FACE DETECTION AND RECOGNITION USING MOVING WINDOW ACCUMULATOR WITH VARIOUS DEEP LEARNING ARCHITECTURE

BY

ANIL KUMAR NAYAK

SUPERVISING PROFESSOR

DR. FARHAD KAMANGAR

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2018

Acknowledgements

I would first like to thank my advisor Dr. Farhad Kamangar of Computer Science Engineering at The University of Texas at Arlington. Professors office was always open whenever I wanted a thesis discussion, or I face any kind of trouble during my thesis research or had a question about my research or writing. He consistently gave me feedback on my progress on monthly basis and steered me in the right the direction whenever he thought I needed it.

I would also like to thank my committee member Dr. Manfred Huber and Dr. Gergely V. Záruba for accepting the invitation be on my committee. I appreciate Dr. Manfred Huber's help. Whenever I need some guidance, he was always there and never hesitated to spare his valuable time for a quick discussion.

I would also like to thank the experts who were involved in the validation of my research work as I have been working on testing phase for this research project. My Friends who have helped me in proving data for testing and supporting me morally throughout my research.

Finally, I must express my very profound gratitude to Dr. Farhad Kamangar, my parents and to my friends here in Arlington as well as in INDIA for providing me with unfailing support and continuous encouragement throughout my MS of study and through the process of researching and writing this thesis. Thesis research would not have been possible without them. Thank you.

March 26, 2018

Abstract

Recent advancement in the field of Computer Vision and Deep Learning is making object detection and recognition easier. Hence, growing research activities in the field of deep learning are enabling researchers to find new ideas in the area of face detection and recognition. Implementation of such systems has a number of challenges when it comes to the current approaches. In this paper, we have presented a system of Face Detection and Recognition with newly designed deep learning classification models like CNN, Inception and various state of art models like SVM and we also compared the result with FaceNet. Multiple approaches to the face recognition were presented, out of which training of deep neural network, SVM on embedding data are optimized for the recognition task by implementing a moving weighted accumulator at the post processing stage. The accumulator helps in storing of past recognized faces for decision making.

For real-world testing, we have implemented a face detection and recognition graphical component, which has helped us in the testing of various deep learning models in real-world scenarios as well as to minimize the data collection efforts for incremental training of deep learning and classification models.

.

Contents

Table of Figures

List of Tables

Chapter 1: Introduction

1. Introduction

There have been many researches in the area of Computer Vision and Deep Learning for decades to make it better in detecting and recognizing objects and their interaction with the environment. Robots of different types are becoming more sophisticated with the use of such methodologies e.g. autonomous cars, Unmanned Aerial Vehicle (UAV), and surgery robots. These robots need large datasets from different sensors such as vision, proximity sensor, etc. to analyze real environment, and implementing deep learning architectures enables them to make better decisions with high accuracy when compared to a human brain. The brain is an extremely complex structure with multiple connections of neurons and sensors such as vision, touch sensors etc. Researchers have been working hard to match the level of human intelligence by improvising the robot's decision-making process with the implementation of deep learning methodologies. This process of improvements in above mentioned field of studies would not have been possible without the help of computer vision and deep learning research community.

1.1. Thesis Objective

The objective of this thesis is to investigate and analyze face detection and recognition system. Various deep learning architecture models have been closely analyzed, implemented and performances of each model have been observed and compared with present state of art classification models. Following models have been considered in this paper:

- FaceNet: A unified embedding for face recognition [16]
- Conventional Neural Network (CNN) deep learning architecture [3]
- Convolutional Inception Model deep learning architecture [3]
- Support Vector Machine, state of art classification model

CNN is widely used as the deep learning architecture for the object detection and recognition and it has helped researchers to improve the performance. These deep learning networks consists of multiple layers of convolution and max pooling. 2-D convolution on both grayscale images and RGB images can be processed through these kinds of networks. These networks have been used in the extraction of all level of features for analysis of an image and video contents for detection and recognition.

1.2. Preliminary Understanding

Recent advancement in the field of computer vision and deep learning has enabled researchers to improvise object detection and recognition. Many papers have been published on detection and recognition task like You Look Only Once (YOLO) [10], Google's object detection and recognition API [11], etc. in which, researchers have achieved a breakthrough and they proposed their model performed better than the state of art models using CNN and inception models. Implementation of CNN and Inception models requires prior knowledge of deep learning architectures, however, for new aspirants, those architectures have been explained in detail in the following section 2 and 3.

1.3. Methodology

Current researches in object detection and recognition inspired us to analyze and build a system of face detection and recognition with the help of LFW dataset. The system primarily comprises of two main components, face detection component and face recognition component with other additional components like pre-processing, post-processing, etc.

Face detection component has been developed using the existing libraries like Multi-Task Cascaded Convolutional Networks (MTCNN) [2] and DLIB [1], to identify the face bounding box and marking points in an image or a video frame, which will be explained in detail in the section [3.1.5.16].

Face recognition component, has been developed using various deep learning classification models like convolutional neural network, convolutional inception 1b model (which has only one inception layer), convolutional inception 5b model (which has 5 inception layers), state of the art classification model like SVM and FaceNet's face recognition functionalities. Theory and implementation of all the components have been explained in detail in the following sections 2 and 3.

1.4. Delimitations

Face detection and recognition system have been developed using following libraries and methodologies.

- SVM, CNN, inception 1b, and inception 5b models for the face recognition
- MTCNN and DLIB for face detection
- Face detection and recognition system GUI, which has helped us in real life testing and automatically capturing of the future dataset for our training.

1.5. Outline

Background of computer vision & deep learning and theories of deep learning architecture designs and terminologies has been explained in Chapter 2. Chapter 3, briefly discusses about our research implementation and experimental set-up. Followed by Chapter 4, which contains experimental result and analysis along with a comparison of various models, bibliography and the future work.

Chapter 2: Background and Theory

2. Background

Computer vision and deep learning is an extensive field of research. There are many articles and papers that have been published in various publications. The purpose of this chapter is to study the significant contributions in computer vision (section 2.1) and deep learning (section 2.2) research areas and recent research work in object detection and recognition. Note, this chapter assumes that the reader has prior knowledge of convolutional neural networks and their terminologies.

2.1. Background of Computer Vision

Computer Vision is an interdisciplinary field that deals with analysis of videos and image contents. In this field of study, researchers have been analyzing the human visual system and vision tasks. These tasks include methods on acquiring, processing, analyzing and understanding of digital images and videos. In order to produce numerical or symbolic information from the data, system extracts high-dimensional feature from real world scenarios to find meaningful information for easy understanding. The image data can take many forms, such as video sequences, stereo vision or multi-dimensional data from a medical scanner. Computer vision is concerned with the theory behind image processing which extracts information from this dataset.

In the late 1960s, this field of study has begun at universities, those were pioneering in artificial intelligence. It was designed to compete with the human visual system, which will be going to be the stepping stone for robots with intelligent behavior. In 1966, a breakthrough has happened while a camera was attached to a computer and allowing it to describe “what it saw”.

In this thesis, various computer vision techniques have been used to process an image and a video frame. These techniques like convolution, pooling, background subtraction, optical flow have been explained later in this document. Following are the pre-work that have been carried out to understand the field of computer vision related to object detection and recognition task.

- Understanding of Viola-Jones face detection framework [17].
- Understanding of Human Pose Estimation system to identify the human pose in video frames.
- Implementation of edge detection and smoothing operation to find features in an image and a video contents.

2.2. Background of Deep Learning

In 1962, Deep Convolutional Neural Networks (CNN) paper, laid the foundation for feature identification. The paper we are referring to is "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex" by Hubel and Wiesel [4]. Their experiment has shown new insights on how brain sees objects and things around us. The experiments were conducted on a sedated cat. They shone the light on to the cat's eyes with electrodes connected to its brain. The authors made very important findings of how the brain interprets visual stimuli. Finally, they observed that complex cells were activated by the same type of light as simple cells. The difference was that they were less dependent on the spatial position. Following sections contain a detail explanation of recent work and the deep learning architectures.

2.3. Recent Work in Deep Learning

Deep learning is very famous in research community for a long time because of its computation power and popularity in analyzing the video contents of huge data size with ease. However, it had lost its creativity and credibility in past because of lack in computation power and processing units. Recent advances in GPU's are enabling researchers to concentrate on deep learning area. Many papers have been published in various deep learning journals since 2012 related to object detection and recognition task, out of those most popular is capsule network [18], YOLO [16], Googles Tensorflow Object Detection API [15]. Following are the pre-work that have been successfully carried out to understand the field of deep learning related to object detection and recognition task.

- Face recognition using FaceNet.
- CNN based classifier to recognize face.
- Perceptron learning in python to understand the deep learning methodologies.
- Object detection and recognition using Google’s Object Detection API

2.4. Understanding CNN

Researchers have been concentrating on improvising CNN architecture. First, Szegedy [15], he showed that small perturbations on the images can cause 100% misclassification on the network that it has been trained on. He also showed that the perturbations are quite general as they significantly decrease the performance of networks trained with the difference in number of layers and using different training datasets. The knowledge they gained proved that it was actually the depth in CNN network that caused the significant leap in performance rather than the supporting tasks used e.g. cropping, the use of data augmentation, and GPUs.

Recently, a lot of progresses has been achieved in the area of image classification and object detection with the help of deep CNN classifiers. It all really took off when Krizhevsky and Hinton [19] crushed the previous state-of-art models and beat the Top-5 error rate in the ImageNet challenge by 10.9% (absolute) compared to the second-best entry in the competition. They found out that, 1x1 convolution in the last layer improved the classification rate drastically, whereas 1×1 convolutions correspond to a multilayer perceptron producing more advanced function approximation.

CNN uses multiple layers in its architecture. Following are the layers used to build convolutional neural network architectures.

- Convolutional Layer
- Activation Layer
- Pooling Layer
- Fully-Connected Layer or Densely Connected Layer
- Output Layer or Softmax Layer for classification

CNN architecture is explained in detail in section 3.

2.5. Understanding Inception Model

Inception model is the breakthrough in the era of deep learning called as Deep Convolutional Neural Network, which was considered as the state of art classification and visual recognition model in ImageNet. The main idea behind the inception model is to improve the utilization of computing resources inside the neural network. This model is used to increase the depth of the feature by keeping computational cost constant. Moreover, it provides parallel computing branch of convolution layers for same input and concatenates the output of all parallel layers before passing to the next layer in the architecture.

The basic inception model consists of 4 parallel layers. First parallel layer has 1x1 convolution. Second parallel layer has 1x1 convolution followed by 3x3 convolution. Third parallel layer has 1x1 convolution followed by 5x5 convolution. Fourth parallel layer has 1x1 convolution followed by max pool layer. The final layer concatenates all outputs of parallel layers, before feeding it to the next layer. Inception models are explained in detail in Section 3.

2.6. Pre-Trained Model

Pre-trained model contains already trained weights for a specific neural network. One of the pre-trained models is ImageNet, which has been trained over 1.3 million images for object detection and recognition task. Normally researcher removes the final classification layers or fully-connected layers of those pre-trained model and replaces them with SVM or KNN classification layer for their research.

Donahue and Jia [13] investigated that to generalize the ImageNet model which can be further used for other datasets at various depths. They did this by visualizing the separation between different categories in the first and sixth layer, showing greater separation in the deeper layers. They have shown that eight-layer network having three fully-connected layers are most expensive when it comes to computational time. However, by tuning the pre-trained network separately for fine-grained bird classification, domain adaptation, or scene labeling, it out-performed the state-of-art models in these categories. Similarly, Oquab's [14] experiment outperformed the state-of-art object detection model when the last classification layer was replaced by Rectified Linear Unit (ReLU) and Softmax for the VOC07 and VOC12 dataset.

In addition to that, FaceNet has achieved success in face recognition by extracting embedding feature from inception model and having the SVM classifier at final layer. In one of our experiment, FaceNet 's pre-trained model has been used to extract embedding features from our dataset to train the SVM model in the final classification layer.

Generally, the pre-trained model comes in a single protobuf file with the meta, checkpoint and graph files. Following are the files that are present in a pre-trained model.

- model.meta
- model.index
- checkpoints
- model data
- model.pbtxt

Following steps have to be performed, while loading the model into tensorflow session before starting the training process.

- The foundation of computation in TensorFlow is the Graph object being loaded first into tensorflow session or any deep neural network training

- Default session of Tensorflow holds a network of nodes, their associated trained weight, operational nodes like softmax, addition, multiplication and these are connected to each other
- Graph object is created, it can be accessed through “as_graph_def()”, which returns a GraphDef object from tensorflow session
- Also, the input placeholders, operations, and variables can be accessed from tensorflow graph object which is present in tensorflow session
- These graph objects are used to run the model

2.7. Understanding SVM

Support vector machines (SVM) is the supervised learning methodology in machine learning field which analyzes the data used for the classification and regression task. SVM is based on finding the best possible hyperplane that gives the largest distance to separate the training class labels.

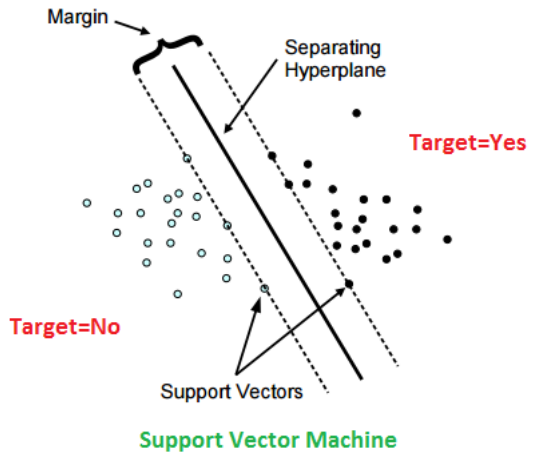


Figure 1: SVM model for linearly separable data

SVM training algorithm builds a model that based on categorical separation, making it a non-probabilistic binary linear classifier. Support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outlier’s detection.

Linearly Separable Data

For an example, let's consider the data which is linearly separable. Imagine a training set of $\{x_i, d_i\}$, where i=0 consisting of an input pattern x_i, for the i^{th} row, and d_i is the desired output to the corresponding class label. A hyperplane separating this training set is described by

$$w^T = x_i + \mathrm{b} = 0$$

where w is an adjustable weight vector and b is the bias term. This means that classes belonging to $d_i = 1$ are described by

$$w^T = x_i + b \geq 0$$

This method known as perceptron will provide a solution which by no means guaranteed to be optimal. Different perceptron might come up with different solutions that would maximize the margin for each class labels.

2.8. Theory

Convolutional Neural Networks (CNN), were first introduced by Yann LeCun's in 1998 for Optical Character Recognition (OCR), where they have shown impressive performance on character recognition. CNN is not just used for image related tasks, they are also commonly used for signals and language recognition, audio spectrograms, video, and volumetric images. Figure 2 shows the high-level block diagram of CNN.

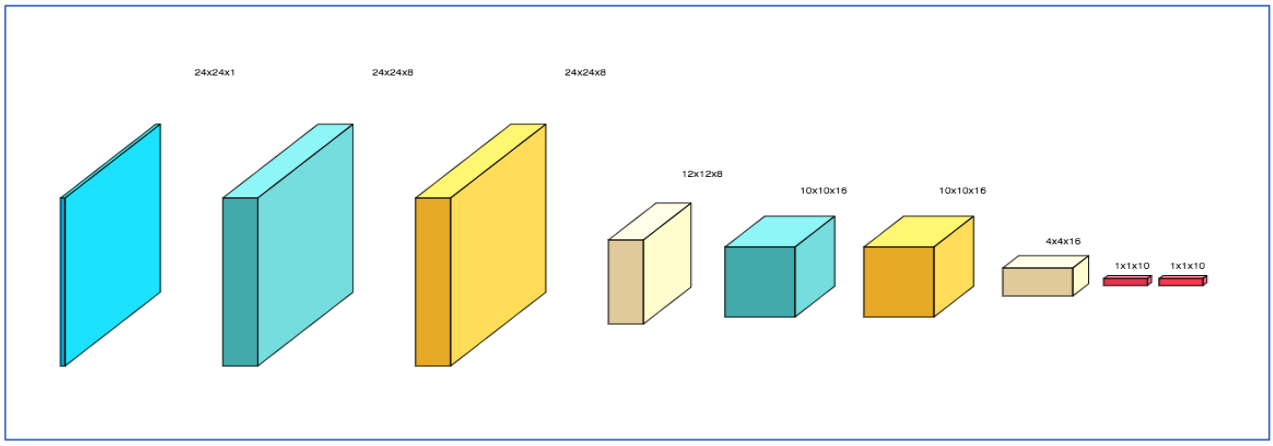


Figure 2: Convolutional neural network architecture block diagram

Above diagram is an overview of a CNN, where the first layer is an input layer, followed by blue color blocks which are convolution layer. The yellow color blocks are the activation layer, and the white color blocks are

the maxpool layers. Finally, the red colored blocks are the fully connected layers in the CNN architecture. Above architecture is a basic model for the initial understanding of CNN. We have explained the CNN in detail in section 3 [3.1.2].

These was a breakthrough in deep learning area when researchers designed Inception model in 2014. According to the paper [3] “Going deeper with convolutions” [3], an inception layer was introduced to the existing CNN model, which has set a benchmark in state of art classification and detection task on ImageNet dataset. The main functionality of this inception layer is to improve the utilization of computing resources inside the neural network. Figure 3, shows the basic state of the art inception layer block diagram.

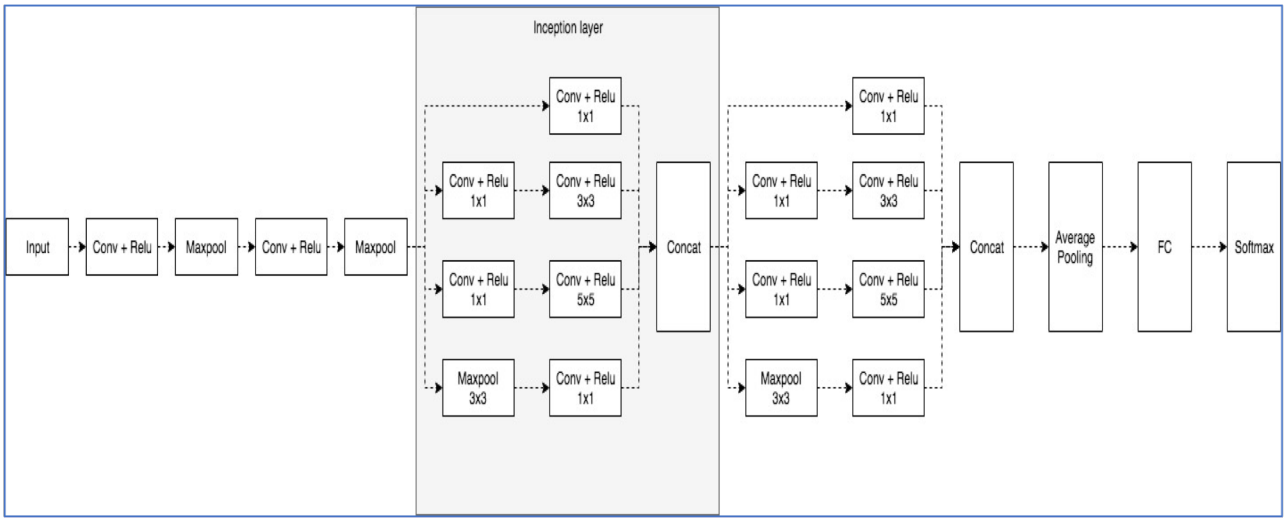


Figure 3: Inception model architecture block diagram

In above diagram the gray color block shows a basic inception layer. The only difference between CNN and Inception model is inception layer, which has 4 parallel connected convolution and max pool layers with the concatenate layer at the end. The architecture of Inception layer model is explained in detail in section 3 [3.1.3].

So far, we have got an overview on CNN and Inception model architecture and what does the deep learning model mean. The common terminologies used in deep learning architectures has explained in the following sections.

2.9. Convolutional Layer

Convolution Layer provides a convolution operation, in which a 2-D or 3-D filter of appropriate size sweeps over an image and apply the filters to each depth of an image. The convolutional layers are restricted version of the Multi-Layer Perceptron (MLP) adapted to take a 2D / 3D inputs instead of 1D. The idea behind

convolutional layers is to detect elementary features such as edges, corners, and endpoints, and combine them using multiple layers to get high-level features that might describe an object completely. Figure 4, shows a sample convolution of an image on 3x3 filter size.

This operation performed in the following manner: first layer f1(·) contains the most elementary features, the second layer f2(f1(·)) is a function of the elementary features in the previous layer. The third layer is then a function f3(f2(f1(·))) of the features in the second layer and so on.

Moreover, this architecture is designed for high-level features extraction from an image at any given layer to describe an object like face, chair, or a car. In addition to this, convolution also provides an important and valuable feature attribute called shift invariance. That is, if the input to the first layer is shifted, then the output of the first layer is also shifted by the same amount. Convolution has 2 main parameters which can change the behavior of convolution, like stride and padding. Padding and Stride is explained in the later sections.

Following figure shows the convolution over an 8x8 image size with STRIDE of 1 and PADDING of 0 (SAME) and kernel size of 3x3.

