

# **FACIAL RECOGNITION USING DEEP LEARNING**

**A PROJECT REPORT**

*Submitted By*

**NIKHILESH.M      312214104062**

**KAUSHIK NARAYANAN      312214104042**

**SAKET RAM      312214104039**

*in partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**SSN COLLEGE OF ENGINEERING**

**KALAVAKKAM 603110**

**ANNA UNIVERSITY :: CHENNAI - 600025**

**April 2018**

**ANNA UNIVERSITY : CHENNAI 600025**

**BONAFIDE CERTIFICATE**

Certified that this project report titled “ **Facial Recognition using Deep Learning**” is the *bonafide* work of “**NIKHILESH.M (312214104062), KAUSHIK NARAYANAN (312214104042), and SAKET RAM (312214104039)**” who carried out the project work under my supervision.

**Dr. Chitra Babu**

**Head of the Department**

Professor,  
Department of CSE,  
SSN College of Engineering,  
Kalavakkam - 603 110

**Dr. P. Mirunalini**

**Supervisor**

Associate Professor,  
Department of CSE,  
SSN College of Engineering,  
Kalavakkam - 603 110

Place:

Date:

Submitted for the examination held on.....

**Internal Examiner**

**External Examiner**

## ACKNOWLEDGEMENTS

We thank GOD, the almighty for giving us the strength and knowledge to do this project.

We would like to thank with a deep sense of gratitude to our guide **Dr.P.MIRUNALINI**, Associate Professor, Department of Computer Science and Engineering, for her valuable advice and suggestions as well as her continued guidance, patience and support that helped us to shape and refine our work.

Our sincere thanks to **Dr.CHITRA BABU**, Professor and Head of the Department of Computer Science and Engineering, for her words of advice and encouragement and we would like to thank our project Coordinator **Dr. S.SHEERAZUDDIN**, Professor, Department of Computer Science and Engineering for his valuable suggestions throughout the phase of the project.

We express our deep respect to the founder **Dr. SHIV NADAR**, Chairman, SSN Institutions. We also express our appreciation to our **Dr. S. SALIVAHANAN**, Principal, for all the help he has rendered during this course of study. We would also like to thank our management for providing us the HPC lab and the GPU system to help carry out our project.

We would like to extend my sincere thanks to all the teaching and non-teaching staffs of our department who have contributed directly and indirectly during the course of our project work. Finally, we would like to thank our parents and friends for their patience, cooperation and moral support throughout our lives.

**NIKHILESH.M**

**KAUSHIK NARAYANAN**

**SAKET RAM**

## **ABSTRACT**

A facial recognition system is a computer application capable of identifying or verifying a person from a digital image or a video frame. It is one of the popular ongoing research fields with applications that require high percentage of accuracy. It presents a challenging problem due to the difficulty that arises from the enormous amount of variability on account of images taken from constrained and unconstrained environments. Hand crafted feature engineering may lead to poor performance and it is time consuming in this domain. We propose three different automated systems based on transfer learning and deep neural network to carry out face recognition and comparing their performances. In the first two methods, we use transfer learning technique which makes use of two pre-trained models: OpenFace model and Google Inception v3 model, for feature engineering. Using the extracted feature vectors, a multi-class Support Vector Machine (SVM) classifier has been trained for classification. In the final method, we develop a deep neural network using different layers of a Convolutional Neural Network (CNN) to train and classify the input images of different classes. The system was evaluated with three different datasets having images obtained from both constrained and unconstrained environments: ORL database, a subset of the LFW dataset and the extended Yale face database B.

# TABLE OF CONTENTS

<b>ABSTRACT</b>	<b>iii</b>
<b>LIST OF TABLES</b>	<b>vii</b>
<b>LIST OF FIGURES</b>	<b>viii</b>
<b>1 INTRODUCTION</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Deep Learning . . . . .	3
1.2.1 Concepts . . . . .	4
1.2.2 Why Deep Learning over ML? . . . . .	5
<b>2 LITERATURE SURVEY</b>	<b>6</b>
<b>3 SYSTEM REQUIREMENTS</b>	<b>8</b>
3.1 Hardware Requirements . . . . .	8
3.2 Software Requirements . . . . .	8
3.3 Operating System . . . . .	9
<b>4 METHODS AND ALGORITHMS USED</b>	<b>10</b>
4.1 Face Detection . . . . .	10
4.2 Preprocessing . . . . .	13
4.2.1 Reshaping . . . . .	13
4.2.2 Denoising . . . . .	13
4.2.3 Dlib - Real TimeFace Pose Estimation . . . . .	14

4.2.4	Affine Transformation . . . . .	15
4.3	Transfer Learning . . . . .	16
4.4	Google's Inception v3 Model . . . . .	18
4.4.1	Inception Modules . . . . .	19
4.4.2	Dimensionality Reduction . . . . .	21
4.4.3	Layers in Detail . . . . .	21
4.5	OpenFace Model . . . . .	23
4.6	SVM Classifier . . . . .	23
4.6.1	Multi-Class SVM . . . . .	24
4.6.2	Categorical Feature . . . . .	25
4.6.3	Scaling . . . . .	26
4.6.4	RBF Kernel . . . . .	26
4.6.5	Cross-Validation and Grid-Search . . . . .	27
4.7	CNN . . . . .	27
4.7.1	Convolutional Layer . . . . .	28
4.7.2	Pooling Layer . . . . .	29
4.7.3	Output Layer . . . . .	29
<b>5</b>	<b>PROBLEM DEFINITION AND PROPOSED SYSTEM</b>	<b>31</b>
5.1	Problem Statement . . . . .	31
5.2	Proposed System . . . . .	31
5.2.1	Preprocessing . . . . .	34
5.2.2	Classification based on transfer learning technique using OpenFace model. . . . .	35
5.2.3	Classification based on transfer learning technique using Google's Inception v3 model . . . . .	36

5.2.4	Classification using CNN . . . . .	38
<b>6</b>	<b>EXPERIMENTS AND RESULTS</b>	<b>39</b>
6.1	Dataset . . . . .	39
6.1.1	ORL Database . . . . .	39
6.1.2	LFW Dataset . . . . .	40
6.1.3	The Extended Yale Face Database B . . . . .	41
6.2	Performance Analysis . . . . .	42
6.2.1	Results - Classification Using Google Inception Model . .	43
6.2.2	Results - Deep Neural Network . . . . .	47
6.2.3	Comparison of classification from all three methods . . .	49
<b>7</b>	<b>CONCLUSION AND FUTURE WORK</b>	<b>50</b>
<b>A</b>	<b>INSTALLATION</b>	<b>51</b>
A.1	Installing Python and Setting up Environment . . . . .	51
	<b>REFERENCES</b>	<b>54</b>

## LIST OF TABLES

6.1	Datasets . . . . .	42
6.2	Accuracy using Google's Inception v3 Model . . . . .	44
6.3	Confusion Matrix of Extended Yale B Database using Inceptionv3	45
6.4	Accuracy using OpenFace Model . . . . .	45
6.5	Confusion Matrix of Extended Yale B Database using OpenFace .	46
6.6	CNN . . . . .	47



## LIST OF FIGURES

4.1	Haar-like features . . . . .	11
4.2	Different stages in visualization . . . . .	12
4.3	Visualizing the 68 facial landmark coordinates . . . . .	15
4.4	Fine Tuning . . . . .	19
4.5	Inceptionv3 Architecure . . . . .	20
4.6	Inception Module . . . . .	22
4.7	FaceNet’s nn4 Architecture . . . . .	23
5.1	Architectural Diagram . . . . .	33
6.1	ORL Database . . . . .	40
6.2	LFW Dataset . . . . .	41
6.3	Extended Yale Database B . . . . .	42
6.4	Hyperparameters Tuning using Grid-Search . . . . .	43
6.5	Result using Google Inception Model . . . . .	44
6.6	Result using OpenFace Model . . . . .	46
6.7	Result using CNN . . . . .	48
6.8	CNN Model Accuracy Graph . . . . .	48
6.9	CNN Model Loss Graph . . . . .	49

## CHAPTER 1

# INTRODUCTION

Face recognition describes a biometric technology that attempts to establish an individual's identity. Also known as facial recognition or face detection, the process works using a computer application that captures a digital image of an individual's face (sometimes taken from a video frame) and compares it to images in a database of stored records. Images can be captured in different environments, say constrained or unconstrained. Images from a constrained environment will have certain constraints such as standoff distance, background colour, lighting conditions etc., whereas images from an unconstrained environment are not subject to any such constraints. It is typically used in security systems, biometric authentication, surveillance systems, face identification, payments, criminal identification etc. It can be compared to other biometrics such as fingerprint or eye iris recognition systems. Recently, it has also become popular as a commercial identification and marketing tool. All these applications are highly sensitive and require a high degree of accuracy, otherwise the purpose of the system breaks down. So an automated system with high accuracy is desirable.

## 1.1 Motivation

For a real face recognition system, the input images can be obtained from constrained or unconstrained environments. A good facial recognition system must achieve high accuracy on both environments. There are many machine

learning techniques that can be used for facial recognition but it requires manual feature selection and feature extraction. Machine learning uses algorithms to parse data, learn from that data, and make informed decisions based on what it has learned. Feature engineering is expensive and difficult operation. For an input 2D face, selection and extraction of features, usually very large in number would be a cumbersome process. Machine learning fails to be robust towards orientation and emotional changes that can be seen on a facial input of the same class. Also, extraction of features from unconstrained images proves to be very challenging, leading to lower accuracies of the final system. Whereas deep learning, structures algorithms in layers to create an artificial neural network that can learn and make intelligent decisions on its own for recognizing facial patterns and classifying them, and also automatically extracts features at every layer. Using Deep learning technique, the system learns the entire data distribution in the case of both constrained and unconstrained images. We apply the concept of transfer learning, a deep learning technique for feature engineering. Transfer learning is a research problem that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. Lower layers are good in identifying low-level features. Top layers distinguish the specific high-level features of each class. Therefore we can build systems with better accuracies, using transfer learning. We also built another system using deep neural networks, Where the system learns automatically from the input images and attains high accuracies. The performances of the proposed systems are then compared.

