ABSTRACT

Facial Recognition can be used to authenticate a person passively and can be also used for mass identification. This quality is extremely useful in Home Automation systems where members of a household can be authenticated in a seamless manner which produces a good balance between security and ease of use.

Traditional methods for Facial Recognition include normalizing a set of facial images and compressing it to data which is useful for face recognition. Then a test image will be compared with the compressed face data for matches. This method does not provide a good performance since it fails to identify a familiar face under various lighting conditions.

With the advancement of Convolutional Neural Network(CNN), it is possible to train it to identify and classify faces much more accurately since it identifies the crucial features in the convolution layer and thus can identify a familiar face under various lighting conditions. Thus, to solve the above problem, the project aims to train a CNN for a particular household to recognize all the members even in various lighting conditions.

TABLE OF CONTENTS

ABSTRACT **i**

LIST OF FIGURES **iv**

LIST OF TABLES **vi**

LIST OF ABBREVIATIONS **vii**

1 INTRODUCTION **1**
- 1.1 FACE RECOGNITION 1
- 1.2 TYPES OF FACE RECOGNITION 1
  - 1.2.1 EigenFace 1
  - 1.2.2 FisherFace 2
- 1.3 NEURAL NETWORKS 3
- 1.4 CONVOLUTIONAL NEURAL NETWORKS 4
- 1.5 PROJECT DESCRIPTION 4

2 LITERATURE SURVEY **6**

3 PROBLEM DEFINITION AND METHODOLOGIES **7**
- 3.1 PROBLEM DEFINITION 7
- 3.2 EXISTING SYSTEM 7
- 3.3 PROPOSED SYSTEM 7
- 3.4 FUNCTIONALITY 8

4 SYSTEM DESIGN **9**
- 4.1 SYSTEM REQUIREMENTS 9
  - 4.1.1 HARDWARE REQUIREMENTS 9
  - 4.1.2 SOFTWARE REQUIREMENTS 10
- 4.2 SOFTWARE DESCRIPTION 10
  - 4.2.1 Python 10
  - 4.2.2 NumPy 10
  - 4.2.3 OpenCV 11
  - 4.2.4 Matplotlib 11

4.2.5 Compute Unified Device Architecture (CUDA) 11
4.2.6 CUDA Deep Neural Network (cuDNN) 12
4.2.7 TensorFlow 12
4.2.8 TensorBoard 13
4.2.9 Jupyter-Notebook 13
4.3 ARCHITECTURE DIAGRAM 14
4.4 FLOW DIAGRAM 15
4.5 STATE DIAGRAM 16

5 IMPLEMENTATION 17
5.1 DATA PREPROCESSING 17
5.2 TRAINING 17
5.2.1 Loading the Dataset 17
5.2.2 Defining the Hyper Parameters 18
5.2.3 Creating a Convolution Layer 18
5.2.4 Creating a Max Pooling Layer 19
5.2.5 Flattening the Convolution Layer 20
5.2.6 Creating a Fully Connected Layer 20
5.2.7 Defining Loss Function for Back Propogation 20
5.2.8 Instantiating the Tensorflow Training 21
5.2.9 Saving and Restoring the Model 21
5.3 VISUALIZING 21
5.3.1 Plotting the Convolution Filters 21
5.3.2 Plotting the Convolution Images 21
5.3.3 Visualizing Accuracy and Loss in TensorBoard 22
5.4 TESTING AND WRAPPER 22

6 RESULT 23
6.1 PREPROCESSING 23
6.2 TRAINING 23
6.2.1 Convolution and Max Pooling Layers 23
6.2.2 Loss and Accuracy 24
6.3 VISUALIZING 27
6.3.1 Convolution Filters 27
6.3.2 Convoluted Images 30
6.4 TESTING 34
6.4.1 Testing by Downloading Images from Internet 34
6.4.2 Testing by Web Camera 35

7 CONCLUSION AND FUTURE WORK 36

REFERENCES 37

APPENDIX 1 38

TECHNICAL BIOGRAPHY

LIST OF FIGURES

1.1 Eigenfaces 2
1.2 FisherFaces 3
1.3 Neural Networks 3
1.4 Convolutional Neural Networks 4

4.1 Architecture of CNN 14
4.2 Flow Diagram of the Face Recognition System 15
4.3 State Diagram of the Face Recognition System 16

6.1 Results after Preprocessing 23
6.2 Convolution, Max Pooling and ReLU in TensorBoard 24
6.3 Training Loss and Accuracy after 100,000 iterations in TensorBoard 25
6.4 Training Loss and Accuracy after 150,000 iterations in TensorBoard 26
6.5 Convolution Layer 1 Filters 27
6.6 Convolution Layer 2 Filters 28
6.7 Convolution Layer 3 Filters 29
6.8 Image of U.S President Barack Obama before Convolution 30
6.9 Convoluted Image of U.S President Barack Obama in Layer 1 31
6.10 Convoluted Image of U.S President Barack Obama in Layer 2 32
6.11 Convoluted Image of U.S President Barack Obama in Layer 3 33
6.12 Face Recognition of Popular Celebrities. 34
6.13 Testing with Web Camera Feed 35

LIST OF TABLES

5.1 Hyperparameters for the Face Recognition Model. 19

LIST OF ABBREVIATIONS

IoT	Internet of Things, page 1
CNN	Convolutional Neural Network, page 4
CUDA	Compute Unified Device Architecture, page 10
cuDNN	CUDA Deep Neural Network, page 10
t-SNE	t-distributed stochastic neighbor embedding, page 13
PCA	Principal Component Analysis, page 13
IDE	Integrated Development Environment, page 13
ReLU	Rectified Linear Unit, page 19
LSTM	Long short-term Memory, page 36
MANN	Memory Augmented Neural Network, page 36

Chapter 1
INTRODUCTION

1.1 FACE RECOGNITION

Face recognition is a system which can identify or recognize a person from a photo or a video automatically. It is a requirement which grows day by day in fields like IoT and Home Automation. Specifically, tagging a person via Facial Recognition can be very useful to identify context in a Home Automation System. This context can be used for customizable settings, content and tasks for each person in a household. It is also needed in systems such as authentication systems, identification systems, and automatic photo tagging in social media.

Since it is a necessity as mentioned above, there has been a lot of research to develop Facial Recognition as a separate domain. However, the focus of the project is to tag a person through Face Recognition and give context to devices in a Home Automation System.

1.2 TYPES OF FACE RECOGNITION

Face Recognition is usually a task which has many stages such as facial detection, feature extraction, and classification. The main stage is feature extraction where one develops mathematical models to feed features to the learning algorithm. There are many methods for extracting features which are described below.

1.2.1 EigenFace

In this feature extraction method, Principal Component Analysis (PCA) is used to reduce M images of one person which has N x N dimensions into a single NxN image. Then the image is used as a feature for the classification algorithm.

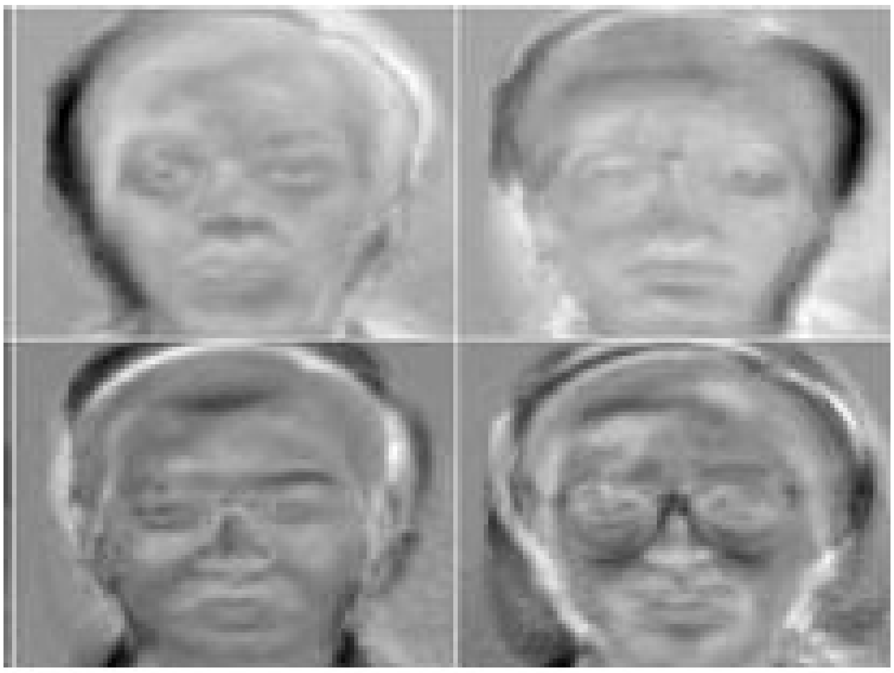


Figure 1.1: Eigenfaces

The advantage of this method is it extracts features automatically from raw image and it does it very fast. The disadvantage of this method is that it performs poorly with various lighting conditions and poses since the feature values oscillates on PCA. Figure 1.1 shows an example of the Eigenface feature extraction technique.

1.2.2 FisherFace

Fisherface reduces the dimension of the data and also preserves the difference between the classes by using PCA and fisher's linear discriminator on the dataset. Then, the Fisher vector that was generated by FisherFace is given to the classification algorithm. Figure 1.2 shows an example of three images where the features were extracted by fisher's linear discriminator.

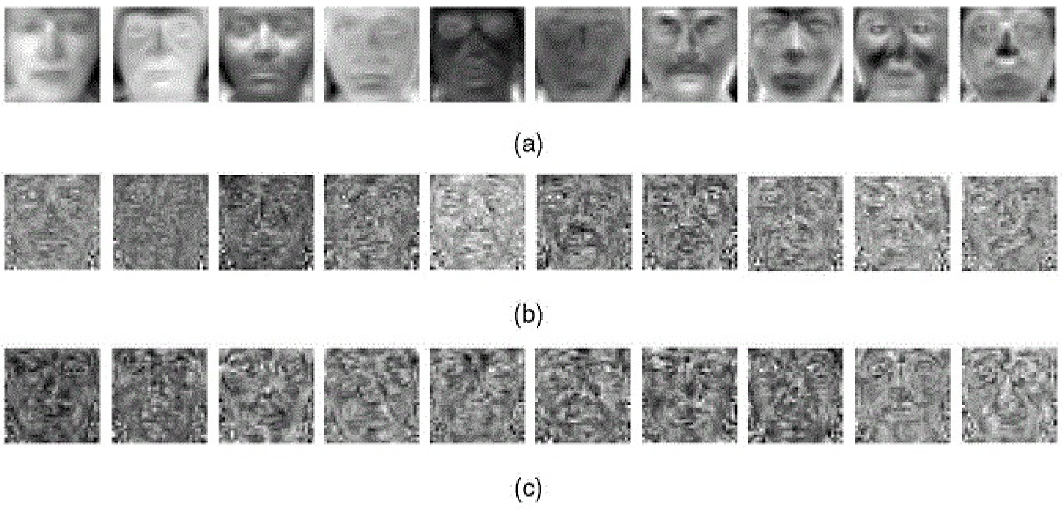


Figure 1.2: FisherFaces

The advantage of this method is it works under various facial expressions, lighting conditions and poses. The disadvantage of this method is that its performance is highly dependent on classification algorithm and the dimensionality reduction also affects the FisherFace technique.

1.3 NEURAL NETWORKS

Neural Networks are a group of neurons which are connected by weights. This algorithm is trained with dataset by optimizing the loss function. The loss function is computed from the true values and the predicted values. This process of calculating loss and adjusting weights is called back propagation. It uses many layers of neurons to learn more complex function.