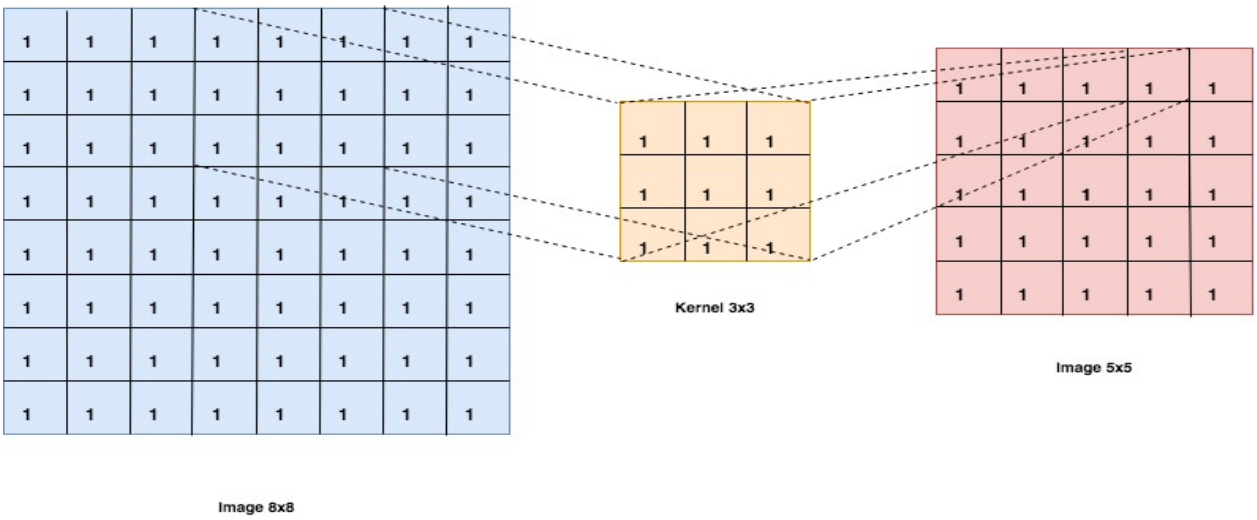


Figure 4: Convolution of a gray scale image on 3x3 kernel size

Output of the convolution layer is calculated as per the following formula.

$$W_{new} = \frac{(W_{old} - F_{width} + 2P)}{S} + 1$$

F_{width}: Filter or Kernel size as in width and height parameter while using respective formula.
P: Padding
S: Stride window size for convolution
W_{new}: New width of the output image
W_{old}: Old width of the input image

2.10. Pooling Layer

Pooling is a method of reducing the feature size in width and height of an input. The pooling operation sweeps a rectangular window over the input feature and computes a size reduction operation for each window (average, max, or max with arg max). Each pooling operation uses rectangular windows of size k, separated by offset strides. For example, if strides are all ones every window is used, if strides are all twos every alternative window is used in each dimension. a simple way of reducing the precision for the position from where distinctive features are located in the feature map. Since the exact position of the feature is irrelevant, only its position in relation to the other features is of importance, especially for classification tasks.

In other words, we do not care where an edge or a corner is located in the image, we only care about its position relative the other corners and edges in the image. The size of the receptive field is critical, most commonly used is 2×2, this is due to the significant information loss that occurs when using the larger receptive field. For an instance, if a 2×2 receptive field is used four pixels are turned into one. Increasing the receptive field to 3×3 would mean nine pixels are turned into one and if 4×4 is used, 16 pixels are turned into one and so forth.

To put it into perspective, imagine you have a 12 × 12 image, giving a total of 144 pixels. A receptive field of 2×2 would leave you a 6×6 image with 36 pixels. You have now lost 75% of the information, but the object might still be recognizable. If instead, you use 3×3 you only get an image of 4×4 with 16 pixels leaving only 11% of the original information.

Now it might be possible to recognize the original shape, but if you are unlucky you have lost too much information. If you decide to push it even further by using 4×4 pooling will get you a 3×3 image with a total of nine pixels, you have now lost 94% of all the information in the image. Figure 5, shows a max pool operation on the 2x2 filter with 2x2 stride.

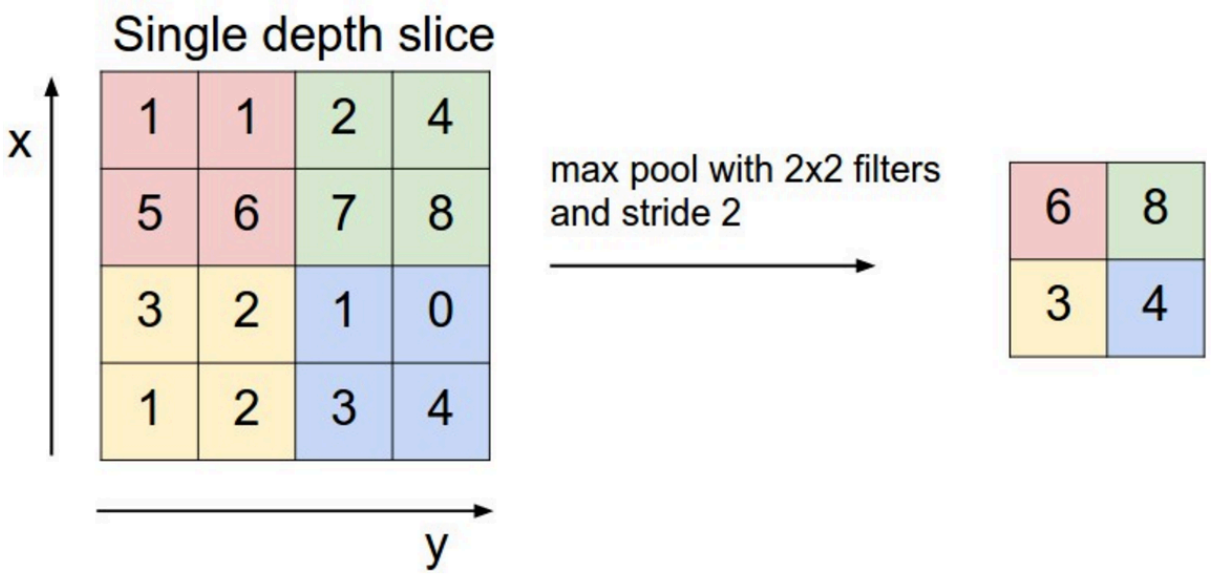


Figure 5: Maxpool layer operation [google]

Pooling layer output is calculated as per the following formula.

$$W_{new} = \frac{(W_{old} - F)}{S} + 1$$

W_{new}: New Width for the output image

W_{old}: Input image width

F: Filter Width size

S: Stride size

This formula is used for the output image width calculation, and same can be used to calculate the resulting height of an output image from the pooling layer by changing width parameter with the height parameter.

2.11. Fully Connected Layer

Fully connected layer is the called as dense layer, where each neuron in one layer is connected to each and every neuron in the following layer. This principle is same as the traditional multi-layer perceptron neural network model and how it works. Fully-connected layers refer to be the final layers in the full CNN model. Fully-connected layers operate as a Multi-Layer Perceptron (MLP) with normally either two or three hidden layers and one classification layer. The properties of the MLP make it a superb function approximation, with only two hidden layers it can approximate any function assuming it has enough hidden neurons. Normally, the number of neurons in the hidden layers is constant, with 4096 being a common number for deep networks with large input images.

The inputs to the first hidden layer originate from all neurons in the previous layer (either a pooling or convolutional layer). In other words, each neuron in the previous layer is connected to each and every neuron in the first hidden layer. Outputs from the first hidden layer are connected to each and every neuron in the second hidden layer, it is fully-connected. Outputs from the last hidden layer are then fully- connected to the final classification layer. The size of the final layer depends on the number of classes used to train the neural network.

The fully connected layer can be flattened and connect to the output layer and so on it get reduced in size for the classification of the images. At the fully connected layer, if the input is coming from convolution layer or max pool layer of size XxYxZ size, we can choose how many nodes do we need in the fully connected flattened layer. It could be the XxYxZ number of nodes or (X*Y*Z)/2 number of nodes and then the output will be reduced feed to the output classification layer.

2.12. Activation Function

The activation function is really important to the deep neural network, which is complicated and complex. They bring non-linearity property to neural networks. The main property of an activation functions is to convert an input signal to output signal. This is used in every node of the deep neural network for abstraction representation of action potential firing the node.

If we don't use the activation function, the output mapping function will be, by default a linear function, which linearity is less effective towards learning of complex function boundaries of the input data. Following are some of the activation functions explained in detail.

2.12.1. ReLU

The activation function plays a central, and very important role in how CNN work and how well they perform. Every neuron in every convolutional and fully- connected layer has a specific non-linear activation function. The type of non-linear function varies, but most networks nowadays are using ReLUs (Rectified Linear Units) mathematically described by $\phi(x) = \max(0, x)$.

It got very popular in recent years over the tanh because its convergence capability surpasses the tanh function by 6 times. It addresses the and rectifies the vanishing gradient problem. I have used the ReLU in all activation function in all the hidden layer. We have capabilities to change the activation function to any other function at any point in time.

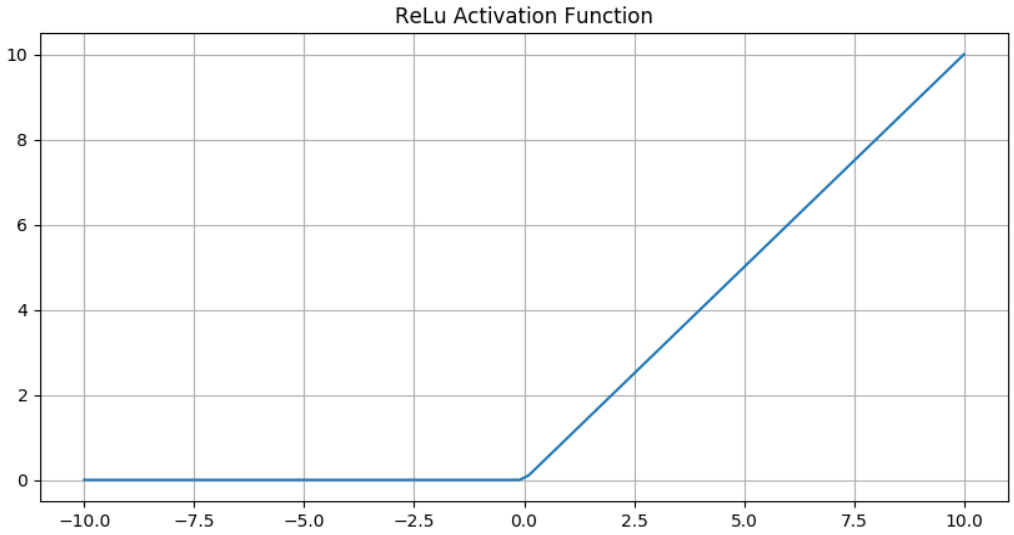


Figure 6: Relu activation function

2.12.2. Sigmoid

A sigmoid function is used as activation function is a special case of the logistic function shown in the first 7 this includes the logistic and hyperbolic tangent functions. Sigmoid curves are also common in statistics as cumulative distribution functions (which go from 0 to 1).

It has major reason why this is fall out of the popularity,

- Vanishing Gradient problem
- Output is not zero centered
- Sigmoid saturates and kills the gradient

- Sigmoid has slow convergence

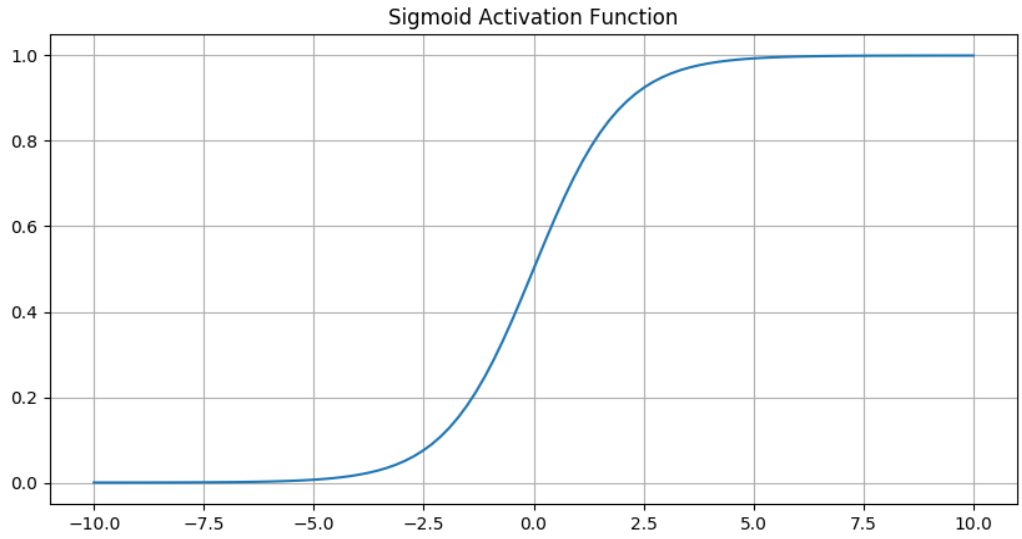


Figure 7: Sigmoid activation function

2.12.3. Softmax

This softmax function also known as normalized exponential function which is used as the activation function in the neural network. This is used as a probability distribution for *K* possible outcomes. This is also used in the final layer with cross entropy in the neural network for the probability distribution for categorical data. Softmax gives the output as 0-1.

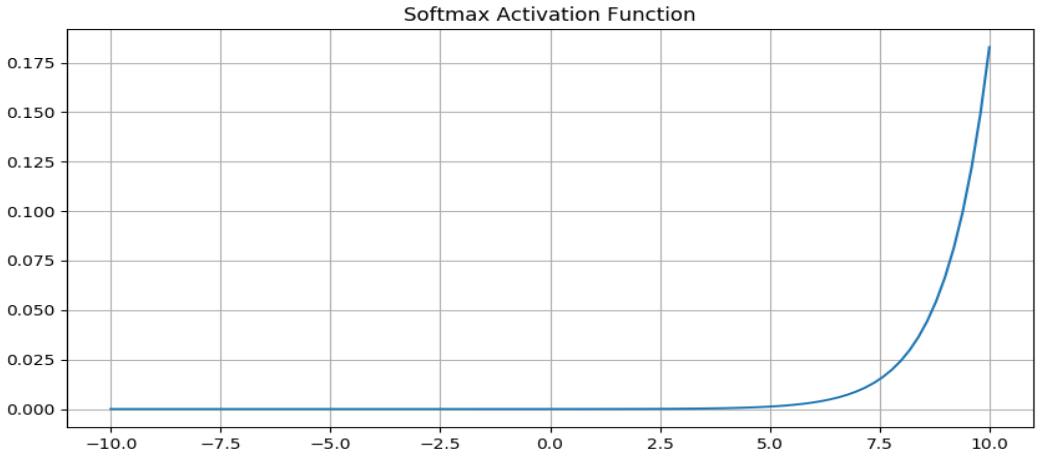


Figure 8: Softmax activation function

2.12.4. TanH

Tanh function is another activation function of kind Logistic sigmoid, which has "s"- shaped curve, but outputs values are range from -1 to 1. This function makes the negative inputs to map strongly to the negative outputs. Additionally, only zero-valued inputs are mapped to nearest non-zero outputs. These properties make the network less likely to get "stuck" during training and eradicate the possibility of vanishing gradient. Calculating the gradient for the tanh function also uses the quotient rule.

The output is zero centered because it falls between -1 to 1. Optimization is easier than the sigmoid but still, it has the vanishing gradient problem.

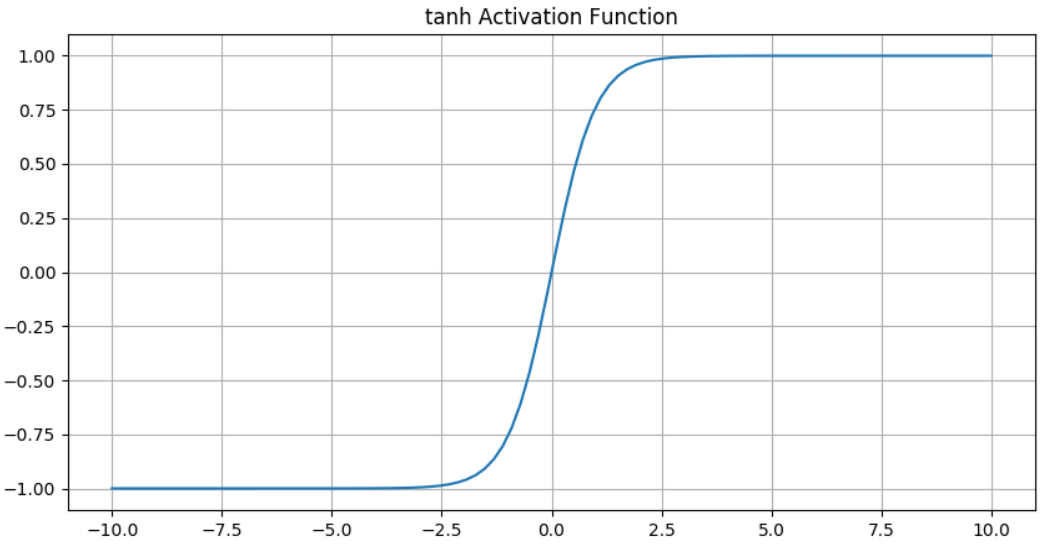


Figure 9: Tanh activation function

2.12.5. Softplus

A smooth approximation to the rectifier linear function is Softplus function: f(x) = ln(1+ex). Both the ReLU and Softplus are largely similar, except near 0 where the softplus is enticingly smooth and differentiable. It's much easier and efficient to calculate derivatives of ReLU than the softplus function which has log(.) and exp(.) in its calculation. Interestingly, the derivative of the softplus function is the logistic function.

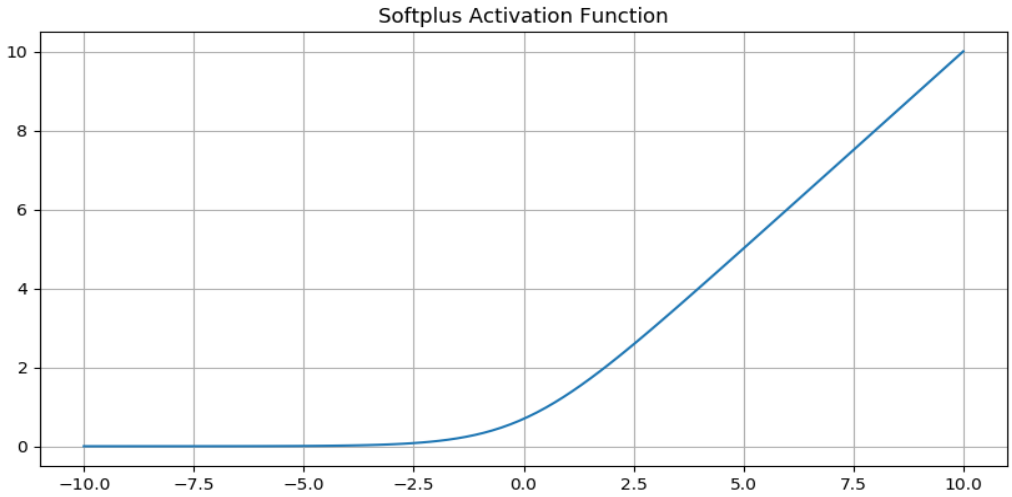


Figure 10: Softplus activation function

2.12.6. Relu6

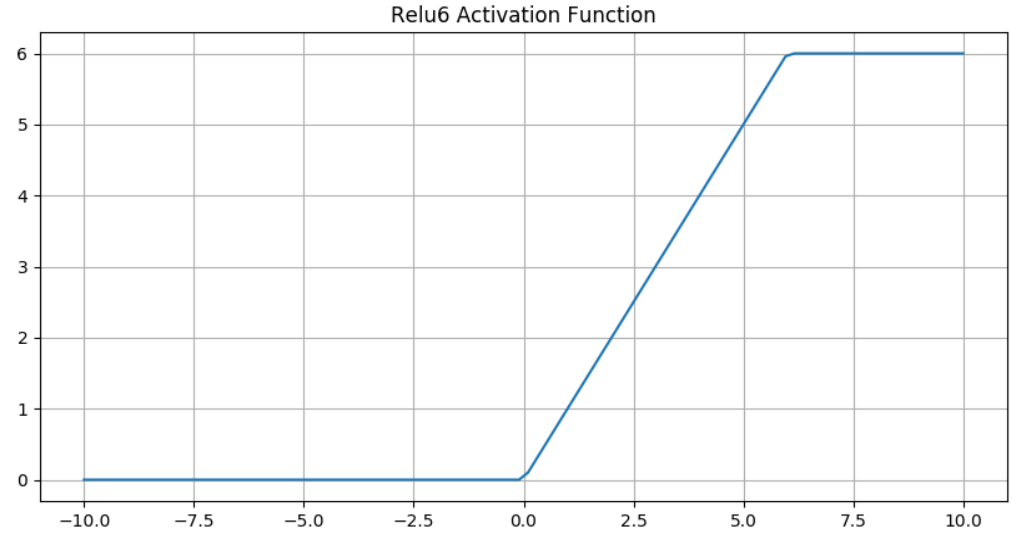


Figure 11: Relu6 activation function

2.13. Elementary Feature

Elementary features are the simplest of shapes, such as edges and corners in an image detected by some feature detectors. The Sobel operator is the simplest example of the feature detector. It approximates the derivative of a pixel value. This type of detector is basic of CNN, there are lots of feature detectors were there, which are responsible for finding elementary features in any layer of the neural network. Elementary

features in one part of the image are used in another part of the image and because of this one feature map has identical Weight Vector (Section 2.8.6) even if their local receptive fields are in the different region of the image. Every unit as in neuron in each feature map has the exact the same weight vector. And this concept is helpful in terms of following in detail.

- Generalization ability
- Reduction in training time
- It allows for parallelism

2.14. Weight Vector

Weight vectors are the filters in the Convolutional Neural Network and help in the convolution process and find out the feature map through feature extraction operation. Initially, the weight matrix is created by the Gaussian distribution of numbers which has 0 mean and 0.001 standard deviations. This initialization also could be achieved by setting all weights to a random number or zeros in all layers of the neural network. This random initialization affects the convergence of loss function, which will take a lot of time to reach local minima or global minima.

