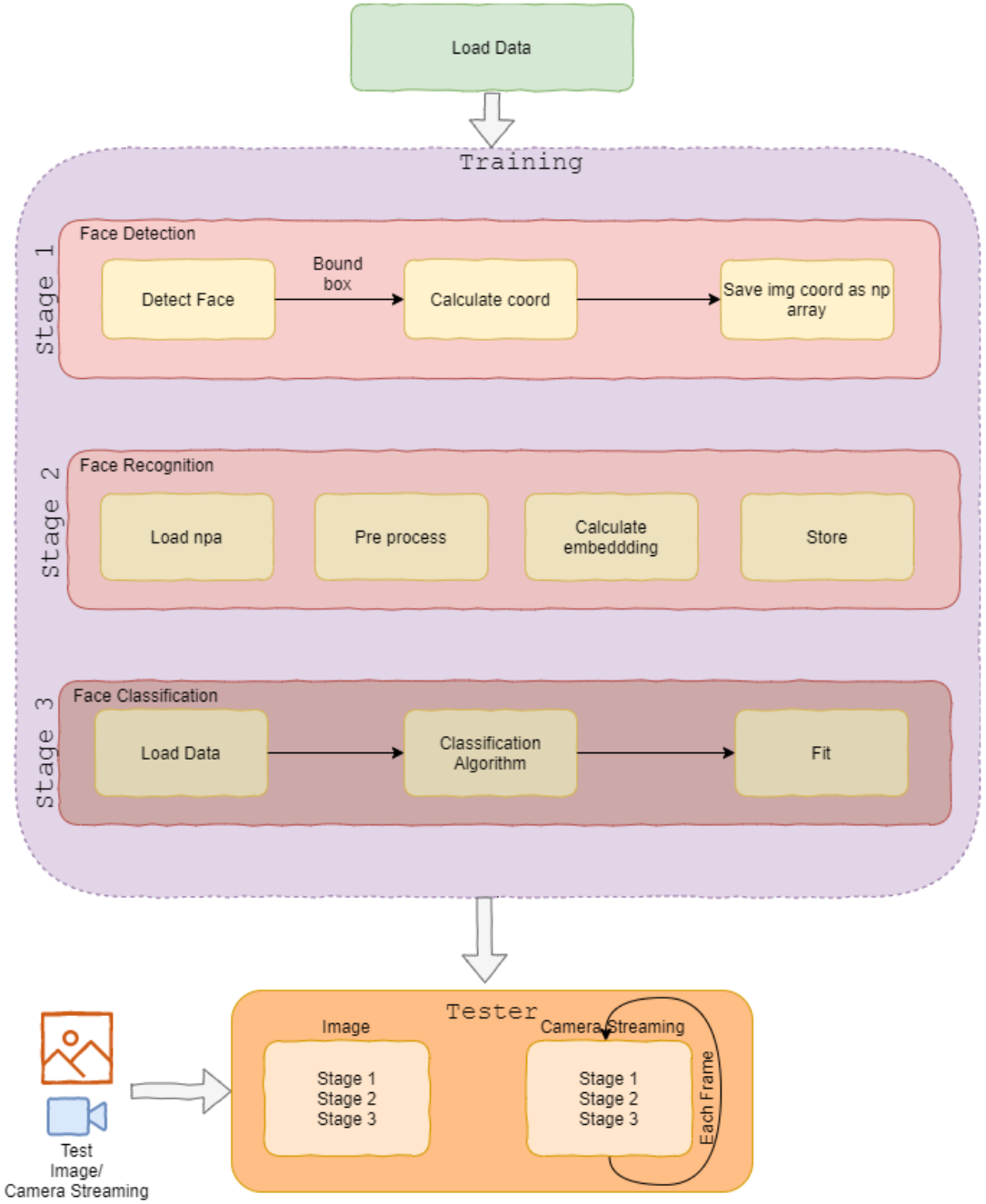

FIGURE 3.1: 3 Staged: Full Architecture of proposed system

- **Load Data:** The data loaded from directories should be arranged and loaded in certain format, more on this in section 4.1.
- **Stage 1:** Faces are detected in this stage, once detected they are extracted and stored.
- **Stage 2:** Face Recognition, basically embedding for faces are calculated here and store.
- **Stage 3:** Face Classification: Faces are identified and classifies into various person as predicted.
- **Tester:** User can pass test image/web cam streaming, it will give the expected output

Chapter 4

Implementation

4.1 Data

For any AI project, the most important component from where the project begins is the data. It is said that data is the new Oil. Looking at the growth of AI with production of such a massive data on daily basis shows that the saying is true.

In this project still image data is used as well as data generated while streaming from web-cam. One of the objective of this project is to collect the data from scratch, which is not an easy task, though it is achieved successfully.

4.1.1 Data Collection

While collecting data, number of things needs to be kept in mind. Few of the issues that were faced and learned during the data collection are discussed below.

4.1.1.1 Conditions

- The image collected should have only a single person in it whose face is being trained for that person's class
- At least 15-20 images of each person is required, which will further be divided into test and train

4.1.1.2 Variations

- **Intra Image Variation:** Each person's pictures should have variation like difference in pose, light, wear facial accessories like glares, nose pin, etc, Men having pictures with beard and without beard.
- **Inter Image Variation:** The collected images should belong to variety of people like kids, women, girls, boys, men, older people, etc

4.1.2 Structure Data Storage

One shouldn't simply store all the collect image together and get started. Data collected should be organised according to how it is planned to be loaded in the system.

For this project, the collected data should be into 2 folders for each person, one will be used for testing while other for training, both folders should have exactly same name, and in this case, name of the person the picture belongs to. The name on the folder will be treated as person's name and displayed in the output. Now make 2 new folders test & train and put the afore created folders into their respective places. The folder structure for this project is shared in Figure 4.1 to give a clear idea.

Note: The collected and structured data is zipped along with code file with same file structure in Figure 4.1, link can be found Section 4.10.

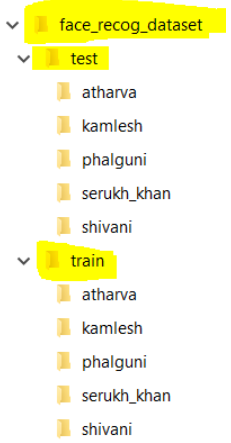

FIGURE 4.1: Structuring the collected data

4.2 Load Dataset

Once the data is collected and also structured into expected folder structure. It is the time to load the data into the system. 'load-dataset' functionality in the code does this.