## 1.2 Deep Learning

Learning multiple levels of representations of the underlying distribution of the data to be modelled. Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Deep learning models are loosely related to information processing and communication patterns in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neuronal responses in the brain.

Deep learning architectures such as deep neural networks, deep belief networks and recurrent neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics and drug design, where they have produced results comparable to and in some cases superior to human experts.

Deep learning is a class of machine learning algorithms that:

- Use a cascade of multiple layers of nonlinear processing units for feature extraction and transformation. Each successive layer uses the output from the previous layer as input
- Learn in supervised (e.g., classification) and/or unsupervised (e.g., pattern analysis) manners

- Learn multiple levels of representations that correspond to different levels of abstraction; the levels form a hierarchy of concepts

### 1.2.1 Concepts

The assumption underlying distributed representations is that observed data are generated by the interactions of layered factors.

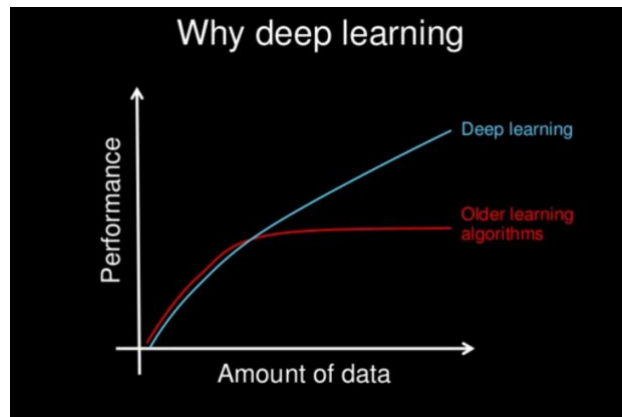
Deep learning adds the assumption that these layers of factors correspond to levels of abstraction or composition. Varying numbers of layers and layer sizes can provide different degrees of abstraction.

Deep learning exploits this idea of hierarchical explanatory factors where higher level, more abstract concepts are learned from the lower level ones.

Deep learning architectures are often constructed with a greedy layer-by-layer method. Deep learning helps to disentangle these abstractions and pick out which features improve performance.

For supervised learning tasks, deep learning methods obviate feature engineering, by translating the data into compact intermediate representations akin to principal components, and derive layered structures that remove redundancy in representation.

Deep learning algorithms can be applied to unsupervised learning tasks. This is an important benefit because unlabeled data are more abundant than labeled data. Examples of deep structures that can be trained in an unsupervised manner are neural history compressors and deep belief networks.



### 1.2.2 Why Deep Learning over ML?

- Automatically extracts the low and high-level features for classification
- Good classification performance in face recognition
- Time consuming hand-crafted feature design is eliminated

## CHAPTER 2

# LITERATURE SURVEY

The work proposed in [1] describes an efficient method for facial image representation based on local binary pattern (LBP) texture features. These features have been used as a face descriptors and the performance of the proposed method is assessed in the face recognition problem under different challenges. The work cited in [11],describes the six experimental grand challenge problem, data corpus, and presents baseline performance on natural statistics of facial imagery. The framework proposed in [15] discusses two crucial issues in face recognition: feature extraction and robustness to occlusion. In this work the recognition problem has been casted as classification using multiple linear regression models.

Another work discussed in [3] achieves 90 percent correct recognition using geometrical features and perfect recognition using template matching.A method has been proposed in [5],which exploits the fact that the set of images of an object in fixed pose, but under all possible illumination conditions, is a convex cone in the space of images. Using a small number of training images of each face taken with different lighting directions, the shape and albedo of the face can be reconstructed. In turn, this reconstruction serves as a generative model that can be used to render (or synthesize) images of the face under novel poses and illumination conditions. The pose space is then sampled and, for each pose, the corresponding illumination cone is approximated by a low-dimensional linear subspace whose basis vectors are estimated using the generative model. Our recognition algorithm assigns to a test image the identity of the closest

approximated illumination cone. Test results show that the method performs almost without error, except on the most extreme lighting directions.

In [2] a theoretical model and a functional model has been proposed for recognizing familiar faces involves a match between the products of structural encoding and previously stored structural codes describing the appearance of familiar faces, held in face recognition units. Identityspecific semantic codes are then accessed from person identity nodes, and subsequently name codes are retrieved. In [6] et al. has proposed an appearance-based face recognition method called the Laplacianface approach. By using locality preserving projections (LPP), the face images are mapped into a face subspace for analysis.

The work in [8] presents a hybrid neural-network for human face recognition which compares favourably with other methods. The system combines local image sampling, a self-organizing map (SOM) neural network, and a convolutional neural network. The goal of [10] was twofold: to discuss the trade off between data purity and time and to traverse through the complexities of deep network training and face recognition to present methods and procedures to achieve comparable state of the art results on the standard LFW and YTF face benchmarks. In [14] a system is proposed , called FaceNet, that directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity.



## CHAPTER 3

# SYSTEM REQUIREMENTS

This chapter discusses the hardware requirements, outlined the software components used (e.g., machine learning libraries / frameworks needed, underlying operating system) and methodologies for implementing the proposed system.

### 3.1 Hardware Requirements

- **Processor:** 4x Intel(R) Core(TM) Xeon CPU @ 1.70GHz
- **GPU:** NVIDIA GTX 1080 Ti
- **GPU MEMMORY:** 12 GB
- **RAM:** 8 GB
- **Hard disk drive:** Seagate 1.0TB HDD(ST1000LM024 HN-M SCSI Disk Device)

### 3.2 Software Requirements

- Conda 4.4.7
  - Python: 3.5.4

- Scipy: 1.0.0
  - Numpy: 1.13.3
  - Matplotlib: 2.1.1
  - Pandas: 0.22.0
  - Statsmodels: 0.8.0
  - Sklearn: 0.19.1
- Tensorflow: 1.4.0
  - Keras: 2.1.3

### **3.3 Operating System**

- Ubuntu 16.04 LTS

## CHAPTER 4

# METHODS AND ALGORITHMS USED

In this chapter, we describe the methods and algorithms used in our three proposed systems. The main sections are as follows:

- Preprocessing
- Transfer Learning
- OpenFace Model
- SVM Classifier
- CNN

## 4.1 Face Detection

The first step in our pipeline is face detection. Obviously we need to locate the faces in a photograph(images) or in videos before we can try to tell them apart.Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images.Face detection can be regarded as a specific case of object-class detection.Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stores in database. Any facial feature changes in the database will

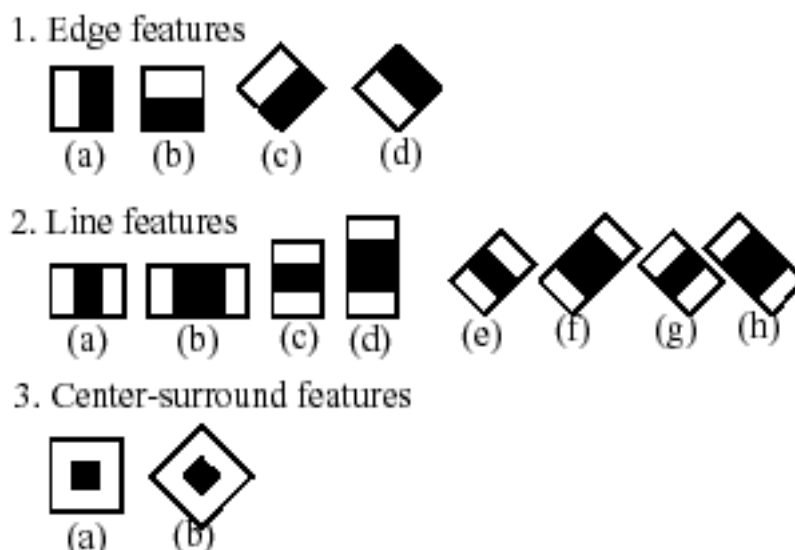


FIGURE 4.1: Haar-like features

invalidate the matching process. Face detection can be done OpenCV. OpenCV provides us with pre-trained and ready to be used for face detection classifiers:

**Haar Classifier** The Haar Classifier is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones; which (as mentioned before) are trained from many many positive images (with faces) and negatives images (without faces).

Each window is placed on the picture to calculate a single feature. This feature is a single value obtained by subtracting the sum of pixels under the white part of the window from the sum of the pixels under the black part of the window.

Now, all possible sizes of each window are placed on all possible locations of each image to calculate plenty of features.

For example, in above image, we are extracting two features. The first one focuses on the property that the region of the eyes is often darker than the area of the nose

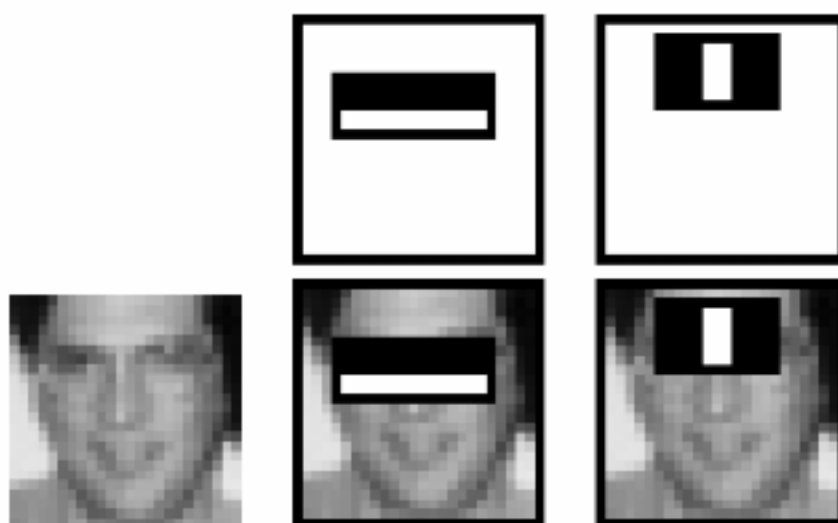


FIGURE 4.2: Different stages in visualization

and cheeks. The second feature relies on the property that the eyes are darker than the bridge of the nose.