In addition to that weights are shared between feature maps. If the two neurons have same feature maps, then they share the same weight vector to reduce the space and computation complexity. Generally, the researcher uses the random uniform weight vector for its normal distribution feature.

2.15. Filters or Kernels

Kernels and filters are used interchangeably in our document. The kernel is an of 2-D or 3-D matrix, which is used in convolution process as described earlier in this section. Each convolution gives rise to a number of the channel as defined by the number of filters. Kernels are generally referred as the feature detectors used in convolution layer.

Kernels are defined as 3x3, 5x5, 7x7 and so forth. This means the height and width of the kernel. Smaller filters or kernels capture minute detail features in an image than the bigger kernels which leave out some important features in the convolution or in max pool layer. So, researcher prefers to use the smaller kernels rather than bigger kernels like 11x11 or 19x19. The kernel is defined in the tensorflow as [1,3,3,1] vector format, this means, [batch, height, width, depth] respectively. Height and width are defined for the kernel size of 3x3 or 5x5. Moreover, the batch size is defined as per the training requirement and the depth is defined by input image depth.

2.16. Strides

Stride is a concept, which controls the movement the kernel over an image in convolution and max pool operation. By Default, the kernel moves over the image by shifting one position at a time in horizontally or vertically. Starts at (0,0) position of the image and if the stride is 1x1 then it will move 1 in both the direction, horizontally or vertically. Stride is defined as [1,2,2,1], that means, each element in the array is defined as [batch, shift in vertical direction, shift in horizontal direction, channels] respectively.

For an example

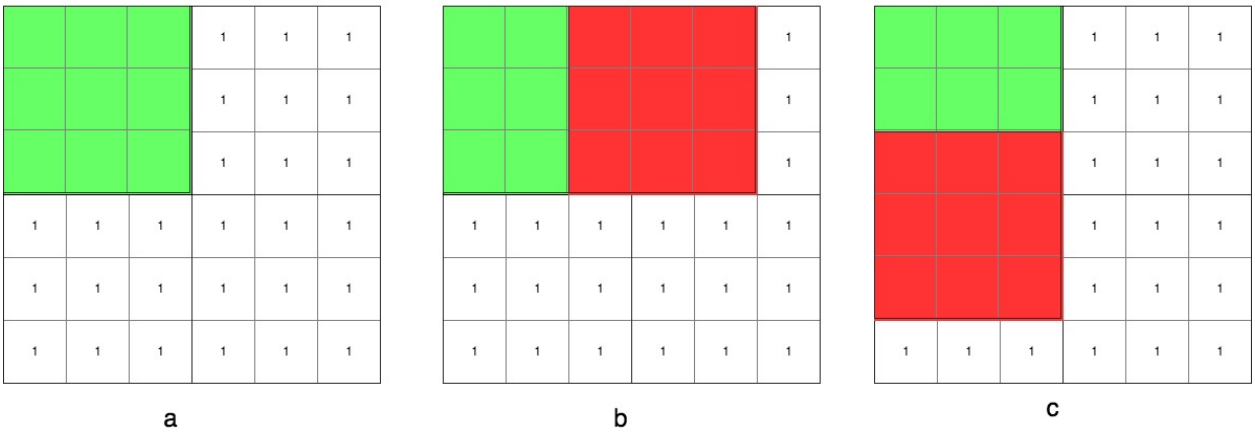


Figure 12: Stride of 2x2 is used for kernel in the convolution process on 6x6 image size

In above convolution, the stride is used as 2x2, in the figure 13b, the kernel moved by 2 positions as shown in red color in horizontal direction and in figure 13c, the kernel moved by 2 positions in vertical direction.

2.17. Padding

Padding the input image is 3D or 2D with zeros, such that the convolution layer does not alter the spatial dimensions of the input image. With the zero padding while convolution controls the spatial size of the output image from convolution layer.

Padding can be calculated as the P = (F-1)/2.
F: Filter size

For an Example convolution without padding or padding = 0

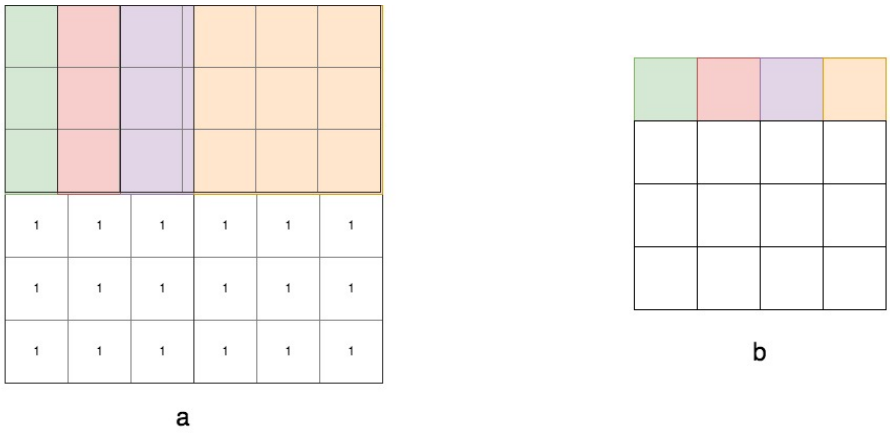


Figure 13: Zero Padding convolution on 6x6 image which reduces the spatial dimension

In figure 13a, convolution is happening without padding, which results in reduction in the spatial dimension. Because of which, most of the convolution process loose information in the image. So padding is added in the convolution process to avoid this issue.

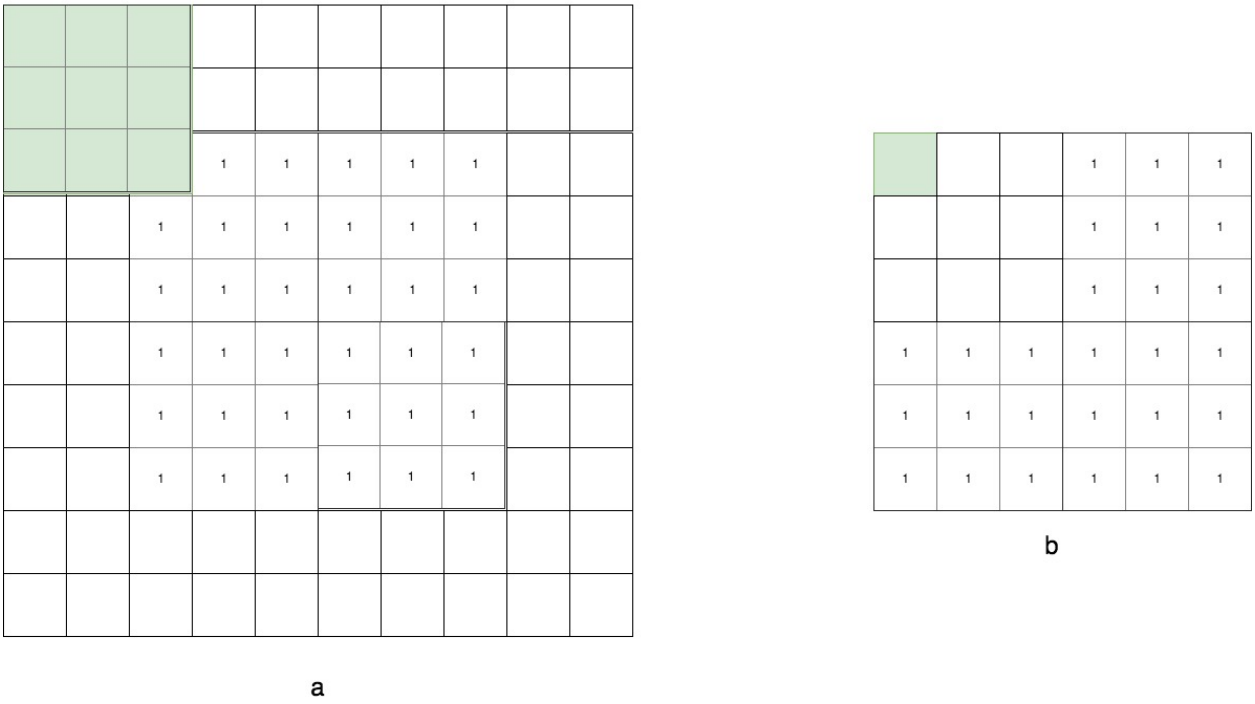


Figure 14: Padding of 2 pixels before convolution on 6x6 image to maintain the spatial dimension

In figure 14a, before the convolution process, padding is added around the image of size 2, which help to keep the spatial dimension of the image after the convolution.

2.18. Dropout

Deep neural networks with a large number of nodes and hyper-parameters are making this model very powerful in machine learning algorithms. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem.

To reduce overfitting, we will apply dropout before the readout layer. We create a placeholder for the probability that a neuron's output is kept during the dropout, which allows us to turn off and on the dropout during training and testing.

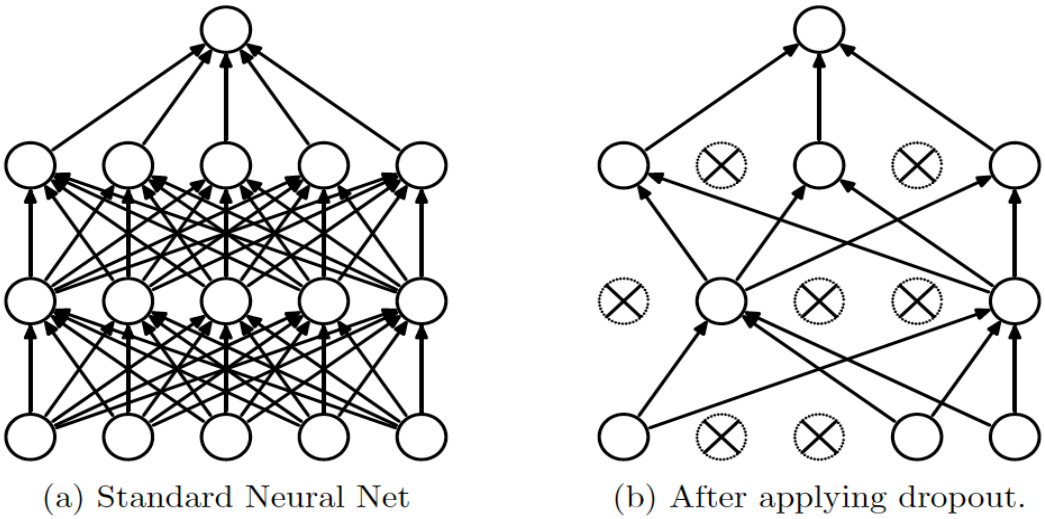


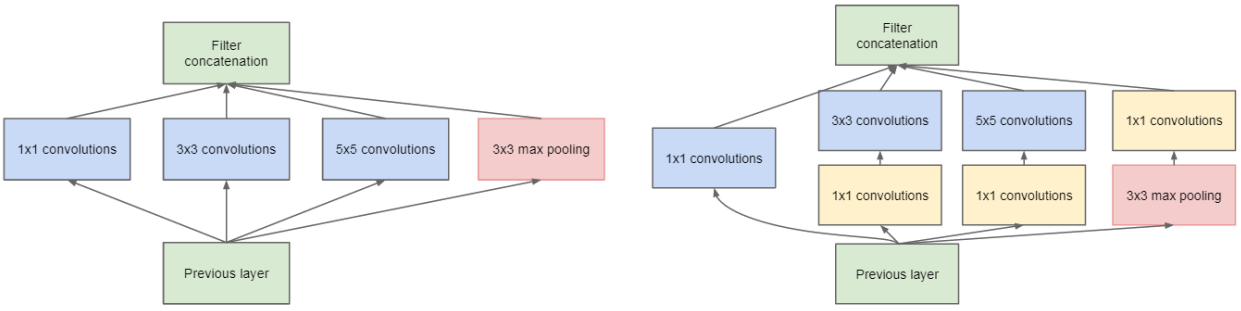
Figure 15: Dropout operation block diagram [google]

2.19. Inception Layer

This architecture has been developed to make the computation faster with dimensionality reduction feature in the neural network design. This Inception model is a breakthrough happened in the ImageNet paper [3]. This model has multiple convolutions with multiple filters and pooling layers in parallel within the same layer called inception layer. In this architecture shown in the figure 13-a is a basic inception model with parallel layers, employs convolution with 1x1 filters as well as 3x3 and 5x5 filters and a max pooling layer. Figure 13-b demonstrates a dimensionality reduction in feature map and explains the use of 1x1 convolution filters can help in dimensionality reduction (since no. of channels is reduced).

The intention behind this type of architecture is to enable the deep neural network to learn deep and find the best possible weights while training the model. This model automatically selects the useful features and

concatenate them all, before passes it to next layer. Additionally, it is intended to reduce the number of dimensions of feature map, so that the number of units and layers can be increased at the later stage [3].



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 16: Inception naive and inception dimension reduction block diagram [3]

2.20. Normalization

Generally, normalization of the data helps in speeding up the training process. If we didn't normalize the input data, the data distributions range of feature values would likely be different for each input. Normalization is important in terms of classification to avoid the overfitting of data. We normalize the data from -127 to 127-pixel values to make it uniform across the all the gray scale images before starting the training process for depth 1. For depth 3 (RGB) image training, the normalization process goes through the process of subtracting the mean pixel value from image and dividing the standard deviation of mean face to bring the normal distribution to the image pixels.

2.21. Regularization

The goal of a deep learning or machine learning algorithm is to ignore the noise. In the noisy data, algorithms will over fit while trying to fit those noise in addition to features or patterns. Since the noise is stochastic and does not generalize for unseen data leads to low training error but the high testing error.

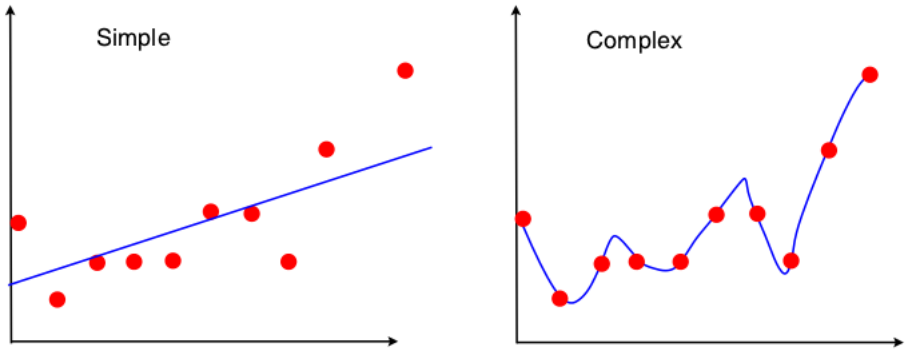


Figure 17: Regularization function fit graph [google]

In the figure above, first, the straight line tries to fit most possible points however the second graph shows all the points fit a k degree polynomial. In the right side, this fit the noise, so by penalizing the higher degree polynomial so that it reduces error significantly as compared to the simpler at left. Then we call it regularization. To avoid overfitting and max reach of weight we have to perform the regularization to keep the weights under some threshold limit.

2.22. Optimization

In deep learning optimization involved in many contexts. Sometimes analytical optimization does help in the writing of proofs. The most optimization problem in deep learning is neural network training. Optimization is a process of finding a best optimizing parameter for which that the neural network cost function reduces significantly. There are many optimizers are available like Adam Optimizer or Gradient Descent to make the loss function optimized, when there is a good feature found and reaches its optimum point and global minima. If it reaches to any kind of local minima, it's very difficult to overcome those local minima. Some of the optimizers are described as follows.

2.22.1. Adam Optimizer

The Adam optimizer uses Kingma and Ba's Adam algorithm to control the learning rate. Adam offers several advantages over the simple Gradient Descent Optimizer. Foremost is that it uses moving averages of the parameters (momentum).

This enables the optimizer to use a larger effective step size, and the algorithm will converge to this step size without fine-tuning. this requires more computation to be performed for each parameter in each training step (to maintain the moving averages and variance and calculate the scaled gradient), and more states to be retained for each parameter.

2.22.2. Gradient Descent Optimizer

A simple Gradient Descent Optimizer could equally be used in your MLP but would require more hyperparameter tuning before it would converge as quickly. Other Optimizer can be used to optimize the cost function are

- Adadelta Optimizer
- Adagrad DAO ptimizer
- Adagrad Optimizer
- Gradient Descent Optimizer
- Adam Optimizer

2.23. Loss Function

The loss function is an important part in deep learning neural networks, sometimes loss function cares about the error in classification but it is not optimized, which is used to measure the inconsistency between the predicted value and actual label. It is a non-negative value comes out of final classification layer, where the robustness and accuracy of deep learning model increase along with the decrease in the value of loss function. The loss function is an empirical risk function, which is consist of empirical risk term and a regularization term.

- Mean Square Error
- Mean Square Logarithmic Error
- L2
- Mean Absolute Error
- Cross Entropy Error
- Hinge Error

2.24. Early Stopping

Early stopping is a form of regularization used to avoid overfitting while training a model in iteratively such as gradient descent. Early stopping is calculated to open the validation sets, and the loss incurred by the network at each iteration. When the neural network observes the loss is increasing from a certain limit at low and try to stop the model training and use the lowest loss model for testing. This method also provides a way to stop the model when it starts overfitting.

2.25. One hot Encoding

One-hot encoded vectors are high-dimensional and sparse. If we have 2000 classes to train and in the final layer, a vector containing 2000 integers and 1999 are zero and as the network is trained with 2000 classes then each class will possess 2000-dimensional vector for each output node and it is very computationally expensive when it comes to the big dataset.

2.26. Embedding

Instead of using one hot encoding, we can specify the size of embedding matrix to keep how many embedding values do we need to classify one class label for the classification purpose. Embedding values are best-chosen values for each class label and, for the future unknown input, images can be classified by measuring the distance from each class embedding values to the new image embedding values.

Embedding is generally a vector representation of an image in specific feature dimension space. If we define an embedding vector of 128-dimension space. Then after several convolutions and max pool and when it the comes to fully connected layer it will flatten the image matrix to the feature vector and convert it to 128 sizes and this can be used to represent the input image in 128-dimensional space.

2.27. Batch Processing

Batch processing is a technique to process multiple numbers if images simultaneously in training. Most of the CNN and other deep learning network architectures supports batch processing through tensorflow. But it comes at a cost. Batch processing needs computing power to process and do mathematical computation at once on huge size of matrixes. For an example, if you have 120x120 image and you have selected batch size as 100 then the Deep Learning Networks will process the input as 100x120x120 simultaneously. For that much of the memory and GPU units has to be placed to handle that number of mathematical operation.

It is very efficient in calculating and updating the weight vectors. In backpropagation, weights associated with each node get updated once in a batch rather than updating for every image as we process in stochastic gradient descent algorithm.

Chapter 3: Implementation

3. Implementation

Several experiments have been performed on face detection and recognition systems using deep learning architectures. We have trained various deep learning architectures and implemented a system from the scratch on deep learning model training, evaluation, testing, GUI modules for the application for real-life testing. Following are the deep learning models have been considered for the face recognition system. Followed by a detail explanation of our system.

- Convolutional Neural Network
- Inception 1b Model
- Inception 5b Model
- Embedding SVM Model
- FaceNet Out of the box functionality

3.1. Face Detection and Recognition System Experiment

This thesis concentration is on face detection and recognition task. This system primarily comprises of two components, face detection component and face recognition component. Face detection component is used to detect the faces in a video frame. DLIB and MTCNN face detectors are used in our application. Face recognition component, for recognizing the faces those are already detected by the existing library as mentioned earlier. For this component, FaceNet out of the box application features have been used along with the newly trained deep learning models like CNN based classifier and Inception model classifiers for face recognition. Following sections contains details explanations of our system.

Our task consisted of detecting and recognizing moving objects, specifically faces in the real world. This also includes detecting and classifying moving objects in images using a camera, video and images. Moving object detection using CNN is a highly complex and computational challenge and solving such advanced problems requires 3 fundamental resources. First of all, hardware that will help us in performing computationally challenging tasks. This means, a faster CPU and GPU (Graphical Processing Unit, e.g. Nvidia GT 1080T) along with large amount of RAM (Random Access Memory). Our complete set-up of GPU cluster is described in section 3.2. Secondly, software which would have a faster implementation, both in terms of writing code as well as execution time, thus enabling us in training deep learning models. The software we used for this project is explained in section 3.3. Lastly, data, which is a primary focus in any

deep learning model training task. The data that are used for training and testing of deep neural network is explained in section 3.5.

3.2. Hardware

It used to be very difficult for the researchers to train complex deep learning architectures earlier. However, the evolution of GPUs and faster CPUs have enabled researchers to concentrate more in the area of deep learning research. The training of various deep learning architectures with huge datasets requires intense processing power, to satisfy that requirement a GPU cluster of 4 NVidia GX Titan 11 GB GPU, overall 45 GB of GPU memory has set up for training.

All training and testing were conducted on the Unix machine with "NVidia GX Titan" 45 GB cluster in the Computer Science department at UTA. It has 4 GPUs allowing for extreme parallelization using multiple cores. The main GPU is an NVidia GX Titan with 11GB RAM that allows us for preparing batch of images at once. It can perform 1.4 TeraFLOPS (FLoating-points Operations Per Second) split over 2,880 CUDA cores with a bandwidth of 288 GB/s. This high-end hardware setup allowed for rapid training and testing of very large networks with thousands of images. Experiments that otherwise would have taken days or weeks or even months could not be performed in a few hours or less. The hardware, therefore, played a crucial role in conducting multiple experiments during the time of the thesis.

3.3. Software

Software also played a crucial role in this face detection and recognition system for training of various deep learning architecture. All the detection & recognition APIs and architectures is explained in the following sections. Software are considered because of its computational power, easy to use and of course it's open source category. Software that have been used is explained in table 1.

