Project Report On

Interacting with Software using Gestures

"A dissertation submitted in partial fulfillment of the requirements of Bachelor of Technology Degree in Computer Science and Engineering of the Maulana Abul Kalam Azad University of Technology for the year 2017-2018"

Submitted by

Sauradip Nag (10200114044)
Pallab Kumar Ganguly (10200114025)
Swati Ghosh Hazra (10200114056)
Aditya Jain (10200114003)

Under the guidance of

Dr. Kousik Dasgupta
Assistant Professor
Dept of Computer Science & Engineering
Kalyani Govt. Engineering College

Mr. Tamojit Chatterjee
Software Engineer
Indeed

Department of Computer Science and Engineering
Kalyani Government Engineering College
(Affiliated to Maulana Abul Kalam Azad University of Technology, West Bengal)
Kalyani - 741235, Nadia, WB

Certificate of Approval

This is to certify that this report of B. Tech. Final Year project, entitled **"Interacting With Software Using Gestures"** is a record of bona-fide work, carried out by **Sauradip Nag, Pallab Kumar Ganguly, Swati Ghosh Hazra and Aditya Jain** under my supervision and guidance.

In my opinion, the report in its present form is in partial fulfillment of all the requirements, as specified by the ***Kalyani Government Engineering College*** and as per regulations of the ***Maulana Abul Kalam Azad University of Technology***. In fact, it has attained the standard, necessary for submission. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report for B. Tech. programme in Computer Science and Engineering in the year 2017-2018.

Guide / Supervisor

____

Dr. Kousik Dasgupta

Department of Computer Science and Engineering
Kalyani Government Engineering College

____

Examiner(s)

____

Head of the Department
Computer Science and Engineering
Kalyani Government Engineering College

Certificate of Approval

This is to certify that this report of B. Tech. Final Year project, entitled **"Interacting with Software using Gestures"** is a record of bona-fide work, carried out by **Sauradip Nag, Pallab Kumar Ganguly, Swati Ghosh Hazra, and Aditya Jain** under my supervision and guidance.

In my opinion, the report in its present form is in partial fulfillment of all the requirements, as specified by the ***Kalyani Government Engineering College*** and as per regulations of the ***Maulana Abul Kalam Azad University of Technology***. In fact, it has attained the standard, necessary for submission. To the best of my knowledge, the results embodied in this report, are original in nature and worthy of incorporation in the present version of the report for B. Tech. programme in Computer Science and Engineering in the year 2017-2018.

Guide / Supervisor

Tamojit Chatterjee

Tamojit Chatterjee

Software Engineer

Examiner(s)

Head of the Department
Computer Science and Engineering
Kalyani Government Engineering College

ACKNOWLEDGEMENT

First of all, we would like to thank our guides, Mr. Tamojit Chatterjee, and Dr. Kousik Dasgupta for their constant support and guidance, which have been invaluable throughout the duration of the project and without whom we would not have been able to bring this project to fruition.
We also thank our parents and families for their sanguine attitude and encouraging us in all our endeavors.
We also express our gratitude to all the professors, technical and laboratory staff and attendants of the Computer Science and Engineering Department, for providing technical and hardware assistance and for their unwavering commitment to help all students as much as possible.
Finally, we express our gratitude to Dr. Saurabh Kumar Das, Principal, Kalyani Government Engineering College, and Prof. Malay Kumar. Pakhira, H.O.D., Department of Computer Science and Engineering.

Sauradip Nag
Roll: 10200114044
Regn. No. 141020110044

Pallab Kumar Ganguly
Roll: 10200114025
Regn. No. 141020110025

Swati Ghosh Hazra
Roll: 10200114056
Regn. No. 141020110056

Aditya Jain
Roll: 10200114003
Regn. No. 141020110003

Declaration by Authors

This is to declare that this report has been written by me/us. No part of the report is plagiarized from other sources. All information included from other sources has been duly acknowledged. I/We aver that if any part of the report is found to be plagiarized, I/we are shall take full responsibility for it.