The functionality requires the path of the dataset to be loaded. Once it gets the path using the OS library of python it goes into the sub-folder belonging to each person, each of these sub-folder have multiple images of same person. Each picture is fed to Stage 1 of the system Face Detection module. After processing each image it looks for more images, if exist it does the same, otherwise it further checks if sub-folder exists in main folder, if yes then load the next folder, otherwise exit as all the data has been loaded. Figure 4.2 shows the whole functionality of load_dataset.

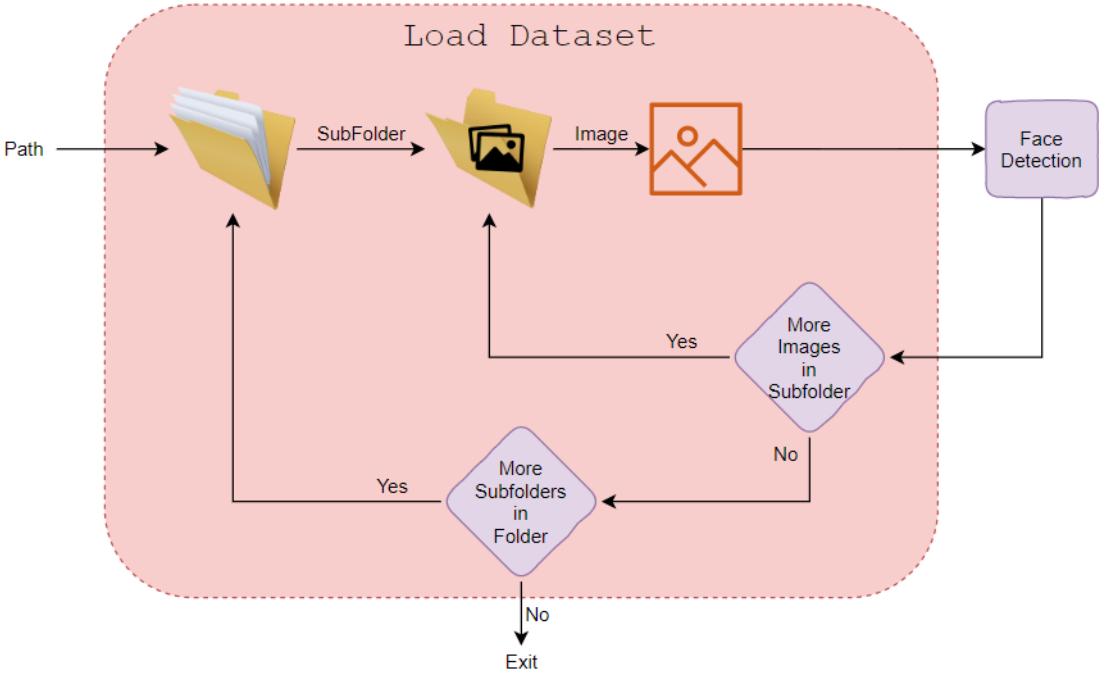

FIGURE 4.2: Loading the files

```
def load_dataset(directory):
    """
    :param directory: path to test/train directory
    :return: Numpy Array of face patches and their labels
    """
    X, y = list(), list()
    # enumerate folders, on per class
    for subdir in listdir(directory):
        # path
        path = directory + subdir + '/'
        # skip any files that might be in the dir
        if not isdir(path):
            continue
        # load all faces in the subdirectory
        faces, face_coord_list = load_faces(path)
        # create labels
        labels = [subdir for _ in range(len(faces))]
        # summarize progress
```

```
        print('>Loaded %d examples for class: %s' % (len(faces), subdir))
        # store
        X.extend(faces)
        y.extend(labels)
    return asarray(X), asarray(y)
```

4.3 Load Models

In this project 2 pre-trained models: YOLOv3 and FaceNet both considered as state of the art, are loaded using dnn module of OpenCV. For the purpose of detecting faces using YOLO we are using a model called YOLOFace[33] whose code is openly available on github. This project incorporates certain parts of that repository as well as taken the weights and config file generated by training the YOLOv3 on WiderFace Dataset[30].

Both of these models are loaded globally and used as required through out the code. This has been done to avoid reloading the models repetitively as this extremely slows down the system. No dedicated functionality is written for this purpose.

4.4 Face Detection

1. It takes each image when data is loading as seen in section 4.3
2. Passes it through YOLOv3 network (as given in [33])
3. Store all the boxes with confidence more than threshold
4. Remove all the overlapping boxes by using Non-Maximum Suppression[45]
5. Filter out unwanted boxes and returns final boxes having coordinates of left, top, width, height
6. Calculate face coordinates
7. Extract face patch
8. Create Image from extracted face array
9. Resize it and Convert it back to array and append to face_array_list
10. Go back to step 1 if more images exist
11. Else, Return list of face array with list of their coordinate