Table 1: Technology stack used in our system

Category	Name	Description
Operative system	Ubuntu 14.0	Operating System used in Ubuntu 14.0 version
Programing language	Python 3.6	
	Anaconda, Conda	
Deep Learning Library	Tensorflow 1.4, 1.5, 1.6	Deep Learning library
	Keras	Deep Learning library and tensorflow as backend

Math Library	Numpy	Numerical Library for python
	Scipy	Numerical Library for python
	Scikit-learn	Scientific Library for python. It has lot of other functionality as in statistical modeling and ML libraries.
Pre-Processing Tools		
GUI Tools	QtPy	Interface Library
Image Tools	ImageIO	
Others	pickle	Pickle file preparation library and it has lots of other functionality too.
	Dlib	Face Detection library
	MTCNN	Face Detection library

3.4. System Architecture

Face detection and recognition system have two major components

- GUI component
- Training component

Both the component's architecture is explained in the following sections

3.4.1. GUI component

GUI component is consisting of multiple modules such as GUI module, Commuter, Context. Each module is designed in a modular fashion by their processing logic. Context module initialization happened at the start of our application and stays till the end. This module possesses other modules instances like data loader, pre-processing, post-processing, recognizer, detector module etc. Moreover, with the help of this module, commuter module creates a process flow pipeline and prepares are communication channel between the modules. Below system block diagram will explain the connectivity between the modules and their dependencies.

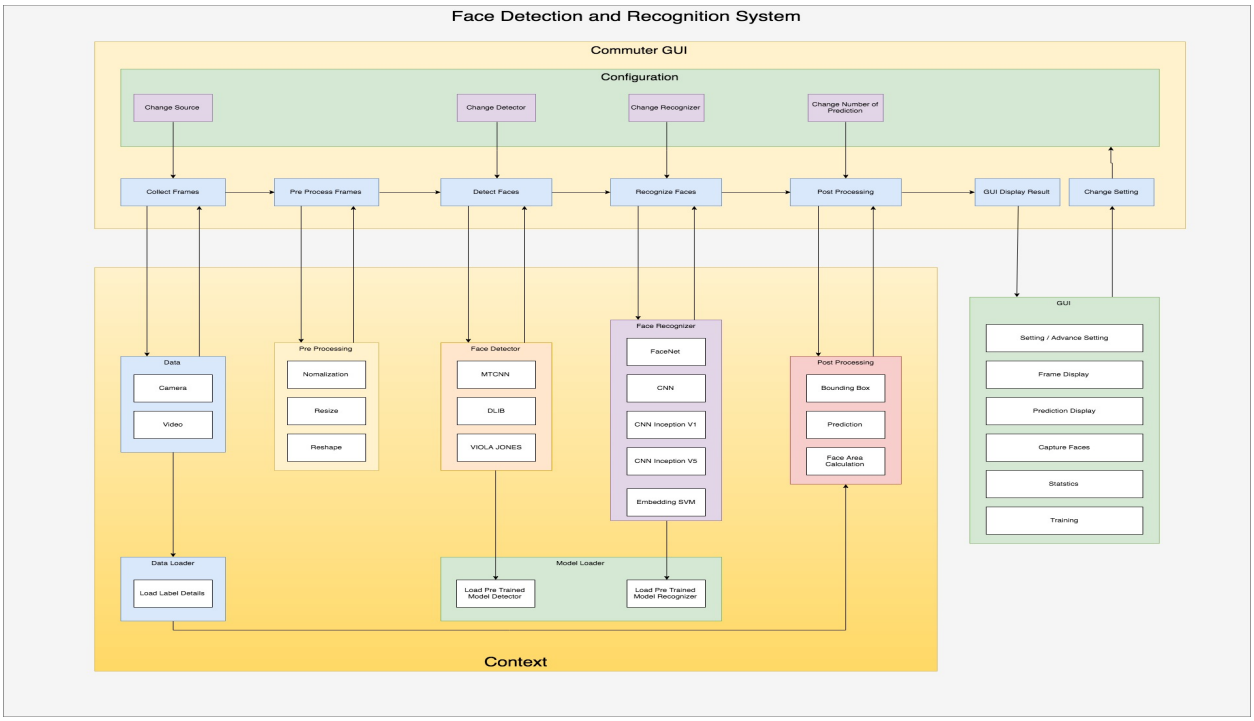


Figure 18: Face detection and recognition GUI system process flow diagram

All modules in the above block diagram are independent to each other, and GUI Commuter module creates a process flow for those independent modules. The GUI Commuter modules and other functionalities are explained in the section starting from 3.6.

3.4.2. Training component

The training component is as like the GUI component; however, the main difference is that the training component does not have instances of some of the modules like the detector, GUI and recognizer as explained in the previous GUI commuter process flow diagram. Each module related to training component are explained in section 3.17.

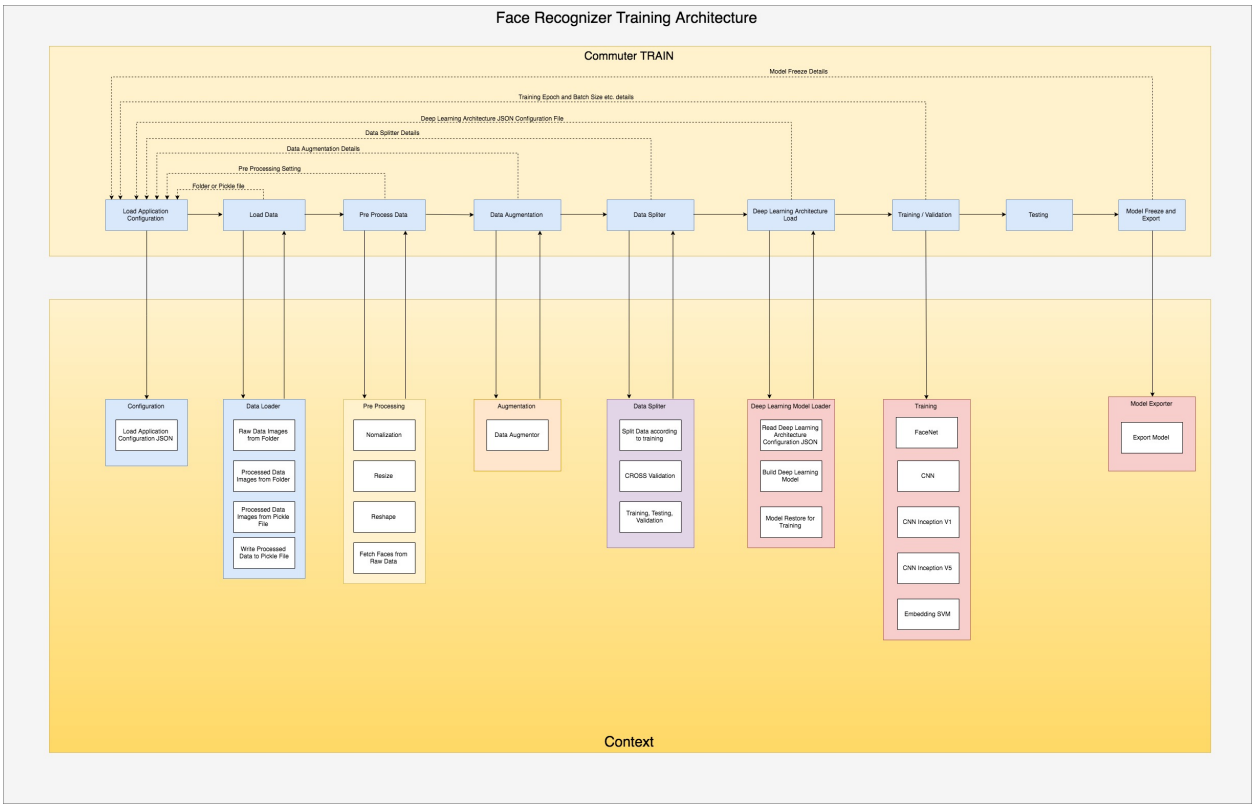


Figure 19: Face detection and recognition training system process flow diagram

Figure 19, shows the training module block diagram. It consists of a training commuter module which creates a process flow for training and a training context module which possesses the instances of supporting modules related to training process. All modules displayed in the above block diagram is explained in the following sections.

3.5. Data

The most important aspect of any deep learning model training is data. Without that the training may not be successful or the model would not be trained properly, or the loss function may not converge to a local or global minimum. LFW face dataset is used for the training process with ~ 5000 face images. Those images have captured in different lighting condition, different angle of rotation of face and with various facial pose.

3.6. Modules

There are several components are designed in a modular fashion depending on their behavior and responsibility. Each module is designed by keeping the scalability of our application too. This whole system of face detection and recognition is going to be an open source software package and will be available on

GitHub for further development and contribution. Following section contains details explanation on each of the modules and their functionalities.

3.7. Commuter

Commuter is the core module, purpose of this module is to create different process flow and inter-module communication channel such as training and user interface for real life face recognition task. Following are the two types of commuter have been developed.

- Training Commuter
- GUI Commuter

3.7.1. Training Commuter

Training commuter is a placeholder for all the training related tasks and this is designed to handle set of instruction specific to the training pipeline. This pipeline creates a process flow for training in which all the modules related for training is connected to each other. For an example, loading configuration file, deep learning architecture and preparing deep learning model for training, loading of data, preprocessing, and post-processing are connected to each other to create a training process. Those module functions are explained in the following sections.

Below block diagram (figure 20), shows the interaction of each module with other and the flow of information between the modules through a face message.

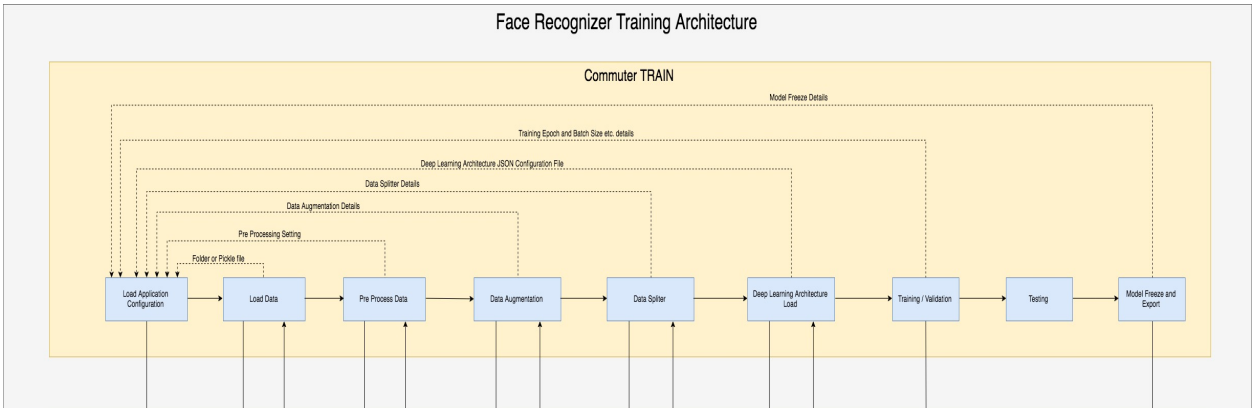


Figure 20: Training commuter process flow block diagram

3.7.2. GUI Commuter

GUI Commuter is as like as training commuter with small difference, is that it loads GUI modules rather than training related module, such as GUI interface component, detector, and recognizer modules are not required in training process. This module is used to create a face detection and recognition process flow for real life testing. Following figure 21, explains the communication channel created by GUI commuter for different modules to communicate for recognition task.

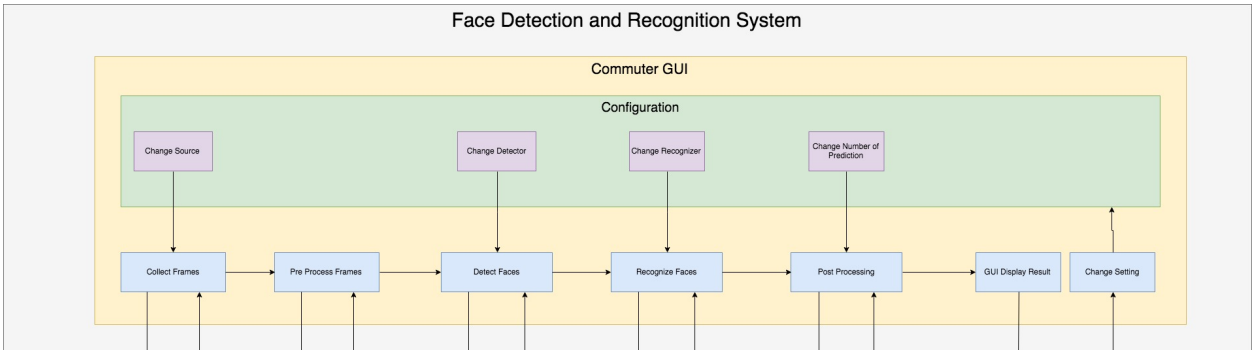


Figure 21: GUI Commuter process flow block diagram

3.8. Context

Context is the primary component of the system. It instantiates and possesses all the modules of the system and make it available for above commuters, to prepare process flow for specific task like training and user interface (GUI) for real life testing. Context creation is the first step of our application. The initialization of the modules in the context will be decided upon the type of contexts is being created such as GUI context or training context.

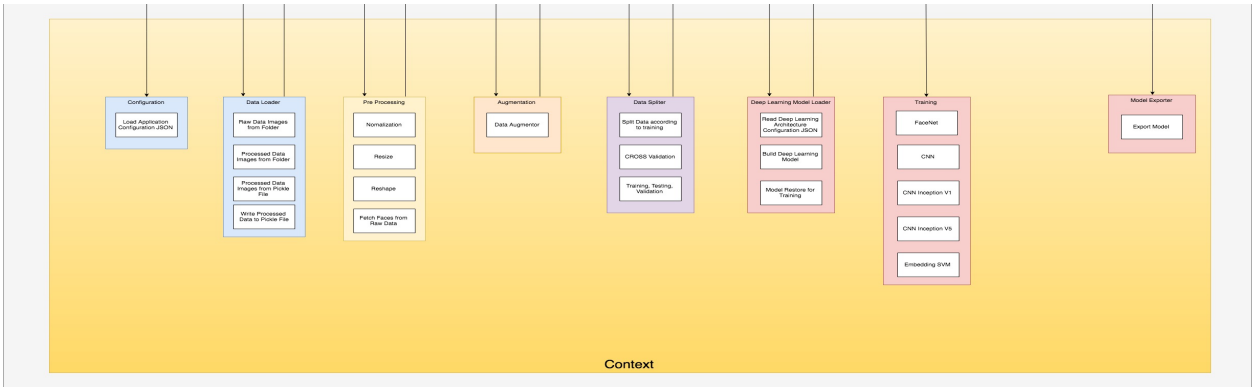


Figure 22: Training context module block diagram

Figure 22, shows modules are instantiated in the training context. Like configuration, data loader, pre-processing, augmentation, data splitter, deep learning model creation, training, and model freeze. All the modules are explained in the following sections.

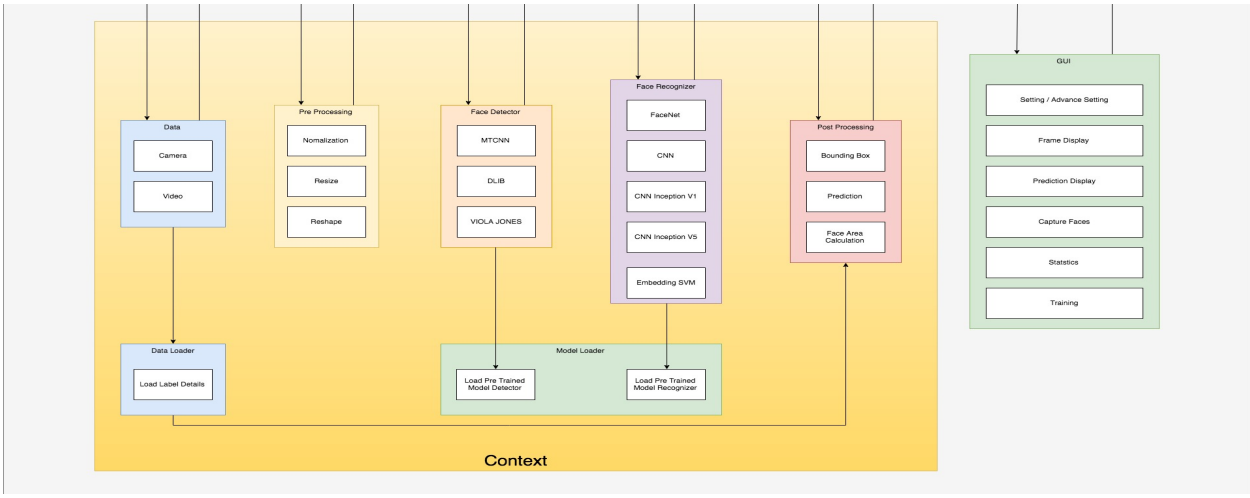


Figure 23: GUI context module block diagram

Figure 23, shows modules are instantiated in the GUI Context. Like Configuration, data loader, pre-processing, face detector, face recognition, post processing and GUI module. All the modules are explained in the following sections

3.9. Configuration

The primary purpose of this module is to load the application specific configuration JSON files into the context. This configuration module's instance is created in the context and later this instance is shared among other modules to access their module specific configuration details. Following is a sample configuration file, which has to be updated and maintained before the application starts.

```
{
  "image_depth":1,
  "training_data_type_info":"FR-FaceRecognition,OD-ObjectDetection",
  "training_data_type": "FR",
  "prepare_data_from_pickle_file": false,
  "pickle_data":{
    "images":"data/training_images.pickle",
    "labels":"data/training_label.pickle"
  },
  "pre_processing_required":false,
```

```
 "raw_data_folder":{
   "images":"data/images/original",
   "labels":"data/labels"
 },
 "processed_data_folder":{
   "images":"data/images/processed120",
   "labels":"data/labels"
 },
 "model_file":{
   "restoration_model_required":false,
   "restore_model":{
     "model":"trained_model/nn_cnn/freeze_conv_net.pb",
     "model_graph":"trained_model/nn_cnn/freeze_conv_net_grpah.pb"
   }
 },
 "image_size":{
   "height": 120,
   "width": 120
 },
 "augmentation_required": true,
 "augmented_data_size": 100,
 "prepare_pickle_file": false,
 "train_method_info":"CNN,INCEPTION,SVM,KNN,CNN-SVM,CNN-KNN",
 "training_method":"CNN",
 "network_config_file":"configuration/nn_architecture/cnn_net.config",
 "classes" : "auto",
 "training_size_percentage":90,
 "random_shuffle":false,
 "data_separation":"NORMAL",
 "data_separation_info": "NORMAL, CROSS_VALIDATION"

}
```

3.10. Data Loader

The main functionality of this module is to read and write data into the file system. Data Loader module is instantiated and the instance of the module in available in the context prior to the training process. Following is the configuration details required by the data module to load the data into the system.

```
"prepare_data_from_pickle_file": false,
"pickle_data":{
  "images":"data/training_images.pickle",
  "labels":"data/training_label.pickle"
},
"pre_processing_required":false,
"raw_data_folder":{
  "images":"data/images/original",
  "labels":"data/labels"
},
"processed_data_folder":{
  "images":"data/images/processed120",
  "labels":"data/labels"
},
```

If the “prepare_data_from_pickle_file = True” option is enabled, then the data will be loaded from the pickle file instead of folder structure. And the pickle file details are mentioned in “pickle_data” option.

If the preprocessing is required to fetch the faces from the larger images the then following configuration has to be pre_processing_required = True.

If all the above configurations are false, then the data for training will be loaded from the processed data folder mentioned in processed_data_folder option.

3.11. Cropping of Face

This module’s primary function is to crop the face image from the larger image as shown in figure 24 where a large image collected for training which has only one face in it and labeled. Face bounding box information is required to crop faces from larger image. For, that DLIB and MTCNN libraries are used to find the face bonding box information. These libraries output the face coordinates in the larger images. Using that face information this module crops and stores those face images in the file system. Figure 25, shows the cropped face image. Which will be later used in the training process for face recognition.

Figure 24: Large data Images collected

Figure 25: After cropped face from large data image

Following are some of the example of cropped faces from large images and properly separated according to the class labels for training.

3.12. Capture Faces

There is another module to capture the faces from a video content having persons face in it. In this case, the cropping of faces happened from the camera frame or video frames in real time. The face detector is used to detect the face in an image and those face images get stored in the directory, which will be used later for the incremental training process. Dlib and MTCNN face detection libraries (Which will be explained in the detector module section 3.18) are used for detection. This module is very helpful in terms of face data collection.

3.13. Pre-Processing

Crucial and important step in our training process is pre-processing. After the data is loaded into the system, it flows through a set of pre-processing steps such as normalization, resize etc. The following section will provide a brief explanation of the preprocessing steps.

3.13.1. Normalization

First step in the preprocessing module is to normalize the face images. Normalization can be achieved by many ways. Following are the options available for the normalization process.

- Keeping the pixel values between -127 to 127
- Keeping the pixel values between -1 to 1
- Keeping the pixel values according to the Gaussian distribution

While keeping the pixel values of an image between -127 to 127. First the face images get converted to a grayscale image for depth 1 training. This typically helps the deep neural network to learn faster, since gradients act uniformly for each channel and scale your inputs in a way that resulted in similarly-ranged feature values so that our gradients don't go out of control and overfit the model.

In addition to that, for RGB image training requires normalization of face image too. For this task the image pixels are normalized by substation the mean pixel value from each and then divided upon the standard deviation of image pixels. By this method of normalization images gets normalized by Gaussian distribution of pixel values, which will be helpful in training process.