But among all these features calculated, most of them are irrelevant. For example, when used on the cheek, the windows become irrelevant because none of these areas are darker or lighter than other regions on the cheeks, all sectors here are the same.

So we promptly discard irrelevant features and keep only those relevant with a fancy technique called Adaboost. AdaBoost is a training process for face detection, which selects only those features known to improve the classification (face/non-face) accuracy of our classifier.

In the end, the algorithm considers the fact that generally: most of the region in an image is a non-face region. Considering this, its a better idea to have a simple method to check if a window is a non-face region, and if it's not, discard it right away and dont process it again. So we can focus mostly on the area where a face is.

## 4.2 Preprocessing

Preprocessing of the input images are done in order to remove the noise and to align the images with different poses using Affine transformation [13].

### 4.2.1 Reshaping

Reshaping the dimensions of image matrix by giving size as argument. The dimensions of the image is 96x96 after resizing.

### 4.2.2 Denoising

**Mean Filter:** Mean filtering is a simple, intuitive and easy to implement method of smoothing images, i.e. reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images. The idea of mean filtering is simply to replace each pixel value in an image with the mean ('average') value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, which represents the shape and size of the neighborhood to be sampled when calculating the mean. Often a 3x3 square kernel is used, as shown in Figure 3, although larger kernels (e.g. 5x5 squares) can be used for more severe smoothing. (Note that a small kernel can be applied more than once in order to produce a similar but not identical effect as a single pass with a large kernel.)

Computing the straightforward convolution of an image with this kernel carries out the mean filtering process.

### **4.2.3 Dlib - Real TimeFace Pose Estimation**

Detecting facial landmarks is a subset of the shape prediction problem. Given an input image (and normally an ROI that specifies the object of interest), a shape predictor attempts to localize key points of interest along the shape. In the context of facial landmarks, our goal is to detect important key facial structures in the face region. There are a variety of facial landmark detectors, but all methods essentially try to localize and label the following facial regions: Mouth, Right eyebrow, Left eyebrow, Right eye, Left eye, Nose, Jaw

The method starts by using: A training set of labeled facial landmarks on an image. These images are manually labeled, specifying specific (x, y)-coordinates of regions surrounding each facial structure. Priors, of more specifically, the probability on distance between pairs of input pixels. Given this training data, an ensemble of regression trees are trained to estimate the facial landmark positions directly from the pixel intensities themselves (i.e., no feature extraction is taking place).

The pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face.

The indexes of the 68 coordinates can be visualized on the image below:

These annotations are part of the 68 point iBUG 300-W dataset which the dlib facial landmark predictor was trained on.

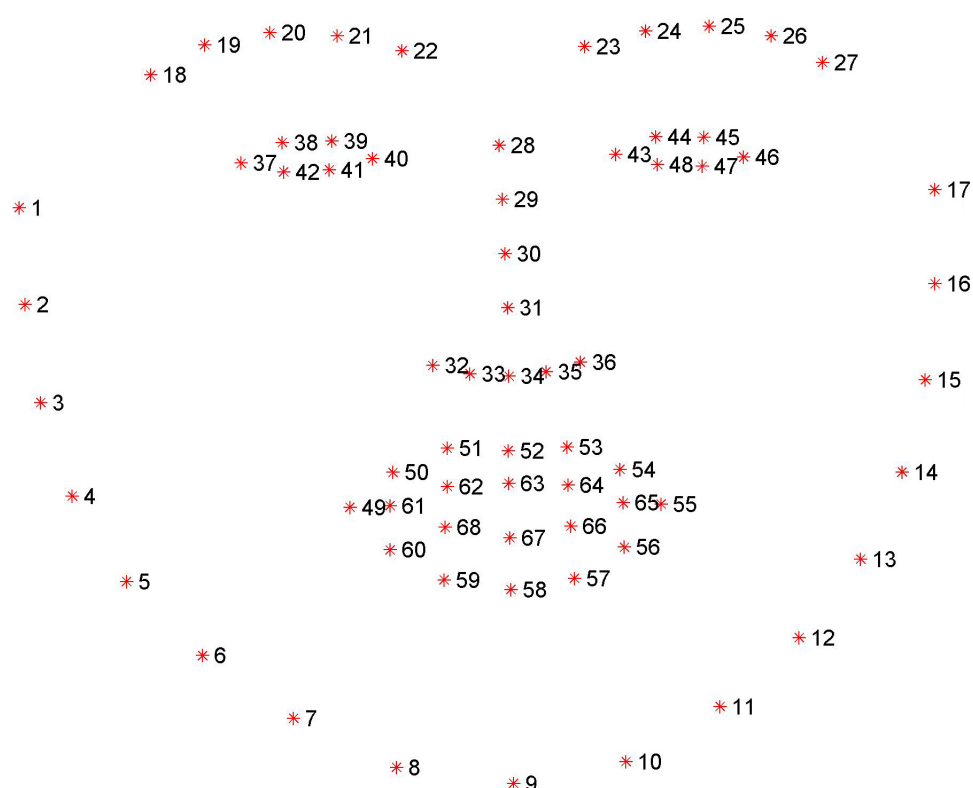


FIGURE 4.3: Visualizing the 68 facial landmark coordinates

It's important to note that other flavors of facial landmark detectors exist, including the 194 point model that can be trained on the HELEN dataset.

Regardless of which dataset is used, the same dlib framework can be leveraged to train a shape predictor on the input training data. This is useful if you would like to train facial landmark detectors or custom shape predictors of your own.

## 4.2.4 Affine Transformation

OpenCV's affine transformation is used to try to make the eyes and bottom lip appear in the same location on each image. This is done to align images with different poses.



## 4.3 Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. Transfer learning is popular in deep learning given the enormous resources required to train deep learning models or the large and challenging datasets on which deep learning models are trained.

### Pre-trained Model Approach

- **Select Source Model.** A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.
- **Reuse Model.** The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
- **Tune Model.** Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

### Example of Transfer Learning with Image Data

It is common to perform transfer learning with predictive modeling problems that use image data as input.

This may be a prediction task that takes photographs or video data as input.

For these types of problems, it is common to use a deep learning model pre-trained for a large and challenging image classification task such as the ImageNet 1000-class photograph classification competition.

The research organizations that develop models for this competition and do well often release their final model under a permissive license for reuse. These models can take days or weeks to train on modern hardware. Three examples of models of this type include:

- Oxford VGG Model
- Google Inception Model
- Microsoft ResNet Model

### **How Pre-trained models are used**

We wish to identify the correct weights for the network by multiple forward and backward iterations. By using pre-trained models which have been previously trained on large datasets, we can directly use the weights and architecture obtained and apply the learning on our problem statement. This is known as transfer learning. We transfer the learning of the pre-trained model to our specific problem statement.

These pre-trained networks demonstrate a strong ability to generalize to images outside the ImageNet dataset via transfer learning. We make modifications in the pre-existing model by fine-tuning the model. Since we assume that the pre-trained network has been trained quite well, we would not want to modify the weights too soon and too much. While modifying we generally use a learning rate smaller than the one used for initially training the model.

## Ways to Fine tune the model

- **Feature extraction** We can use a pre-trained model as a feature extraction mechanism. What we can do is that we can remove the output layer( the one which gives the probabilities for being in each of the 1000 classes) and then use the entire network as a fixed feature extractor for the new data set.
- **Use the Architecture of the pre-trained model** What we can do is that we use architecture of the model while we initialize all the weights randomly and train the model according to our dataset again.
- **Train some layers while freeze others** Another way to use a pre-trained model is to train is partially. What we can do is we keep the weights of initial layers of the model frozen while we retrain only the higher layers. We can try and test as to how many layers to be frozen and how many to be trained.

## 4.4 Google's Inception v3 Model

Google Inception-v3 model is trained on Imagenet dataset, which consists of 1000 general objects. The pre-trained Inception-v3 model achieves state-of-the-art accuracy for recognizing general objects with 1000 classes, like Zebra, Dalmatian, and Dishwasher. The model extracts general features from input images in the first part and classifies them based on those features in the second part. The model without training from scratch, yields poor results on our dataset. But, the model can be used to extract feature vectors at the result of each layer. We used feature engineering on Google's Inception v3 model and input the same to the SVM classifier.

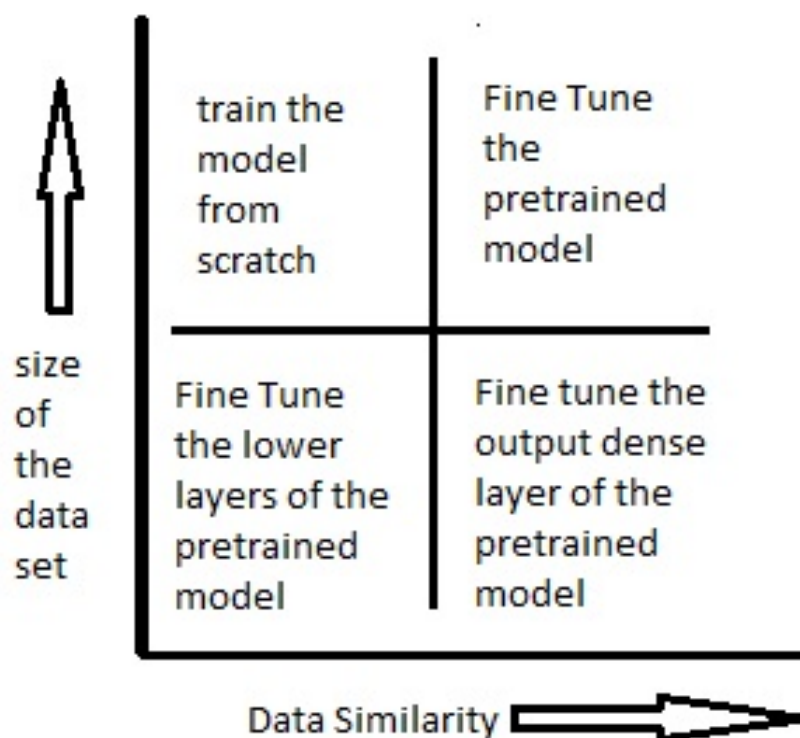


FIGURE 4.4: Fine Tuning

### 4.4.1 Inception Modules

The main idea of the Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components. Note that assuming translation invariance means that our network will be built from convolutional building blocks. All we need is to find the optimal local construction and to repeat it spatially. Arora et al. suggests a layer-by-layer construction in which one should analyze the correlation statistics of the last layer and cluster them into groups of units with high correlation. These clusters form the units of the next layer and are connected to the units in the previous layer. We