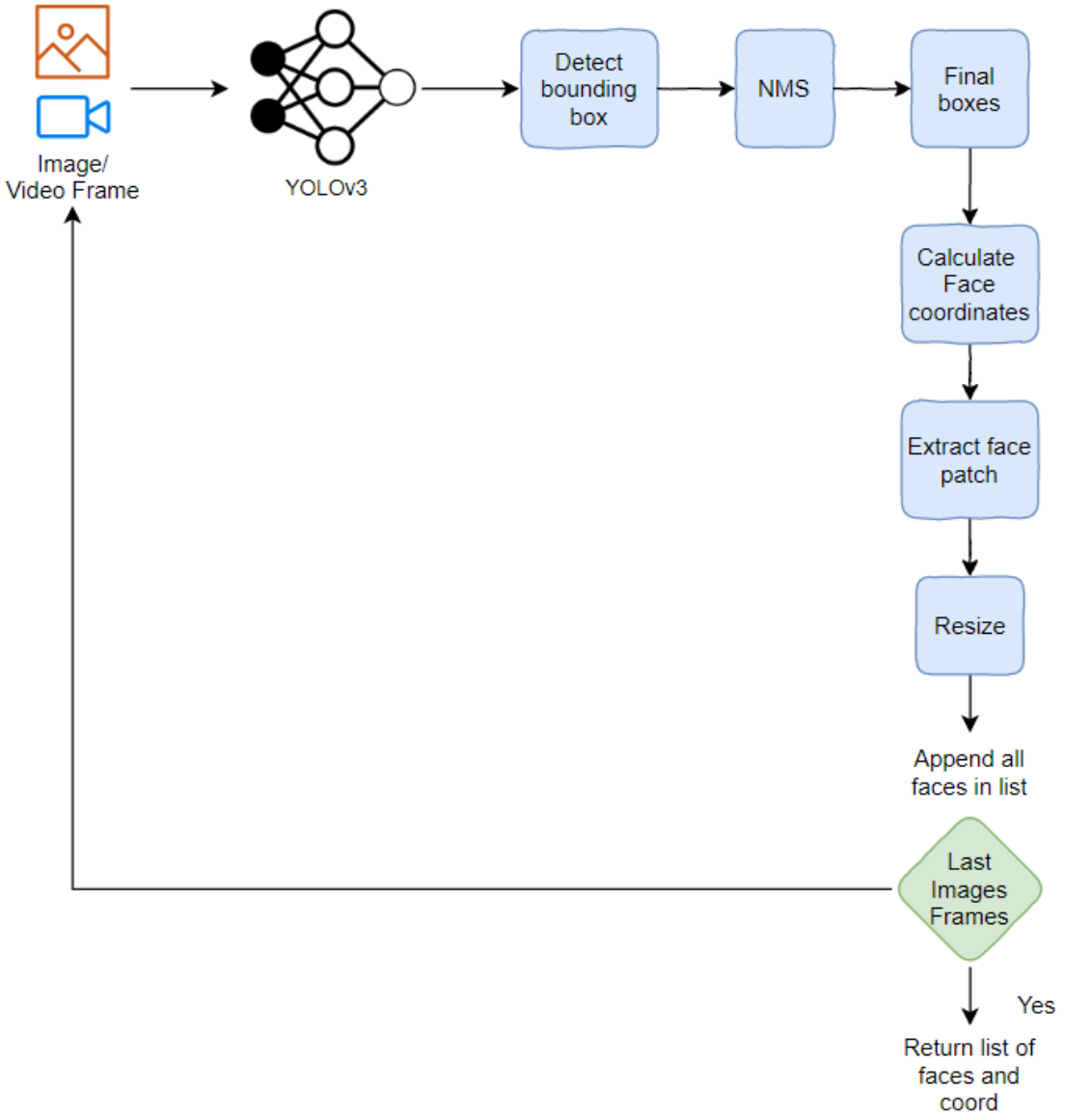

FIGURE 4.3: Stage 1: Face Detection Architecture

```
def face_detection(train_direc, test_direc):
    """
    :param train_direc: path to train directory
    :param test_direc: path to test directory
    """
    trainX, trainy = load_dataset(train_direc)
    print(trainX.shape, trainy.shape)
    # load test dataset
    testX, testy = load_dataset(test_direc)
    # save arrays to one file in compressed format
    savez_compressed('face_recog_dataset.npz', trainX, trainy, testX, testy)
```

4.5 Face Recognition

It is Stage 2 of the proposed architecture and at this stage the second state-of-the-art based network called FaceNet is used.

1. Load data from numpy array stored after Stage 1
2. Loop through face_pixel from data loaded
   (a) Scale, Standardize and Transform the Face Patch
   (b) Pass it through the FaceNet[12] model loaded earlier
   (c) Once the FaceNet model is trained, prediction is done to generate embedding.
   (d) These are 128-dimension embedding which are generated on the basis on triplet loss as explained in section 2.9.3
   (e) Append this in the list
   (f) If next face patch exist then go to step 2, else go to 3
3. Repeat step 2 for test data
4. Convert both train and test embedding into numpy arrays
5. Save the numpy arrays on the disk

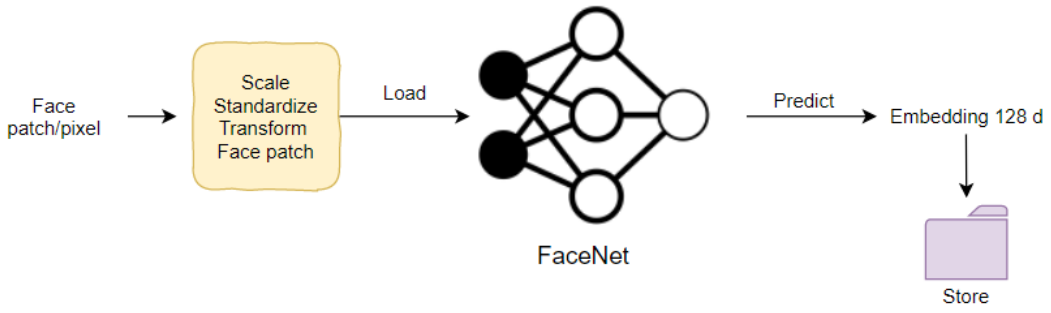

FIGURE 4.4: Stage 2: Face Recognition Architecture

```
def face_recog():
    data = load('face_recog_dataset.npz')
    trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
    print('Loaded: ', trainX.shape, trainy.shape, testX.shape, testy.shape)

    # convert each face in the train set to an embedding
    newTrainX = list()
    for face_pixels in trainX:
        embedding = calculate_embedding(model_facenet, face_pixels)
```

```
        newTrainX.append(embedding)
    newTrainX = asarray(newTrainX)
    print(newTrainX.shape)
    # convert each face in the test set to an embedding
    newTestX = list()
    for face_pixels in testX:
        embedding = calculate_embedding(model_facenet, face_pixels)
        newTestX.append(embedding)
    newTestX = asarray(newTestX)
    print(newTestX.shape)
    # save arrays to one file in compressed format
    savez_compressed('face_recog_dataset_embedding.npz', newTrainX, trainy, newTestX, testy)
    print('Embedding saved')
```

4.6 Face Classification

1. Load the embeddings from stored numpy arrays on disk
2. Normalize and transform the data
3. Initialize the classification model - KNN in this scenario
4. Fit the model
5. Store the model and label encoder on the disk for use in test
6. Calculate accuracy