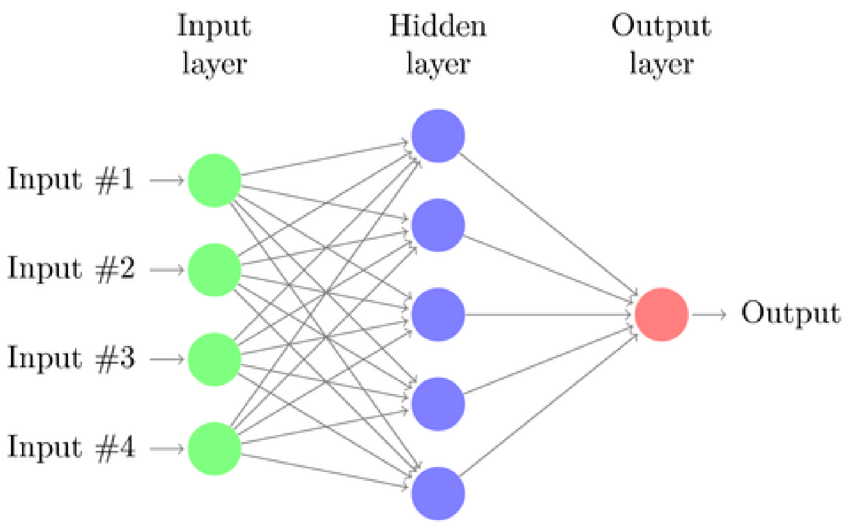


Figure 1.3: Neural Networks

The performance of neural network depends on the dataset. The behavior of neural network is a black box. The performance of the neural network is proportional to the size and variety of the dataset. Also, increasing the layers increases the complexity of the model. This can also lead to over fitting the dataset thus, it is very important to select the appropriate number of layers for a model.

1.4 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural network is also a type of neural network which contains learnable filters. The filters are in the form of kernel which convolves with the input layer by computing dot product. The results of the dot product contain deep information about the input.

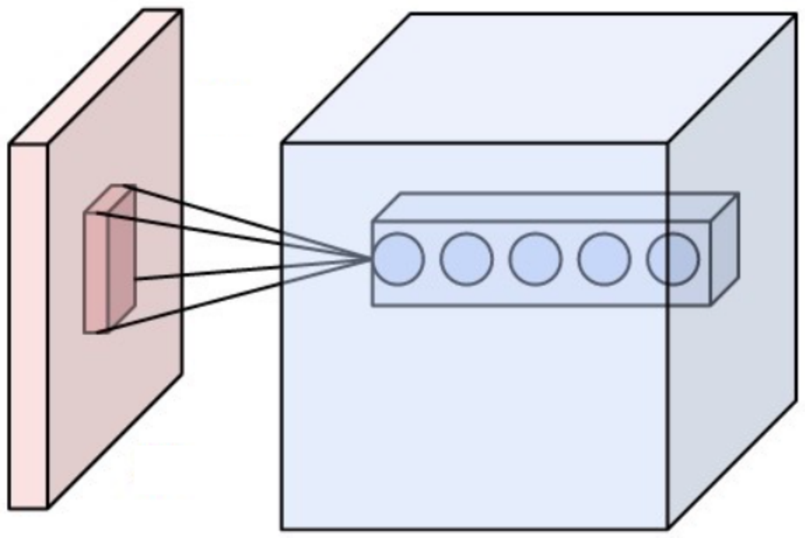


Figure 1.4: Convolutional Neural Networks

This is flattened and given as input to the neural network. From this idea exploring the depth of input leads to better feature extraction method. The advantage of this algorithm is that the filters are trained without any supervision and it leads to better feature extraction.

1.5 PROJECT DESCRIPTION

The project aims to apply Convolutional Neural Networks (CNN) to Facial Recognition which will make it easy to adapt to dynamic environments by extracting crucial features and performing classification by a single algorithm. The CNN will be trained and tested using 100x100 images of 1000 different faces which are taken in various environments such as lighting, facial expression etc. Then the performance of the CNN will be measured and the best performing CNN architecture will be selected for deployment. Then it will be deployed in an embedded device such as Raspberry Pi with a camera to see if it can perform Facial Recognition for IoT and Home Automation Systems.

Chapter 2
LITERATURE SURVEY

The four main stages of the traditional face recognition process are face detection, face alignment, feature extraction and classification [1]. The main stage out of these four is feature extraction [1]. In constrained environments, hand-crafted features such as Local Binary Patterns and Local Phase Quantisation (LPQ) have achieved respectable face recognition performance [1].

However, the performance using these features degrades dramatically in unconstrained environments where face images cover complex and large intra-personal variations such as pose, illumination, expression and occlusion [1]. It remains an open problem to find an ideal facial feature which is robust for face recognition in unconstrained environments (FRUE) [1]. In the last three years, convolutional neural networks (CNN) rebranded as deep learning have achieved very impressive results on FRUE. Unlike the traditional hand-crafted features, the CNN learned features are more robust to complex intra-personal variations [1].

Moreover, since the feature extraction stage and the recognition stage are separate in traditional methods, they cannot be jointly optimized [2]. Once useful information is lost in feature extraction, it cannot be recovered in recognition [2]. Also, without the help of classification, the best way to design feature descriptors to capture identity information is not clear [2]. For example, fisher vector encodes a high set of features into high dimensional vector representation which cannot be optimized without dimensionality reduction [3].

In a convolutional hybrid model, the feature extraction and recognition stages are unified under a single network architecture [2]. The parameters of the entire pipeline (weights and biases in all the layers) are jointly optimized for the target of face recognition [2].

Chapter 3
PROBLEM DEFINITION AND METHODOLOGIES

3.1 PROBLEM DEFINITION

To create a face recognition system which can be deployed in a home automation system which can be retrained and customized for multiple households.

3.2 EXISTING SYSTEM

Existing face recognition systems which are deployed in IoT are based on handcrafted feature detection systems such as eigenfaces, principal component analysis, Fisherface algorithm. These systems are not suitable for unconstrained faces with various illumination, pose, expression. Deep Learning models solves these above problems by learning the crucial features through learnable filters.

However, Deep learning models contain large number of layers which makes it harder to deploy in embedded device. Therefore there are larger servers serving these models through API. Embedded devices uses these API to identify or recognize the people. This leads to lot of network usage and network overhead in the process of recognition as the video or image data should be uploaded to the server in real time.

3.3 PROPOSED SYSTEM

The proposed facial recognition system will be based on a deep learning model which is more robust to faces with varying pose, illumination, and expression. It will also be deployed in an embedded device such as Raspberry Pi which can used for home automation systems. This will make the system work offline which will significantly reduce network overhead and usage.

3.4 FUNCTIONALITY

There will be a web camera attached to the embedded device (Raspberry pi) which will constantly look for faces in a certain radius. Once a face gets detected, it will be given to the deep learning face recognition model which will output its prediction. If the prediction crosses a certain threshold, it will classify the person which can be used to authenticate a person.

The model can also be retrained and customized for a specific household since the crucial features are learned by the convolutional layers. This makes the system vastly scalable and applicable to Home Automation Systems.

Chapter 4
SYSTEM DESIGN

4.1 SYSTEM REQUIREMENTS

4.1.1 HARDWARE REQUIREMENTS

For Training the model

- Intel Processor i5/i7/Xeon.
- Minimum Nvidia Geforce 980M, Recommended 4 Nvidia 980M.
- Minimum 16GB RAM, Recommended 32GB.
- Minimum 80GB HDD, Recommended 128GB SSD.

For Deployment 10,000 Request per Second.

- Intel Processor Xeon.
- Nvidia Tesla P100 8X.
- 64GB RAM.
- 256GB SSD.

For Deployment 1 Request per Second.

- Intel Processor i3/i5 any similar processor.
- 512MB GPU.
- 512MB RAM.
- 8GB SSD.

4.1.2 SOFTWARE REQUIREMENTS

- Python.
- NumPy.
- OpenCV.
- Matplotlib.
- Compute Unified Device Architecture(CUDA).
- CUDA Deep Neural Network(cuDNN).
- TensorFlow.
- TensorBoard.
- Jupyter-Notebook.

4.2 SOFTWARE DESCRIPTION

4.2.1 Python

Python is the high level programming language with simple and elegant syntax. It is an interpreter based language which makes it easy to script and test. It supports different programming paradigm like object-oriented, imperative, functional, procedural and reflective. It has inbuilt high level efficient data structures. It is the ideal language for rapid application development.

Python has rich set of library support for different domains. It has latest support for majority of Machine Learning, Artificial Intelligence and Data Science libraries. Python emphasizes readability and brevity in writing programs. It helps to integrate different systems easily. It provides standard library which is written in C programming language. It has cross platform support.

4.2.2 NumPy

NumPy is the fundamental package for numerical and scientific computation. It can create and operate on N-Dimension Matrix data. It has ready to use mathematical, logical, shape manipulation, sorting, selecting, I/O, linear algebra, Fourier transform, and random number functions. The efficient multi dimensional array can hold generic data but should be same for whole array. It can easily and effectively integrate with variety of databases. It helps developer when the program needs to handle large size of numerical data. Most of the scientific library uses NumPy for handling data.

4.2.3 OpenCV

OpenCV (Open Source Computer Vision) is a library for Computer Vision library which has real time computer vision functions. It is cross platform and supports languages like C, C++, Java and Python. It has more than hundreds of computer vision algorithms. It is implemented in C++. It has computation acceleration with OpenCL and CUDA GPU interfaces. Operations on Image and Video made simple and efficient with OpenCV.

4.2.4 Matplotlib

Matplotlib is the python plotting which works on NumPy. It provides object-oriented API to integrate plots and graphs in applications. Pyplot is the module which produces interface like MATLAB. It has different visualization packages like generate plots, histograms, power spectra, bar charts, errorcharts, scatterplot, 3D plot, and image plot. It uses general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+ to visualize.

4.2.5 Compute Unified Device Architecture (CUDA)

CUDA is a parallel computing platform created by Nvidia. It allows developers to exploit the computation ability of Graphics Processing Unit (GPU). It is the software layer which gives access to virtual instruction sets and parallel computational elements. It works with C,C++ and Fortran. Programmers run sequential process or single thread optimized process on Central Processing Unit (CPU) and parallel or multi threaded computation are moved GPU. CUDA helps to directly execute C/C++/Fortran code in General purpose GPU. Scattered reads, unified memory, shared memory, faster downloads and read-backs to and from the GPU, full support for integer and bitwise operations makes it better option for performance in parallel processing. It can processes matrix and large chunk of numerical data more efficiently than CPU. It is supported in majority of Nvidia GPUs.

Firstly, CUDA copies data from main memory to GPU memory. Then, CPU directs the executables to GPU. Then, GPU executes the instructions in parallel in each core. Finally, CUDA copies the result from GPU memory to main memory. It is released as free-ware by Nvidia.

4.2.6 CUDA Deep Neural Network (cuDNN)

It is a library for deep neural network on CUDA platform. This library holds all the essential primitives of deep neural networks like forward and backward convolution, pooling, normalization, activation layers, Rectified Linear Unit (ReLU), Sigmoid, softmax and Tanh. It has the optimized and GPU accelerated implementation of deep neural network functions. It provides abstract layer hiding complex low level performance tuning on GPUs. It has Context-based API allows for easy multithreading and Tensor transformation functions. cuDNN accelerates widely used deep learning frameworks like TensorFlow, Theano, Torch, Caffe and Keras. cuDNN helps developer to create deep learning models avoiding complex implementation of mathematical function optimized for GPU and GPU performance.

4.2.7 TensorFlow

TensorFlow is an open source software library for machine learning developed by Google and released under the Apache 2.0 open source license. It can run on multiple CPUs and GPUs. It supports Microsoft Windows 64bit, Linux 64bit, Mac OS 64bit, Android and iOS. TensorFlow API is used as library in C++/Python .

Every Component in TenorFlow is represented as multidimensional array called Tensor. TensorFlow computations are expressed as graphs. The graphs operates on Tensors like data flow. In TensorFlow the computation graph is defined with initialized Tensors. Then, to execute the defined computation a session is created and using using the session the computation is executed. Values in tensor can be accessed or feed using session. Name of the Tensor is specified as first argument in session to return the value from TensorFlow. Dictionary with Tensor name as its key and Input to tensor as its value is passed as the second argument in session. Multiple session can be executed for same graph. TensorFlow has CPU and GPU versions.