……………………………...
SAURADIP NAG
ROLL: 10200114044

……………………………...
PALLAB KUMAR GANGULY
ROLL: 10200114025

……………………………...
SWATI GHOSH HAZRA
ROLL: 10200114056

……………………………....
ADITYA JAIN
ROLL: 10200114003

Abstract

Gesture recognition is a field in Computer Science which deals with interpreting gestures performed by humans by the use of mathematical models and algorithms. Current work in this field comprises of hand-gesture recognition, and emotion classification from the face. This report presents a method for hand gesture identification and classification of hand gestures, so that they can be used to interact with software. The proposed method involves two stages. First, from video feed, samples are extracted, and the area containing the hand is detected using a HAAR-like cascade classifier. Thereafter, we use a Convolutional Neural Network to classifier to classify the detected hand region (which is performing a gesture) into one of four gestures, namely, the closed fist, the index finger raised, the first two fingers raised in a V-shape, and the thumb raised. The identified gesture can then be mapped to a software action. In this report we describe in detail the datasets used to train our models, the methods used to detect the hand region from the video feed, as well as the design, implementation and working of the Convolutional Neural Network used to classify the gestures. Finally, we describe the results of our work and compared them against standard Architectures for judging the Effectiveness of t Proposed Method.

Keywords: Computer Vision, Deep Learning, Gesture recognition, HAAR Cascade, Convolutional Neural Network.

CONTENTS

CHAPTER 1 INTRODUCTION 1

1.1 Motivation 1

1.2 Background 3

1.3 Summary of prior works 3

1.4 Summary of present work 5

1.5 Organization of the thesis 6

1.6 Resources used. 6

CHAPTER 2 THEORETICAL BACKGROUND 7

2.1 Hand detection 7

2.2 Gesture Recognition 10

CHAPTER 3 PROPOSED WORKFLOW 15

3.1 Hand Object Detection using HAAR Cascade Classifiers 16

3.2 Gesture Recognition using Convolutional Neural Network 17

3.3 Mapping of Software Applications to Labeled Gesture 19

CHAPTER 4 RESULTS AND EXPERIMENTATION 22

4.1 Collection of Raw Data for Creating Images 22

4.2 Preparation of Dataset for HAAR Classifier 22

4.3 Training a HAAR Cascade Classifier for Hand Detection 24

4.4 Testing and Evaluation of Cascade Classifier 25

4.5 Data Augmentation for Convolutional Neural Network 30

4.6 Data Preprocessing for Convolutional Neural Network 31

4.7 Architecture of Convolutional Neural Network and Training 33

4.8 Testing and Evaluation of Convolutional Neural Network 35

CHAPTER 5 CONCLUSION AND FUTURE WORK 39

5.1 Conclusion 39

5.2 Future Work 39

REFERENCES **40**

PLAGIARISM REPORT **42**

1. INTRODUCTION

Since the invention of Computers, it slowly became tightly integrated with our daily lives. And with every passing day, with every new technology, innovation, we must upgrade ourselves. Nowadays focus has been given to make everything digital from analog. Hence reliance of Human Beings on Analog Devices must be reduced. Vision based systems are very popular nowadays and now computer can "see". Hence the interaction of Human with the Machines has been quite enriching and user friendly affair. Computer Vision based systems must ensure that the Working of the Software is Real Time and there exists no lag, must be accurate and robust to any environment. In the 20th Century, there has been a huge influx of Man-Machine Interactions. However, it brings along a complex set of challenges. Researchers are currently working on such challenging problems. Hence, cost of productions of such software, controllers, devices are quite high . This is widely used in Image, Gesture, Speech, and Sign Language Recognition. Just like speech, gestures are a mode of communication between humans. In fact, some research suggests that human communication may have evolved from gestures rather than speech. It is therefore unsurprising that several technologies in the recent past have tried to revolutionize human-machine interaction (HMI) using gestures.

1.1 Motivation

Ever since the advent of computing tools, there has been a consistent endeavor to improve on the communication barriers between man and machine. Computers of the 60's had only one mode of interaction between the user and the computer, the command-line interface. This prevented the computer from achieving the widespread use it has achieved today, and computers were used only in niche applications such as defense and space projects. In the mid-80's however, graphical user interfaces, or GUIs were introduced, and the computer was accessible and usable by the general public for everyday tasks.

In the last decade, there has a been a boom in mobile computing, where users interact with using touch-screen display, another path breaking human-machine interaction method. The next big thing in Human Computer Interaction (HCI) could be touchless devices, where users interact with their mobile devices or traditional computers using gestures, thus greatly simplifying the process. Another area where gesture recognition could find use is in the fledgling field of Augmented Reality (AR). Indeed, as of today, several of the leading consumer electronics manufacturers are using AR natively in their products.