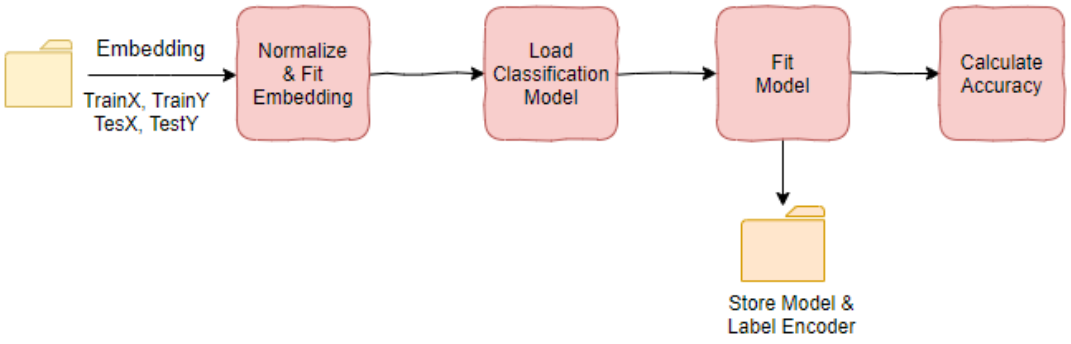

FIGURE 4.5: Stage 3: Face Classification Architecture

```
def face_class():

    # load face embeddings
    data = load('face_recog_dataset_embedding.npz')
    trainX, trainy, testX, testy = data['arr_0'], data['arr_1'], data['arr_2'], data['arr_3']
    # normalize input vectors
    in_encoder = Normalizer(norm='l2')
```

```
trainX = in_encoder.transform(trainX)
testX = in_encoder.transform(testX)
# label encode targets
out_encoder = LabelEncoder()
out_encoder.fit(trainy)
trainy = out_encoder.transform(trainy)
testy = out_encoder.transform(testy)
# fit model
model = KNeighborsClassifier()
model.fit(trainX, trainy)
calc_accuracy(trainX, trainy, testX, testy, model)
# Store the model
modelSave = open('model_knn_saved.pkl', 'wb')
pickle.dump(model, modelSave)
np.save('classes.npy', out_encoder.classes_)
print('Model & Encoder Saved')
```

4.7 Tester

1. Input to the tester is an image or frame from video
2. We also pass the model used for classification, so we can use same model to predict and out label encoder with 0 - for image and 1 - for webcam
3. This frame is passed through a pipeline similar to Stage1 and Stage2
   (a) Detect faces, Extract faces, covert to numpy arrays and add all these faces to a list, also make a list of coordinated for each face
      i. Extract face from each image as coordinates are known
      ii. Calculate embedding for each detected face from the above generated list
      iii. Now, using the trained classification model, classify each image if it belongs to any of the people trained on our dataset using model.predict and model.predict_proba
      iv. If probability less than threshold then it is marked as unknown otherwise, name is extracted using inverse_transform
      v. bounding box of green with confidence & name is displayed and similar for unknown except it is red.
4. This is repeated for each frame if it is video