TensorFlow has inbuilt functions for optimization like Gradient descent, Adadelta, Adagrad, AdagradDA, Momentum, Adam, Ftrl, Proximalgradientdescent, ProximalAdagrad and RMSProp Optimizer. It has set of configurable deep learning functions like convolution, pooling, normalization, activation layers, ReLU, sigmoid, softmax and dropouts.

TensorFlow allows to run learning algorithm in batch mode to run large training dataset. It offers ability store/restore session and model. This make it easy to train large models which requires more iterations. It allows developer to tune hyper parameters in machine learning models to get better model. It also logs the values at specified iterations for visualizing the model performance and correctness. It stores the ml file which has all session values which is instructed to be stored.

TensorFlow contains many trained models like QuocNet, AlexNet, Inception (GoogLeNet), BN-Inception-v2, Inception-v3 and ResNet. TensorFlow allows to retrain or transfer learning to new model which saves developers and engineers time for training models.

TensorFlow runs various modes. It runs on normal CPU mode, GPU mode, Distributed CPU mode, Distributed CPU and GPU mode. It offers developers to ability to run it on clusters which saves time. Distributed mode helps engineers to fine tune the model with better parameters. It can execute Distributed model mode and distributed data mode.

TensorFlow offers Serving module which offers organization to deploy models in Real-time applications. It runs on grpc which is a remote procedure call. Each serving process runs the model in isolated TensorFlow environment. All the above features make it the popular and best machine learning framework.

4.2.8 TensorBoard

TensorBoard is a suite of web applications for inspecting and understanding your TensorFlow runs and graphs. It is tool which is used to visualize Scalar, Images, Audio, Graph, Distributions, Histograms and Embeddings. Tensor values are shown as distribution plots and histograms. Scalar values like loss, accuracy and iterations are plotted in Scalars. Graph shows the TensorFlow Model Architecture as a flow chart. Embeddings shows the t-SNE and PCA of the Data and Weights. TensorBoard requires the log file generated by TensorFlow Session to visualize. It plots different log file with different colors.

4.2.9 Jupyter-Notebook

It is the Python IDE which helps developer to maintain code as cells which can be executed as bunch. The User Interface of Jupyter-Notebook is from browser which makes it sophisticated. It has ability to hold markdown and other rich text which helps developer to document on the go while prototyping. It can also convert the python notebook to python file.

4.3 ARCHITECTURE DIAGRAM

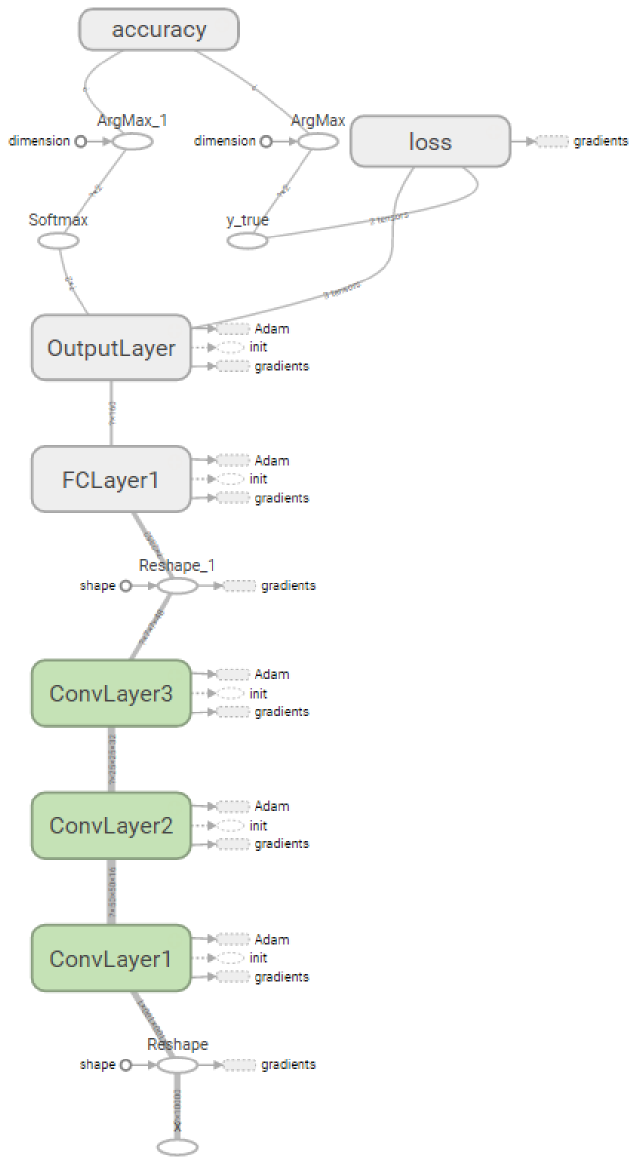


Figure 4.1: Architecture of CNN

The above figure shows the Architecture Convolution Neural Network. This model uses 100x100 as input features. In ConvLayer1 it uses 5x5x16 kernel to map 100x100 to 100x100x16. Then pooled or reduced to 50x50x16 in Pooling layer. In ConvLayer2 it uses 4x4x32 kernel to convolute 50x50x16 to 50x50x32. Then Pooling layer reduce 50x50x32 to 25x25x32. In ConvLayer3 it uses 3x3x48 kernel to convolute 25x25x32 to 13x13x48.Then pooling layer reduces 13x13x48 to 7x7x48. Then fully connected layer (FCLayer1) flattens 7x7x48 to 2352. FCLayer1 maps 2352 to 160. The Output Layer maps the 160 to Number of Output labels.

4.4 FLOW DIAGRAM

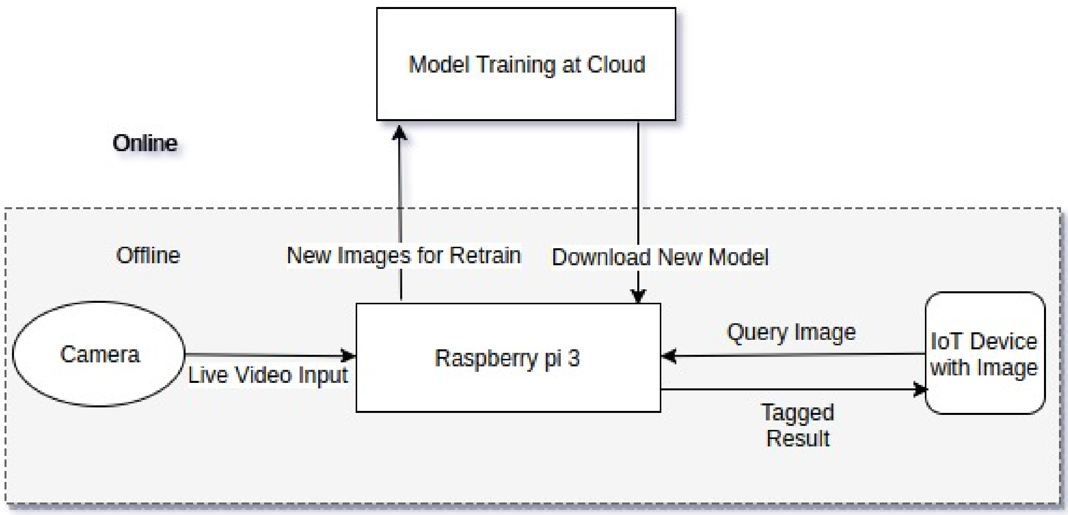


Figure 4.2: Flow Diagram of the Face Recognition System

Figure 4.2 shows the flow of the system. The model is deployed on the Raspberry pi 3 which accepts image from two different input such as camera and an image query from an IoT device. It is also primarily an offline model for the system to work and only when the model has to be trained or retrained again, the system will go online.

4.5 STATE DIAGRAM

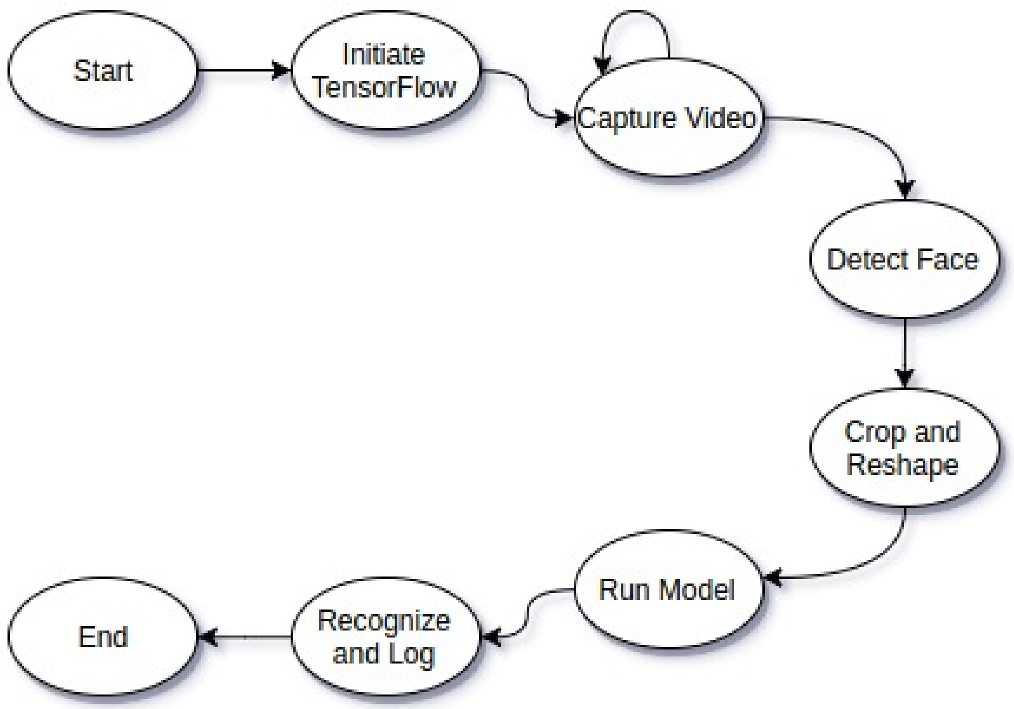


Figure 4.3: State Diagram of the Face Recognition System

Figure 4.3 shows the state diagram of the deployed model in Raspberry Pi and how it will recognize images from a webcamera feed. First, TensorFlow library will get initiated since it is the deep learning framework which contains the model. Then, the video will be captured live from the webcam which will give input to the face detection state. Once a face gets detected, it will be reshaped and given as input to the model. The model will output its prediction which will be given as a tag to the end user.

Chapter 5
IMPLEMENTATION

There are mainly four modules where were implemented in the project. They are Data Preprocessing, Training, Visualizing and Testing.

5.1 DATA PREPROCESSING

The main data set provided by Microsoft which is used to train our model is MSRA-CFW: Data Set of Celebrity Faces on the Web [4]. It contains over 200,000 color images of popular celebrities under various pose, expression, dimension and lighting conditions.

This dataset was reshaped to a fixed size of 100X100 pixels and mapped to gray scale using OpenCV and Haar Cascade representation. The mirror of the preprocessed image was also taken to increase the data set. Haar Cascade is a face detection classifier which was based on an object detection cascade proposed by Paula Viola and Michael Jones [5]. It works similar to convolution filter where a kernel will look for specific set of features and if it passes the filter, it will go to the next set of filters. If it does not pass, then the image will be discarded.

Finally, The data set was also normalized using Min-Max Normalization as the images were fed into the input layer. This vastly improved the performance of the Face Recognition System as images were taken in various lighting conditions. The result of the preprocessing is shown in Figure 6.1 in Result section.