3.13.2. Resize

The second task of preprocessor is to resize the faces as per the input tensor needs in the deep neural network training process. Images those are captured in the previous capture modules could be of any size. This module resizes the face images before it gets passed to input tensor of the neural network. The resizing parameters such as height and width details are declared in the application configuration file (app.config file under configuration->application folder). Below is the sample configuration required by this module.

```
"image_size": {
  "height": 120,
  "width": 120
}
```

3.13.3. Reshaping

Reshaping comes after the resizing process in the preprocessor; this functionality helps in reshaping of face images as per the input tensor needs in deep learning architecture. Input tensor in tensorflow receives the image data in the following format [batch, height, width, depth], batch considered as 1, while declaring the input tensor, height and width as per the configuration which is by default 120x120 and the depth as 1 because of grayscale images. The depth parameter of input tensor can be changed according to the training process for grayscale or RGB images.

3.14. Augmentation

Data augmentation is a crucial functionality for any deep learning architecture training. The sole purpose for this module is to augment the data and increase the number of images, which is essential for neural network training. Following are the augmentation properties affects the data augmentation process.

3.14.1. Rotation Angle

Rotation angle will help us in rotating the existing data set and create more rotated faces by 30 degrees we have used in our application. Those configurations can be changed in augmentation module.

3.14.2. Width Shift Range

Width shift range will shift the image content to the specific range but the shifting of the pixel values to right and left horizontally. Width shift range value used in this augmentation process is 0.2. however, this property value can be changed at any time in the augmentation module, before starting the training process.

3.14.3. Height Shift Range

This is same as the width shift range, but the shift of pixel values occurs in the vertical direction. Height shift range value used in this augmentation process is 0.2. however, this property value can be changed at any time in the augmentation module, before starting the training process. This augmentation helps in training of the faces, which are shifted a little from the center.

3.14.4. Shear Range

This option enables augmenter to shear the image from various angle and this augmentation helps in training of the network for blurring effect in the faces. This shearing value used in this system is 0.1. We

don't shear much, because the chance of finding feature will be minimized because of the higher blurring effect. Which will be bad for the training of deep neural network training. This property value can be changed at any time in the augmentation module, before starting the training process.

3.14.5. Zoom Range

The zoom range of 0.1 is used for augmenting the face images so that the network can be trained for the bigger face images. This property value can be changed at any time in the augmentation module, before starting the training process.

3.14.6. Horizontal and Vertical Flip

Horizontal flip functionality is used to create augmented face flipped horizontally and vertically as well. It will help in the way that in the real world sometimes we stand in front of the camera and we flip ourselves horizontally and vertically. This could help us in training those kinds of real-life scenarios. This property value can be changed at any time in the augmentation module, before starting the training process.

3.15. Data Splitter

This module helps in splitting of the dataset into training, validation and testing set. Data Split will commence after the augmentation process is complete. This data splitting process requires the percentage parameter to split the dataset, which is declared in the configuration file with the property name “training_size_percentage”. If this property’s value is set to 90 then the data split happens in 90% training data, 5% validation data and 5% testing data. This property’s value has to be updated before training process starts. Out of nearly 16000 of images generated for training after the augmentation process are segregated into 90% is 14000 images are kept for training purpose and other 1000 will be for validation and rest 1000 will be for the testing task.

3.16. Deep Learning Model Preparation

This is the core module of our training process. Which loads the neural network architecture JSON configuration file into the system and prepare the tensorflow deep learning model layers like convolution, max pool and fully connected layer, and binds the hyper-parameters to the model for the training. The configuration JSON for the neural network architecture is detailed as follows,

3.16.1. Hyper-parameter details:

The hyper-parameters are mentioned and declared in the configuration file under the key "training". Some of the hyper-parameters are like batch size to be performed, learning rate to be used, optimizer, loss function, regularization parameter and dropout percentage and the number of the class label has to be trained. Following is the sample of the configuration file that used in one of our training processes.

```
"training":{
  "training_steps":1000,
  "batch_size":30,
  "learning_rate":0.001,
  "stopping_loss_threshold_from_previous": 0.5,
  "optimizer":"adam",
  "loss":"softmax_cross_entropy",
  "regularization_beta":0.01,
  "nn_input_size":"auto",
  "nn_output_size":"auto",
  "tensor_name":"auto",
  "class_label_no": 37
},
```

3.16.2. Model File Details:

Model file details section in the JSON configuration holds the details about the model to be freeze after the training is over or the pre-trained model to be restored before training. Following is the example of the model information.

```
"model_file":{
  "model_dir": "trained_model/nn_cnn",
  "model_name": "freeze_conv_net_120.pb",
  "model_graph":"freeze_conv_net_grpah_120.pb",
  "restoration_model_required":false,
  "restore_model":{
    "model":"trained_model/nn_cnn/freeze_conv_net_120.pb",
    "model_graph":"trained_model/nn_cnn/freeze_conv_net_grpah_120.pb"
  }
},
```

In the above configuration, if the option restoration_model_required is set to True, in this case, before the training starts, it will restore that pre-trained model instead of creating a new from scratch. If that option is False, then a new neural network model will be created, and weight and biases are to be re-initialized by random normal function to start the training process. After the training process completed, the model will be saved as per the file name provided in the configuration option "model_name" under the directory mentioned as "model_dir".

3.16.3. Image Size

Another configuration is image size, the width and the height in this configuration, will be used to resize an image, and along with that, this will be used to declare the input tensor size of the neural network.

```
"image": {
  "width":120,
  "height":120,
  "resize_required":false
},
```

3.16.4. Deep Neural Network

This configuration provides the network architecture declaration. In which, all the layers like convolution and max pool are declared in blocks of JSON string in sequence. And the network preparation module will create the layers and automatically binds them in sequence. "deep_neural_network" is the key in the JSON file which holds the network JSON configuration. This JSON configuration is designed to ease our process of designing neural networks. Following is the sample JSON file configuration for the neural network and followed by different blocks information used while declaring the network layers.

```
"deep_neural_network":[
            {
            "name": "conv0",
             "type": "conv",
            "filters": 32,
            "kernel": [3, 3],
            "strides": [1, 1],
            "padding": "SAME",
            "activation": "relu",
            "output":"120x120x32"
```

```
    }
      […] ]
```

3.16.5. Convolution layer configuration JSON block

```
{
  "name": "conv0",
  "type": "conv",
  "filters": 32,
  "kernel": [3, 3],
  "strides": [1, 1],
  "padding": "SAME",
  "activation": "relu"
}
```

In the above configuration, which is used to declare a convolution layer with relu as the activation function. Properties are explained in detail as follows

Table 2: Convolution layer JSON configuration

Property	Values and Description
name	This property will be used to name the tensor that will be created in tensorflow as per the layer definition
type	This property defines the type of layer in the neural network for that block
filters	This property depicts how many filters to be used in a specific block of convolution in neural network.
kernel	Kernel or filter size that will be used in the convolution
strides	This property will be used to declare the stride size of the convolution
padding	Declares the padding of the convolution
activation	This help is declaration of activation for that layer

3.16.6. Maxpool layer configuration JSON block

```
{
  "name": "maxpool0",
  "type": "maxpool",
  "pool_size": [2, 2],
  "strides": 2
}
```

In the above configuration, which is used to declare a maxpool layer for the neural network. Maxpool layer properties are explained in detail as follows

Table 3: Maxpool layer JSON configuration

Property	Values and Description
name	This property will be used to name the tensor that will be created in tensorflow as per the layer definition
type	This property defines the type of layer in the neural network for that block
pool_size	This property shows the pooling size of the max pool layer.
strides	This property will be used to declare the stride size of the max pool layer

3.16.7. Flat layer configuration JSON block

```
{
  "name": "flat",
  "type": "flat",
  "output":"1x1x28800"
}
```

In the above configuration, which is used to declare a flat layer. Flat layer properties are explained in detail as follows

Table 4: Flat layer JSON configuration

Property	Values and Description
name	This property will be used to name the tensor that will be created in tensorflow as per the layer definition
type	This property defines the type of layer in the neural network for that block

3.16.8. Dense layer configuration JSON block

```
{
  "name": "dense0",
  "type": "dense",
  "activation": "relu",
  "units": 1024
}
```

In the above configuration, which is used to declare a dense layer. Dense layer properties are explained in detail as follows

Table 5: Dense layer JSON configuration

Property	Values and Description
name	This property will be used to name the tensor that will be created in tensorflow as per the layer definition
type	This property defines the type of layer in the neural network for that block
activation	This help is declaration of activation for that layer
units	This property is the output of that layer.

3.16.9. Inception layer configuration JSON block

Inception layer same as that of the convolution or max pool layer as per the configuration but the main difference is that the blocks have to declare in parallel. The following design will show who to align the blocks in parallel in JSON file.

```
{
  "name": "inception_1a",
  "type": "inception",
  "block": [[{Convolution Block}], [{Convolution Block}, {Convolution Block}], […], […]]
}
```

Table 6: Inception layer JSON configuration

Property	Values and Description
name	This property will be used to name the tensor that will be created in tensorflow as per the layer definition of inception layer Under the inception layer, there will be many convolution and max pool layers. All the layers below inception block will be named prefix with this name property of inception layer. Ex: in inside inception block named as "inception_1a", if we have one convolution block with the name as "conv0" the conv0 name will be changed to "inception_1a_conv0"
type	This property defines the type of layer in the neural network for that block of inception layer
block	This property shows the inception blocks. As you can see, this a list of lists. Each parallel unit will be declared in a list containing that layer's information and then all the parallel unit will be put together in another list. So that in this way we can create a parallel layer configuration in JSON for the network creation.

3.16.10. Output layer configuration JSON block

Output layer is the final dense layer of the network, which will be used to classify the different class. Following are the configuration details in the JSON file.

```
{
  "name": "output",
  "type": "output",
  "units": 37
}
```

Table 7: Output layer JSON configuration

Property	Values and Description
name	This property will be used to name the tensor that will be created in tensorflow as per the layer definition
type	This property defines the type of layer in the neural network for that block
units	This property is the number of classes information for the classification.

3.17. Training

In this thesis, various deep learning architecture designed, trained and compared the result with the existing FaceNet system, which is trained over our dataset. Following are the deep learning architecture have been considered for the training. Following sections consists of detail explanation of architecture and training process of each model.

- CNN Model
- Inception 1b model
- Inception 5b Model
- Embedding SVM Model [CNN as feature / embedding extractor]
- FaceNet Training

3.17.1. CNN Model Training

Convolutional Neural Network, the basic model used for object detection and recognition. Carried out some research to understand the feature extraction and classification using CNN. The following section will be explained in detail about the implementation of CNN architecture that we have used in the system and model training. [3]

Description

Convolutional Neural Network is used heavily for the object detection now a day. We have designed our own CNN architecture model to train from scratch for our face recognition system.

Architecture:

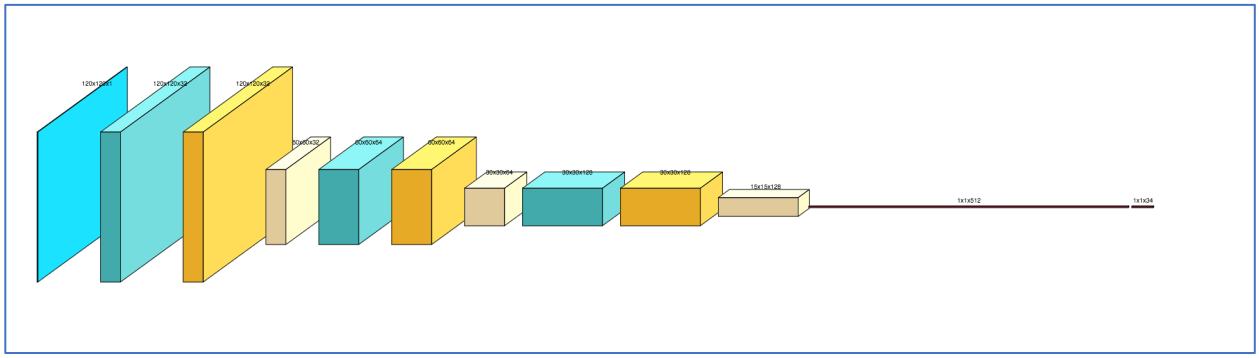


Figure 26: CNN architecture model diagram

Figure 22, shows the architecture of CNN, and each layer's input, output, and responsibilities are explained in detail as follows

First Layer is the input layer (input tensor), which will accept the input images in batch for the training to go through. The size of the tensor decided upon the input configuration provided before the training starts in the input height and width section of configuration file. However, for our training, we have used the shape of the 120x120x1 image, and input tensor shape would be 1x120x120x1 [batch, height, width, depth].

The second layer is the convolution layer, in which filter size we have kept is of 3x3, with 32 filters and stride is of 1x1 and padding as SAME. The output of the second layer is 120x120x32 (32 being the number of the filtered has been used for the convolution).

The third layer is relu activation layer, which will accept the input size of 120x120x32, and apply relu activation function on each weight vector and output produced is of the same size.

The fourth layer is the max pooling layer, of filter size of 2x2 and stride, is of 2x2. To make the input size of the image to be reduced by half in the max pooling layer. The input to the layer is 120x120x32 and output of the layer is 60x60x32. As we are not changing the number of filters in max pooling layer, so it won't have any effect on the number of channels we have used to the output.

The fifth layer is the second convolution layer, with 3x3 size filter and the number of filters used is 64 as per the processing standards with a stride of 1x1. The input to the layer is 60x60x32 and the output of the layer is 60x60x64 as this number of channel is changed from 32 to 64 depends upon the number of filters we have used in the fifth layer.

The sixth layer is again a relu layer applied after the second convolution layer as an activation function. Input size to the relu layer is 60x60x64 and the output from the layer is 60x60x64.

The seventh layer being a max pool layer we again reduced the size of the input to the layer by half again to get the feature again fine-grained further. With stride 2x2 and filter size of 2x2. The input to the layer is 60x60x64 and the output is 30x30x64.

The eighth layer is a convolution layer which has input as per the previous layer's output as 30x30x64, we have applied 3x3 filter size of 128 filters, which will find the local features in the 64 channels and in 30x30 feature size. The input to the layer is 30x30x64 and output of the layer is 30x30x128 because if 128 numbers of filters.

The ninth layer is a max pool layer which has input as per the previous layer's output as 30x30x128, we have applied 2x2 pooling filter size, which will reduce the feature size to 15x15x128. Which is the input to the next layer?

After the Ninth layer, we have decided to stop because we have reached the size of 15x15 feature size and we applied a flattened relu layer. In this layer will flatten our input features to 28800 number of nodes by the calculation of 15*15*128 (28800) and it will be feed to the next layer which is a final connected layer.

In the tenth layer, we have applied the final connected layer, in which the input from the previous flattened relu layer being converted to 512 feature vectors from 28800 feature vectors.

In the eleventh layer, we applied again a fully connected layer which in turn convert the input feature vector from 512 to the number of classes got into the training phase as in 37 class labels.

After the last fully connected layer, we applied the cross-entropy softmax loss function and Adam optimizer to train our CNN model network for the classification. At the last layer, we have softmax to get the probability of class prediction for all the class labels to find the validation accuracy and to minimize the loss function.

Training

All parts of the CNN deep neural network have been trained with error back-propagation using stochastic gradient descent as the optimizer. This optimizer can be changed in the configuration file before training starts. The hyper-parameters those we have used are like, batch size is 24. This means that during training a batch of 24 images are feed-forwarded through the network until each of the images has been trained or classified to one of the possible classes with the networks current weights. Then every image classification is compared to the ground-truth for that image. Then the error will be calculated if the prediction is same as the ground truth then back-propagated along the network changing the weights one by one in the direction that would minimize the error, also known as the steepest gradient descent.

The amount of training data plays a huge role in the performance of CNN. When training a network from scratch, a few thousand annotated images will not be enough. We have seen with the lesser number of the images the accuracy is not enough as compared to the larger dataset. One needs tens of thousands or hundreds of thousands, preferably millions of images. This amount of annotated data is hard to come by; As per of student dataset we have used for training with over a 2500 annotated image with over 37 classes.

Our training involved from scratch, pre-training modes have not considered for this architecture. However, there are other models for which the pre-trained network has been used to train on our dataset. Which will be explained later in the section.

While in training, we kept 1000 epochs for training before we validate the accuracy on the validation set in each epoch and get the loss and to decide for an early stop to avoid the overfitting.

3.17.2. Inception 1b Model

Inception model, that we have trained on 120x120 face images using one inception layer. Following sections will provide insight into the inception 1b model architecture and training.

Description

This is simple, inception model having one inception stack. The inception stack has three convolutions and one max pool with convolution in parallel and concatenates the resulting feature maps from the parallel layers before going to the next layer.

Now let's assume the next layer is also an Inception layer. Then each of the convoluted feature maps will be passed through the mixture of convolutions again and so forth, If the network has multiple inception layers stacked to each other. The idea is we don't need to know ahead of time that there was a better chance of finding best feature in the layers of convolution as, a 3×3 then a 5×5 convolution. Instead, this model applies parallel convolution and maxpool and automatically pick the best feature for the model training.

In this model a variety of convolutions is used; specifically, 1×1, 3×3, and 5×5 convolutions along with a 3×3 max pooling. If you're wondering what the use of max pooling layer with all the other convolutions, is that pooling is added to the Inception layer for the feature reduction as all the network design has at least one pooling layer. The larger convolutions are more computationally expensive, so the paper suggests first doing a 1×1 convolution reducing the dimensionality of its feature map, passing the resulting feature map through a relu layer, and then performs a larger convolution (in this case, 5×5 or 3×3). The 1×1 convolution is key because it will be used to reduce the dimensionality of its feature map.

It's also designed to be computationally efficient, using 12x fewer parameters than other competitors, allowing Inception to be used on less-powerful systems.

Architecture

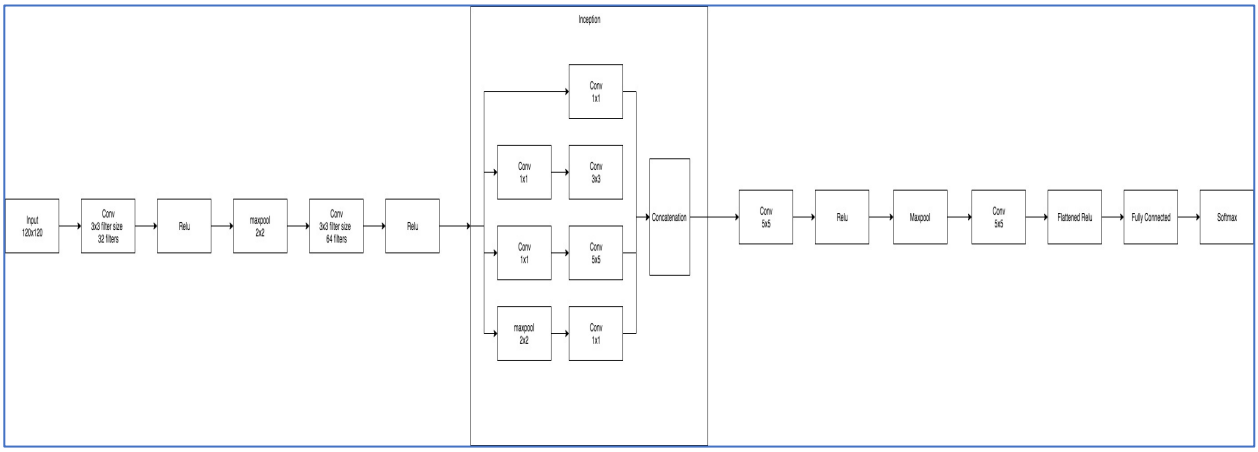


Figure 27: Inception 1b architecture diagram

First Layer is the input layer (input tensor), which will accept the input images in batch for the training to go through. The size of the tensor decided upon the input configuration provided before the training starts in the input height and width section of configuration file. However, for our training, we have used the shape of the 120x120x1 image, and input tensor shape would be 1x120x120x1 [batch, height, width, depth].

The second layer is the convolution layer, in which filter size we have kept is of 3x3, with 32 filters and stride is of 1x1 and padding as SAME. The output of the second layer is 120x120x32 (32 being the number of the filtered has been used for the convolution).

The third layer is relu activation layer, which will accept the input size of 120x120x32, and apply relu activation function on each weight vector and output produced is of the same size.

The fourth layer is the max pooling layer, of filter size of 2x2 and stride, is of 2x2. To make the input size of the image to be reduced by half in the max pooling layer. The input to the layer is 120x120x32 and output of the layer is 60x60x32. As we are not changing the number of filters in max pooling layer, so it won't have any effect on the number of the channels we have used to the output.

The fifth layer is the second convolution layer, with 3x3 size filter and the number of filters used is 64 as per the processing standards with a stride of 1x1. The input to the layer is 60x60x32 and the output of the layer is 60x60x64 as this number of channel is changed from 32 to 64 depends upon the number of filters we have used in the fifth layer.

The sixth layer is the inception layer, here we have used the dimensionality reduction inception model, which is different than the usual state of art inception model. In this inception layer, we have 4 parallel processing and one concatenation layer. Among 4 parallel layers, the first layer holds the 1x1 convolution with stride 1x1, the output is the same as input and it will find the feature map. Second, it comes to dimension reduction, in this second parallel layer, we have two sequential convolutions one with 1x1 convolution and the second one with 3x3 convolution. In the third parallel layer is also a dimension reduction layer, we have one 1x1 convolution followed by a 5x5 convolution and the final parallel layer has 1x1 convolution with a max pool layer and the all the parallel layers are connected to a concatenation layer. Which will concatenate and feature maps and keep the all the feature maps and removed the duplicate ones?

The seventh layer being a max pool layer, we again reduced the size of the input to the layer by half again to get the feature again fine-grained further. With stride 2x2 and filter size of 2x2. The input to the layer is 60x60x64 and the output is 30x30x64.