Therefore, we see that there is ample scope and reason for research in the field of gesture recognition, and as a result, several approaches toward gesture recognition, both hand gestures and facial gestures are being explored. Our work is a unique approach to the same goal, and we believe it is a positive step toward the solution of this problem.

1.2 Background

Most of the early work in gesture recognition is very hardware dependent, and in some cases requires the user to wear some type of apparatus to track and recognize his/her movements and gestures. There are several problems to this approach, the most obvious being the user having to wear the apparatus, and this carries out the natural and intuitiveness that one would want. Secondly, this type of approach is not cost effective, as it requires specialized sensors to detect and recognize human gestures. Thus, most of the recent works have tried to capture footage/images of the users performing the gesture and then classifying that gesture from the footage/images using mathematical models and algorithms. This eliminates the need of specialized hardware, as most computers and mobile devices are equipped with a digital camera, enabling easy access to video/images.

Gestures can be static or dynamic. Dynamic gestures are basically a sequence of static gestures. Dynamic gestures are much more complex to analyse but are more suitable for real time applications. Static gestures, on the other hand are simple, yet useful for basic tasks, and they have the additional advantage of being less computationally intensive. Several methods have been explored for recognition of static gestures from images. Our work focusses on recognition of static gestures from video feed.

1.3 Summary of previous work

Gesture Recognition has grabbed many eyeballs in recent history due to its wide application in the fields of Human Computer Interaction (HCI), Robotics and User Experience (UX). We can divide the existing approaches related to this literature broadly into hardware intensive approaches and non-hardware intensive approaches based on how features are extracted. Hardware Intensive approaches normally used special hardware like depth cameras, Kinect sensors for capturing features from video stream and then process the Images to predict the gestures. non-hardware Intensive based approaches on the other hand used real time locally available and cheap source such as web-cam, smartphone cameras, digital cameras etc. to fetch data and features are extracted using standard Image Processing or Deep Learning Approaches and then used various other techniques to predict the gestures.

The works which have been done in recent past has the usage of both traditional Image Processing and Deep Learning techniques for gesture recognition, however Hidden Markov Model has been used more frequently. Hardware intensive approaches like Ren et al. [1] used Kinect Sensor to detect hand shapes and then proposed finger Earth Movers distance metric to remove noise from data stream. However, this approach will work only for open hand gestures. Zhao et al. [2] proposed Depth Camera to obtain Structure Steaming Skeletons (SSS) features for human gesture recognition. Plawiak et al [3] adopted a methodology to track gestures using a special device called DG5 VHand Glove which has 10 sensors. The above mentioned approaches mainly extracts feature using some special hardware, which makes these approaches far from practical. Since the cost of the apparatus used is high, these methods find very less application in day-to-day usage. However, these methods are quite accurate owing to dependence on hardware. Less hardware intensive approaches are also used recently to recognize gestures. For example, Sefat et al. [4] proposed the use of color space and HOG feature to detect gestures, but the color space is very sensitive to lighting conditions, hence the robustness of the approach cannot be guaranteed. Simao et. al. [5] introduced a novel concept of detecting hand gestures using unsupervised approach. Motion based features and use of Genetic Algorithms to detect gesture makes this work quite interesting. Hence less work on interaction of softwares with machines has motivated us to work on this interesting and complex problem.

1.4 Summary of present work

The approach we propose consists of two stages. First from the video, frames are extracted, and from the frame, the position of the hand is detected. For this purpose, a HAAR-cascade classifier was trained to distinguish hand vs. non-hand images. These trained classifiers are then used to get a bounding box indicating the region of the hand in the frame. Using the co-ordinates of the bounding box, the hand region is the cropped out of the frame. If no hand is detected (i.e., no gesture is performed) no gesture is reported.

However, if a hand region is detected in the frame, it is passed on to the next stage in which we use a CNN to classify the gesture. For simplicity, only four gestures have been used: the hand closed in the form of a fist, the index finger raised, the first two fingers raised in a V-shape, and the thumb raised. A CNN was trained on several thousand images for this purpose. The cropped out image is then fed as the input to the Convolutional Neural Network. Now we are not passing the cropped out Image directly , we mask out the Skin Region from the Cropped Image using HSV Colorspace and eliminate background Noise . After this step we pass the Masked out Skin Image into 6-Layered Deep CNN . This performs feature extraction and classification of the image and gives a label for the image provided to it. This label is then used to initiate a VLC Media Player Application which must be Preinstalled in the Computer . The Gestures are Mapped to Open , Play Video and Close the Application .