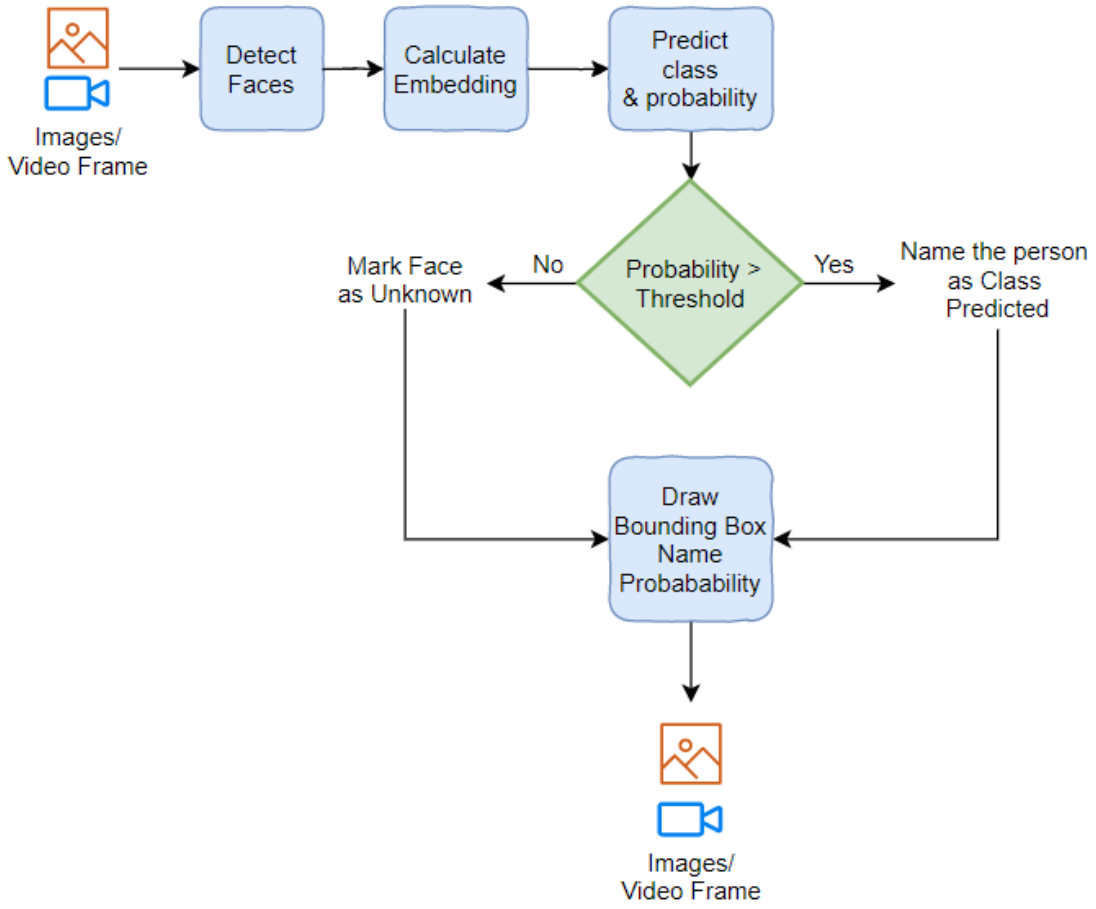

FIGURE 4.6: The Testing Architecture

```
def tester(model, out_encoder, img, test_face_frame):
    '''
    :param model: Prediction Model e.g SVM, knn, etc
    :param out_encoder: LabelEncoder on output
    :param img: int - 1--> Image Testing, 0/Else --> Web cam Testing
    :param test_face_frame: Image frame is passed to be tested
    :return: test_face_frame: The frame with bounding box
             no_detect: # Detected Faces
             no_recog: # Recognized Faces
    '''
    THRESHOLD_PROB = 95.0
    unk = 0

    if img == 1:
        test_face = test_face_frame
        image = Image.open(test_face)
        image = image.convert('RGB')
        test_face_frame = asarray(image)
        test_face_pixel_list, test_face_coord = get_face_patch(test_face_frame)
    else:
        test_face_pixel_list, test_face_coord = get_face_patch(test_face_frame)

    if isinstance(test_face_pixel_list, int):
```

```
        return test_face_frame, 0, 0

    for (i, test_face_pixel) in enumerate(test_face_pixel_list):

        test_face_embedding = calculate_embedding(model_facenet, test_face_pixel)
        samples = expand_dims(test_face_embedding, axis=0)

        yhat_class = model.predict(samples)
        yhat_prob = model.predict_proba(samples)
        # get name
        class_index = yhat_class[0]
        class_probability = yhat_prob[0, class_index] * 100
        predict_names = out_encoder.inverse_transform(yhat_class)

        if class_probability < THRESHOLD_PROB:
            predict_names[0] = 'Unknown'
            unk += 1

        print('Predicted: %s (%.3f)' % (predict_names[0], class_probability))

        x1, y1, width, height = test_face_coord[i][0], test_face_coord[i][1], test_face_coord[i]
        x1, y1 = abs(x1), abs(y1)
        x2, y2 = x1 + width, y1 + height

        no_detect = len(test_face_coord)
        no_recog = no_detect - unk

        if predict_names[0] == 'Unknown':
            cv2.rectangle(test_face_frame, (x1, y1), (x2, y2), (0, 0, 255), 2)
            cv2.putText(test_face_frame, predict_names[0] + " " + str(class_probability), (x1 -
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        else:
            cv2.rectangle(test_face_frame, (x1, y1), (x2, y2), (0, 255, 0 ), 2)
            cv2.putText(test_face_frame, predict_names[0] +" "+str(class_probability), (x1-4, y1
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    return test_face_frame, no_detect, no_recog
```

4.8 Training & Testing

4.8.1 Training

Training is a process of teaching data, what is wrong and what is right. Just like humans need to learn by doing and check if what they did matches the expected, machines also go through the same loop. Just like most humans don't become best and efficient at something, they have to learn it by doing it repetitively, similarly machines learn the pattern of data by training over no of times. For the purpose of this project, pre-trained

models have been used, yet to make it relevant to us. It is trained on custom dataset which is collected in the scope of the project.

In this thesis, the training takes place in 3 Stages: Detection(YOLOFace) [Section 4.4], Recognition(FaceNet) [Section 4.5] and Classification(KNN) [Section 4.6]. And after each stage the trained data is stored in numpy array and loaded in the next stage. Once the training data is passed through all the stages, a final trained model is ready to use, this model is further saved as pickle, so that once the system is trained it doesn't have to be trained each time it is executed.

Hence, training the model is a one time thing and it is advised to train the model whenever there is change in dataset, for instance, addition of more pictures of different people, change in name of the folder(remember this will be shown as name of the person on the display to user).

```
def train():
    face_detection(args.train, args.test)
    face_recog()
    face_class()
```

4.8.2 Testing

Once the system is trained, it is ready to be used and tested. It takes 2 types of input: Image or Web Camera Streaming. The type of input user wants to try can be given in command line arguments while running the system. If the user wants to test an image, they can give path to the image or a default image will be taken for testing. While for web-cam, it directly starts it and begin detecting and recognizing faces. In implementation the testing is divided into 2 functionalites, test_cam_image which takes 0 for image input or 1 for webcam as arguments to the function. In this function it loads the model saved after full training and also the label encoder which will be used to inverse transform to get person name. The second functionality is the where the actual testing happens, in tester [Section 4.7]. test_cam_image simply calls the tester once for image and keeps it in loop for streaming and shows the output image/video using OpenCV.

```
def test_cam_img( webcam):
    """
    :param img_path: path to test image
    :param webcam: 1 - test from web cam | 0 - test from image
    """
    img_path = args.img
    model = pickle.load(open('model_knn_saved.pkl', 'rb'))
    out_encoder = LabelEncoder()
    out_encoder.classes_ = np.load('classes.npy')
```

```
print('Model & Encoder Loaded')
if webcam:
    cap = cv2.VideoCapture(0)
    while True:
        has_frame, frame = cap.read()
        test_face_frame, no_detect, no_recog = tester(model, out_encoder, 0, frame)

        cv2.putText(test_face_frame, 'No. of faces detected:' + str(no_detect), (10, 15),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        cv2.putText(test_face_frame, 'No. of faces recognized:' + str(no_recog), (10, 45),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

        cv2.imshow('Real-time Detection', test_face_frame)

        key = cv2.waitKey(1)

        if key == 27 or key == ord('q'):
            print('[i] ==> Interrupted by user!')
            print('[i] ==> Done processing!!!')
            break

    cap.release()
    cv2.destroyAllWindows()

else:
    test_face_frame, no_detect, no_recog = tester(model, out_encoder, 1, img_path)
    cv2.putText(test_face_frame, 'No. of faces detected:' + str(no_detect), (10, 15),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    cv2.putText(test_face_frame, 'No. of faces recognized:' + str(no_recog), (10, 45),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
    cv2.imshow("Face Recognition", test_face_frame)
    cv2.waitKey()
```

4.9 Data Saved

Throughout the system wherever possible and needed the data has been captured and stored. Specially after each of 3 Stages of the system architecture. The data stored were:

1. **face_recog_dataset.npz:** Stored after Stage 1 - Face Detection, contains numpy arrays of detected and extracted faces, along with their names.
2. **face_recog_dataset_embedding.npz:** Stored after Stage 2 - Face Recognition, calculated embedding for each face is converted in numpy array and stored
3. **model_knn_saved.pkl:** Stored after Stage 3 - Face Classification, it is the final trained model which can be loaded and directly used for precition in tester.

4. **classes.npy:** Stored after Stage 3 - Face Classification, it is used to get the name of the class from the label encoded values.

4.10 Code

The code for this project is saved in CIt's google drive, can be seen and downloaded by anyone within CIT.
Link: `shorturl.at/lwJK4`
If there is any issue, please drop a mail to phalguni.rathod@mycit.ie with subject 'MSc Project - Help'

Chapter 5

Testing and Evaluation

5.1 Testing

The model has been tested rigorously on variety of data. The testing has been done on test set of collected data [Section 4.1.1]. As well as the the system have option for users to give image of their own choice for testing the system, they just have to give path of image to be tested. Along with image testing, the system can take the real time live stream and detect & recognize faces via web-camera. For ease of understanding which face are recognized in the image, they have green coloured bounding box while unrecognized ones are red.