5.2 TRAINING

5.2.1 Loading the Dataset

After preprocessing the data into a 100x100 normalized gray scale image, it was loaded into NumPy Array object. The corresponding labels of the images were also loaded into another Numpy Array Object. This will be later given to the tensorflow session in a dictionary known as feed dict where the training will be instantiated. There are over 300,000 images in the data set where 10% of the data was allocated to testing and the remaining data was allocated to training.

5.2.2 Defining the Hyper Parameters

In Machine learning, one trains models which are mathematical functions that learn some parameters which will be used for classification, and regression. However there are some parameters which cannot be learned by the model and has to be given during the training stage. These parameters are called hyper parameters and are essential for a machine learning model's capacity to learn and its complexity.

This project required many hyper parameters to be defined such as

- Number of Convolution Layers
- Number of Max Pool Layers
- Number of Convolution Filters
- Convolution Filter Size
- Convolution Stride
- Learning Rate
- Batch Size.
- Number of Iterations to train the model
- Size of the Fully Connected Layer
- Number of Classifiers

These hyper parameters were defined with guidance from the base paper [1] and it is shown in Table 5.1 below.

5.2.3 Creating a Convolution Layer

As discussed previously, convolution layers are where crucial features of the images are detected by its weights. These weights are known as filters and these are trained to identify some specific features from facial images.

In Tensorflow, there is a function called `tf.nn.conv2d()` which takes input from previous layer, number of input channels,filter size, list of strides, padding

Hyperparameter	Value
Number of Convolution Layers	3
Number of Max Pool Layers	3
Learning Rate	0.001
Batch Size	100
Number of Iterations	150,000
Fully Connected Layer Size	160
Classifiers	1581
Layer 1 Hyperparamters	**Value**
Number of Convolution Filters	16
Convolution Filter Size	5
Stride	[1,1]
Layer 2 Hyperparameters	**Value**
Number of Convolution Filters	32
Convolution Filter Size	4
Stride	[1,1]
Layer 3 Hyperparameters	**Value**
Number of Convolution Filters	48
Convolution Filter Size	3
Stride	[2,2]

Table 5.1: Hyperparameters for the Face Recognition Model.

and whether to perform it on cuDNN, the deep neural network from NVIDIA, on GPU.

Thus, using the above function provided by tensorflow, three convolution layers were created.

5.2.4 Creating a Max Pooling Layer

In Max pooling layers, the size of the convolution filters are reduced by picking the largest weight value from a 2x2 pixel window. This ensures that only crucial features which have the most impact remain when going to the next convolutional layer.

In Tensorflow, there is a function called `tf.nn.max_pool()` which takes input from previous layer, list of strides and padding. Thus, using the above function three max pooling layers were created after each convolution layer was created.

Rectified Linear Unit (ReLU) operation is also performed after max pooling. It calculates max(x,0) for every input pixel x and thus adds some non linearity to the model. This makes the model learn some complicated functions. In tensorflow, `tf.nn.relu()` performs the ReLU operation which takes the output of max pooling layer as input.

5.2.5 Flattening the Convolution Layer

The output of the max pooling layer has to be flattened before giving it as input to the fully connected layer. The input's shape is [num_images, img_height, img_width, num_channels] and is converted to [num_images, img _height * img_width * num_channels]. This is done using a Tensorflow function known as `tf.reshape()` which takes in the max pool layer output as input and the output shape of the flattened layer.

5.2.6 Creating a Fully Connected Layer

This is the final layer of the CNN and where weights and bias have to be created. They are created using `tf.Variable()` command which takes in either constant values or randomized values. Usually, the weight values are randomly initialized by `tf.truncated_normal()` function and the bias values are constantly initialized with tf.constant() function.

After defining the weight and bias variables, they will be combined using this function

$$y = WX + b$$

where y is the output layer and W is the weights, X is the previous layer and b is the bias value. Now this can be activated by sigmoid function also known as softmax or by ReLU function. For this model, they are two fully connected layers where the first layer is activated by ReLU and the second layer is activated by softmax.

5.2.7 Defining Loss Function for Back Propogation

This is the last step of setting up the model before instantiating it in Tensorflow graph. In this step, one chooses how to calculate loss which will be minimized in back propagation. This model uses cross entropy method to calculate the loss and it is given by `tf.nn.sparse_softmax_cross_entropy _with_logits()`. This function takes in the last fully connected layer and its corresponding labels where the softmax function will be applied and loss will be calculated.

Then to minimize loss, back propagation will begin and it will use the gradient descent algorithm to update weights and bias values. In this model, Adam optimizer is used which is the most popular gradient descent algorithm used for Convolutional Neural Network.

5.2.8 Instantiating the Tensorflow Training

Finally, this is where all the operations gets instantiated and all the previous steps results are actually calculated. It is done using a tensorflow session which is given as `tf.Session()`. Then we provide the input images along with the corresponding labels in `session.run()` which takes input in a form of a python dictionary known as `feed_dict`. The output of loss and accuracy can taken which will be used to evaluate the model.

5.2.9 Saving and Restoring the Model

Finally, after training, the model has to be saved for using to classify faces in real time. It is provided by Tensorflow as `tf.trainSaver()` and one can use it to save the model. Another reason to save the model is make the training process easier to run with limited resources. One can train for a few hundred thousand iterations and save the best model. Then one can run restore the model and train it for some more iterations. This reduces the computation and running time of training.

5.3 VISUALIZING

After training, it is important to visualize the model to see if it can be deployed in real time. Tensorflow and Python offer several different packages to visualize the model. The three main visualization which are essential for this model are convolution filters, convoluted images with filters and the training accuracy and loss of model over time.

5.3.1 Plotting the Convolution Filters

As discussed previously, Matplotlib is a plotting library in python which works with NumPy objects seamlessly. Therefore one uses the weights and the corresponding filter and filter size of each convolution layer to output the convolution filters. This is a key indicator to see if all filters are unique and if there is any crucial features learned by the convolution layers. The filters are shown in Result section in Figure 6.5,6.6 and 6.7 for each convolution layer respectively.

5.3.2 Plotting the Convolution Images

After getting the convolution filters, one also needs to look at the convoluted images with respect to the filter to see if they are any meaningful features which are learned by the convolutional layer. This is also plotted using matplotlib where the image along with the filter is combined to produce a interpolation image. These images are also shown in Result section in Figure 6.9, 6.10 and 6.11 for each convolution layer respectively.

5.3.3 Visualizing Accuracy and Loss in TensorBoard

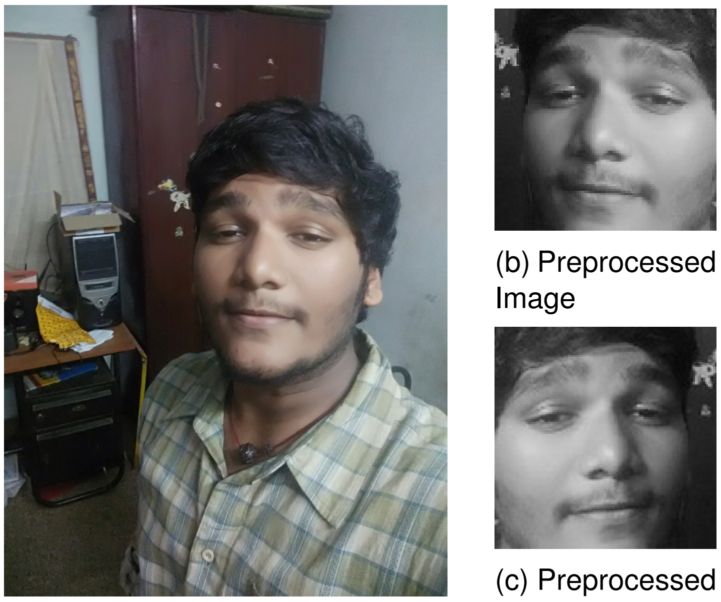
Finally, one has to visualize how the accuracy and loss during training fluctuate over time. This can be achieved using TensorBoard which is a visualizing tool provided along with Tensorflow. One will use `tf.summary.scalar()` function to write all the summaries to an external file. Then one can point this log directory to Tensorboard which will visualize the loss and accuracy functions. These graphs are also shown in Result section in Figure 6.4.

5.4 TESTING AND WRAPPER

Finally, after training and visualizing the model,there were two testing files which were written to test the model. The first testing file will get its input from a folder where lots of images of popular celebrities from the Internet will be downloaded. It will be evaluated by how will it can classify all the images from the folder. In the next testing file, the input will be given in the form of a web camera image and the model has to classify people who were trained. It should ignore people who were not trained in the model. The results of both the testing are shown in Result section in Figure 6.12 and Figure 6.13 respectively.

Chapter 6
RESULT

6.1 PREPROCESSING



(a) Original Image

(b) Preprocessed Image

(c) Preprocessed Mirror Image

Figure 6.1: Results after Preprocessing

Figure 6.1 explains how the dataset was preprocessed using OpenCV and Haar Cascade representation. As one can see in Figure 6.1b, it is reshaped to 100x100 in gray scale and Figure 6.1c shows the mirror image which was also included in the dataset.

6.2 TRAINING

6.2.1 Convolution and Max Pooling Layers

In Figure 6.2 , the convolution layer, max pooling layer and ReLU layer are shown as a Graph object in TensorBoard. It shows a flow of tensors between various operations.

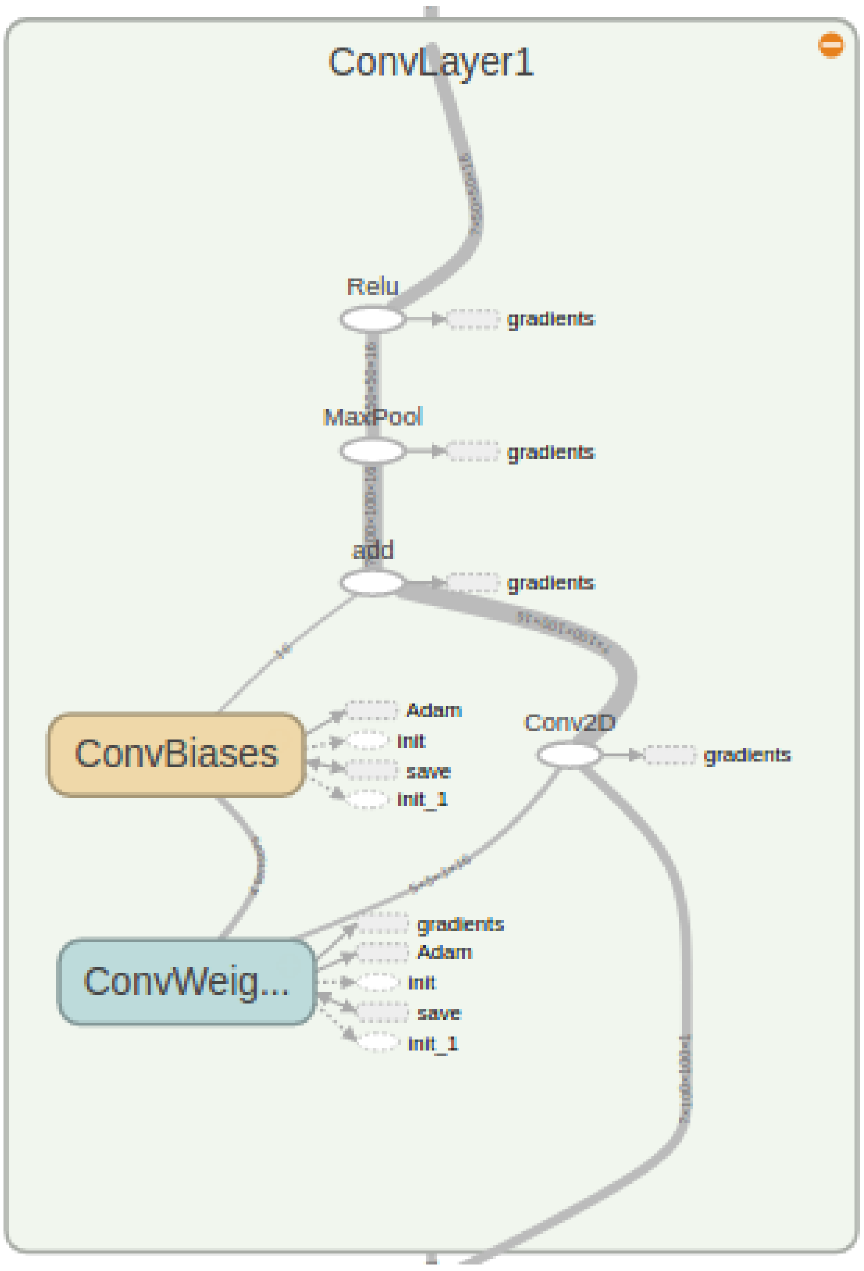
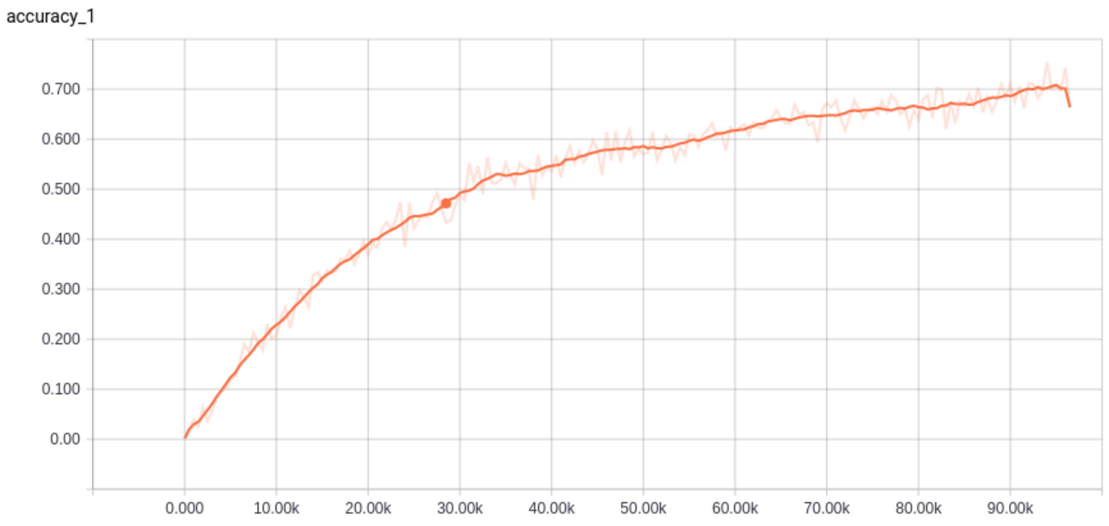