type	patch size/stride or remarks	input size
conv	$3 \times 3 / 2$	$299 \times 299 \times 3$
conv	$3 \times 3 / 1$	$149 \times 149 \times 32$
conv padded	$3 \times 3 / 1$	$147 \times 147 \times 32$
pool	$3 \times 3 / 2$	$147 \times 147 \times 64$
conv	$3 \times 3 / 1$	$73 \times 73 \times 64$
conv	$3 \times 3 / 2$	$71 \times 71 \times 80$
conv	$3 \times 3 / 1$	$35 \times 35 \times 192$
$3 \times$ Inception	As in figure 5	$35 \times 35 \times 288$
$5 \times$ Inception	As in figure 6	$17 \times 17 \times 768$
$2 \times$ Inception	As in figure 7	$8 \times 8 \times 1280$
pool	$8 \times 8$	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

FIGURE 4.5: Inceptionv3 Architecture

assume that each unit from the earlier layer corresponds to some region of the input image and these units are grouped into filter banks. In the lower layers (the ones close to the input) correlated units would concentrate in local regions. This means, we would end up with a lot of clusters concentrated in a single region and they can be covered by a layer of 11 convolutions in the next layer, as suggested in. However, one can also expect that there will be a smaller number of more spatially spread out clusters that can be covered by convolutions over larger patches, and there will be a decreasing number of patches over larger and larger regions. In order to avoid patch alignment issues, current incarnations of the Inception architecture are restricted to filter sizes 11, 33 and 55, however this decision was based more on convenience rather than necessity. It also means that the suggested architecture is a combination of all those layers with their output

filter banks concatenated into a single output vector forming the input of the next stage

## **4.4.2 Dimensionality Reduction**

One of the main beneficial aspects of this architecture is that it allows for increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity. The ubiquitous use of dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously. The improved use of computational resources allows for increasing both the width of each stage as well as the number of stages without getting into computational difficulties. Another way to utilize the inception architecture is to create slightly inferior, but computationally cheaper versions of it. We have found that all the included the knobs and levers allow for a controlled balancing of computational resources that can result in networks that are 2-3 faster than similarly performing networks with non-Inception architecture, however this requires careful manual design at this point.

## **4.4.3 Layers in Detail**

1. INPUT will hold the raw pixel values of the image, in this case an image of width , height , and with three color channels R,G,B.

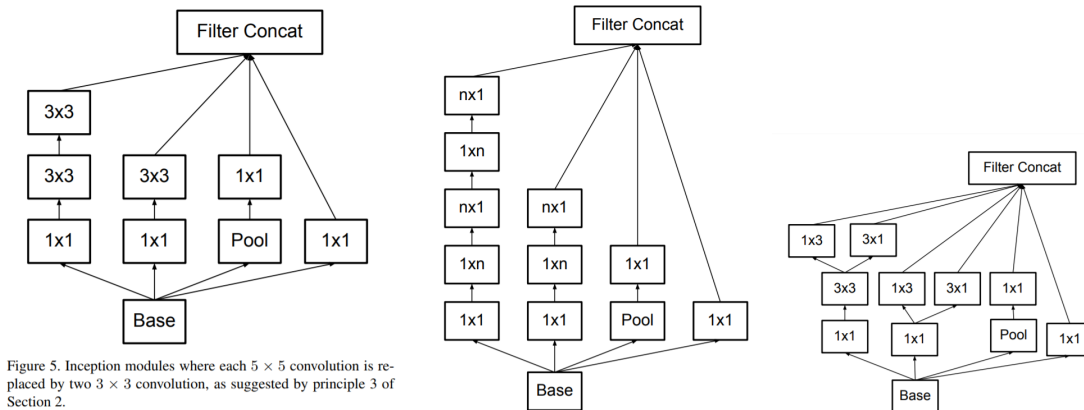


FIGURE 4.6: Inception Module

2. CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and 13 a small region they are connected to in the input volume. This may result in volume such as if we decided to use 32 filters.
3. RELU layer will apply an element-wise activation function, such as the  $\max(0, x)$  threshold at zero. This leaves the size of the volume unchanged.
4. POOL layer will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as  $[8 \times 8 \times 2048]$ .
5. FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

Among these layers, we have decided to take the vectors from the linear layer(logits layer). The vector size is  $[1 \times 1 \times 2048]$ .

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

FIGURE 4.7: FaceNet's nn4 Architecture

## 4.5 OpenFace Model

Openface model is trained on 500k images from Casia-webface and Face-scrub dataset. The dataset contains face images taken in unconstrained environment. The model extracts a 128 dimensional feature vector, which is then used for multi-class classification using sklearn's SVM. The OpenFace model uses a modified version of FaceNets[13] nn4 network. The neural network training and inference portions has been done using Torch, Lua and luajit. The neural network used is similar to Siamese networks which share weights

.

## 4.6 SVM Classifier

Support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for



classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). This SVM can be used for Multi-class Classification.

We have used libsvm (SVM) to classify the faces. The feature vectors extracted from the above method is fed into the svm classifier. The training is done using the training dataset. We used grid search for Multi-class Classification and found the  $c$  and  $\gamma$  value to make the classification accurate. We have scaled the input vectors and trained using the train command in libsvm. Final Prediction is done using the predict command in libsvm.

### **4.6.1 Multi-Class SVM**

In machine learning, multiclass or multinomial classification is the problem of classifying instances into one of three or more classes. While some classification algorithms naturally permit the use of more than two classes, others are by nature binary algorithms; these can, however, be turned into multinomial classifiers by a variety of strategies.

Multiclass SVM aims to assign labels to instances by using support vector machines, where the labels are drawn from a finite set of several elements. The dominant approach for doing so is to reduce the single multiclass problem into

multiple binary classification problems. Common methods for such reduction include:

- Building binary classifiers which distinguish (i) between one of the labels and the rest (one-versus-all) or (ii) between every pair of classes (one-versus-one). Classification of new instances for the one-versus-all case is done by a winner-takes-all strategy, in which the classifier with the highest output function assigns the class (it is important that the output functions be calibrated to produce comparable scores). For the one-versus-one approach, classification is done by a max-wins voting strategy, in which every classifier assigns the instance to one of the two classes, then the vote for the assigned class is increased by one vote, and finally the class with the most votes determines the instance classification.
- Directed acyclic graph SVM (DAGSVM)
- Error-correcting output codes

### 4.6.2 Categorical Feature

SVM requires that each data instance is represented as a vector of real numbers. Hence, if there are categorical attributes, we first have to convert them into numeric data. We recommend using  $m$  numbers to represent an  $m$ -category attribute. Only one of the  $m$  numbers is one, and others are zero. For example, a three-category attribute such as red, green, blue can be represented as  $(0,0,1)$ ,  $(0,1,0)$ , and  $(1,0,0)$ .

### 4.6.3 Scaling

Scaling before applying SVM is very important. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. We recommend linearly scaling each attribute to the range  $[-1, +1]$  or  $[0, 1]$ . We have to use the same method to scale both training and testing data. For example, suppose that we scaled the first attribute of training data from  $[10, +10]$  to  $[-1, +1]$ . If the first attribute of testing data lies in the range  $[11, +8]$ , we must scale the testing data to  $[-1.1, +0.8]$ .

### 4.6.4 RBF Kernel

In general, the RBF kernel is a reasonable first choice for our system as this kernel nonlinearly maps samples into a higher dimensional space so it, unlike the linear kernel, can handle the case when the relation between class labels and attributes is nonlinear. The second reason is the number of hyperparameters which influences the complexity of model selection. The polynomial kernel has more hyperparameters than the RBF kernel. Finally, the RBF kernel has fewer numerical difficulties. One key point is  $0 \leq K_{ij} \leq 1$  in contrast to polynomial kernels of which kernel values may go to infinity or zero. Moreover, it is noted that the sigmoid kernel is not valid under some parameters.

### 4.6.5 Cross-Validation and Grid-Search

There are two parameters for an RBF kernel:  $C$  and  $\gamma$ . It is not known beforehand which  $C$  and  $\gamma$  are best for a given problem; consequently some kind of model selection (parameter search) must be done. The goal is to identify good  $(C, \gamma)$  so that the classifier can accurately predict unknown data (i.e. testing data). Note that it may not be useful to achieve high training accuracy (i.e. a classifier which accurately predicts training data whose class labels are indeed known). As discussed above, a common strategy is to separate the data set into two parts, of which one is considered unknown. The prediction accuracy obtained from the unknown set more precisely reflects the performance on classifying an independent data set. An improved version of this procedure is known as cross-validation. In  $v$ -fold cross-validation, we first divide the training set into  $v$  subsets of equal size. Sequentially one subset is tested using the classifier trained on the remaining  $v - 1$  subsets. Thus, each instance of the whole training set is predicted once so the cross-validation accuracy is the percentage of data which are correctly classified. The cross-validation procedure can prevent the overfitting problem. We recommend a grid-search on  $C$  and  $\gamma$  using cross-validation. Various pairs of  $(C, \gamma)$  values are tried and the one with the best cross-validation accuracy is chosen.

## 4.7 CNN

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been

applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. We need three basic components to define a basic convolutional network.