Each image/video-frame has to go through a pipeline for testing and shown in Figure 5.5

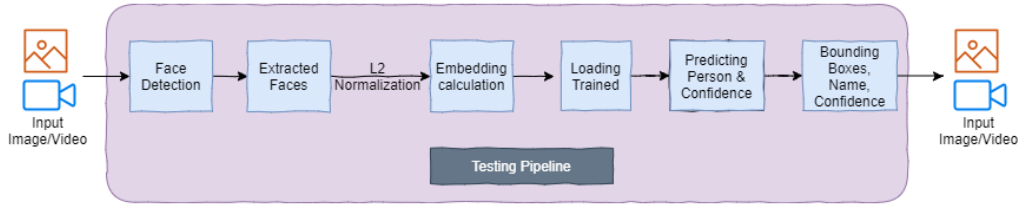

FIGURE 5.1: Testing Pipeline

Flow of Pipeline

1. Send the Image/Frame for detect all the faces
2. Cut out all the face patches
3. Do L2 Normalization and Calculate embedding for each face patch

4. Load the trained model
5. Predict person and name accordingly
6. Draw bounding box around each face, Green: Recognized, Red: Unknown
7. Display Image/Frame

5.2 Evaluation

The only component where experimentation can be done is Face Classification [Section 4.6]. Experimentation can be done by running the system on variety of classifiers. For the purpose of doing experiments and evaluation 5 classifiers have been chosen, namely: KNN, Decision Tree, Random Forest and Gradient Boosting.

The system was executed with each of these classifier as part of Stage 3 of the system and classification report for both train and test is taken for evaluating their performance.

Accuracy: We compare the predicted target values with actual labels and find out what percentage of values are correctly predicted.

Confusion Matrix: It gives us deeper insight of the predicted actual classes. It tells us about how many True Positives, True Negative, False Positives False Negatives are found in the prediction.

Recall: It shows us how confident we are that all of the instances that belong to a specific class have been classified correctly using our model.

Precision: It shows us how confident we are that all any instance predicted as belonging to a certain class actually belongs to that class.

F1-Score: Rather than having Recall Precision as 2 different metrics, they can be combined to form the F1-Score. It is harmonic mean of precision recall. For F1 to be high both precision recall has to be high. F1 doesn't get skewed on just a single value.

5.2.1 KNN

```
Test:
              precision    recall  f1-score   support

           0       0.78      1.00      0.88         7
           1       1.00      1.00      1.00         5
           2       1.00      0.92      0.96        13
           3       1.00      0.83      0.91         6
           4       1.00      1.00      1.00         4

    accuracy                           0.94        35
   macro avg       0.96      0.95      0.95        35
weighted avg       0.96      0.94      0.94        35

Accuracy: train=98.413, test=94.286
Model Saved: KNN
```

FIGURE 5.2: KNN Classification Report

5.2.2 Decision Tree

```
Test:
              precision    recall  f1-score   support

           0       0.80      0.57      0.67         7
           1       0.62      1.00      0.77         5
           2       1.00      0.62      0.76        13
           3       0.60      1.00      0.75         6
           4       0.75      0.75      0.75         4

    accuracy                           0.74        35
   macro avg       0.76      0.79      0.74        35
weighted avg       0.81      0.74      0.74        35

Accuracy: train=100.000, test=74.286
Model Saved: DT
```

FIGURE 5.3: DT Classification Report

5.2.3 Gradient Boosting

```
Test:
              precision    recall  f1-score   support

           0       1.00      0.57      0.73         7
           1       0.45      1.00      0.62         5
           2       1.00      0.62      0.76        13
           3       0.62      0.83      0.71         6
           4       1.00      1.00      1.00         4

    accuracy                           0.74        35
   macro avg       0.82      0.80      0.77        35
weighted avg       0.86      0.74      0.75        35

Accuracy: train=100.000, test=74.286
Model Saved: GBM
Model & Encoder Saved
```

FIGURE 5.4: GBM Classification Report

5.2.4 Random Forest

```
Test:
              precision    recall  f1-score   support

           0       0.75      0.86      0.80         7
           1       0.83      1.00      0.91         5
           2       1.00      0.92      0.96        13
           3       1.00      0.83      0.91         6
           4       1.00      1.00      1.00         4

    accuracy                           0.91        35
   macro avg       0.92      0.92      0.92        35
weighted avg       0.93      0.91      0.92        35

Accuracy: train=98.413, test=91.429
Model Saved: RF
```

FIGURE 5.5: RF Classification Report

5.2.5 Model Comparison

The models are evaluated are on the basis of f1-score because it is combination of both precision and recall, f1-score is low if even one of them is low, hence to get the comprised version on precision and recall, we use f1-score.

From the above classification report for each model, it can see that KNN is leading with highest test score of about 94.286% followed by Random Forest having score of 91.429%. Both Decision Tree and Gradient Boosting seems to be lagging a lot in terms of score as each have 74.286%. Another point observed is that training accuracy for Decision Tree and Gradient boosting is reaching 100% which is a clear sign of overfitting. Some digging for the reason behind such behaviour is done but no proper reason can be given.

Hence, For the final system KNN was chosen as the classifier for predicting people. Even if KNN mis-classify certain faces some times, but most of the times it does the work correctly.

The extent of success in term of accuracy was not decided or had any fixed expectations of it, given that it reached decent accuracy score of 94.268% when used KNN as classifier for Stage 3.

5.3 Test Run

1. **Step 1- run:** YOLO_FaceNet_Classification_MS_Project.py -h

```
(venv) E:\phalguni\MSc\FaceRecognition\YoloFaceNet_MSC_Project>YOLO_FaceNet_Classification_MS_Project.py -help
2020-05-17 18:34:32.256912: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudart64_100.dll
usage: YOLO_FaceNet_Classification_MS_Project.py [-h] [--training TRAINING]
                                                 [--oper OPER] [--train TRAIN]
                                                 [--test TEST] [--img IMG]
```

FIGURE 5.6: Step - 1

2. **Step 2:** Pass desired combination of arguments, if doesn't want anything, simply run - it will take default parameters

```
parser.add_argument('--training', type=int, default=0, help='0 - Use Pre-Trained | 1 - Train on default data')
parser.add_argument('--oper', type=int, default=0, help='1 - Webcam | 0 - Image - Give test image path --img')
parser.add_argument('--train', type=str, default='./face_recog_dataset/train/', help='path of the training folder')
parser.add_argument('--test', type=str, default='./face_recog_dataset/test/', help='path of the testing folder')
parser.add_argument('--img', type=str, default='./face_recog_dataset/test1/group_05.jpg', help='path of the test image')
```