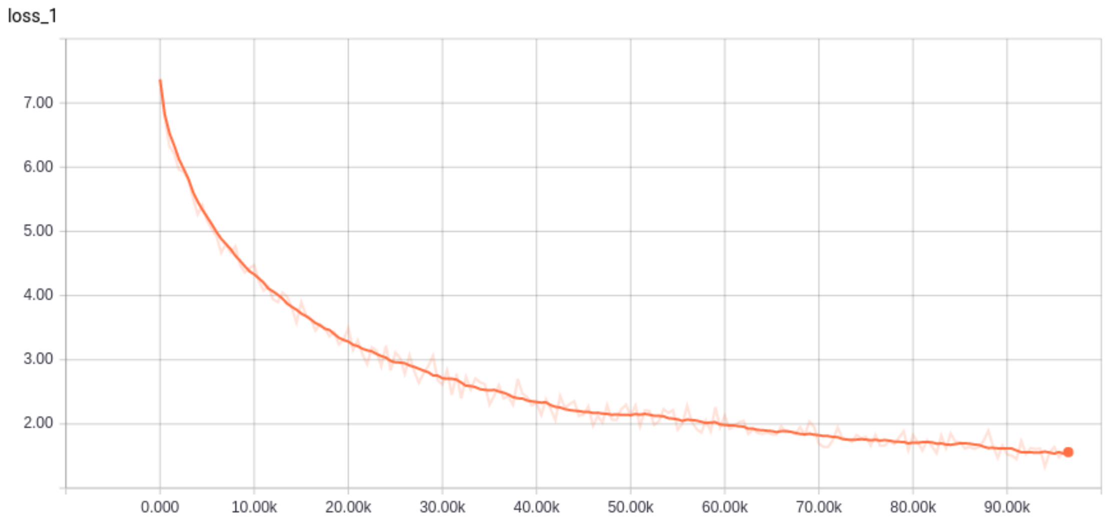
Figure 6.2: Convolution, Max Pooling and ReLU in TensorBoard

6.2.2 Loss and Accuracy

The loss and accuracy functions for the first 100,000 iterations has been shown in Figure 6.3. One can observe that accuracy function is gradually increasing and the loss function is gradually decreasing. This process took about seven hours to complete and therefore it was saved before running another 50,000 iterations.

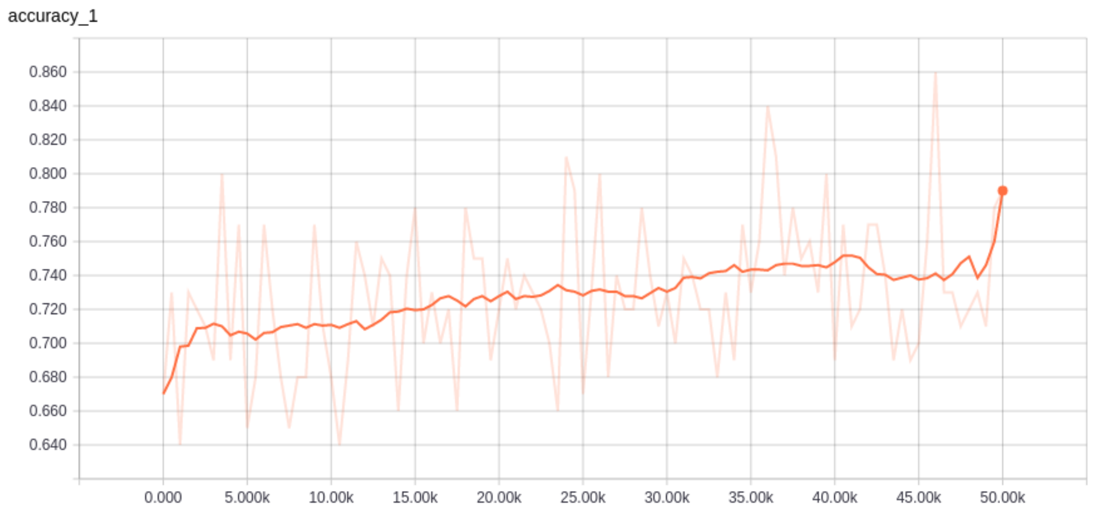


(a) Accuracy

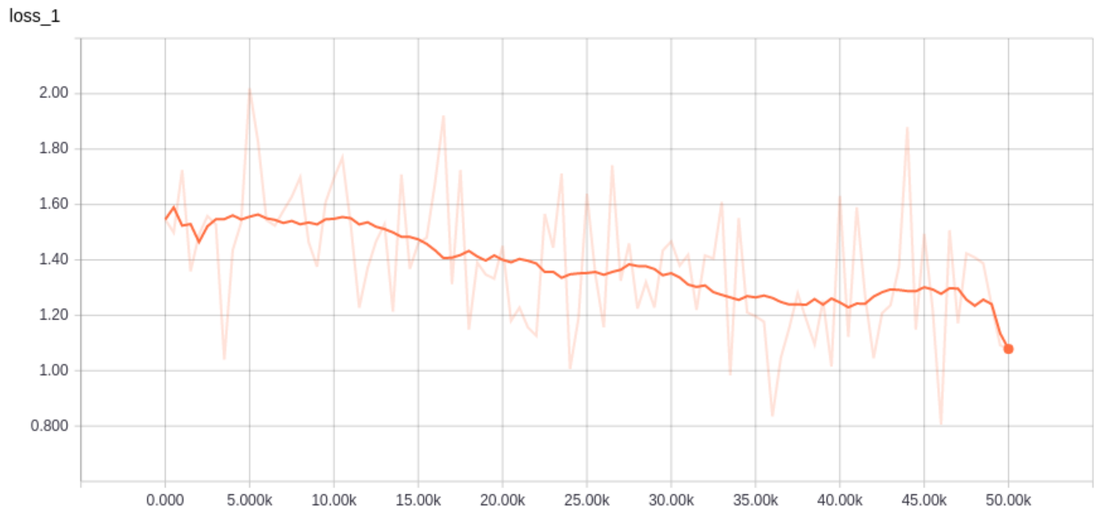


(b) Loss

Figure 6.3: Training Loss and Accuracy after 100,000 iterations in TensorBoard



(a) Accuracy



(b) Loss

Figure 6.4: Training Loss and Accuracy after 150,000 iterations in TensorBoard

As mentioned previously, the last 50,000 iterations were done separately since it takes a lot of time to train. The Loss and Accuracy Functions after the last 50,000 Iterations in training are shown in in Figure 6.4. It reaches around 79% training accuracy and loss function goes to 1.10 after the end of 150,000 iterations.

6.3 VISUALIZING

6.3.1 Convolution Filters

Layer 1 Filters

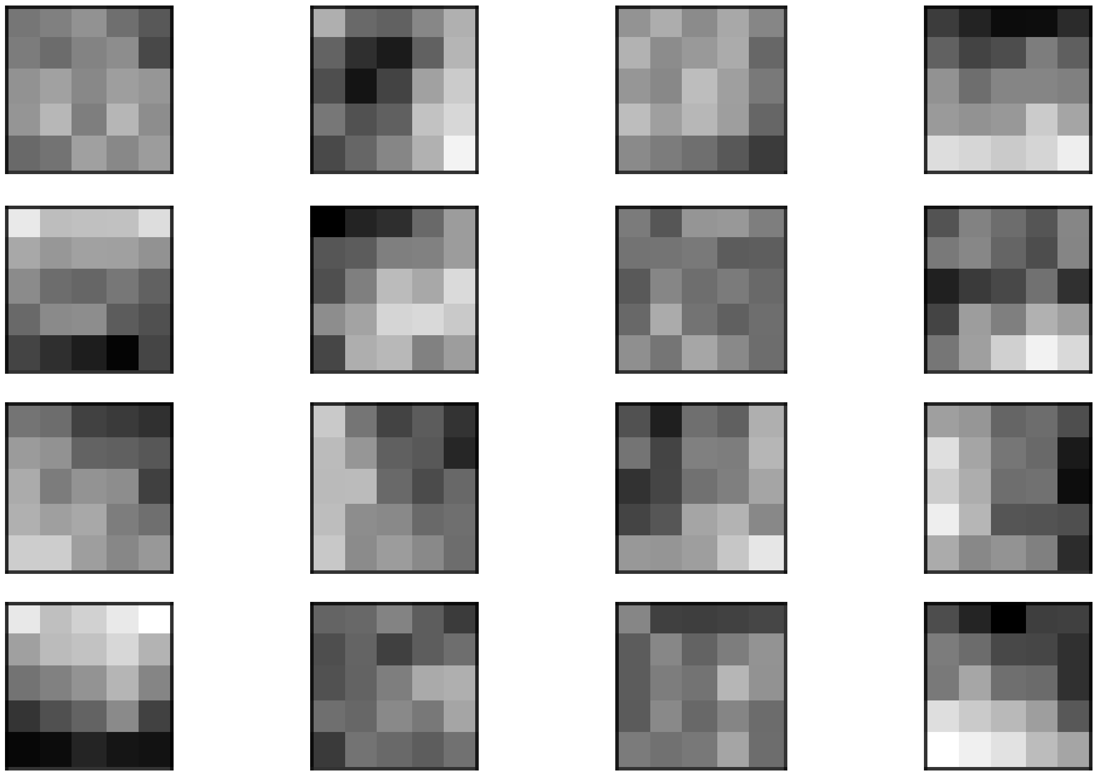


Figure 6.5: Convolution Layer 1 Filters

As one can see in Figure 6.5 there are totally 16 filters in the first convolution layer and the white color represents the maximum weight values and black represents minimum weight values.