The eighth layer is a convolution layer which has input as per the previous layer's output as 30x30x64, we have applied 3x3 filter size of 128 filters, which will find the local features in the 64 channels and in 30x30 feature size. The input to the layer is 30x30x64 and output of the layer is 30x30x128 because if 128 numbers of filters.

The ninth layer is a max pool layer which has input as per the previous layer's output as 30x30x128, we have applied 2x2 pooling filter size, which will reduce the feature size to 15x15x128. Which is the input to the next layer?

After the ninth layer, we have decided to stop because we have reached the size of 30x30 feature size and we applied a flattened relu layer. In this layer will flatten our input features to 28800 number of nodes by the calculation of 15*15*128 (28800) and it will be feed to the next layer which is a final connected layer.

In the tenth layer, we have applied the final connected layer, in which the input from the previous flattened relu layer being converted to 512 feature vectors from 28800 feature vectors.

In the eleventh layer, we applied again a fully connected layer which in turn convert the input feature vector from 512 to the number of classes got into the training phase as in 37 class labels.

After the last fully connected layer, we applied the cross-entropy softmax loss function and Adam optimizer to train our Inception 1b model network for the classification. At the last layer, we have softmax to get the

probability of class prediction for all the class labels to find the validation accuracy and to minimize the loss function.

3.17.3. Inception 5b Model

This is the very complex architecture of training a deep learning network. We have 5 inception layers before we classify the images. Images have to go through the 5-inception layer and computation wise this the heaviest model to train. Following sections contains architecture and training of inception 5b model in details.

Description

This is very deep inception layer for the training of face recognition system. Earlier we have seen the inception model for only 1 layer having 4 parallel convolutions and max pool and concatenation layer before passing it on to the next layer. However here, we have five inception layers as described in next architecture section.

Architecture

Here we have five inception layers connected to each other and each inception layer has 4 parallel blocks. Following architecture, the diagram shows the 5b inception model architecture.

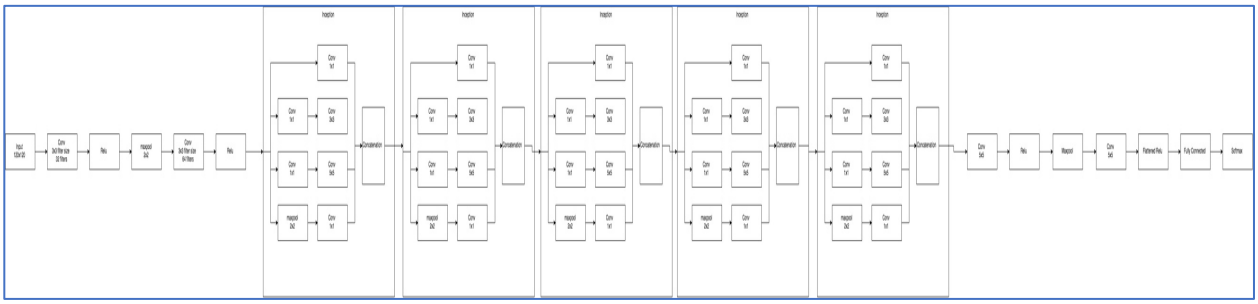


Figure 28: Inception 5b architecture model diagram

First Layer is the input layer (input tensor), which will accept the input images in batch for the training to go through. The size of the tensor decided upon the input configuration provided before the training starts in the input height and width section of configuration file. However, for our training, we have used the shape of the 120x120x1 image, and input tensor shape would be 1x120x120x1 [batch, height, width, depth].

The second layer is the convolution layer, in which filter size we have kept is of 3x3, with 32 filters and stride is of 1x1 and padding as SAME. The output of the second layer is 120x120x32 (32 being the number of the filtered has been used for the convolution).

The third layer is relu activation layer, which will accept the input size of 120x120x32, and apply relu activation function on each weight vector and output produced is of the same size.

The fourth layer is the max pooling layer, of filter size of 2x2 and stride, is of 2x2. To make the input size of the image to be reduced by half in the max pooling layer. The input to the layer is 120x120x32 and output of the layer is 60x60x32. As we are not changing the number of filters in max pooling layer, so it won't have any effect on the number of the channels we have used to the output.

The fifth layer is the second convolution layer, with 3x3 size filter and the number of filters used is 64 as per the processing standards with a stride of 1x1. The input to the layer is 60x60x32 and the output of the layer is 60x60x64 as this number of channel is changed from 32 to 64 depends upon the number of filters we have used in the fifth layer.

The sixth layer is the inception layer, here we have used the dimensionality reduction inception model, which is different than the usual state of art inception model. In this inception layer, we have 4 parallel processing and one concatenation layer. Among 4 parallel layers, the first layer holds the 1x1 convolution with stride 1x1, the output is the same as input and it will find the feature map. Second, it comes to dimension reduction, in this second parallel layer, we have two sequential convolutions one with 1x1 convolution and the second one with 3x3 convolution. In the third parallel layer is also a dimension reduction layer, we have one 1x1 convolution followed by a 5x5 convolution and the final parallel layer has 1x1 convolution with a max pool layer and the all the parallel layers are connected to a concatenation layer. Which will concatenate and feature maps and keep the all the feature maps and removed the duplicate ones?

From 7th to 10th layers, are the inception layers, as mentioned earlier. Each having same inception block architecture of dimensionality reduction inception model as per the sixth layer.

The eleventh layer being a max pool layer we again reduced the size of the input to the layer by half again to get the feature again fine-grained further. With stride 2x2 and filter size of 2x2. The input to the layer is 60x60x64 and the output is 30x30x64.

The twelfth layer is a convolution layer which has input as per the previous layer's output as 30x30x64, we have applied 3x3 filter size of 128 filters, which will find the local features in the 64 channels and in 30x30

feature size. The input to the layer is 30x30x64 and output of the layer is 30x30x128 because if 128 numbers of filters.

The thirteenth layer is a max pool layer which has input as per the previous layer's output as 30x30x128, we have applied 2x2 pooling filter size, which will reduce the feature size to 15x15x128. Which is the input to the next layer?

After the Thirteenth layer, we have decided to stop because we have reached the size of 30x30 feature size and we applied a flattened relu layer. In this layer will flatten our input features to 28800 number of nodes by the calculation of 15*15*128 (28800) and it will be feed to the next layer which is a final connected layer.

In the tenth layer, we have applied the final connected layer, in which the input from the previous flattened relu layer being converted to 512 feature vectors from 28800 feature vectors.

In the fourteenth layer, we have applied the final connected layer, in which the input from the previous flattened relu layer being converted to 128 feature vectors from 28800 feature vectors.

In the fifteenth layer, we applied again a fully connected layer which in turn convert the input feature vector from 128 to the number of classes got into the training phase as in 37 class labels.

After the last fully connected layer, we applied the cross-entropy softmax loss function and Adam optimizer to train our Inception 1b model network for the classification. At the last layer, we have softmax to get the probability of class prediction for all the class labels to find the validation accuracy and to minimize the loss function.

Training

This training is same as the above-mentioned steps, but the difference is the image has to go through the 5 layers of inception blocks before it classifies the image passed to the network. We have seen the fall of the loss function is faster as compared to the above two models and smooth. All the results and analysis will be explained in section 4.

3.17.4. SMV Classifier

Support Vector Machine is the simplest and time-consuming Machine Learning state of art classification model when the data is more. Following sections will explain, how the SVM is used to train the model on our dataset in detail.

Description

Support Vector Machines (SVMs) is a binary feedforward neural network that can be used for pattern classification given both linearly and non-linearly separable data. Given the simplest scenario with two classes that are linearly separable the main idea of SVMs can be summarized as "Given a training sample, the support vector machine constructs a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized." We have used the radial basis function in SVM for training. Radial basis function provides non-linearity to the data, so this is very useful for us in this kind of training.

Architecture

Here the architecture is very simple as compared to the previous models. SVM requires feature vectors so that it can construct the hyperplanes to separate each class features for classification. The feature vector is of 128-embedding data extracted from the final layer of Inception 5b model. Those 128-embedding data used to train the SVM with RBF function along with the beta as 0.001 and C as 1. we have used the Scikit-learn SVM trainer to train our SVM model. Figure 25, shows the architecture of SVM model.

Figure 29: SVM Model for face image classifier architecture diagram

Training

The first thing is to load a pre-trained network which provides 128 embedding feature vectors. In the first method, the final classification layer is removed from the pre-trained model and then the images are sent through the network and the 12-embedding features are recorded along with the class labels. Feature vectors generated from the training images are then used to train a Support Vector Machine (SVM) classifier. This classifier is adapted specifically to our training data since the feature vectors generated from the feature extraction are high-level representations of our images.

Evaluation of the SVM classifier is simple, we use our feature vectors from the test images and run them through the SVM. It will then predict which class each of the images they belong to or provide a probability

estimate of which class it belongs to. This relatively simple method has proven itself by providing outstanding results.

3.18. Detector

This module's function is to detect a face in the given frame. This Module is independent as like other modules. The module architecture is as follows.

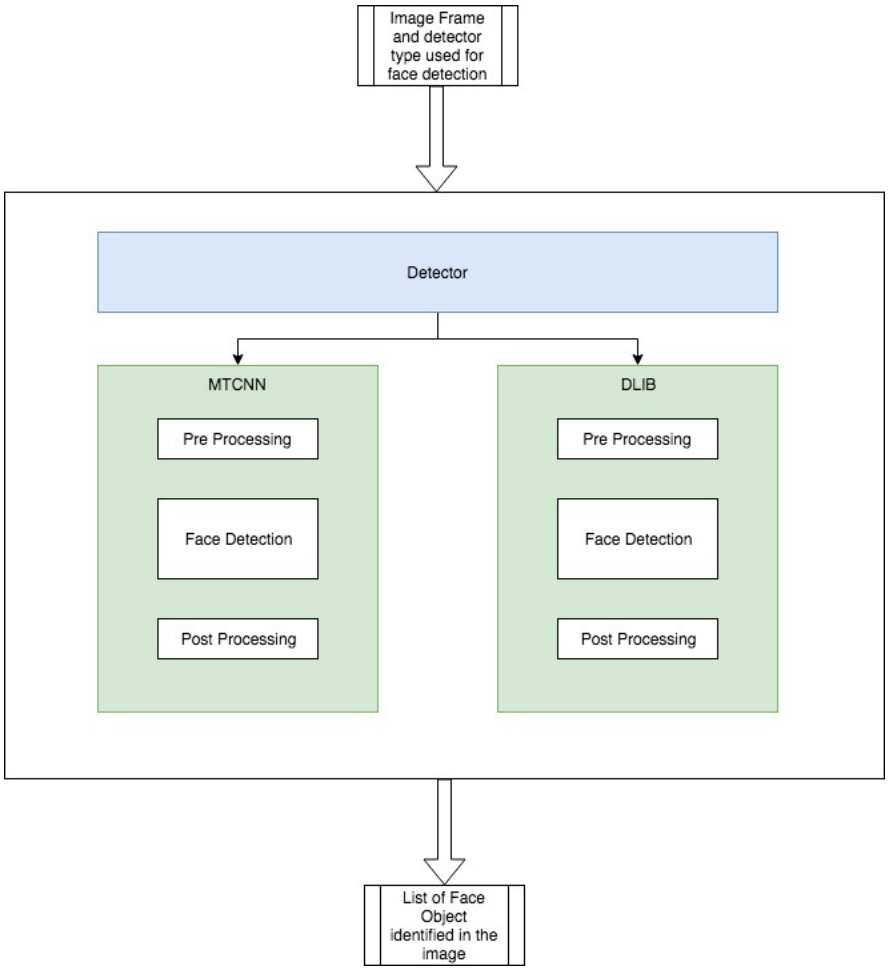


Figure 30: Detector module block diagram

The input to the modules in the image in which faces has to be detected and type of detection method to be used for face detection. This module is very scalable if we would like to add a new detection library, we just have to create an underlying module structure as defined above. The output of the detection module is the list of face object which holds the face image and the bounding box of the face image relative to the original image passed to the detector module. Each component of this module is explained in detail in the following sections.

3.18.1. Method and Implementation

Face Detection has been used so many places right now and now a day. In addition to that MTCNN and DLIB library are used to detect the faces as primary module, with all the kind of face detection technology and try to improve the accuracy of the face detection from the recognition standpoint.

3.18.2. DLIB Library

Dlib is an open source API. It has many functions, but we have used the face detection function out of it. And created a wrapper around the Dlib so that it will handle the preprocessing and post-processing steps before it creates the list of face objects and passes it to other modules in the pipeline of face detection and recognition system. [1]

3.18.3. MTCNN Library

MTCNN is the multi-task cascaded convolutional network, is an open source library designed and published by Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, Yu Qiao. Which is a Face detection and alignment in an unconstrained environment? This has various challenging due to various poses, illuminations, and occlusions around the faces in the image frame. However, this performs better than the Dlib 19.4 version. [2]

3.18.4. Post-processing of Detector

We have implemented a naïve approach in the post processing module of our application. Usually detector detects the faces in an image and passes it to recognizer module (section 3.1.5.16). It does not rotate the face as in, if the face is tilted with 30 degree it will pass the tilted face image to the recognizer, and we have seen a performance issue. So, to avoid the rotated issue, in the post processing section of detector, we have implemented an anti-rotation functionality. This anti-rotation functionality rotates the face in opposite direction of face rotation and makes the face vertically straight before it passes it to the recognizer. And we have seen some improvement over the application of anti-rotation.

This anti-rotation has problem in padding of pixel values. If we rotate a face image of the size 120x120x3, then after the rotation, the corner of the face image has black area, which is not good for neural network.

So, we have added some padding before rotation and then rotate and after the rotation we have performed a crop operation if rotated image to get the straight face without black area.

3.18.5. Anti-rotation of Face without padding

Figure 31: Tilted face detected

Figure 32: Anti-rotation on tilted face with black corners

Figure 33: Cropped face after anti-rotation

Above pictures shows, in figure 27, the face is detected as tilted and in figure 28 we have performed an anti-rotation of face in the opposite direction of persons movement and in figure 29 it shows the cropped face without the back corners which will be the input to the neural network. In this scenario the features of a person get damaged by cropping and anti-rotation. To Solve this problem, we have taken an approach to add padding before the anti-rotation.

3.18.6. Anti-rotation of face with padding

Figure 34: Face detected with padding 20

Figure 35: Anti-rotation on image padded with 20 pixels

Figure 36: After cropping of anti-rotated image with padding 20

Above pictures shows, in figure 30, the face is detected as tilted but with padding of 20 and in figure 31 we have performed an anti-rotation of face in the opposite direction of persons movement and in figure 32 it shows the cropped face without the back corners which will be the input to the neural network. In this scenario the features of a person did not get damaged by cropping and anti-rotation as compared to the previous without padding.

3.19. Recognizer

This module is also an independent and wrapper module is our system and it has many recognizers inside it wrapped inside one single recognizer module. CNN recognizer, FaceNet, Inception model recognizer module and SVM model recognizer are considered for the recognition components. Below is the recognizer module architecture.

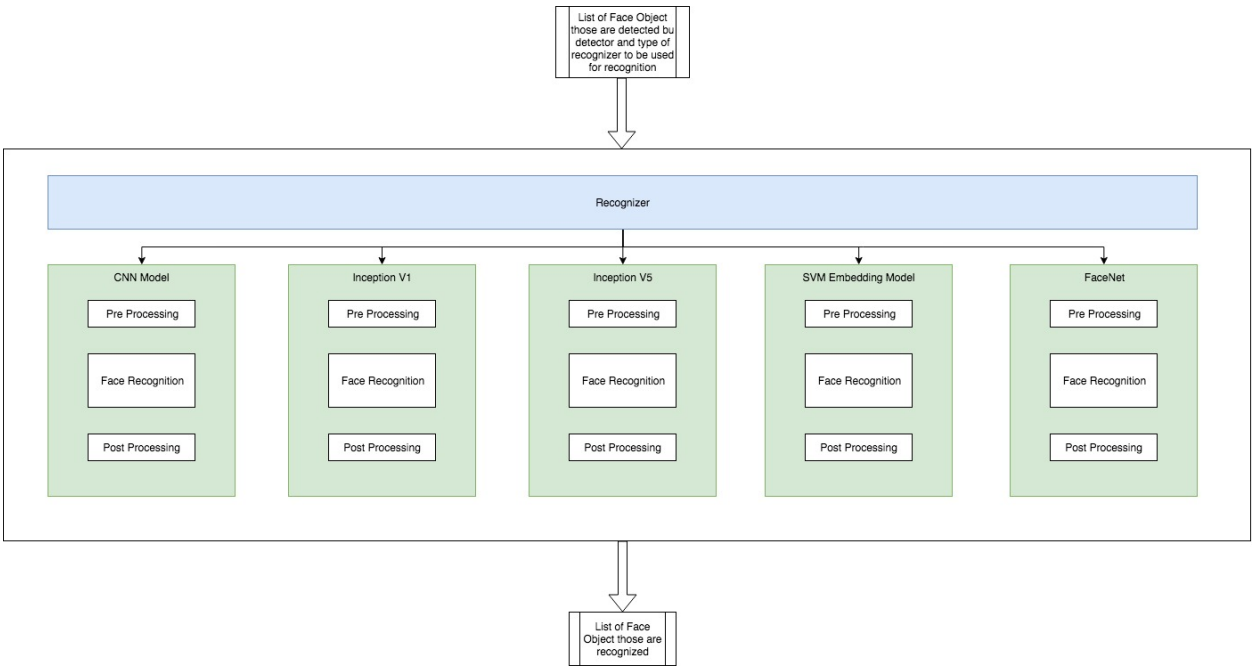


Figure 37: Recognizer module wrapper architecture

The input to the recognizer is the list of face objects those are detected by the detector module so that this module could process the faces detected and recognize the faces of the persons trained. And the output of the module is the list of faces recognized by the recognizer.

Each of the recognizer modules inside the recognizer wrapper is independent of each other. Each module inside the recognizer has their own preprocessing and post-processing components depending on their implementation. In future, if anyone would like to add new recognizer, they have to add a separate module with preprocessing, recognizer and post-processing and register to the existing recognizer wrapper module. Recognizer module each component will be explained in the following sections.

3.19.1. Method and Implementation

Various components have been designed and implemented for the face recognition system, the following are the face recognition components have been used independently

- CNN Model Recognizer
- Inception 1b Model Recognizer
- Inception 5b Model Recognizer
- SVM-Embedding Model Classifier Recognizer
- FaceNet Recognizer

All of the above recognizer modules and their usage have been explained in detail in the following sections and as we have already mentioned each recognizer component has their own preprocessing and post-processing component in regardless of recognizer component. Following are the preprocessing steps those are part of preprocessing steps and post-processing steps.

3.19.2. FaceNet

As FaceNet does not support the recognition functionality for image size below 160x160 height and width. So, in the preprocessing step in the FaceNet component, we have filtered those faces which do not have 160x160 size.

In the FaceNet post-processing component, we have taken the faces and their predictions from the FaceNet and prepared the face objects so that our system could understand the message being flown to the subsequent module in the system.

3.19.3. CNN Model, SVM & Inception 1b and 5b Model

As above models are concerned, there is some more task has to be done in the preprocessing step before we pass the face image to the deep neural network for the recognition. First, we normalize the face pixel values from -127 to 127, then resize the face to 120x120 so that the reshape component can reshape the face images to the 1x120x120x1 dimension so that this can be processed by our own designed deep learning models. In the post-processing step, we just accumulate the al the face recognition and prediction

output from the neural network and prepare the face object so that it can be passed to the subsequent modules.

3.20. Pre-Processing

Pre-Processing being the first layer as such in the deep learning feature extraction process from deep learning architectural point of view for the unseen data. In this layer, the data will be converted to the appropriate shape so that it can be fed to the neural network for the feature extraction. Without that the neural network may throw an exception for undetected data format giving to the network. Several pre-processing tasks is done before our deep learning layers accept the input, the following is the task done and explained in detail.

3.20.1. Normalization

Normalization is the pre-processing task in which, we normalize the input image so that the mathematical calculation stays in the limit and don't go into the overfitting step. I have normalized each face image pixel values to be constrained into -127 to 127-pixel value. Because of its Gaussian nature of pixel value distribution and keeping the pixel value between that to handle the

3.20.2. Resizing

Resizing the images is very important from the neural network standpoint so that the network can handle the proper input size and dimensionality of the input data. I have used 120x120 pixel image format that can be feed into the neural network. It can be changed at any time however it depends on the training process and how the training has been carried out with how many dimensions of the image for the feature extraction. If we provide the image of 200x170 it will be resized to 120x120 dimension, it may reduce the feature extraction and loses some of its property however it does not matter a lot because that way I have designed the neural network to handle the feature extraction is to the sheering and resizing of the face.

3.20.3. Reshaping

Reshaping is required by the preprocessor because of the nature of handling the input by the tensorflow, it requires the input image should be proper shape before it enters into the first layer. The shape of the image depends on the batch size, height, width, depth of the image. So as per my training process, I have used various batch size but the height and width as of 120x120 and depth of the image to be 1 because I have converted the image to the grayscale image before pushing it for feature extraction.

3.21. Post Processing

The last but previous module in our system is the post-processing module, which comprises of various tasks to be handled before it can be viewed by the users. Following are the post-processing task those are designed for our system.