1.5 Organisation of thesis

This report/thesis is organized as follows: In Chapter 2, a brief theoretical background of the theoretical concepts which are used is discussed. Specifically, a framework for real-time object detection, and the use of Convolutional Neural Networks for image classification are discussed. In Chapter 3, a workflow is proposed to recognize gestures from video feed by the authors. In Chapter 4, the experimentation performed in order to implement the proposed framework is elaborately discussed. In the next chapter, Chapter 5, the results of the experimentation performed are dealt with, and some advantages and disadvantages of the proposed method are discussed. Finally, in Chapter 6, future possibilities are explored, and some conclusions are drawn.

1.6 Resources used

For the purposes of hand detection, we used a machine with a 2.7 GHz, Intel Core i7 CPU, with 16 GiB of memory, running Linux (Ubuntu). For the classification and feature extraction using CNN we used a machine with 2.3 GHz Intel Core i5 CPU, with 4 GiB of memory and a Nvidia GeForce 940M GPU on Linux (Ubuntu). Images for training were mostly captured on a mobile phone camera at 1280x720 resolution. The following libraries and software were used:

Table. 1.1: Software Libraries used

Purpose	**Library/Software Used**
Extraction of frames from video	FFmpeg
Hand Detection	OpenCV 3.1.0
	Pillow (forked from PIL)
	Numpy
	Matplotlib
Gesture Classification	TensorFlow
	TFLearn
	Numpy
	OpenCV 3.1.0

2. THEORETICAL BACKGROUND

A method for interacting with software using static gestures from video feed is proposed, requiring minimal hardware, but achieving high accuracy comparable to state-of-the-art algorithms. The proposed approach is broken into three stages: first hand detection is discussed, then gesture classification is covered, and finally, mapping of gestures to software is performed. In this chapter, the underlying theoretical concepts are explained, which will enable the reader to relate with the forthcoming matter.

2.1 Object Detection using HAAR Cascade Classifiers

The first stage of the proposed approach consists of hand detection from video feed. It is pointed out to the reader that is essentially a binary classification problem. Most of the work in the field of object detection is based on the seminal work of Paul Viola and Michael Jones [Ref. No.]. In [Ref. No.], an algorithm to achieve robust, real-time detection is explored. The most important advantage that HAAR Cascade classifiers based on Viola-Jones algorithms have is their speed of detection. The Viola-Jones algorithm has four features of interest:

a. HAAR features

Like previous approaches [Ref. Papageorgiou], Viola-Jones' approach uses HAAR features. HAAR features are computed by sliding a window with each HAAR-like feature kernel over the image. The value of the feature is calculated as the difference of the sum of the pixel values under the regions defined by the kernel. For example, in Fig. 2.1, the HAAR feature is calculated as:

$$value = \left| \sum_{pixels} Black\_Region - \sum_{pixels} White\_Region \right| \dots (1)$$

Simple HAAR-features were used: two-rectangle features, three-rectangle and four-rectangle features, to compute a set features for the frame. However, for a standard window of 24x24, the set of features is overcomplete, with over $1.8\text{x}10^5$ features.

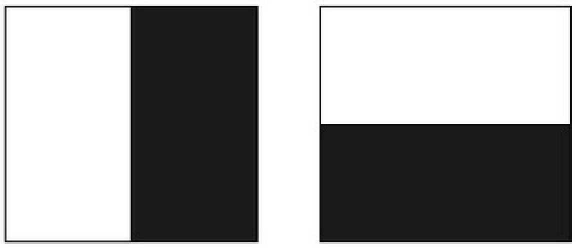


Fig. 2.1 : Example of HAAR-like feature for edge

b. Creating Integral Image

One of the main contributions of [Ref. No.] is the concept of Integral Images. To compute HAAR features, it is required to calculate the sum of pixel values very frequently. To speed up this process, Viola-Jones framework introduced the concept of integral images, which are summed area tables. For each pixel value (x, y), the sum of all pixel values above and to the left of (x, y) are calculated and stored in a look-up table. This look-up table is therefore a intermediate representation of the original image This intermediate representation of an image enables calculation of HAAR features in constant time. Any time a sum over an area is required, it can be done in constant time over a single pass over the image as :

$$I(x, y) = i(x, y) + I(x, y-1) + I(x-1, y) - I(x-1, y-1) \quad \ldots (2)$$

$$area\ of\ ABCD = I(D) - I(C) - I(B) + I(A) \quad \ldots (3)$$

Where I(x, y) is the value of the sum of all pixels above and left of the pixel at (x, y), and i(x, y) is the pixel value. Refer Fig. 2.2 for the explanation of (2) and (3)