- The convolutional layer
- The Pooling layer[optional]
- The output layer

### 4.7.1 Convolutional Layer

Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Suppose we have an image of size  $6 \times 6$ . We define a weight matrix which extracts certain features from the images

We have initialized the weight as a  $3 \times 3$  matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output. The value 429 above, is obtained by the adding the values obtained by element wise multiplication of the weight matrix and the highlighted  $3 \times 3$  part of the input image.

The  $6 \times 6$  image is now converted into a  $4 \times 4$  image. Think of weight matrix like a paint brush painting a wall. The brush first paints the wall horizontally and then comes down and paints the next row horizontally. Pixel values are used again when the weight matrix moves along the image. This basically enables parameter sharing in a convolutional neural network.

The weight matrix behaves like a filter in an image extracting particular information from the original image matrix. A weight combination might be extracting edges, while another one might a particular color, while another one might just blur the unwanted noise.

### **4.7.2 Pooling Layer**

Sometimes when the images are too large, we would need to reduce the number of trainable parameters. It is then desired to periodically introduce pooling layers between subsequent convolution layers. Pooling is done for the sole purpose of reducing the spatial size of the image. Pooling is done independently on each depth dimension, therefore the depth of the image remains unchanged. The most common form of pooling technique generally applied is the max pooling, which is what we have used in our CNN.

### **4.7.3 Output Layer**

After multiple layers of convolution and padding, we would need the output in the form of a class. The convolution and pooling layers would only be able to extract features and reduce the number of parameters from the original images. However, to generate the final output we need to apply a fully connected layer to generate an output equal to the number of classes we need. It becomes tough to reach that number with just the convolution layers. Convolution layers generate 3D activation maps while we just need the output as whether or not an image belongs to a particular class. The output layer has a loss function like categorical

cross-entropy, to compute the error in prediction. Once the forward pass is complete the backpropagation begins to update the weight and biases for error and loss reduction.

## CHAPTER 5

# PROBLEM DEFINITION AND PROPOSED SYSTEM

### 5.1 Problem Statement

1. To detect faces from the input images and align the faces by preprocessing.
2. To extract representation vectors by transfer learning technique and to classify faces using multi-class SVM classifier.
3. To learn and classify faces using CNN.
4. To compare the classification accuracies between the proposed methods.

### 5.2 Proposed System

We have proposed three different systems to perform facial recognition in both constrained and unconstrained environments as follows:

1. Classification based on transfer learning technique using Google's Inception v3 model.
2. Classification based on transfer learning technique using OpenFace model.
3. Classification using Convolutional Neural Network (CNN).



The proposed systems are based on transfer learning and deep neural networks. In **Transfer learning based systems**, we have used pre-trained models such as Google's Inception v3 and OpenFace to extract the representation vectors of the input images. The vectors are used to train a multi-class SVM classifier and build a model. Using the built model, the test images have been classified.

In the **Deep neural network based system**, multi layer CNN has been used to automatically learn and train the network to classify the input images. The overall architectural design of the proposed systems have been depicted in Figure5.1.

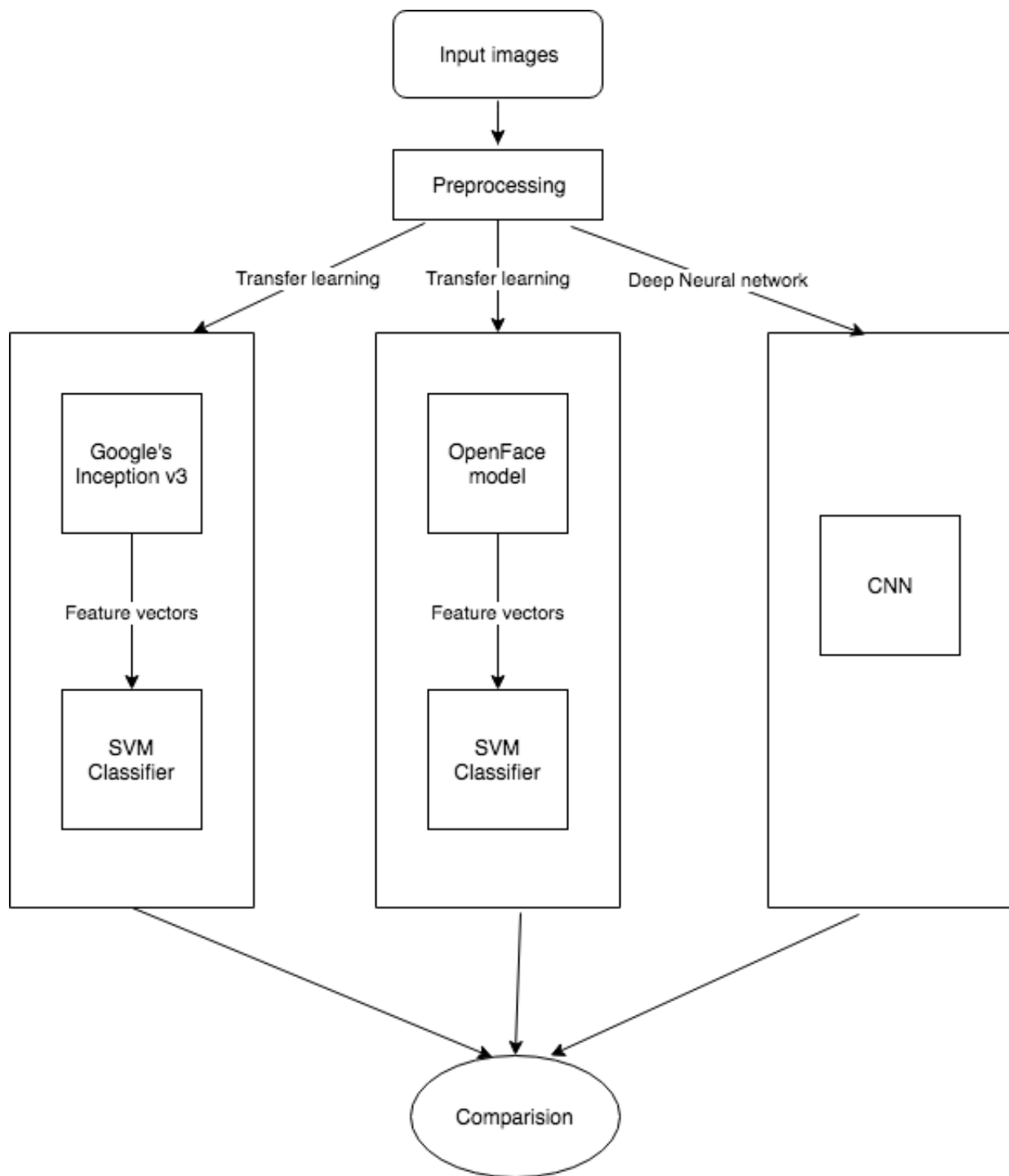


FIGURE 5.1: Architectural Diagram

### 5.2.1 Preprocessing

The input image obtained are preprocessed to obtain high accuracy. The aim of preprocessing is to locate the face in the input photograph or video frame, determine the facial landmarks and align the faces of different poses. We have used Dlib library [7] to detect the face and to determine the facial landmarks. The Dlib algorithm makes use of Haar classifier which is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones; which are trained from many positive images (with faces) and negatives images (without faces). The detection is based on Haar-like features. **A Haar-like feature** considers adjacent rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. Since this technique uses a cascade of classifiers, the features are grouped into different stages of classifiers and applied one-by-one. If a window fails in the first stage, the remaining groups of features are not considered.

After a face is detected, the pre-trained facial landmark detector inside the dlib library is used to estimate the location of 68 (x, y)-coordinates that map to facial structures on the face. The facial landmarks are obtained which helps us to detect important key facial structures in the face region. For the training set of images, the specific (x,y)-coordinates of the landmark regions and the probability of distance between the pair of input pixels are determined. After estimating the facial landmarks, Affine transformation is applied which normalizes the faces by making the eye corner and nose close to the mean location. This is done to align images with different poses. It resizes and crops the images to the edges of the

landmarks to a size of 96x96. The preprocessed images have been given as input to the three proposed systems.

### **5.2.2 Classification based on transfer learning technique using OpenFace model.**

The proposed system consists of the following steps

1. Feature Extraction from OpenFace model
2. Classification using a Multiclass SVM classifier

#### **5.2.2.1 Feature Extraction from OpenFace model**

Once a face is detected, the system preprocess each face in the image to create a normalized and fixed-size input to the neural network. The preprocessed images are too high-dimensional for a classifier to take directly on input. The neural network is used as a feature extractor to produce a low-dimensional representation that characterizes a persons face. A low-dimensional representation is key so it can be efficiently used in classifiers or clustering techniques.

The original nn4 neural network was used in FaceNet's architecture [13] which works based on the triplet loss. The OpenFaces end-to-end network training flow uses triplet loss, which consists of a parameter  $\theta$  that maps an image  $I$  onto a unit hypersphere of dimension  $m$ . A triplet consists of an anchor image  $a$ , a

positive image of the same person  $p$ , and a negative image of a different person  $n$ . The distance between the anchor and positive should be less than the distance between the anchor and negative. The OpenFace model uses the modified version of FaceNet's architecture, nn4.small2, which consists of only 3733968. The reduced number of parameters allows faster training of the model. Using the OpenFace model representation vectors are extracted from the preprocessed images. The feature vectors are extracted from the last layer of the model, which is a L2 normalization layer. The dimensions of the extracted features are  $1 \times 1 \times 128$  which is then classified using SVM classifier.

#### **5.2.2.2 Classification using a SVM classifier**

The representation vectors of the training images obtained from OpenFace model is fed into the multiclass SVM classifier and a model is built. After building the model the test images are classified using the model with the help of the representation vectors obtained from the OpenFace model.