Layer 2 Filters

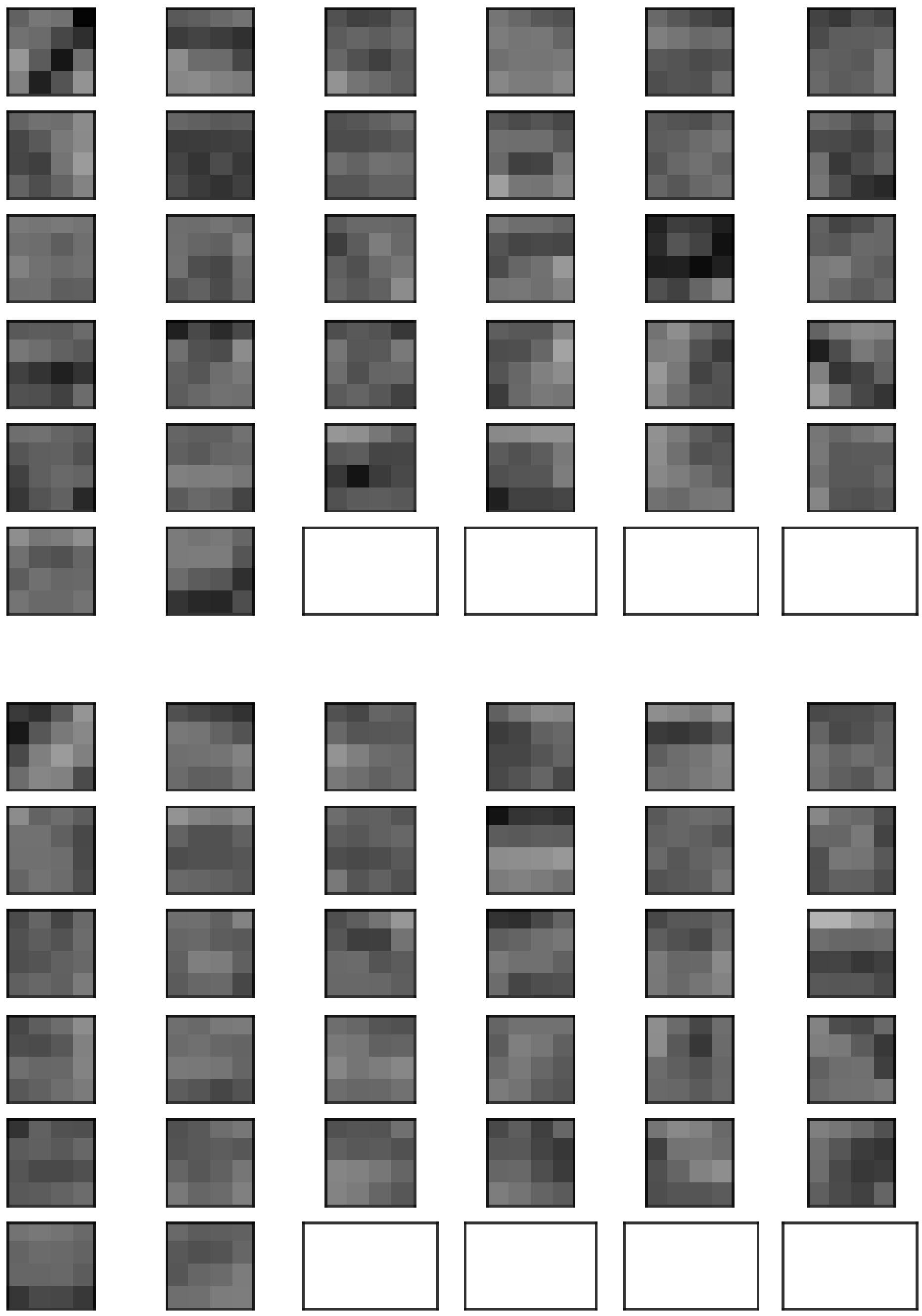


Figure 6.6: Convolution Layer 2 Filters

As one can see in Figure 6.6 there are totally 32 filters for each of the input channel in the second convolution layer which makes it a total of 512 filters. Only two input channels' filters (64 filters) have been displayed.

Layer 3 Filters

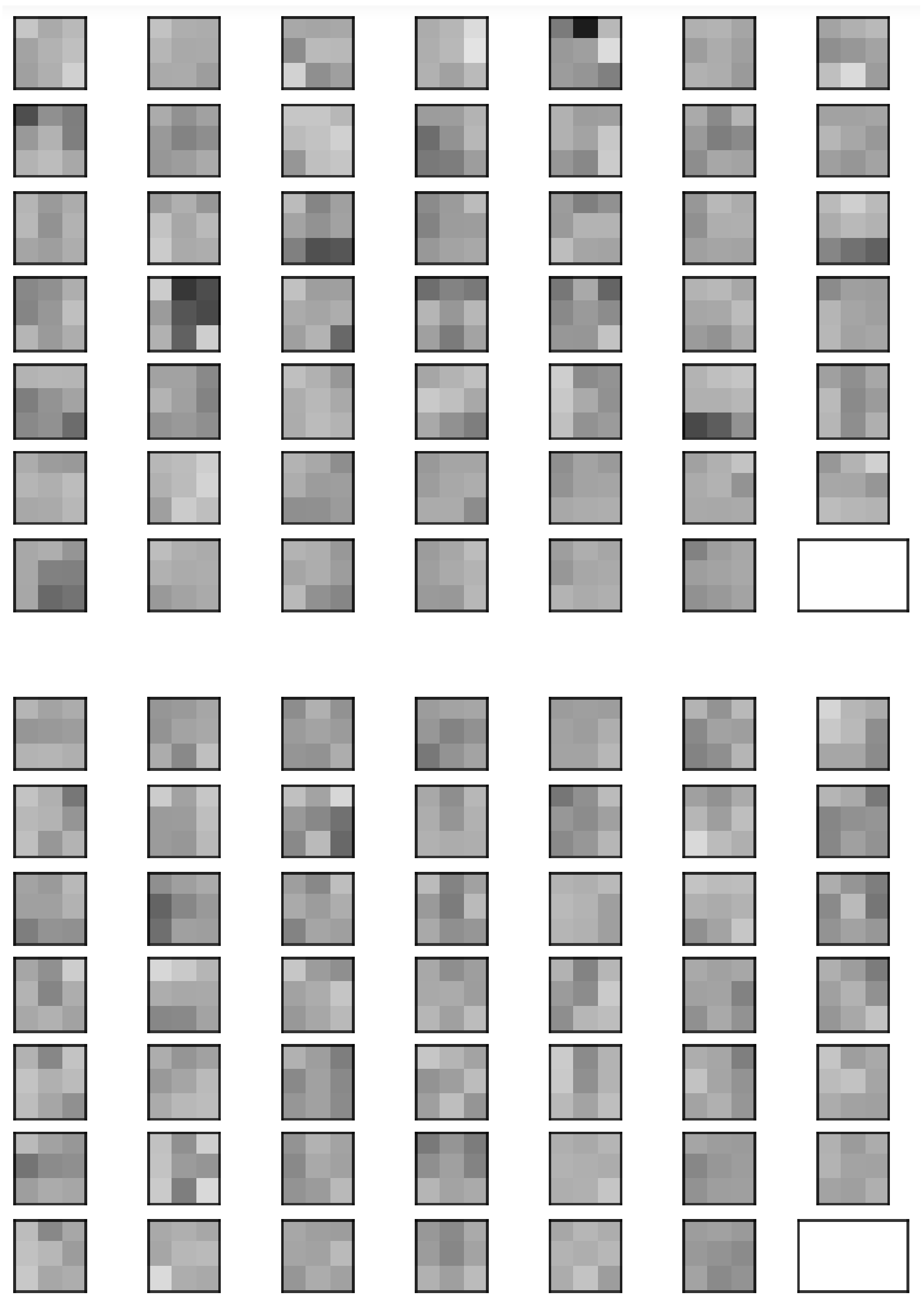
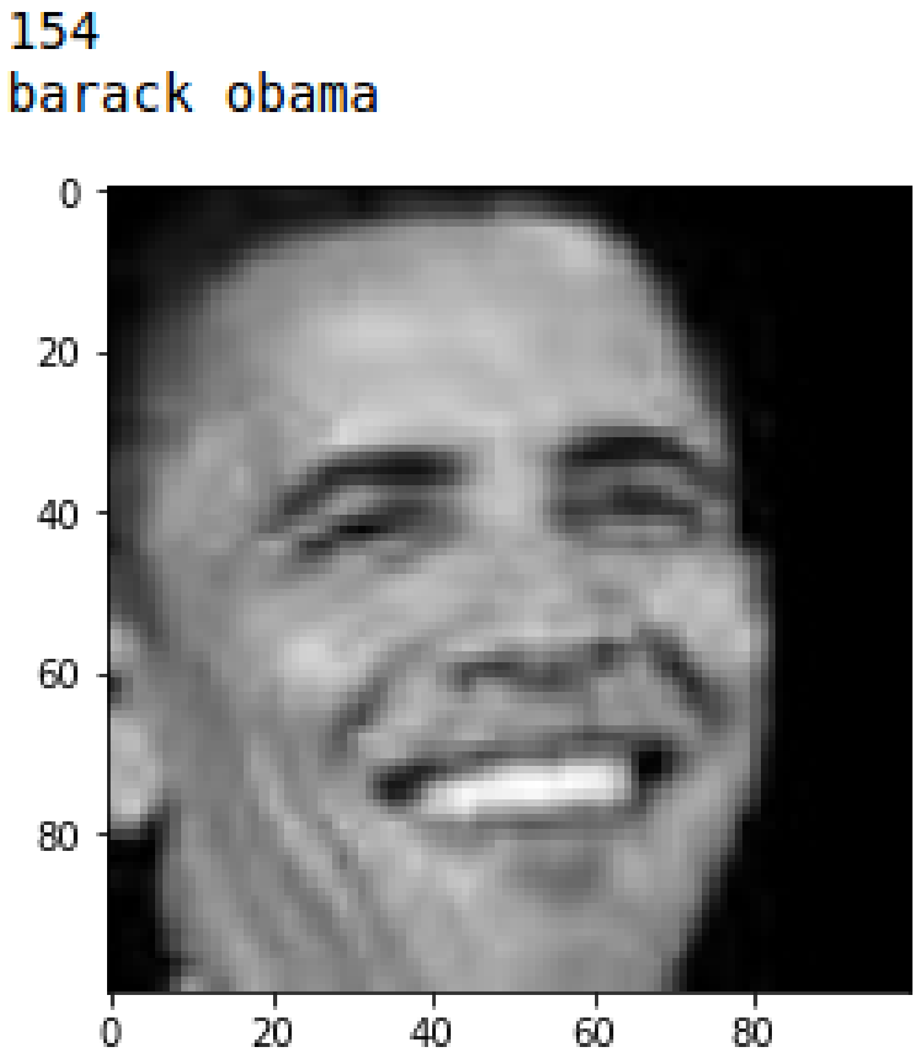


Figure 6.7: Convolution Layer 3 Filters

As one can see in Figure 6.7 there are totally 48 filters for each of the input channel in the second convolution layer which makes it a total of 24576 filters. Only two input channels' filters (96 filters) have been displayed.