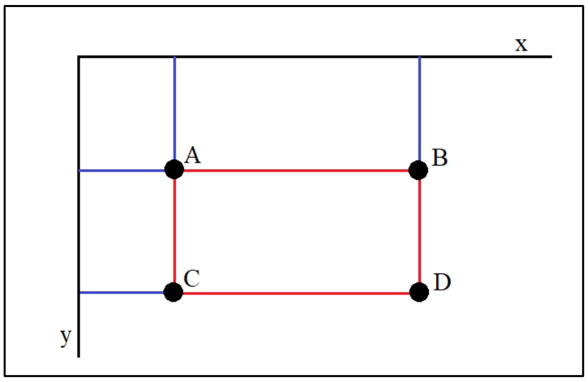


Fig. 2.2: Calculation of Integral Images

c. Adaptive boost Training

Another significant contribution of [Ref. No.] is the algorithm for building a classifier by choosing a small set of significant features using Adaboost [Adaerf]. As was pointed out earlier, the number of HAAR features calculated is over complete, the learning algorithm must sift out a large number of unimportant features and select a small set of important ones. This is done by having a weak classifier algorithm, such that it can only use a single feature. This weak classifier is only just better than a random guess. However, on many iterations, a new feature, and hence a new weak classifier is chosen. The final classifier, the boosted classifier, is a weighted combination of all such weak classifiers. After each iteration in the training process, a weight equal to the error or loss on the current data item is assigned to the data item. Finally, the boosted classifier is created as the weighted sum of the above classifiers.

d. Cascading Classifiers

To increase the speed of classification, a method is developed by combining a cascade of classifiers, in which each stage uses successively more complex classifiers. The detector prepared by Viola and Jones had 38 such stages. Each stage has more features than the previous stage. If a sample passes this stage, it is passed on to the next stage. Thus, if a sample passes all stages, it is classified as containing a positive. It is therefore noted that each stage eliminates false positives. Thus, a good stage should have 100% true positives and some false positives. This cascade of classifiers eliminates obvious negatives at early stages, and only retains more promising regions that pass earlier stages for more complex computation.

In our proposed approach, the concept of object detection was modified to detect hands from videos. For this purpose, a HAAR cascade classifier was trained using several thousand images. This is discussed in detail in Chapter 3. The resultant classifier was used to get a bounding box from the image, marking the position of the hand in the image (which is actually a video frame). This bounding box is then passed on to the next module in the pipeline, which performs gesture recognition.

2.2 Image Recognition and Convolutional Neural Networks

The second stage of the proposed approach is the recognition of a gesture from the image. At this stage, the bounding box is obtained from the output of the object detection module described earlier. In this stage, this bounding box is passed on to a image classifier, which recognizes it as one of four valid gestures, or as an invalid gesture.

One of the most revolutionary ideas in image classification is due to Alex Krizhevsky *et al.* [17]. While most existing approaches to image classification relied on traditional machine learning techniques, they fail to encompass the complexity and variability of objects in realistic settings. The novelty of [17] was in that a Convolutional Neural Network (CNN) was used to classify images from the ILSVRC-2012 dataset. An error rate of 15.3% was achieved using a CNN by Krizhevsky. A Convolutional Neural Network is essentially a few convolutional layers in front of an Artificial Neuron (ANN). The design of a CNN generally consists of the following three stages:

1. Convolutional Layer: Convolutional Layers apply some kind of convolutional operation to the image and passes the result to the next layer. Each convolutional layer is like a filter applied to the input. The convolution operation is explained in Fig 2.3. A small kernel can be thought to be convolved over the entire image, producing an activation map. For each position of the kernel over the image, the pixel values and the weights of the kernel are multiplied element-wise and added. It is also

to be noted that the depth of the filter should be the same as the depth of the image, so if the input image has dimensions 32x32x3 for a 3-channel (RGB) image, the kernel also has to have depth 3. After the entire convolutional operation, an array with depth 1 is obtained. This is called the feature map for one filter. Stacking all the activation maps for the entire depth of the input using various filters forms the output of the convolutional layer.