### **5.2.3 Classification based on transfer learning technique using Google's Inception v3 model**

The proposed system consists of the following steps

1. Feature Extraction from Inception v3 model
2. Classification using a SVM classifier

### 5.2.3.1 Feature Extraction from Inception v3 model

Google Inception model consists of different layers, these layers help to extract multidimensional features such as low level to high level features from the images. The Google Inception V3 module consists of multiple convolution and pooling layers. The CONV layer consists of a set of learnable filters. Each filter is spatially small but extends through full depth of the input volume. The filters stride over the width and height of the input volume and produce an activation map. The activation map are the responses estimated by the filter at each spatial position. The convolution layer consists of three filters of size  $3 \times 3$ , four filters of size  $1 \times 1$  and of varying strides 1 and 2. The original inception module had filters of size  $5 \times 5$  and  $7 \times 7$ . These filters have been replaced by two consecutive  $3 \times 3$  filters stacked on top of each other which reduces the number of parameters to a great extent and increases efficiency. Each convolution filter convolves over its input and computes the weighted sum. The model also consists of different pooling layers of size  $3 \times 3$  and  $1 \times 1$  with stride 2. Pooling can be achieved using the filters and pooling strategies such as max pooling and average pooling. These pooling layers reduce the spatial size of the representation and control over fitting by reducing the amount of parameters and computation in the network. The training images of our dataset were fed into the Google inception model. The model extracts the representation vectors of our input images. The representation vectors have been extracted from the last but before layer, logits layer. The size of the vector extracted from the layer is  $[1 \times 1 \times 2048]$ . These features have been used by the SVM classifier to build a model to classify the faces. The same process is repeated for the test images to obtain the representation vectors.

### **5.2.3.2 Classification using a SVM classifier**

The representation vectors of the training images obtained from Google inception model is fed into the multiclass SVM classifier and a model is built. After building the model the test images are tested using the model with the help of the representation vectors obtained from the Inception V3 model.

### **5.2.4 Classification using CNN**

We have built a Convolution Neural Network (CNN) model using Keras and TensorFlow as the Backend. The preprocessed image have been resized to dimension 64x64. The input images have been augmented using the Image generator, by shearing, zooming, and flipping the images. The CNN consists of different layers such as convolution layers, pooling layers, flatten and softmax layer. The architecture of the CNN model consists of three convolution layers, with 64 filters. Each convolution layer is followed by a max pooling layer of stride 2. The activation function used is ReLu. The loss function is optimized using adam optimizer. A fully connected layer with 64 dense units is added in the end, followed by a softmax layer consisting of n number of nodes, where n equals the number of classes.

## CHAPTER 6

# EXPERIMENTS AND RESULTS

In this chapter, we have discussed about the dataset used and the experimental setup and the results obtained using the proposed system.

## 6.1 Dataset

In order to evaluate the performance of our proposed system, we have used three different sets of databases obtained from constrained and unconstrained environments.

- ORL database - constrained environment
- Labelled Faces in the Wild(LFW) dataset - unconstrained environment
- Extended Yale Faces database B - unconstrained environment

### 6.1.1 ORL Database

Database of Faces of ATT Laboratories Cambridge [12], (formerly 'The ORL Database of Faces'), contains a set of face images taken between April 1992 and April 1994 at the lab. The database was used in the context of a face recognition project carried out in collaboration with the Speech, Vision and Robotics Group of the Cambridge University Engineering Department. There are ten different



images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). The sample images of the database have been depicted in Figure 6.1. The files are in PGM format and the size of each image is 92x112 pixels, with 256 grey levels per pixel.



FIGURE 6.1: ORL Database

### 6.1.2 LFW Dataset

LFW [9] is a database of face photographs designed for studying the problem of unconstrained face recognition. The data set contains more than 13,000 images of faces collected from the web. We have used only a part of this dataset (23 classes) for our work. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set. Most of the classes in the database contain an average of 5 images which is insufficient



FIGURE 6.2: LFW Dataset

for training our model, hence only 23 classes have been used from the dataset. The sample images of the database have been depicted in Figure 6.2.

### 6.1.3 The Extended Yale Face Database B

The extended Yale Face Database B [5] contains 16128 images of 28 human subjects under 9 poses and 64 illumination conditions. The data format of this database is the same as the Yale Face Database B. All test image data used in the experiments are manually aligned, cropped, and then re-sized to 168x192 images. The images are in jpg format. Each class contains an average of 500 images. The sample images of the database have been depicted in Figure 6.3.



FIGURE 6.3: Extended Yale Database B

The overall information about the database and the corresponding information such as No. of classes and No of images are shown in Table 6.1.

TABLE 6.1: Datasets

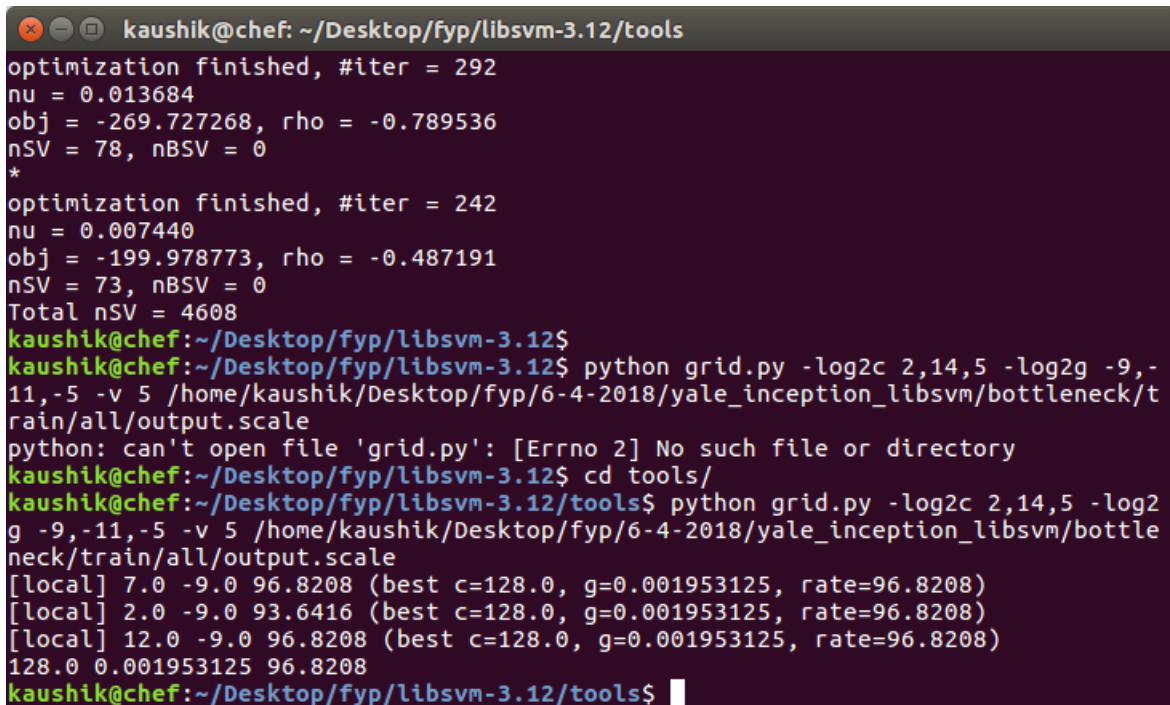
Dataset	No. of classes	No. of images
Extended Yale Faces	28	16128
ORL Database	40	400
LFW dataset	23	1955

## 6.2 Performance Analysis

Accuracy is a statistical measure which specifies how well a classification mechanism correctly predicts the samples. The system has been analyzed using different metric measures such as accuracy and validation loss.

## 6.2.1 Results - Classification Using Google Inception Model

The input images in the dataset are divided into training and test set in the ratio of 70:30. The representation vectors from the training images are obtained through Googles Inception model. We have performed grid search using LIBSVM tool [4] in order to tune the hyper parameters such as  $C$  and  $\gamma$  to obtain higher accuracy. We also validated our training set by performing 10-fold cross validation. We have obtained the  $C$  and  $\gamma$  value as 128.0 and 0.001953125. The result of the grid search and the cross validation accuracy have been depicted in Figure 6.4. Using the fine tuned parameters a multiclass model is built and the test images are tested using the model.



```

kaushik@chef: ~/Desktop/fyp/libsvm-3.12/tools
optimization finished, #iter = 292
nu = 0.013684
obj = -269.727268, rho = -0.789536
nSV = 78, nBSV = 0
*
optimization finished, #iter = 242
nu = 0.007440
obj = -199.978773, rho = -0.487191
nSV = 73, nBSV = 0
Total nSV = 4608
kaushik@chef:~/Desktop/fyp/libsvm-3.12$
kaushik@chef:~/Desktop/fyp/libsvm-3.12$ python grid.py -log2c 2,14,5 -log2g -9,-
11,-5 -v 5 /home/kaushik/Desktop/fyp/6-4-2018/yale_inception_libsvm/bottleneck/t
rain/all/output.scale
python: can't open file 'grid.py': [Errno 2] No such file or directory
kaushik@chef:~/Desktop/fyp/libsvm-3.12$ cd tools/
kaushik@chef:~/Desktop/fyp/libsvm-3.12/tools$ python grid.py -log2c 2,14,5 -log2
g -9,-11,-5 -v 5 /home/kaushik/Desktop/fyp/6-4-2018/yale_inception_libsvm/bottle
neck/train/all/output.scale
[local] 7.0 -9.0 96.8208 (best c=128.0, g=0.001953125, rate=96.8208)
[local] 2.0 -9.0 93.6416 (best c=128.0, g=0.001953125, rate=96.8208)
[local] 12.0 -9.0 96.8208 (best c=128.0, g=0.001953125, rate=96.8208)
128.0 0.001953125 96.8208
kaushik@chef:~/Desktop/fyp/libsvm-3.12/tools$