6.3.2 Convoluted Images

Another way to visualize the model is to see how the convolution filters absorb crucial features of an image. Figure 6.8 shows an image which will be convoluted by the three layers



```
154
barack obama
```

Figure 6.8: Image of U.S President Barack Obama before Convolution

Layer 1 Convoluted Images

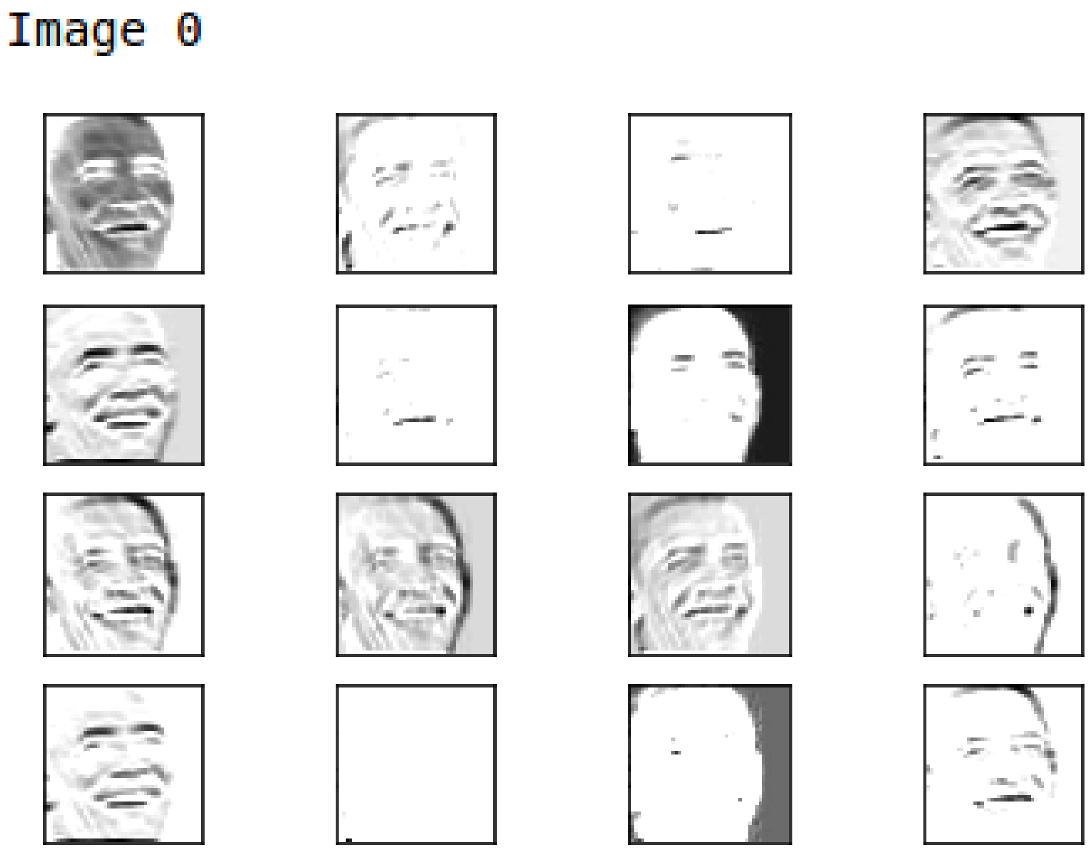


Figure 6.9: Convoluted Image of U.S President Barack Obama in Layer 1

As one can see, the first convolution layer learned about outlines and edges of the face. It also recognizes background from foreground and the background is largely filtered out by the first layer.

Layer 2 Convoluted Images

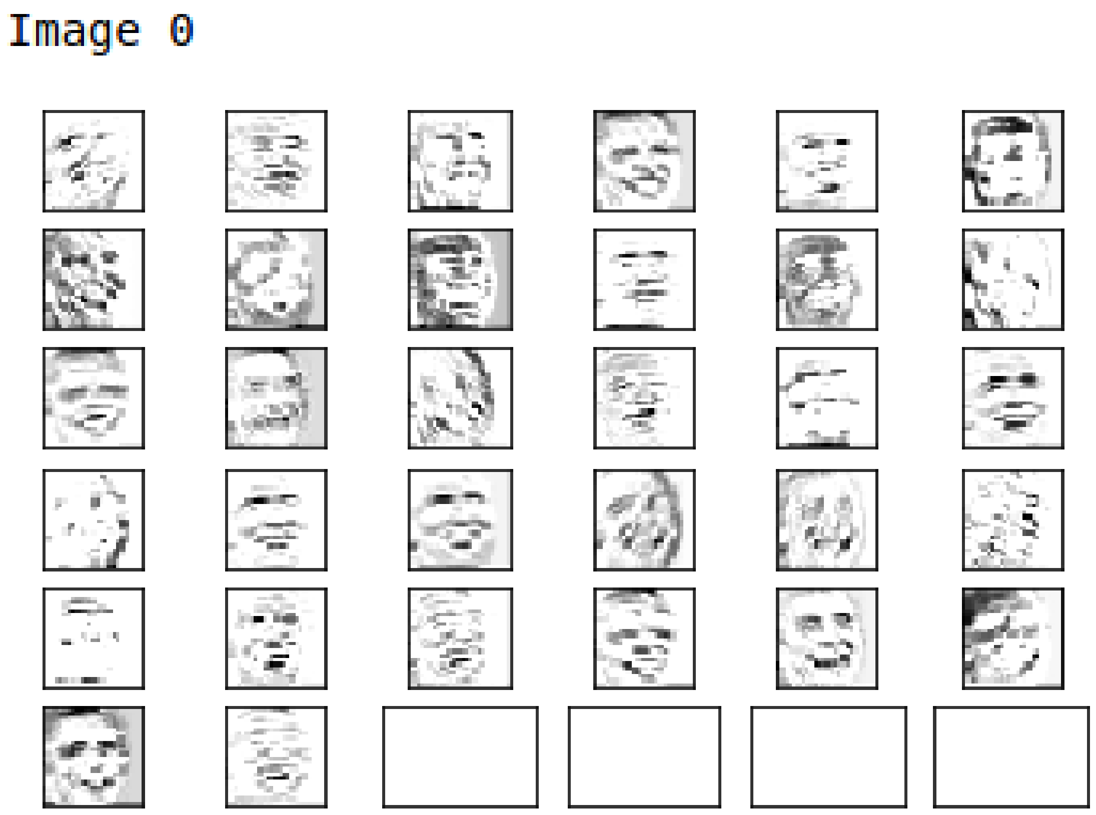


Figure 6.10: Convoluted Image of U.S President Barack Obama in Layer 2

As one can see, the second convolution layer learned about eyes, mouth, nose being part of face. Thus, it is increasing in complexity for identifying crucial features for Face Recognition.

Layer 3 Convoluted Images

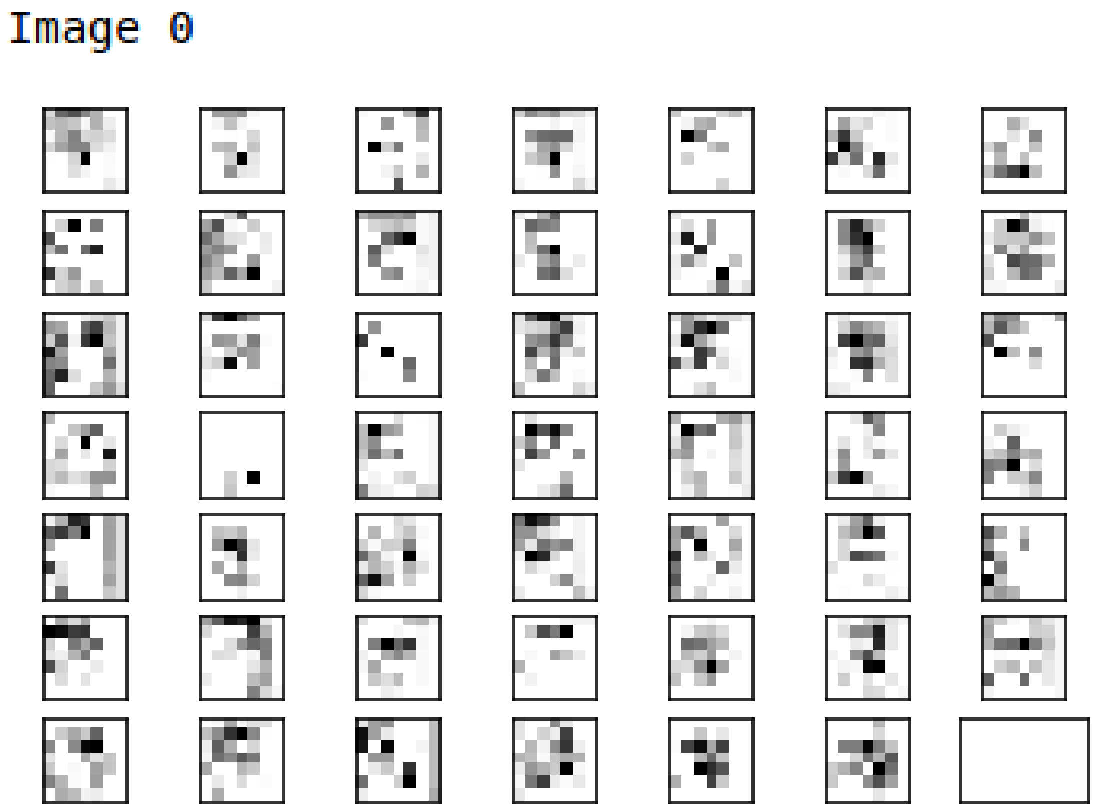


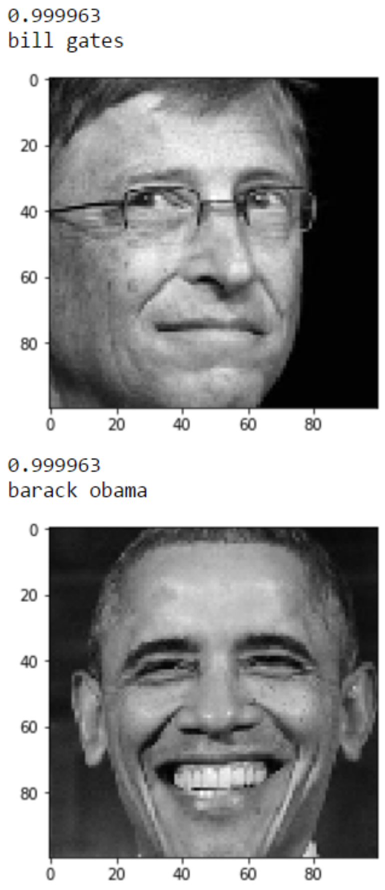
Figure 6.11: Convoluted Image of U.S President Barack Obama in Layer 3

As one can see, the third convolution layer is even more complex in feature identification for Face Recognition. The parts of the face which were not filtered in the first two layers are rigorously considered by the third layer and these filters are flattened and given to the fully connected layer.

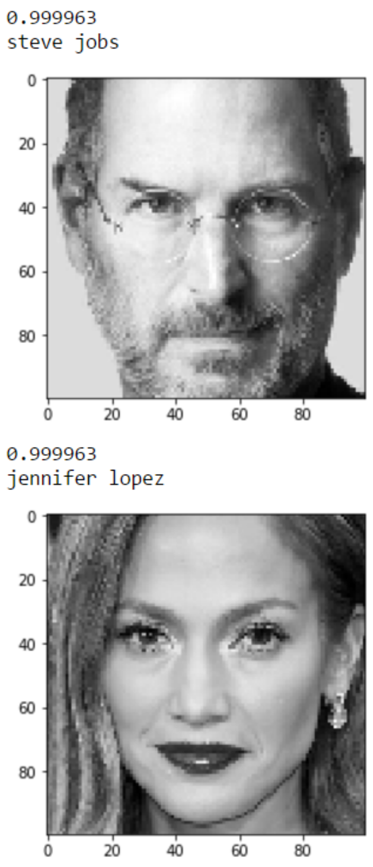
6.4 TESTING

As mentioned previously, there were two tests which was used to evaluate the performance of the face recognition system. The first test takes input from a folder which is filled with photos of celebrities downloaded from the Internet. The second test takes input from a live web-camera feed and recognizes people who were trained in the model.