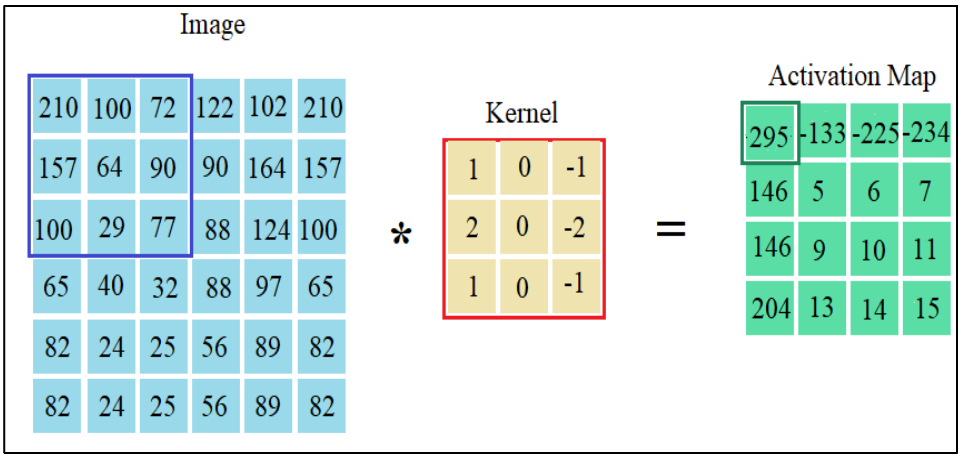


Fig 2.3: Convolution with stride size 1. Each time, the kernel is shifted one pixel

2. Pooling Layer: Most often, a convolutional layer is followed by a pooling layer. In this layer, down-sampling is performed. The most common form of pooling is max pooling. In max-pooling, the image is partitioned into non-overlapping rectangles of size 2, and the maximum of the four pixel values is taken. Another common technique of pooling is L2-pooling. L2 pooling is similar to max-pooling except that instead of taking the maximum value from the region of pixels, the square root of the sum of squares of the pixel values is taken. The purpose, however, is the same as max-pooling: To get a representative value for that region over which it is applied, to reduce the feature space. The pooling layer serves to reduce the size of the output of the convolutional layer, thereby reducing the number of parameters. This reduces the computation and also helps prevent overfitting. Pooling also provides translational invariance []. Pooling is explained in Fig 2.4.

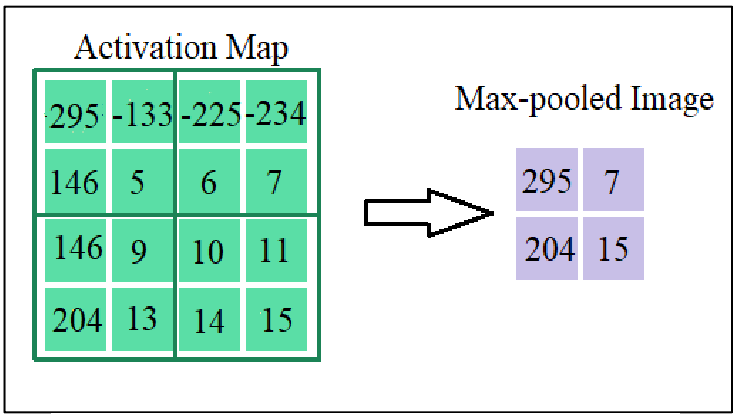


Fig 2.4: Max-pooling with a 2x2 kernel and stride of 2

3. Fully Connected Layer(s): This layer is essentially a hidden layer of the ANN, which performs classification from the features selected from the convolutional layers. This layer takes an input volume which is passed to it by the preceding layer (pooling or ReLU/Softmax) and outputs a k-dimensional vector, for a k-class problem. In this part of the CNN, each unit or neuron is connected to every other neuron in the next and previous layer. It is pointed out that this is in contrast to the Convolutional Layers, where neurons are sparsely connected to each other. Activations are calculated by matrix multiplication of pixel values and weights. The final Fully Connected layer output the scores/predictions for each class.

Initially the network is set up with random weights (and biases). When the input images are fed through the CNN, it predicts a class for each of the images. The prediction is compared against the actual label, and a loss is calculated over the images. Several loss functions are used commonly, prominently SVM loss [ref], Cross Entropy Loss [ref] Mean Squared Error Loss[ref]. They are described as:

$$Multiclass\ SVM\ Loss\ \mathcal{L}_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta) \qquad \dots (4)$$

$$Cross - entropy\ Loss\ \mathcal{L}_i = -f_{y_i} + log \sum_j e^{f_j} \qquad \dots (5)$$

Where in each case, x_i is the input pixel-array and y_i is the label that specifies the index of the correct class, and the hypothesis function s = h(x_i, W), and Δ is a hyperparameter. It is noted that the choice of the loss function is not fixed but is chosen by the author according to his problem.