```

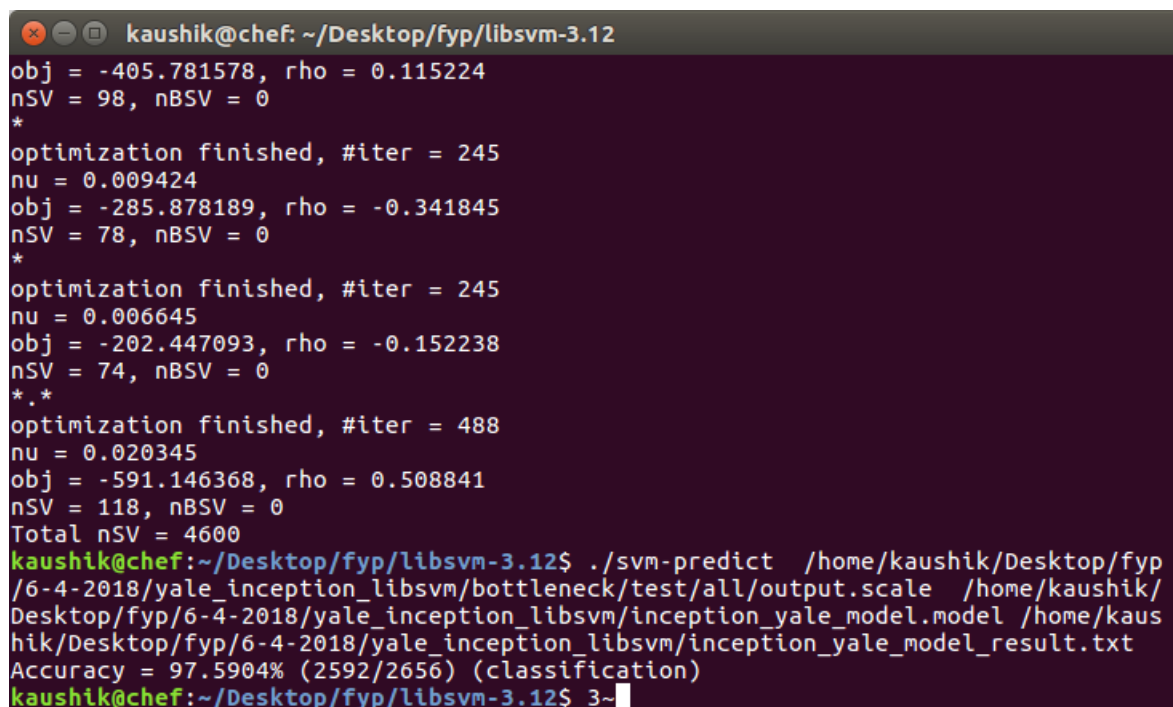
FIGURE 6.4: Hyperparameters Tuning using Grid-Search

The training and testing accuracies for each dataset have been listed in Table 6.2.

Model	Dataset	Training Accuracy	Testing Accuracy
Google Inception model	Extended Yale Faces	97.62%	97.54%
	ORL Database	100%	100%
	LFW dataset	85%	82%

TABLE 6.2: Accuracy using Google's Inception v3 Model

The Yale dataset consists of enough number of training and testing images and it has been taken in an unconstrained environment. We have discussed the results of the database using confusion matrix. The confusion matrix describes the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It tells the predicted class index for each image. The testing accuracy and the confusion matrix of the Extended Yale Database has been depicted in Figures 6.5, 6.3.



```

kaushik@chef: ~/Desktop/fyp/libsvm-3.12
obj = -405.781578, rho = 0.115224
nSV = 98, nBSV = 0
*
optimization finished, #iter = 245
nu = 0.009424
obj = -285.878189, rho = -0.341845
nSV = 78, nBSV = 0
*
optimization finished, #iter = 245
nu = 0.006645
obj = -202.447093, rho = -0.152238
nSV = 74, nBSV = 0
*,*
optimization finished, #iter = 488
nu = 0.020345
obj = -591.146368, rho = 0.508841
nSV = 118, nBSV = 0
Total nSV = 4600
kaushik@chef:~/Desktop/fyp/libsvm-3.12$ ./svm-predict /home/kaushik/Desktop/fyp/6-4-2018/yale_inception_libsvm/bottleneck/test/all/output.scale /home/kaushik/Desktop/fyp/6-4-2018/yale_inception_libsvm/inception_yale_model.model /home/kaushik/Desktop/fyp/6-4-2018/yale_inception_libsvm/inception_yale_model_result.txt
Accuracy = 97.5904% (2592/2656) (classification)
kaushik@chef:~/Desktop/fyp/libsvm-3.12$ 3~

```

FIGURE 6.5: Result using Google Inception Model

TABLE 6.3: Confusion Matrix of Extended Yale B Database using Inceptionv3

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	105	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
2	0	94	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	104	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	104	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
7	0	0	0	0	0	0	95	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	102	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	1	1	0	0	0	0	115	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	1	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	99	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	1	0	0	0	1	0	100	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	93	0	0	0	0	0	0	0	0	2	0	0	0	1	0
15	0	1	0	0	0	0	0	0	0	0	2	1	0	0	93	0	0	0	2	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	96	0	0	0	0	0	0	1	0	0	1	0	0
17	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	83	0	0	0	0	1	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	99	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86	0	1	1	0	0	0	0	0	0
20	0	2	1	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	91	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	87	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	96	1	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	86	0	0	0	0	0	0
24	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	80	0	0	0	0	0
25	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	2	1	0	93	0	0	0
27	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	2	0	0	0	96	0	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	0	0	0	0	32	0

We also proposed another system using transfer learning technique with the help of Openface model. Using the obtained features a multiclass SVM is trained to classify the images of particular class by building a model. With the help of testing images the model is tested. The results of the Openface model is listed in Table 6.4. The confusion matrix of the Extended Yale Faces Dataset B has been listed in Table 6.5.

Model	Dataset	Training Accuracy	Testing Accuracy
OpenFace model	Extended Yale Faces	95%	94%
	ORL Database	100%	100%
	LFW dataset	99%	99%

TABLE 6.4: Accuracy using OpenFace Model

TABLE 6.5: Confusion Matrix of Extended Yale B Database using OpenFace

Class	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	103	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	2	0	0	1	0
2	0	88	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0
3	0	0	80	0	0	0	0	0	0	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	101	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	85	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	102	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	91	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	99	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	117	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	95	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	90	3	2	0	1	0	0	0	0	0	0	0	0	3	0	0	0	0
12	0	1	0	0	0	0	0	0	0	0	2	89	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0
13	2	0	0	0	0	0	0	0	0	0	3	1	94	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0
14	3	0	0	0	0	0	0	0	0	0	1	0	0	88	0	0	1	0	0	0	0	0	0	2	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1	1	0	0	0	91	0	3	0	1	0	0	0	0	0	0	1	0	0
16	2	0	0	0	0	0	0	0	0	0	0	1	0	0	1	95	0	0	0	1	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	83	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0	0	0	0
19	0	1	0	0	0	0	0	0	0	0	2	1	0	1	2	0	2	0	76	0	1	0	0	0	1	0	0	0
20	0	1	1	0	0	0	1	0	0	0	1	0	0	0	0	0	3	0	0	86	1	0	0	0	0	2	0	0
21	0	0	1	0	0	0	0	0	0	0	2	1	0	1	0	0	1	0	1	0	78	0	1	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	3	0	1	0	81	0	0	0	0	0
24	0	0	0	0	1	0	0	1	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	79	0	0	0	0
25	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	94	0	0	0
26	0	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	92	0	0
27	3	0	0	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	91	0
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	33

The testing accuracy of the Extended Yale Dataset B using OpenFace model has been depicted in Figure 6.6.

```

kaushik@chef: ~/Desktop/fyp/libsvm-3.12
obj = -382.687090, rho = -0.185089
nSV = 11, nBSV = 2
*
optimization finished, #iter = 67
nu = 0.051224
obj = -1914.895024, rho = -0.572020
nSV = 29, nBSV = 17
*
optimization finished, #iter = 97
nu = 0.061182
obj = -1494.149489, rho = -0.863873
nSV = 25, nBSV = 11
*
optimization finished, #iter = 57
nu = 0.013879
obj = -273.544336, rho = -0.074386
nSV = 11, nBSV = 0
Total nSV = 2165
kaushik@chef:~/Desktop/fyp/libsvm-3.12$ ./svm-predict /home/kaushik/Desktop/fyp/6-4-2018/yale_openface_libsvm/generated-embeddings/test/output.scale /home/kaushik/Desktop/fyp/6-4-2018/yale_openface_libsvm/openface_yale_model.model /home/kaushik/Desktop/fyp/6-4-2018/yale_openface_libsvm/openface_yale_model_result.txt
Accuracy = 94.903% (2495/2629) (classification)
kaushik@chef:~/Desktop/fyp/libsvm-3.12$

```

FIGURE 6.6: Result using OpenFace Model

We have used two different transfer learning techniques in which the weights from the pretrained models were used as extracted features and helps in classifying using SVM classifier. The Inceptionv3 model performs better than OpenFace model. Although, OpenFace model has been trained on face images, the inception model has been trained on billions of general object images which is tuned to handle various illumination conditions.