FIGURE 5.7: Step - 2

3. Step 3: Get Desired output, more in section 5.4

5.4 Outputs

5.4.1 Image

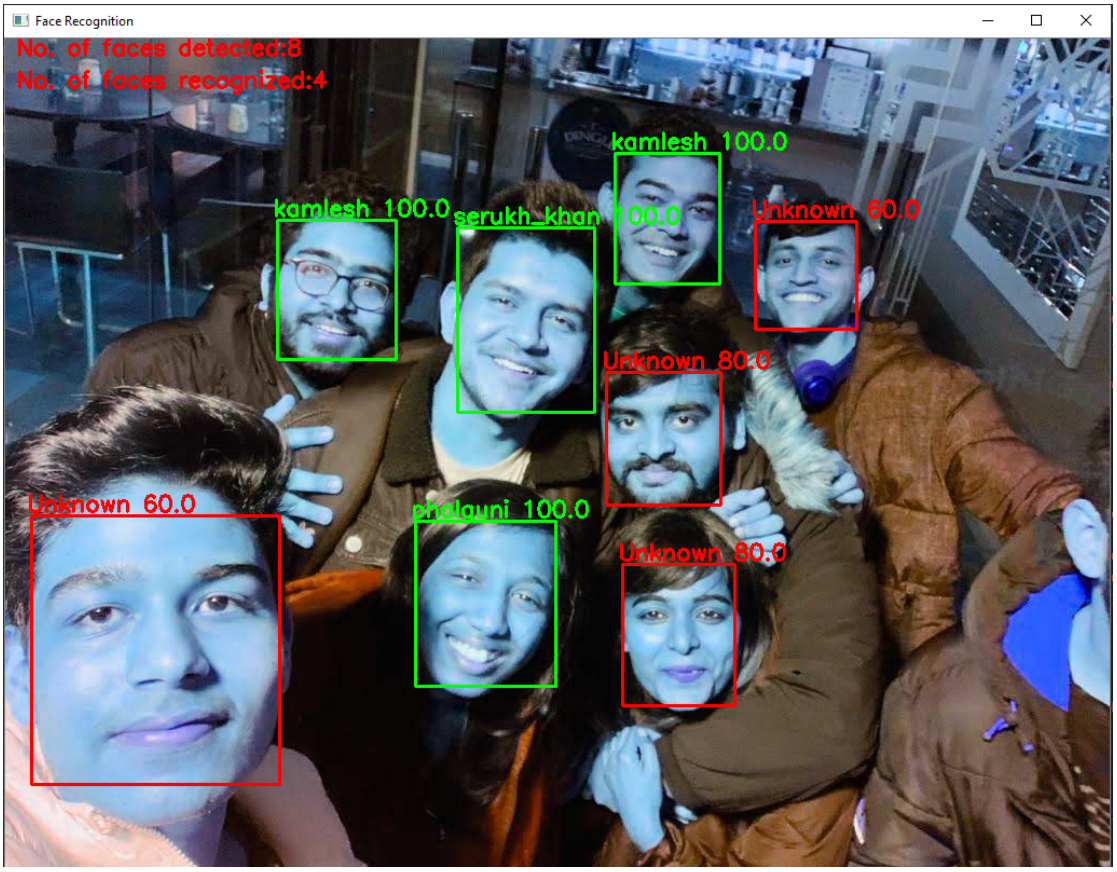

FIGURE 5.8: Test Output - 1

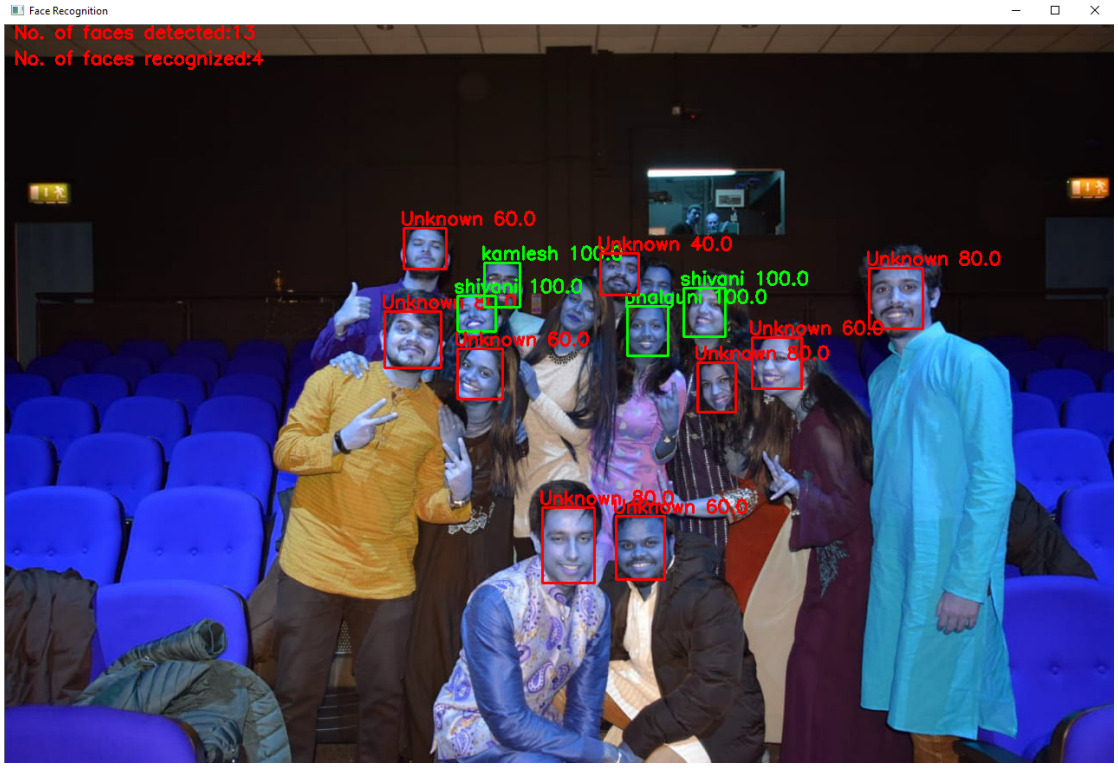

FIGURE 5.9: Test Output - 2

5.4.2 Web Cam

To show this test a acreen recoding of this is uploaded on Youtube, can be found here: `https://youtu.be/83F3ZW480x0`

Chapter 6

Discussion and Conclusions

This chapter finally concludes the thesis by discussing the solution to research questions, also talks about how the motivation have driven the thesis and how this thesis can be taken to next step as suggestions given in future work.