This preliminary process of passing training data to get predictions and comparing is termed as a forward pass over the network. Next, a backward pass is performed over the network, in order to calculate the gradients over each layer progressively.

The gradient is calculated such that that the overall loss function is minimized. The weights are then updated so as to minimize the objective function (loss). One of the popular methods in which this is done is the Gradient Descent algorithm and its many forms. (Online, Stochastic, etc.). In particular, for the i-th epoch (one entire forward and backward pass is called an epoch), gradient descent updates the weight of the j-th layer as:

$$w_j^i = w_j^{i-1} - \alpha * \nabla_{w_j^{i-1}} \mathcal{L} \qquad \dots (6)$$

where $\mathcal{L}$ is the total loss, α is the learning rate, and $\nabla_{w_j^{i-1}} \mathcal{L}$ denotes the gradient of the loss function $\mathcal{L}$ w.r.t. the weights of the current layer j, evaluated as per (4), (5).

The learning rate is an important hyperparameter, as it controls the time taken for training as well as the accuracy of the gradient descent algorithm. When the learning rate is small, the gradient descent makes a smaller step in the direction of the gradient, and as a result it takes time to converge to the minimum. However, when the learning rate is larger, the algorithm takes larger steps in the direction of the gradient. While this generally results in faster convergence, taking larger steps may cause the algorithm to overshoot the minimum and the loss may not decrease with more iterations, thus giving inaccurate results. Also, it is noteworthy that while from a theoretical perspective, the Loss is calculated over the entire dataset, in practice this is hardly the case. This is because evaluating the loss over a dataset consisting a million or more images consumes far too much computational resources than is practical. Therefore, some variation of Gradient Descent is commonly employed in which the loss over a small ‘batch’ of training examples is evaluated. The batch size is also a hyperparameter in such cases. If one example is used at a time, the process is called on-line gradient descent, because weights are updated constantly after each training example has been processed.

It is pointed out that in the problem of image classification, there arise problems due to rotational variance, translational variance, scaling, illumination variations. In a CNN, these problems are handled very robustly if there is adequate training data available. This is due to the feature extraction performed in the convolutional layers enable intermediate representations that capture the above mentioned variances to a large extent.

In our proposed approach, image classification is used to classify the bounding box obtained from the previous section into a gesture. It is to be remembered that from the result of the previous section, a bounding box is obtained which contains a hand performing a gesture. For classification, a Convolutional Neural Network is designed, consisting of five (5) Convolutional Layers and two (2) Fully Connected Layers. This is described in further detail in Chapter X. This CNN was trained with images of gesturers performing four different gestures, namely, the fist closed, the index finger raised, the first two fingers raised in a V-shape, and the thumb raised. The output of the hand detection module is fed into the trained model. The CNN predicts a gesture for the bounding box that is fed to it, and this gesture is passed on to the next module, which performs some software action based on the gesture.

3. PROPOSED WORKFLOW

A method for interacting with software using static gestures from video feed is proposed, requiring minimal hardware, but achieving high accuracy comparable to state-of-the-art algorithms. Since past works related to this involved hardware in extraction of Gestures , we relied on Deep Learning based Methods which outperforms the existing methods simply because Neural Networks can learn Features which are invisible to Human Eye and Hardware sensors . The proposed Model processes 15 Frame per second (15 fps) . The proposed approach is broken into three stages: first hand detection is discussed, then gesture classification is covered, and finally, mapping of gestures to software is discussed. The Overall Architecture of the Proposed Method is Given in Fig 3.0