## 6.2.2 Results - Deep Neural Network

We have built a CNN model with three layers consisting of three convolution layers, three max pooling layers and one fully connected layer. In this experiment we have divided the dataset into 6:3:1 ratio as training, validation and test set. We have obtained the following accuracies. The training and testing are depicted in Figures 6.6. Loss function calculates the error difference between the actual and the predicted value. Accuracy represents the number of images correctly predicted by the model from the testing set. The accuracy and the loss function have been plotted in Figures 6.8, 6.9

Model	Dataset	Training Accuracy	Testing Accuracy
CNN	Extended Yale Faces	97%	95%
	ORL Database	100%	100%
	LFW dataset	94%	89%

TABLE 6.6: CNN



```

kaushik@chef: ~/Desktop/fyp
6979/7000 [=====>.] - ETA: 1s - loss: 0.1600 - acc: 0.949
6980/7000 [=====>.] - ETA: 1s - loss: 0.1600 - acc: 0.949
6981/7000 [=====>.] - ETA: 1s - loss: 0.1600 - acc: 0.949
6982/7000 [=====>.] - ETA: 1s - loss: 0.1600 - acc: 0.949
6983/7000 [=====>.] - ETA: 1s - loss: 0.1600 - acc: 0.949
6984/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
6985/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6986/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6987/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6988/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6989/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6990/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6991/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6992/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6993/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
6994/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
6995/7000 [=====>.] - ETA: 0s - loss: 0.1600 - acc: 0.949
6996/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
6997/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
6998/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
6999/7000 [=====>.] - ETA: 0s - loss: 0.1599 - acc: 0.949
7000/7000 [=====] - 430s 61ms/step - loss: 0.1600 - acc
: 0.9492
Accuracy = 0.9559487951807228

```

FIGURE 6.7: Result using CNN

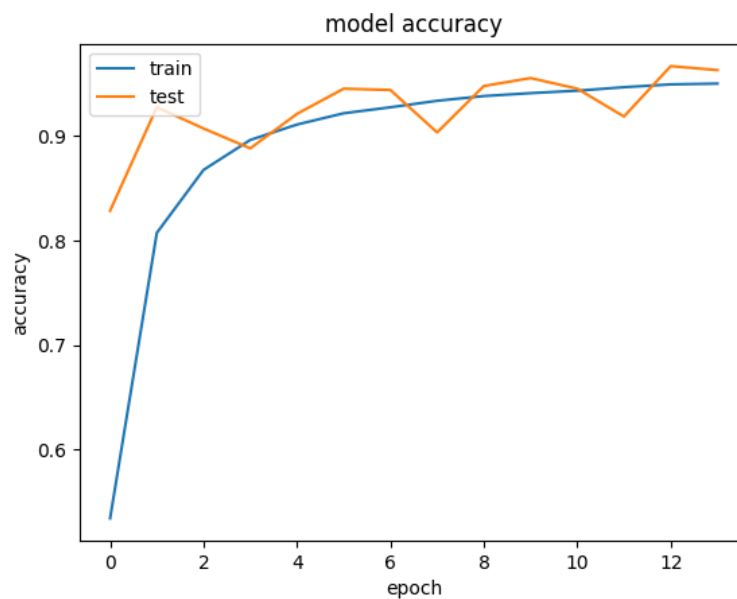


FIGURE 6.8: CNN Model Accuracy Graph

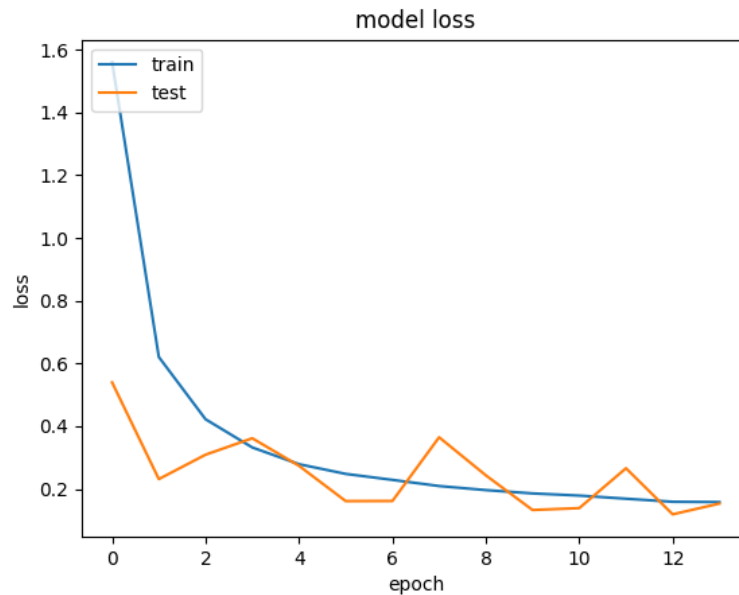


FIGURE 6.9: CNN Model Loss Graph

The CNN model extracts low level and high level features. The model has a very simple architecture and performs comparably well on the Extended Yale Faces Dataset B since the network has been trained from scratch using the input images taken from unconstrained environment.

### 6.2.3 Comparison of classification from all three methods

Face recognition systems must have a high accuracy as it is used in sensitive applications like security and video surveillance systems. The transfer learning technique which uses Inceptionv3 model for feature extraction gives better accuracy than the other two models.

## CHAPTER 7

# CONCLUSION AND FUTURE WORK

The face detection algorithm was run using the HAAR cascade classifier for the face. Once that is done, we limit our ROI and thereby extract the Facial Landmark Features. The cropped images were preprocessed by Reshaping, Denoising, Dlib's real time face pose estimation and Affine transformation techniques.

We have extracted feature vectors from Google's Inception v3 model and from the OpenFace model by passing the preprocessed images of the Yale dataset. The extracted features were classified using LibSVM classifier. The other method of CNN classifier was also implemented successfully.

From our research, we conclude that since the OpenFace pre-trained model, though trained on Face images, gives a comparatively lesser accuracy of 94% on the Extended Yale Face database (unconstrained dataset). The Google Inception model, though built for general objects, has been developed in a constrained environment with illumination corrections. Thus it produces a better accuracy of 97% for classification of the same dataset. Our CNN performs relatively well with an accuracy of 95%.

Ensembling is a technique of combining two or more algorithms of similar or dissimilar types called base learners. This is done to make a more robust system which incorporates the predictions from all the base learners. As part of our future work we would ensemble of the feature vectors obtained from the two different transfer learning techniques used. This helps in achieving better accuracy by complementing the prediction of classes by the two methods.

## Appendix A

# INSTALLATION

## A.1 Installing Python and Setting up Environment

Target Environment consists of following configurations:

- UBUNTU 16.04
- Python 3 and dependencies
- Anaconda
- Keras
- TensorFlow

Setting up the environment

1. Ubuntu 16.04 version ships with both Python 3 and Python 2 pre-installed. To make sure that the versions are up-to-date, update and upgrade the system with apt-get:

```
$sudo apt-get update
```

```
$sudo apt-get -y upgrade
```

2. Check version of Python 3 with the following command:

```
$ python3 -V
```

3. To manage software packages for Python, install pip:

```
$ sudo apt-get install -y python3-pip
```

4. Install the following dependencies :SciPy, NumPy, Matplotlib, Pandas, Statsmodels, and Scikit-learn as follows:

```
$ pip3 install scipy
```

```
$ pip3 install numpy
```

```
$ pip3 install matplotlib
```

```
$ pip3 install pandas
```

```
$ pip3 install statsmodels
```

```
$ pip3 install scikit-learn
```

5. Install anaconda as follows:

- (a) Use curl to download the link to anaconda for python 3 from the official anaconda downloads page

- (b) Run the downloaded script to install Anaconda using the below command. While Running the script give the necessary location for the installation location and when prompted prepend Anaconda3 install location to PATH in your /home/sammy/.bashrc

```
$ bash Anaconda3-5.0.1-Linux-x86_64.sh
```

- (c) Activate the installation using

```
$ source ~/.bashrc
```

(d) Confirm conda is installed correctly, by typing:

```
$conda -V
```

(e) Confirm your conda environment is up-to-date,by typing:

```
$ conda update conda
```

```
$ conda update anaconda
```

## 6. Update sciki-learn library

```
$conda update scikit-learn
```

## 7. Install Tensorflow by

```
$ conda install -c conda-forge tensorflow
```

## 8. Install Keras by typing:

```
$pip install keras
```

## 9. Check if Keras and TensorFlow as installed properly by saving the following code as deep\_versions.py and running the same on the terminal

```
# tensorflow
import tensorflow
print ('tensorflow: %s' % tensorflow.__version__)

# keras
import keras
print ('keras: %s' % keras.__version__)
```

The output of the above code should be as follows: tensorflow: 0.12.1 Using TensorFlow backend. keras: 1.2.1

## Bibliography

- [1] T. Ahonen, A. Hadid, and M. Pietikainen, “Face description with local binary patterns: Application to face recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 12, pp. 2037–2041, Dec. 2006, ISSN: 0162-8828.
- [2] V. Bruce and A. Young, “Understanding face recognition,” British Journal of Psychology, vol. 77, no. 3, pp. 305–327, DOI: 10.1111/j.2044-8295.1986.tb02199.x.
- [3] R. Brunelli and T. Poggio, “Face recognition: Features versus templates,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 15, no. 10, pp. 1042–1052, Oct. 1993, ISSN: 0162-8828. DOI: 10.1109/34.254061.
- [4] C.-C. Chang and C.-J. Lin, “Libsvm: A library for support vector machines,” ACM transactions on intelligent systems and technology (TIST), vol. 2, no. 3, p. 27, 2011.
- [5] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman, “From few to many: Illumination cone models for face recognition under variable lighting and pose,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 6, pp. 643–660, Jun. 2001, ISSN: 0162-8828. DOI: 10.1109/34.927464.
- [6] X. He, S. Yan, Y. Hu, P. Niyogi, and H.-J. Zhang, “Face recognition using laplacianfaces,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 3, pp. 328–340, Mar. 2005, ISSN: 0162-8828.

- [7] D. E. King, “Dlib-ml: A machine learning toolkit,” Journal of Machine Learning Research, vol. 10, no. Jul, pp. 1755–1758, 2009.
- [8] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, “Face recognition: A convolutional neural-network approach,” IEEE Transactions on Neural Networks, vol. 8, no. 1, pp. 98–113, Jan. 1997, ISSN: 1045-9227. DOI: 10.1109/72.554195.
- [9] E. Learned-Miller, G. B. Huang, A. RoyChowdhury, H. Li, and G. Hua, “Labeled faces in the wild: A survey,” in Advances in face detection and facial image analysis, Springer, 2016, pp. 189–248.
- [10] O. M. Parkhi, A. Vedaldi, A. Zisserman, et al., “Deep face recognition,” in BMVC, vol. 1, 2015, p. 6.
- [11] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, “Overview of the face recognition grand challenge,” in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 1, Jun. 2005, 947–954 vol. 1.
- [12] F. S. Samaria and A. C. Harter, “Parameterisation of a stochastic model for human face identification,” in Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on IEEE, 1994, pp. 138–142.
- [13] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in CVPR, IEEE Computer Society, 2015, pp. 815–823.
- [14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in



Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.

- [15] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, “Robust face recognition via sparse representation,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 2, pp. 210–227, Feb. 2009, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.79.