3.21.1. Accumulator

Post-processing accumulator is implemented to store the past recognized faces for the future prediction. An accumulator is an object that stores the previously recognized data as per the accumulator size defined for the post-processing. When the accumulator gets full then its takes maximum occurrence of faces being recognized and on the basis of the maximum vote, it will predict the face. For an example, If the accumulator size is defined as 15. In this case, accumulator stores the recognized faces for 15 frames and at the 16th frame, it starts the prediction. On the 17th frame, the first recognized face from the accumulator is deleted and a 17th frame recognition details get appended to the accumulator and so on. Example of the accumulator size 10

Figure 38: Accumulator of size 10

In the above accumulator example of size 10, it has 10 positions where the recognized faces get stored for initial 10 frames before it predicts. On the 11th frame. Accumulator gives a voting of all the accumulated faces for last 10 frames. As per the example above

 This face gets 8/10 vote, which is the face we have in all the frames

 This face gets 2/10 vote which is the incorrect recognition

3.21.2. Weighted Accumulator

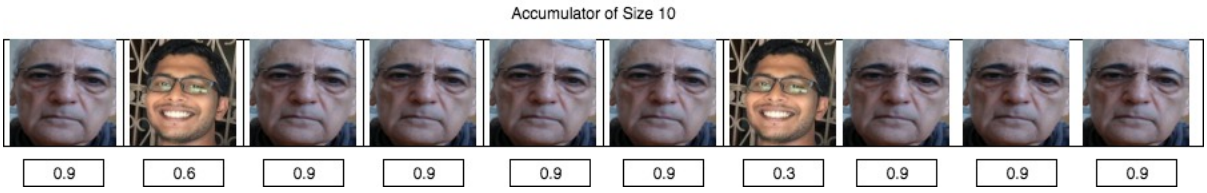


Figure 39: Weighted accumulator example

In the above accumulator example of size 10, it has 10 positions where the recognized faces get stored for initial 10 frames before it predicts for the current recognized frame. On the 11th frame. Accumulator gives a voting of all the accumulated faces for last 10 frames on their weighted sum over each face accumulated. As per the example above

 This face gets weighted sum of probabilities 7.2, which is the face we have in all the frames

 This face gets 1.8, which is the incorrect recognition

3.21.3. Overlay of Bounding Box

As the recognizer and detector, detects and recognized the face and prepared the face object. It has bounding box information, face image and original frame with the prediction details.

To display the bounding box around the face in the original frame, so that the user can see the bounding box around the face, this module adds an overlay in the original frame at the specific position mentioned in the face object on the image and draws the bounding box. Which after that can be displayed in the GUI, so that user could able to see the bounding box information on the screen.

3.21.4. Prediction Details enhancement

Face object prepared in the previous recognition module has the prediction probability details from the softmax layer of the neural network. In this section, we process to find out the top 5 predictions made of the

given face by taking top 5 probabilities from the 37-probability values from the last layer of the neural network or the SVM predictor.

After we have finalized the top 5 probabilities, the system assigns a label according to the probability index. Those indices are stored in a mapping pickle file before we started our training process. That mapping pickle file has the mapping of the label to the output node index. For example, node 0 in the output layer mapped to a specific label X10 and node 1 might get mapped to label X34, because we don't decide the mapping process, because of that we have kept a mapping of pickle file of those class label to node index mapping.

3.22. GUI

The last module in our system is the GUI module, we have used the PYQT framework to design the GUI module. It has many components regarding our system and how user-friendly the system would be for face detection and recognition system. Following are the components we have designed in the GUI module.

3.22.1. Camera QT Frame

Camera frame is a PYQT frame, which is designed to display the camera captured frame and processed frame after the bounding box has been drawn on the original frame after the face gets recognized. This just acts as the display of images captured from the camera in real time.

3.22.2. Toolbar QT Frame

This holds the basic button and functionality selection options, through which we can change the whole system options. Such as displaying the camera frame without recognition, recognize the faces in frames read from the camera or video file. We could able to change the source of the videos from setting window. And after we select the type of action, we would like to perform, then by clicking the start button, we could start the specific function of real-world testing. E.g. Just display the frame read from video or camera or start to capture the faces from the frames or start the recognition system to recognize the persons.

3.22.3. Recognition System Flow QT Frame

This advanced setting is a QT frame, has many options and options in our system. Like we can change, which face detector and recognizer to be used and how many predictions we would like to see in the GUI for a recognized face.

While capturing the face images from the frame, which face has to be captured, is this near to the camera or all faces those are detected in the current frame read from the camera or video file.

By the help of the number of recognition setting, we could able to decide how many persons we would like to recognize. This preference has 1 to 5 and all. Depending on the number of selection, the system will decide how my face has to go through the recognition process.

There is another preference for capturing report of our recognition activity. This report capturing functionality can be turned on but selection the Capture Video Analysis option to Yes. After the process gets stopped, a video analysis report pickle file gets stored in the specific recognizer module folder under result directory.

3.22.4. Prediction QT Frame

Which displays the predicted face images, after a person gets recognized successfully. So, that the user can easily visualize the recognized person by seeing the predicted faces.

It's a setting in the advance setting frame, that how many predicted faces that user would like to display in the prediction frame. This preference can be changed from the advance setting frame labeled as "Number of Predictions".

Chapter 4: Experiment

4. Experiments

We have carried out several experiments on the CNN and Inception 1b, 5b and SVM model along with the FaceNet model and compared the results of each model with another. Following are the statistics of our result analyzed from the training. All the experimentation and result and graphs will be explained in detail in following sections.

4.1. Data

Out of whole dataset collected for training of deep learning models, 90% of those data considered for training set, 5% is considered for the validation set and the rest is taken as testing set. Validation dataset is used to calculate the validation accuracy of the model in each epoch of training. While testing dataset is used to find the model final accuracy before freeze the model for future use.

4.2. Data Augmentation Details

Below statistics shows, the information about the data collected per class label. This statistic is taken before the data get augmented for training.

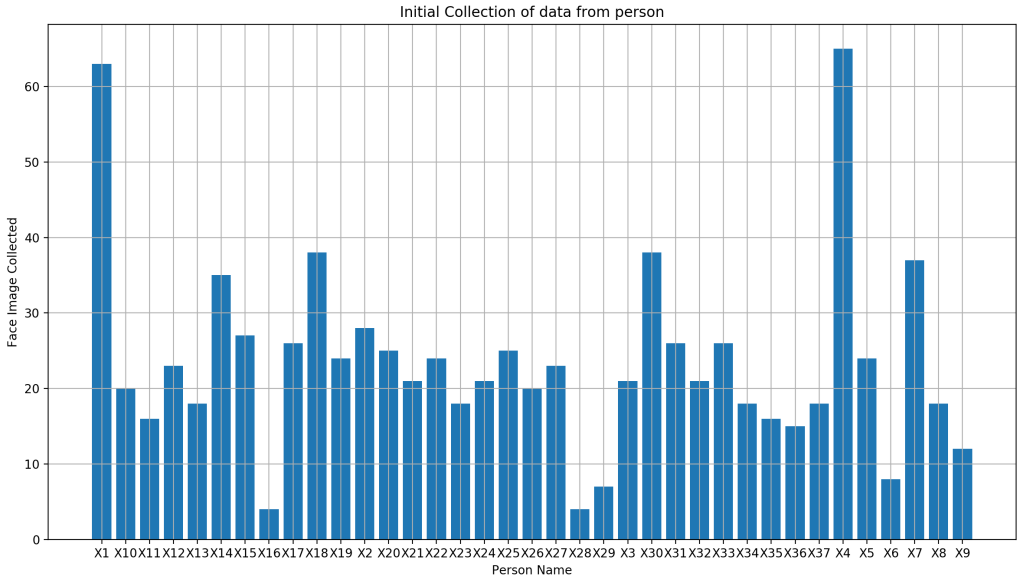


Figure 40: Class Label wise data distribution before Augmentation

Below statistics shows, after the data augmentation completed per class label.

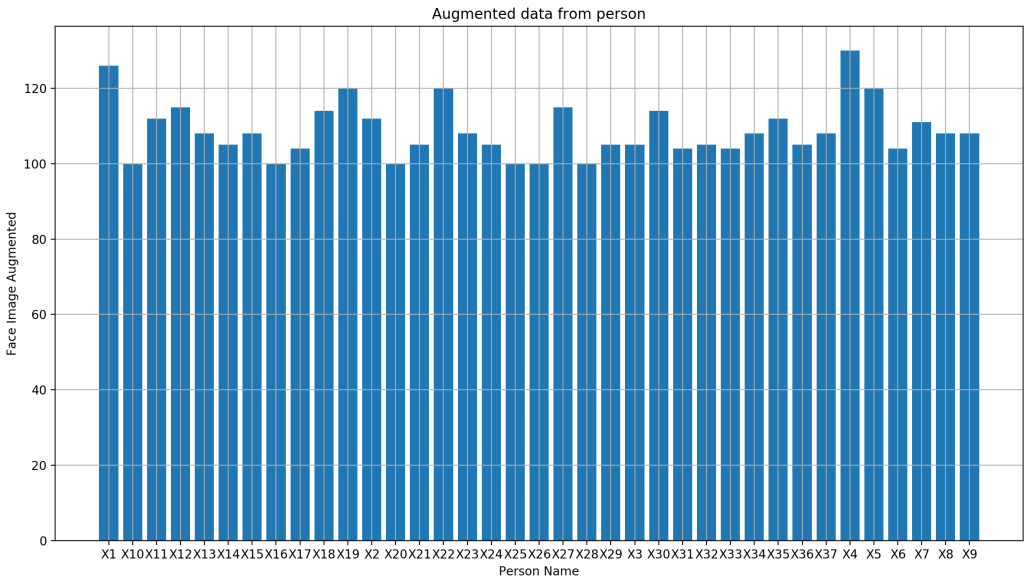


Figure 41: Class label wise data distribution After Augmentation

4.3. Result

4.3.1. Testing Accuracy for gray scale images of Depth 1

Our CNN testing accuracy is 90%.

Inception 1b outperform CNN with the accuracy of 91%.

Inception 5b even outperform Inception 1b with the accuracy of 92%.

SVM model trained over Inception 5b embedding detail accuracy is of 89%.

FaceNet accuracy is of 89% in our dataset.

Our SVM model trained over FaceNet embedding details which gives us testing accuracy of 77%.

4.3.2. Testing Accuracy for RGB images of Depth 3

Our CNN testing accuracy is 80%.

Inception 1b outperform CNN with the accuracy of 84%.

Inception 5b even outperform Inception 1b with the accuracy of 80%.

SVM model trained over Inception 5b embedding detail accuracy is of 77%.

FaceNet testing accuracy is of 79%.

Our SVM model trained over FaceNet embedding details which gives us testing accuracy of 76%.

When we have tested with the sample videos of unknown but labeled dataset, we have observer the following information.

4.4. Comparison without accumulator

Following analysis is carried out for all the models and represented in tabular format. This analysis is done for down samples images by factor 4, which reduces the size of frames by quarter before starts the recognition.

4.4.1. CNN Model

Following are the statistics of unlabeled and unseen dataset tested through CNN model.

Table 8: CNN model video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	86	133
Video2	X3	219	203	197	71	126
Video3	X31	362	316	311	301	10
Video4	X4	511	465	465	445	20
Video5	X4	327	181	181	70	111

4.4.2. Inception 1b

Following are the statistics of unlabeled and unseen dataset tested through Inception 1b model.

Table 9: Inception 1b model video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	55	164
Video2	X3	221	205	199	22	177
Video3	X31	309	309	303	302	1
Video4	X4	504	466	465	463	2
Video5	X4	264	147	147	49	98

4.4.3. SVM – Inception 5b Embedding

Following are the statistics of unlabeled and unseen dataset tested through SVM and Inception5b embedding model.

Table 10: SVM model with inception 5b embedding video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	220	219	34	185
Video2	X3	229	213	208	0	208
Video3	X31	326	316	310	303	7
Video4	X4	510	465	465	462	3

Video5	X4	327	181	181	56	125

4.4.4. SVM – FaceNet Embedding

Following are the statistics of unlabeled and unseen dataset tested through our SVM model trained over FaceNet embedding data.

Table 11: SVM model with FaceNet embedding video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	173	46
Video2	X3	223	207	202	9	193
Video3	X31	347	316	310	59	251
Video4	X4	240	238	237	237	0
Video5	X4	295	161	161	81	80

4.4.5. FaceNet

Following are the statistics of unlabeled and unseen dataset tested through FaceNet model.

Table 12: FaceNet video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	182	37

Video2	X3	268	252	247	219	28
Video3	X31	341	315	309	66	243
Video4	X4	510	465	465	463	2
Video5	X4	327	181	181	93	88

4.5. Comparison with moving accumulator

Following analysis is carried out for all the models and represented in tabular format. this analysis is done for down samples images by factor 1, in short process the actual image received from camera frames and we have used an accumulator of size 10 in the post processing to store the past recognized faces for future recognition and we have seen a better performance as compared to other experiments.

4.5.1. CNN Model

Following are the statistics of unlabeled and unseen dataset tested through CNN model.

Table 13: CNN Model video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	185	34
Video2	X3	219	203	197	181	16
Video3	X31	362	316	311	310	1
Video4	X4	511	465	465	462	3
Video5	X4	327	181	181	85	96

4.5.2. Inception 1b

Following are the statistics of unlabeled and unseen dataset tested through Inception 1b model.

Table 14: Inception 1b model video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	93	126
Video2	X3	221	205	199	180	19
Video3	X31	309	309	303	302	1
Video4	X4	504	466	465	463	2
Video5	X4	264	147	147	55	92

4.5.3. SVM – Inception 5b Embedding

Following are the statistics of unlabeled and unseen dataset tested through SVM and Inception5b embedding model.

Table 15: SVM model with inception 5b embedding video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	220	219	192	27
Video2	X3	229	213	208	102	106
Video3	X31	326	316	310	303	7
Video4	X4	510	465	465	462	3

Video5	X4	327	181	181	75	106

4.5.4. SVM – FaceNet Embedding

Following are the statistics of unlabeled and unseen dataset tested through our SVM model trained over FaceNet embedding data.

Table 16: SVM model with FaceNet embedding video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	173	46
Video2	X3	223	207	202	9	193
Video3	X31	347	316	310	59	251
Video4	X4	240	238	237	237	0
Video5	X4	295	161	161	81	80

4.5.5. FaceNet

Following are the statistics of unlabeled and unseen dataset tested through FaceNet model.

Table 17: FaceNet video analysis statistics

Video Number	Labeled With	Number of total Frame	Face Detected in Number of Frame	Face Recognized in Number of Frame	Correct Number of Recognition	Incorrect Number of Recognition
Video1	X4	245	219	219	182	37

Video2	X3	268	252	247	219	28
Video3	X31	341	315	309	66	243
Video4	X4	510	465	465	463	2
Video5	X4	327	181	181	93	88

4.6. Losses

4.6.1. CNN Loss Function

Following loss graph shows the gradient descent and loss function graph of Convolutional Neural Network training. In this training process we have used the adam optimizer and softmax cross entropy as loss function. We have seen that the loss function approaches to zero after 41st iteration.

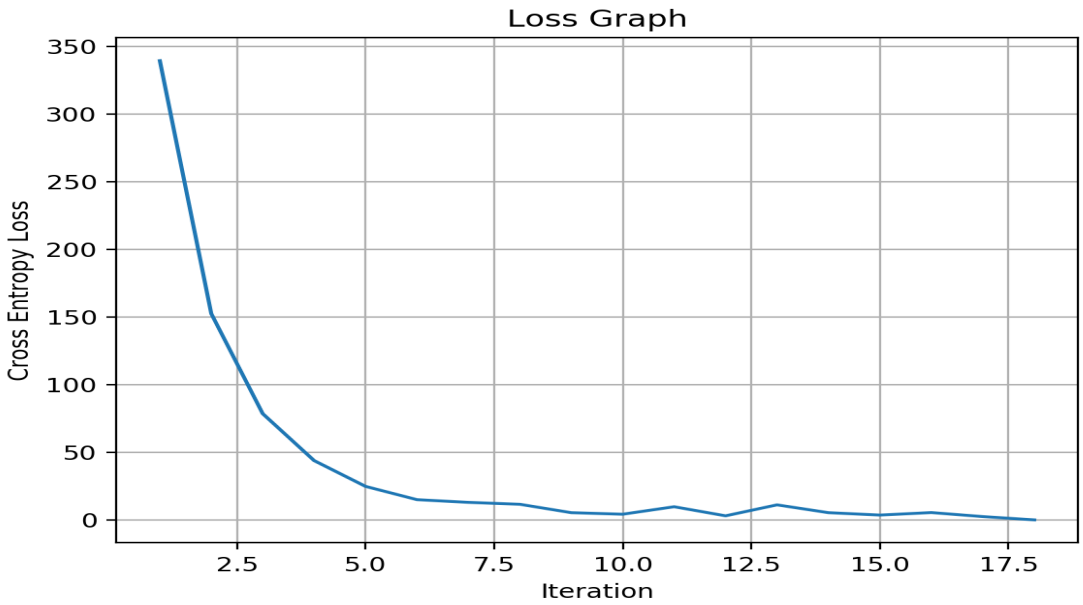


Figure 42: CNN loss function

4.6.2. Inception 1b Loss Function

Following loss graph shows the gradient descent and loss function graph of Inception 1b model training. In this training process we have used the adam optimizer and softmax cross entropy as loss function. We have seen that the loss function approaches to zero after 38th iteration.

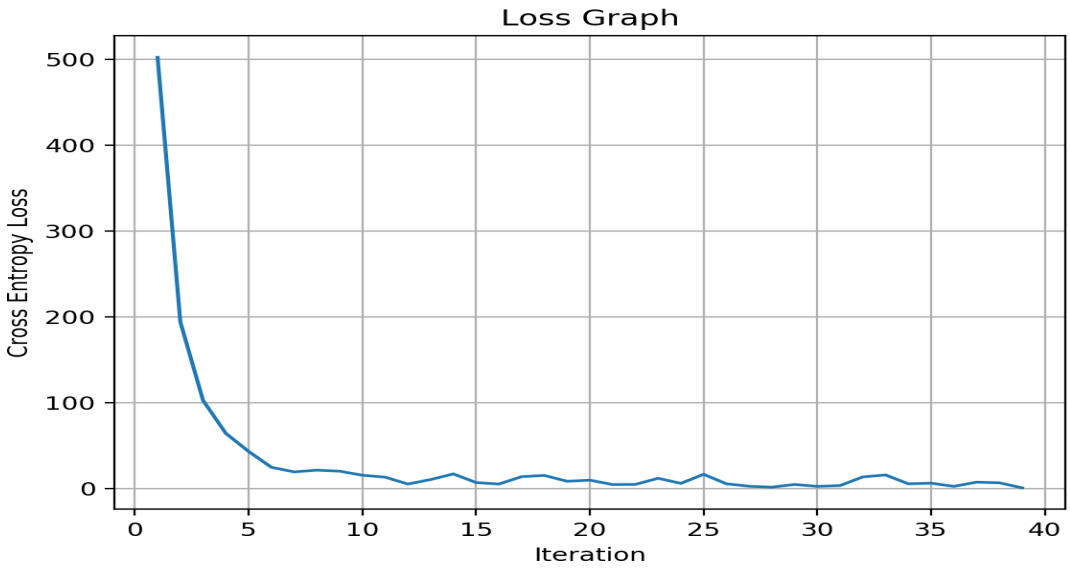


Figure 43: Inception 1b loss function

4.6.3. Inception 5b Loss Function

Following loss graph shows the gradient descent and loss function graph of inception 5b model training. In this training process we have used the adam optimizer and softmax cross entropy as loss function. We have seen that the loss function approaches to zero after 46th iteration.

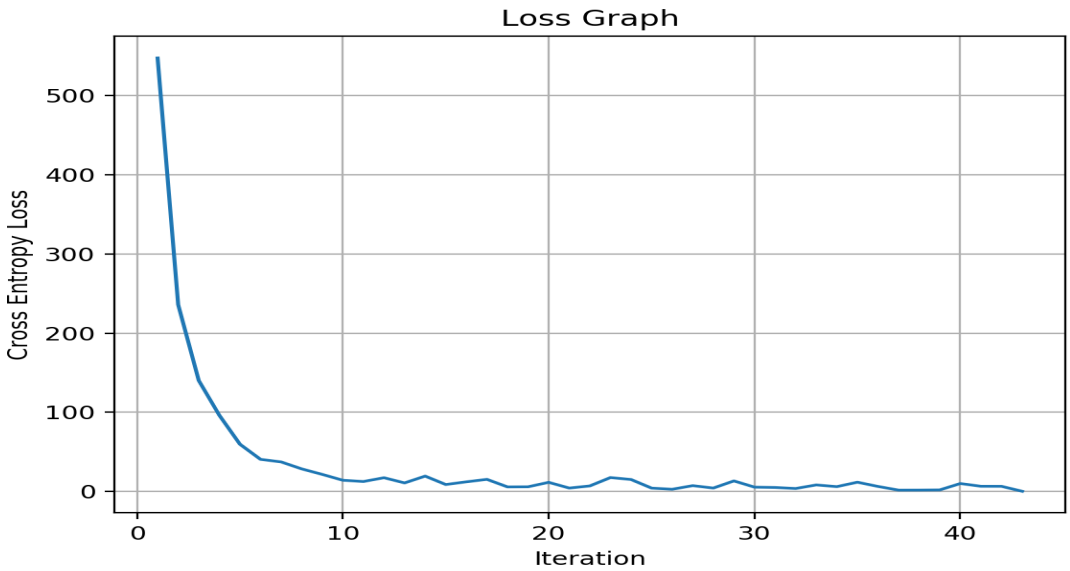


Figure 44: Inception 5b loss function

4.7. Validation Accuracy

Validation accuracy is measured in every iteration of deep learning model training. This accuracy shows that how our model is getting trained over the dataset iteratively and at each stage of training how well our model is.

4.7.1. CNN Validation Accuracy

Following validation accuracy curve have been observed for CNN3 model. It reaches to 85% validation accuracy at 41st iteration where the loss is zero and we have stopped our training process. Its shows that the learning process has been getting better and better over the iteration.