6.1 Discussion

The project was able to successfully achieve all of its objectives and could find answer to all the research question.

- **How data is collected for building face related dataset?**
  Done. Variety of images were collected from friends and family, few challenges were face. More of this is discussed in detail in section 4.1
- **How YOLO can be used to detect faces?**
  Completed. This has been discussed in section 2.6 and its usage is done in section 4.4.
- **How FaceNet is implemented?**
  This has been done and implemented in 4.6
- **How we can integrate YOLO based face detection with FaceNet?**
  Successfully done. To answer this research question, whole new 3 stage architecture is built from scratch, discussed in section 3.4 and implement in section 4.4 and section 4.6
- **Which algorithm to choose for classifying the faces?**
  Achieved. Discussion on this is done in section 5.2

- **How to do face recognition in real-time camera stream?**
  This goal is achieved and discussed in sections - 4.7 and 5.1.

6.2 Conclusion

The motivation for the thesis is to learn more about Face Detection and Recognition technology which is successfully achieved as, to complete the project many resources have been referred and a lot of time was spent of reading academic and state-of the-art papers. The knowledge gained from papers appeared at various places in the thesis while at places like while deciding the architecture of the model some ideas came from paper like rcnn and few came from facenet, while none wholely holds the credit, knowledge gained by reading papers came handy like this.

Another motivation was to work along with 2 fascinating state-of-the-art algorithms YOLO and FaceNet which was done successfully as one is used for face detection and other Face Recognition, respective. 3 Staged architecture is proposed, built and implemented from scratch in order to achieve this.

This thesis was able to achieve all its objectives and could find answers to all the research questions.

6.3 Future Work

There is a great scope of improvement in this system, few of them like:

- Making it more optimised, faster and smoother
- Compatible with videos
- Downsize the stages
- Try clubbing all the 3 stages into a single stage and make end-to-end face detection system
- Experimenting with more classifiers and hyper tuning them for best results
- Improving latency in realtime live streaming
- Experiment with various architectures of YOLO as well as FaceNet

It's ok if you haven't got your letter to *Hogwarts*!
I'm waiting for mine too...
Until then lets create our own magic with *AI*

FIGURE 6.1: Let's Create Magic!

Bibliography

[1] The history of artificial intelligence.

[2] The magic behind new tech: Big data, ai, and machine and deep learning.

[3] Unraveling artificial intelligence for the non-geeks—the non-technical insight.

[4] A. Vahab, M. S. Naik, P. G. Raikar, and P. SR, "Applications of object detection system," 2019.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2015.

[6] More than one billion smartphones to feature facial recognition in 2020.

[7] I. G. N. M. K. Raya, A. N. Jati, and R. E. Saputra, "Analysis realization of viola-jones method for face detection on cctv camera based on embedded system," *2017 International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics)*, pp. 1–5, 2017.

[8] Face detection for beginners.

[9] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multi-task cascaded convolutional networks," 2016.

[10] D. Garg, P. Goel, S. Pandya, A. Ganatra, and K. Kotecha, "A deep learning approach for face detection using yolo," in *2018 IEEE Punecon*, 2018, pp. 1–4.

[11] B. Amos, B. Ludwiczuk, and M. Satyanarayanan, "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, CMU School of Computer Science, Tech. Rep., 2016.

[12] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298682

[13] "Turing, alan mathison, (23 june 1912–7 june 1954), reader in mathematics, manchester university, since 1948," 12 2007.

[14] I. B. A. TURING, "Computing machinery and intelligence-am turing," *Mind*, vol. 59, no. 236, p. 433, 1950.

[15] A. Newell and H. Simon, "The logic theory machine–a complex information processing system," *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 61–79, 1956.

[16] How to understand what is what is not artificial intelligence?

[17] Introduction to the world of ai.

[18] A 6 minute intro to ai.

[19] A. Ulhaq, A. Khan, D. Gomes, and M. Pau, "Computer vision for covid-19 control: A survey," 2020.

[20] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.

[21] Council of European Union, "Council regulation (EU) no 269/2014."

[22] R. Girshick, "Fast r-cnn," 2015.

[23] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[24] Search by people, things places in your photos0.

[25] Breaking down facial recognition: The viola-jones algorithm.

[26] L. Cuimei, Q. Zhiliang, J. Nan, and W. Jianhua, "Human face detection algorithm via haar cascade classifier combined with three additional classifiers," in *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, 2017, pp. 483–487.

[27] A. Srivastava, S. Mane, A. Shah, N. Shrivastava, and B. Thakare, "A survey of face detection algorithms," in *2017 International Conference on Inventive Systems and Control (ICISC)*, 2017, pp. 1–4.

[28] W. Yang and Z. Jiachun, "Real-time face detection based on yolo," in *2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII)*, 2018, pp. 221–224.

[29] M. S. A. Vigil, M. M. Barhanpurkar, N. R. Anand, Y. Soni, and A. Anand, "Eye spy face detection and identification using yolo," in *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, 2019, pp. 105–110.

[30] S. Yang, P. Luo, C. C. Loy, and X. Tang, "Wider face: A face detection benchmark," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[31] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[32] V. Jain and E. Learned-Miller, "Fddb: A benchmark for face detection in unconstrained settings," University of Massachusetts, Amherst, Tech. Rep. UM-CS-2010-009, 2010.

[33] Yoloface - deep learning based face detection using the yolov3 algorithm.

[34] M. D. Kelly, *Visual identification of people by computer.* Department of Computer Science, Stanford University., 1970, no. 130.

[35] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," 2015.

[36] S. Mallat, "Understanding deep convolutional networks," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150203, 2016.

[37] Euclidean distance (l2 norm).

[38] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," 2013.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[40] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM computing surveys (CSUR)*, vol. 35, no. 4, pp. 399–458, 2003.

[41] G. Van Rossum and F. L. Drake Jr, *Python tutorial.* Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[42] T. E. Oliphant, *A guide to NumPy.* Trelgol Publishing USA, 2006, vol. 1.

[43] A. Clark, "Pillow (pil fork) documentation," 2015. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[44] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[45] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," 2017.