6.4.1 Testing by Downloading Images from Internet



(a) Celebrity Test One.



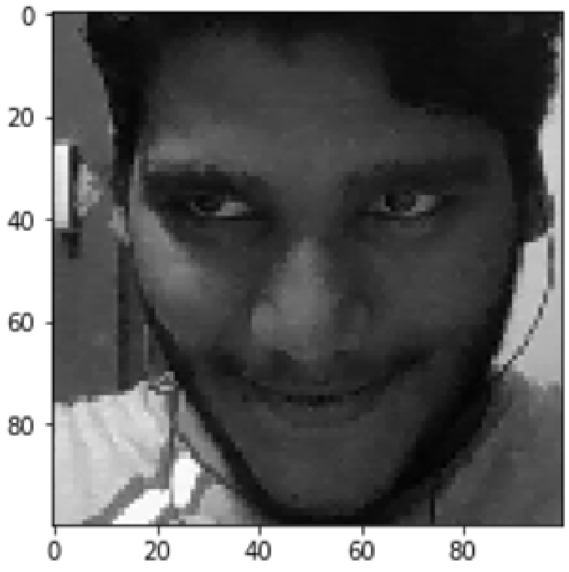
(b) Celebrity Test Two.

Figure 6.12: Face Recognition of Popular Celebrities.

As one can see in 6.12, the face recognition model correctly recognizes popular celebrities with very high confidence values.

6.4.2 Testing by Web Camera

```
0.999963
vigneshwaran
```



```
0.999963
vigneshwaran
```

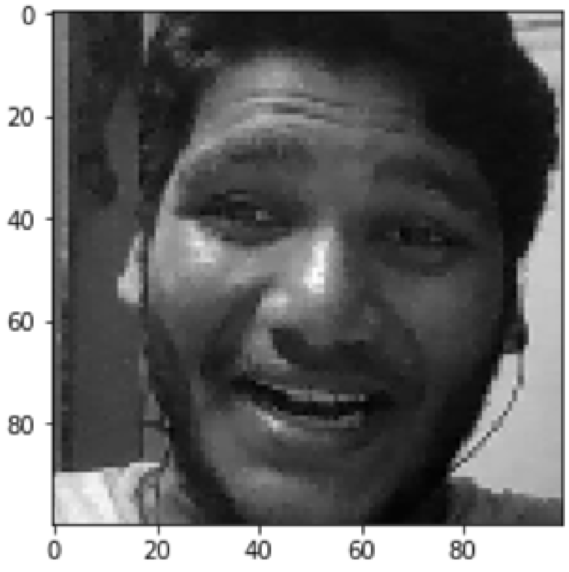


Figure 6.13: Testing with Web Camera Feed

The model also predicts with high confidence in a live web-cam feed as well. Thus, it can be deployed in real-time systems like Home Automation.

Chapter 7
CONCLUSION AND FUTURE WORK

Face Recognition System was successfully modeled by CNN and was implemented in an embedded device. It was trained with 300,000 images from the MSRA-CFW dataset and customized data. It achieved up to 81% validation accuracy with just three convolution layers, three max pooling layers and two fully connected layers.

In future, one can use regularization in the form of L1 regularization or dropouts to avoid over-fitting during the training process. This will help in increasing the validation accuracy of the model. Another way to avoid over-fitting is to increase the training dataset with facial images under various lighting conditions, expression, and pose. This will also generalize the model and it will make it easier to retrain the model for a particular household.

In Addition, one can also retrain popular deep learning architectures such as AlexNet, Inception (GoogLeNet), BN-Inception-v2, Inception-v3 and ResNet to perform Facial Recognition rather than creating a unique model just for Facial Recognition. This will add complexity to the model which might also improve the validation accuracy of Face Recognition.

Furthermore, other forms of deep neural networks which have memory such as Long short-term Memory (LSTM), Memory Augmented Neural Networks (MANN) can be used for Face Recognition. These neural networks will reduce the size of the training dataset and will make it even easier to retrain for a particular household.

Finally, face recognition can be further extended to a person identification by fusing voice recognition into the model. This will make it even more robust and can be used for systems which require very strict authorization and authentication.

REFERENCES

[1] G. Hu, Y. Yang, D. Yi, J. Kittler, W. Christmas, S. Z. Li, and T. Hospedales, “When face recognition meets with deep learning: An evaluation of convolutional neural networks for face recognition,” in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, Dec 2015, pp. 384–392.

[2] Y. Sun, X. Wang, and X. Tang, “Hybrid deep learning for face verification,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1997–2009, Oct 2016.

[3] K. Simonyan, O. M. Parkhi, A. Vedaldi, and A. Zisserman, “Fisher Vector Faces in the Wild,” in *British Machine Vision Conference*, 2013.

[4] X. Zhang, L. Zhang, X.-J. Wang, and H.-Y. Shum, “Finding celebrities in billions of webpages,” *IEEE Transaction on Multimedia*.

[5] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–511–I–518 vol.1.

APPENDIX 1
Sample Code

```
#Test.py
# coding: utf-8
import numpy as np;
import matplotlib.pyplot as plt;
import pandas as pd;
import os;
import tensorflow as tf
import cv2;
import time
import math

d = pd.read_csv('labels.txt')
directory = list(d['labels'])
directory=sorted(directory)

#Hyper Parameters
label_count=1581
image_size = 100
image_size_flat = image_size * image_size
image_shape = (image_size, image_size)
num_channels=1
num_labels = label_count #Currently there are 30 Labels
print(num_labels)
num_iterations = 10000
batch_size = 100
alpha = 0.0001;
# Convolutional Layer 1.
filter_size1 = 5
# Convolution filters are 5 x 5 pixels.
num_filters1 = 16
```

```
# There are 16 of these filters.
conv_stride1=[1,1,1,1]
# Convolutional Layer 2.
filter_size2 = 4
# Convolution filters are 4 x 4 pixels.
num_filters2 = 32
# There are 32 of these filters.
conv_stride2=[1,1,1,1]
# Convolutional Layer 3.
filter_size3 = 3
# Convolution filters are 3 x 3 pixels.
num_filters3 = 48
# There are 48 of these filters.
conv_stride3=[1,2,2,1]
# Fully-connected layer.
fc_size = 160

def variable_summaries(var):
    with tf.name_scope('summaries'):
      mean = tf.reduce_mean(var)
      tf.summary.scalar('mean', mean)
      with tf.name_scope('stddev'):
        stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
      tf.summary.scalar('stddev', stddev)
      tf.summary.scalar('max', tf.reduce_max(var))
      tf.summary.scalar('min', tf.reduce_min(var))
      tf.summary.histogram('histogram', var)

def new_weights(shape):
    return tf.Variable(tf.truncated_normal(shape, stddev=0.05))
def new_biases(length):
    return tf.Variable(tf.constant(0.05, shape=[length]))
def new_conv_layer(name,input,
                   num_input_channels,
                   filter_size,
                   num_filters,
                   list_strides,
                   use_pooling=True.
                   ):
```

```
    with tf.name_scope(name):
        shape = [filter_size, filter_size,
                 num_input_channels, num_filters]
        # Create new weights aka. filters with the given shape.
        with tf.name_scope('ConvWeights'):
            weights = new_weights(shape=shape)
            variable_summaries(weights)
        # Create new biases, one for each filter.
        with tf.name_scope('ConvBiases'):
            biases = new_biases(length=num_filters)
            variable_summaries(weights)
        # Create the TensorFlow operation for convolution.
        layer = tf.nn.conv2d(input=input,
                             filter=weights,
                             strides=list_strides,
                             padding='SAME',
                             use_cudnn_on_gpu=True)

            layer = tf.nn.max_pool(value=layer,
                                   ksize=[1, 2, 2, 1],
                                   strides=[1, 2, 2, 1],
                                   padding='SAME')

    return layer, weights


def flatten_layer(layer):

    layer_shape = layer.get_shape()
    num_features = layer_shape[1:4].num_elements()
    layer_flat = tf.reshape(layer, [-1, num_features])
    return layer_flat, num_features
def new_fc_layer(name,input,
                 num_inputs,
                 num_outputs,
                 use_relu=True):

    with tf.name_scope(name):
        # Create new weights and biases.
        with tf.name_scope('Weights'):
            weights = new_weights(shape=[num_inputs, num_outputs])
            variable_summaries(weights)
        with tf.name_scope('Biases'):
            biases = new_biases(length=num_outputs)
            variable_summaries(biases)
        if use_relu:
            layer = tf.nn.relu(layer)
    return layer

x = tf.placeholder(tf.float32, shape=[None, image_size_flat]
                   , name='x')
x_image = tf.reshape(x, [-1, image_size, image_size, num_channels])
y_true = tf.placeholder(tf.int64, shape=[batch_size], name='y_true')
#Convolutional Layer 1
layer_conv1, weights_conv1 =
new_conv_layer('ConvLayer1',input=x_image,
                   num_input_channels=num_channels,
                   filter_size=filter_size1,
                   num_filters=num_filters1,
                   list_strides=conv_stride1,
                   use_pooling=True
                   )
layer_conv2, weights_conv2 =
new_conv_layer('ConvLayer2',input=layer_conv1,
                   num_input_channels=num_filters1,
                   filter_size=filter_size2,
                   num_filters=num_filters2,
                   list_strides=conv_stride2,
                   use_pooling=True
                   )
layer_conv3, weights_conv3 =
new_conv_layer('ConvLayer3',input=layer_conv2,
                   num_input_channels=num_filters2,
                   filter_size=filter_size3,
                   num_filters=num_filters3,
                   list_strides=conv_stride3,
                   use_pooling=True
```

```
                    )
layer_flat , num_features = flatten_layer(layer_conv3)
layer_fc1 = new_fc_layer(name='FCLayer1',input=layer_flat ,
                         num_inputs=num_features ,
                         num_outputs=fc_size ,
                         use_relu=True)
layer_fc2 = new_fc_layer(name='OutputLayer',input=layer_fc1 ,
                         num_inputs=fc_size ,
                         num_outputs=num_labels ,
                         use_relu=False)
y_pred = tf.nn.softmax(layer_fc2)
y_pred_cls = tf.argmax(y_pred, dimension=1)
with tf.name_scope('loss'):
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=layer_fc2 ,labels=y_true)
    cost = tf.reduce_mean(cross_entropy)
tf.summary.scalar('loss', cost)
optimizer = tf.train.AdamOptimizer(learning_rate=alpha)
            .minimize(cost)

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(y_pred_cls, y_true)
    accuracy = tf.reduce_mean(tf.cast(correct_prediction ,
                                      tf.float32))
tf.summary.scalar('accuracy',accuracy)

session = tf.Session()
saver = tf.train.Saver();
#save_path = str(input("enter the model directory"));
save_dir = 'Model/'
save_path = os.path.join(save_dir, 'model1')
saver.restore(sess=session, save_path=save_path)
face_cascade = cv2.CascadeClassifier
               ('haarcascade_frontalface_alt2.xml')
cnt= 0;
APPROX = 15;
pixel=100

images = os.listdir("test");
```

```
print(images)
l = len(images)
test_data = np.zeros([l,100,100])
for i in images:
    img = cv2.imread('test/'+i,0);
    faces = face_cascade.detectMultiScale(img, 1.3, 5)
    for (x1,y,w,h) in faces:
        roi_gray = np.array(img[y-APPROX:y+h+APPROX,
                    x1-APPROX:x1+w+APPROX],dtype=float)
        img = cv2.resize(roi_gray, (pixel, pixel))
        test_data[cnt] = (img-img.min())/(img.max()-img.min());
        cnt = cnt+1
        break;
[recognized,val] = session.run([y_pred_cls,y_pred],
                  feed_dict={x:test_data.reshape(l,100*100)})
for i in range(cnt):
    if val[i].max() > 0.5:
        print(val[i].max())
        print(directory[int(recognized[i])])
        plt.imshow(test_data[i],cmap='gray')
        plt.show()
```