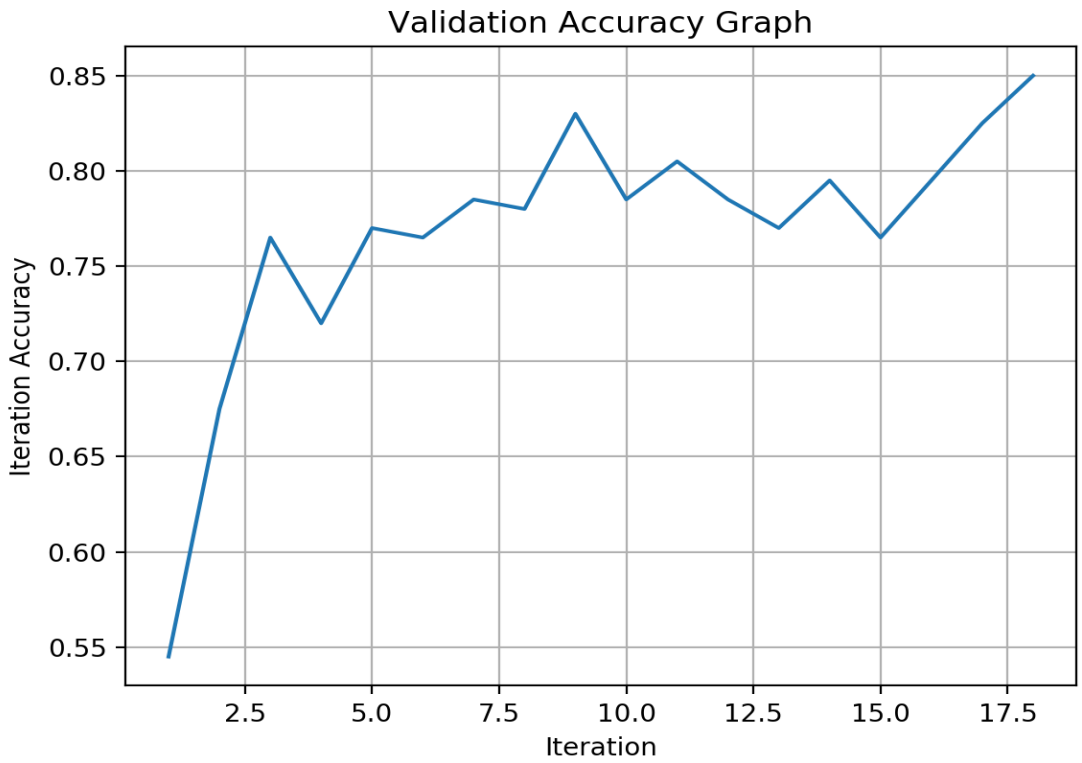


Figure 45: CNN validation accuracy curve

4.7.2. Inception 1b validation accuracy

Following validation accuracy curve have been observed for inception 1b model. Accuracy curve approaches to 90% mark at 38th iteration where the loss is zero and we have stopped our training process. It's also shows that the learning process has been getting better and better over the iteration.

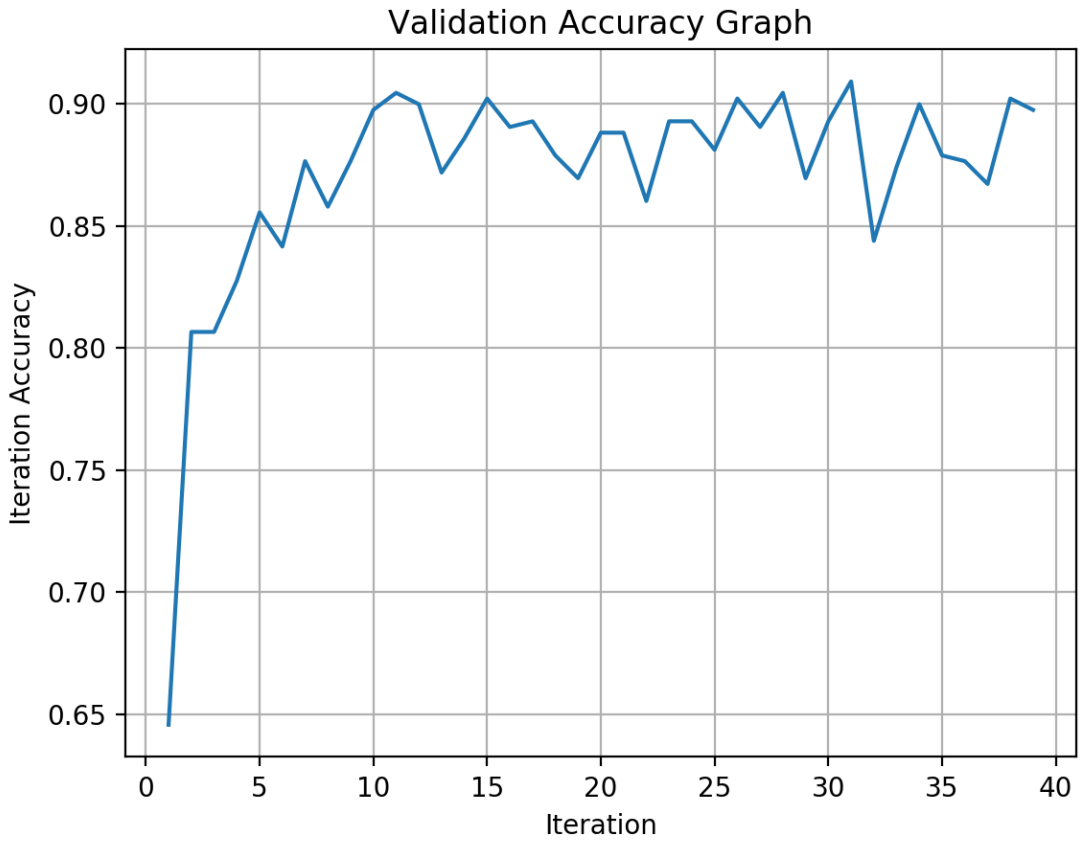


Figure 46: Inception 1b validation accuracy curve

4.7.3. Inception 5b Validation Accuracy

Following validation accuracy curve have been observed for inception 5b model. Accuracy curve approaches to 88% mark at 46th iteration where the loss is zero and we have stopped our training process. It's also shows that the learning process has been getting better and better over the iteration

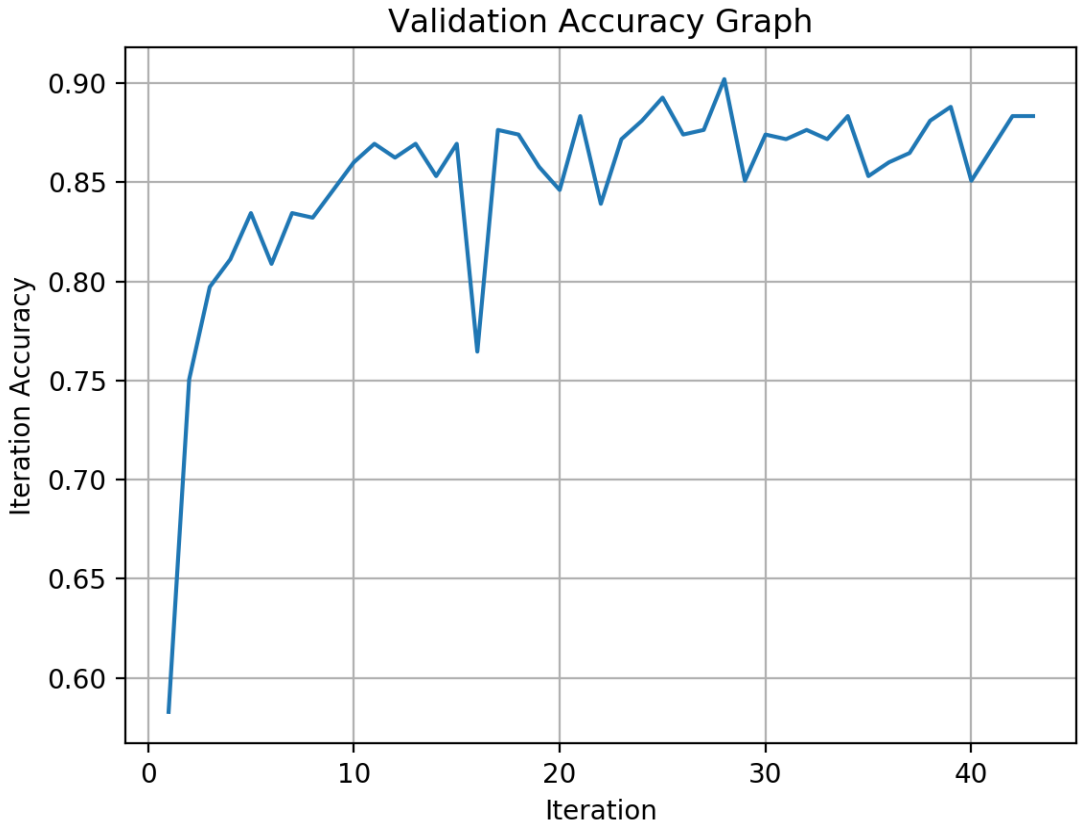


Figure 47: Inception 5b validation accuracy curve

4.8. Conclusion

All the model has been trained with same dataset and the same number of classes. We have seen our models outperforms some of the state of art model and previous implemented model FaceNet in some scenarios. Section 4.4 and 4.5 has the statistics of the analysis for unseen video.

Our CNN and Inception 1b model accuracy which beats the previous state of art models like FaceNet for 90% and 91% respectively and outperforms FaceNet model in some scenarios when the lighting condition is very good, and faces are clearly visible.

With the use of weighted accumulator in the post processing stage also improves the recognition process as compared to the without accumulator.

4.9. Future Work

There are numerous opportunities for future work in moving object detection and face recognition. The most time-consuming part of our research is training deep learning architecture every time when new data set arrives, only if we don't have SVM trained for embedding. I would like to work on this system and improve it further on face recognition and moving object identification as well as object recognition apart from faces. The following improvements can be achieved as follows

- Implement triplet loss function to test the accuracy of the model.
- This system can be enhanced to identify the moving objects rather than only face. It is designed so that, we can train any number of class and visualize in the software as well to test the accuracy.
- It can train classes and would like to improve it by adding sub class prediction. The sub class would be facial pose estimation.
- As we have implemented the reporting tool inside this software, we could improve it to display the video analysis report in future to a great extent.
- Design capsule network to test the face recognition system, as this is the recent paper published by Dr. Hinton.
- Currently our system could detect and recognize any number of face from camera frame or already existing videos. But our system could be improved or enhanced to detect and recognize the small faces which are far away from the view point.
- Currently we have implemented the weighted accumulator for one face recognition, this feature can be enhanced for any number of face recognition.

4.10. Bibliography

1. Vuong Le, Jonathan Brandt, Zhe Lin, Lubomir Boudev, Thomas S. Huang. Interactive Facial Feature Localization
2. Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, Senior Member, IEEE, and Yu Qiao, Senior Member. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. IEEE
3. Going deeper with convolutions, Christian Szegedy Google Inc. Wei Liu University of North Carolina, Chapel Hill Yangqing Jia Google Inc. Pierre Sermanet Google Inc. Scott Reed University of Michigan Dragomir Anguelov Google Inc. Dumitru Erhan Google Inc. Vincent Vanhoucke Google Inc. Andrew Rabinovich Google Inc. CVPR 2014
4. David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of physiology, 160(1):106, 1962.

5. Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4):193–202, 1980.
6. Chunhui Gu, Joseph J Lim, Pablo Arbelaez, and Jitendra Malik. Recognition using regions. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 1030–1037. IEEE, 2009
7. Apple Inc. Kernel convolution, 2011.
8. Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Re- turn of the devil in the details: Delving deep into convolutional nets. arXiv preprint arXiv:1405.3531, 2014.
9. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In Computer Vision–ECCV 2014, pages 346–361. Springer, 2014.
10. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection.
11. GOOGLE Object Detection API. https://research.googleblog.com/2017/06/supercharge-your-computer-vision-models.html
12. Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision.
13. Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. arXiv preprint arXiv:1310.1531, 2013
14. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.
15. Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
16. FaceNet : A Unified Embedding for Face Recognition and Clustering, Florian Schroff, Dmitry Kalenichenko, James Philbin, CVPR 2015.
17. Rapid Object Detection using a Boosted Cascade of Simple Features, Paul Viola; Michael Jones in COMPUTER VISION AND PATTERN RECOGNITION 2001
18. Dynamic Routing Between Capsules; Geoffrey E. Hinton; Google Brain; CVPR 2017
19. ImageNet Classification with Deep Convolutional Neural Networks; Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton
20. Tensorflow API documentation, for learning of tensorflow related functionality for object detection and recognition and basic concept of deep learning, https://www.tensorflow.org/api_docs/

Appendix A

User Manual

Our Face detection and recognition system consists of 6 major application processes.

1. Display Frame
2. Capture Faces
3. Preparing data for training
4. Training of deep learning model user interface
5. Training from console
6. Recognize Faces

Display Frames

This action, is developed to display the frames captured from camera or video in the application screen. In the main screen of GUI application if user deselects the “Recognizer” and “Capture” check box and clicks the start button in the toolbar. The application start displaying the real time camera frame in the application window.

To display a video, user has to select the “video file” option in the Process Flow setting window, which can be open by hot key Ctrl+A. After selecting the video file option from the setting window. Application will open one browse window to select a video file. Supported video files are .avi, .mov, .mpeg and .mp4. Then the application we will read the frames in the video and displays on the screen. At point of time user can change the video file by clicking the “Change Video File” button the setting window (Ctrl+A).

Capture Faces

As frames gets displayed on the from source, by this functionality user can able to instruct application to capture the face that has been detected, for future training. To enable the capture functionality following steps has to be performed.

Note:

1. Setting window can be accessed through hot key Ctrl+A or by selecting toggle button from the menu bar.
2. Capture Window can be accessed through the hot key Ctrl+C or by selecting toggle button from the menu bar as Capture, from View menu.

Following are the steps for capturing faces

1. Select the Source from setting window video file or camera from
2. Select the “Capture” checkbox from the toolbar from the main window.
    a. Selecting both the “Recognize” and “Capture” option from main window, also captures the faces, however this time the recognizer functionality is also in action.
    b. If the “Capture” alone is selected, at this time only capture process will run.
3. Click the “Start” button to start the capture process
4. Capture process options can be changed from capture window as well as from setting window. Following are the options can be changed
5. “Face to capture” which has two options such as “Near to Camera” or “All faces”.
    a. Near to camera: captures the face which is nearer to camera
    b. All faces: captures all the identified faces in the viewing frame.
6. “Detector” which can be changed from setting window
    a. Changing the face detector library for capture process by selecting the detector from the setting window as Dlib or MTCNN for face detector.
7. “Size of the face” can be changed by altering “padding slider” from the setting window in detector post processing block
8. After the face is captured, it will get stored in the following folder.
    a. data -> images -> captured -> <date:time>
    b. There will be a current date time folder, in which all the captured faces will be stored for that period of time. Along with the face images, system will the store the video file for that time period.

Preparing data for training

For a new user the preparation of data is very important. Following are the steps to start the training process from scratch.

1. Open the application app.py
2. Select capture option from the main window and capture as many faces as you can.
    a. Remember while capturing the face images of a person. Only one person has to be present in front of camera. Once the faces of one person is captured then click the stop button and start it again for next person and so on.
3. After the capture process is completed. You can find captured face images under folder “data->images->captured-> **<<many folders with date and time>>**”. There will be many folders depending on how many time the capture process has ran. Each folder holds a particular person image. Make sure specific person’s faces images are not duplicated in multiple folders.
4. Then move or copy all the folders from captured folder to the “data->images->processed” folder.
5. After the data is placed in the processed folder, the class labels of the person will be chosen as per the folder names.

6. Then follow the steps mentioned in the “Training of deep learning model user interface” and “Preparing data for training” section to start the training process.

Training of deep learning model user interface

This module is graphical user interface for training process. Following are the steps to use the GUI for training of deep learning models or svm models

1. Run train_gui.py from base folder
2. This screen contains for each stage of training process and with their configurations such as training method selection, data preparation, pre-processing, data augmentation, data separation, hyper parameter selection, model preparation and training blocks
3. Initially all the block will be populated with the default configuration as per the configuration file “configuration->application->app.config”
4. First block is “Deep Learning Architecture”, it contains following options
    a. Training method selection
        i. Neural Network
        ii. SVM
    b. Neural Network model selection
        i. NN_CNN_3 etc.
    c. SVM Model selection
        i. SVM_RBF_INCEPTION_5B etc.
5. Second block is “Data Preparation”, it contains following options
    a. Data Folder
        i. Raw data folder: used for data pre-processing task
        ii. Processed data folder: from which the training face images will be loaded into the system
    b. Information Button: Click to view the data information those are loaded for training process
        i. Number of class labels for training
        ii. Each class has how many faces to be trained
6. Third block is “Pre-Processing”, it contains following options
    a. Normalization: select the normalization method
    b. Resize: select the resize height and width for the training
7. Fourth block is “Augmentation”, which contains following options
    a. Rotation Angle: how much of rotation is required for augmentation
    b. Vertical and Horizontal flip: is vertical and horizontal flip is required for the augmentation process.
    c. Shearing range, zoom range, fill mode, etc.

    d. Information button: to visualize the augmentation details about the data after the data get augmented
8. Fifth block is “Data Separation”, which contains the following options
    a. Separation logic: used for data separation
    b. Percentage of Separation: Used for separation of training, validation and testing dataset
    c. After the data is separated, an information block will appear with the separation details
9. Sixth block is “Hyper Parameter”, this block has following options
    a. Learning Rate
    b. Regularization beta
    c. Dropout percentage
    d. Optimizer selection
    e. Loss function selection
10. Seventh block is “Model Preparation” this block is used to visualize the model is prepared as per the configuration selected in each block and network configuration as per the configuration file for respective model selected in the first block (All the model configuration files can be accessed from configuration->nn_architecture folder)
11. If the model name is not listed, user has flexibility to create a new model configuration file for training as per the configuration details mentioned in the section 3. Following are the steps to create a new model file
    a. Create a model configuration file under “configuration -> nn_architecture” folder.
    b. Name the configuration file as XYZ.config, XYZ could be any name without any special character or spaces in it.
    c. Go to the configuration.py and add the same name into the respective model list
        i. If the new model file is of SVM type then add the name to self.svm_model_name_list variable
        ii. If the new model is of type neural network then add the name to self.deep_learning_model_name_list variable
12. After all the above steps are successfully completed, then the training process can be started by clicking the “Prepare Model” button followed by “Train” button
13. The progress of the training process will be displayed in the progress bar on the top of each block.

Training from console

Training process can be carried out without the GUI as well. Following steps has to be performed for the training of deep learning models from console.

1. Update the configuration files “configuration->application->app.config” and “configuration->nn_architecture->**<<model_name>>**.config” as per the training requirements and how to update refere to the section 3.
2. If new model is required to train, then follow the 11th step in the previous section
3. Open train.py
4. Update the following line **Train(svm_model='SVM_LINEAR_FACENET', depth=3, skip=True)**
    a. “svm_model”: this named parameter passed to train SVM model, the value passed to this parameter is the configuration file name present in “configuration->nn_architecture” folder.
    b. “deep_learning_model”: this named parameter is passed to train Deep learning model, the value passed to this parameter is the configuration file name present in “configuration->nn_architecture” folder
    c. “depth”: this named parameter is used to instruct the training process to use the depth of the image for training, the default depth value is 1.
        i. 1-grayscale image
        ii. 3-RGB image
    d. “skip”: if this parameter is set to true then system start to train the FaceNet out of the box Linear SVM model from their application folder
5. Run the train.py from base folder

Recognize Faces

Third and the most important feature of our system is recognizing person. “Recognize” checkbox in the main window has to be selected to enable the recognition task. The system starts recognizing the person after clicking the start button and the recognized person details can be seen in the prediction box (Ctrl+P to open the prediction box window if it not visible).

This module also has some options, following are the options those can be accessed and changed as per the user need in real time.

1. Number of Recognition: this option instruct system, to recognize those many faces in the frame if available. This preference has 1 to 5 and all option.
2. Number of Prediction: this option instruct system, to change the number of prediction to be viewed in the prediction frame. Like, if the Number of Prediction is selected as 2, then in the prediction frame, each detected face will have top 2 predictions displayed on the screen.
3. Change Camera: by changing this option system will swap the available camera connected to computer for the input. Default is 0.

4. Face Recognition Method: this option will change the method of recognition in the application. By default, this option is set to “nn”. All the recognition activity will go through CNN model. We have other options like
    a. “cnn”: CNN model
    b. “inception1b”: inception 1b model
    c. “inception5b”: inception 5b model
    d. “svm”: svm model trained over inception 5b embedding
    e. “svm_FaceNet”: SVM model trained over FaceNet embedding
    f. “FaceNet”: FaceNet out of the box recognition platform trained over our dataset.
5. Face Detection method: On the selection of detector, the face detector will be changed in the application in real time. Two detector API have been used such as MTCNN and DLIB.
6. Capture Report: On the selection of this combo box to Yes, video analysis report for the period of recognition activity will commence.
7. Accumulator Status: On the selection of this checkbox, post processing module will start to use the accumulator functionality
8. Accumulator Size: On the selection of accumulator size, post processing module will accumulate those many recognized faces before it produces the final prediction.
9. Rotation of face: By selecting this option user can instruct the system to pass the rotated face for recognition task instead of passing the original tilted face.
10. Weighted Accumulator: On the selection of this checkbox, post processing module will start to use the weighted accumulator for the recognition task.
11. Display Feature Points: On the selection of this checkbox, the feature points such as eyes, nose and mouth point will be displayed on screen.