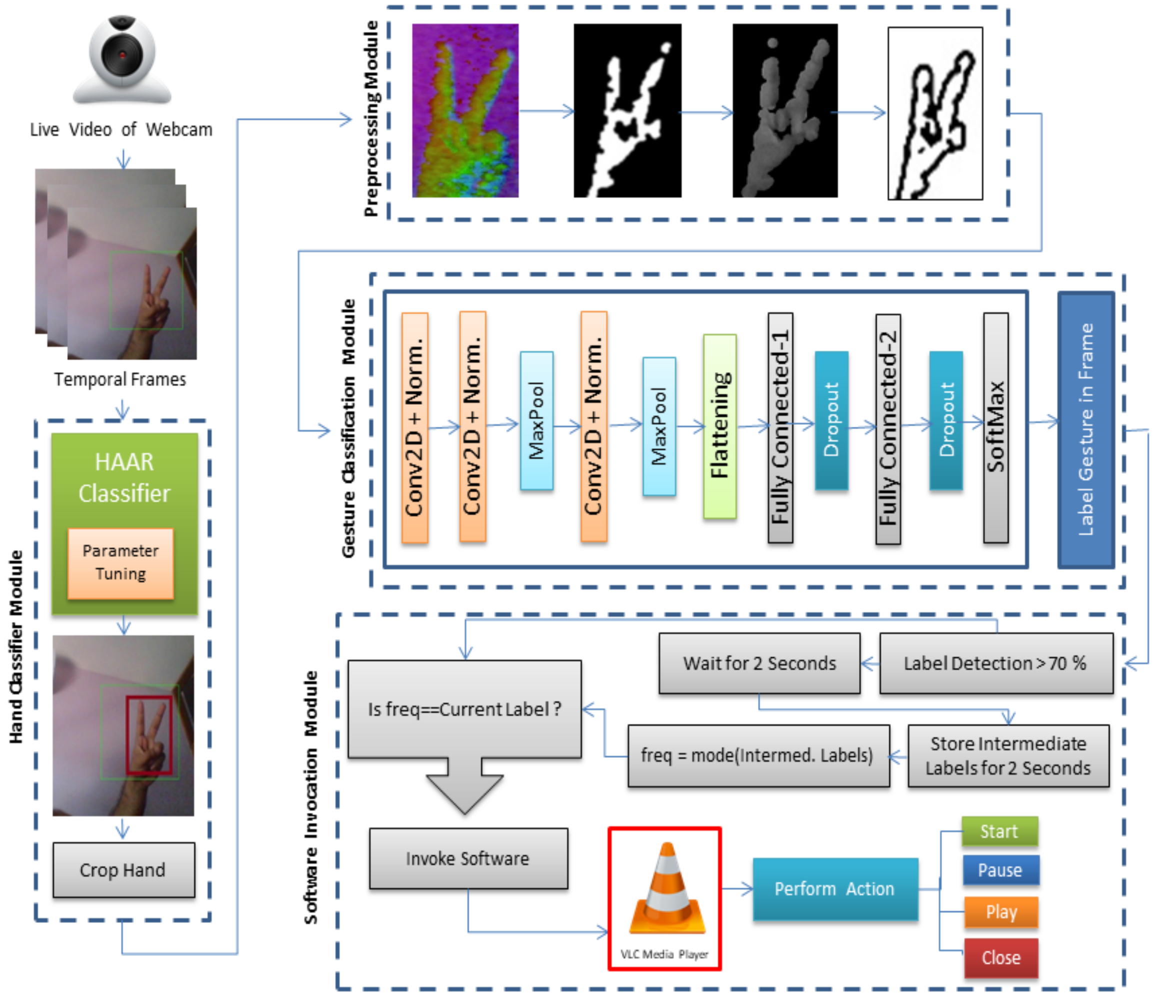


Fig 3.0: Illustrates the Architecture of Proposed Method

3.1 Hand Object Detection using HAAR Cascade Classifiers

Since the important part of this literature is hand gestures, so effective way of detecting hand object is of paramount importance. There are many existing methods of detecting Hand from a live video stream as discussed in previous works but having a dedicated classifier for hand object is crucial for gesture recognition. The methods which involves skin color matching and segmentation is not robust to skin color of various races. Here deep learning outperforms this approach and makes this process of hand object detection free from background, orientation and skin colour.

For creating a dedicated hand classifier, we selected HAAR Classifier which uses HAAR like Features which are discussed in Section 2.1(a). Using this classifier, we can detect hand images from live video stream. So, input in this stage is the temporal frames from Live Video Stream captured using standard Laptop Web Camera. These temporal frames are fed directly to the Trained Hand Classifier. The Training of the classifier is explained in details in next Chapter. In this step, these temporal frame images are preprocessed by gray scaling them and resizing them to 100 x 100 size. The Trained Object Detector scans the image in a sliding window fashion to return Confidence Score and the Bounding Box coordinates of the Hand Object as Displayed in Fig 3.1. The detected box of Hand Object is padded with a fixed threshold and cropped out of the Temporal Frame. This will be the input to the Next step.

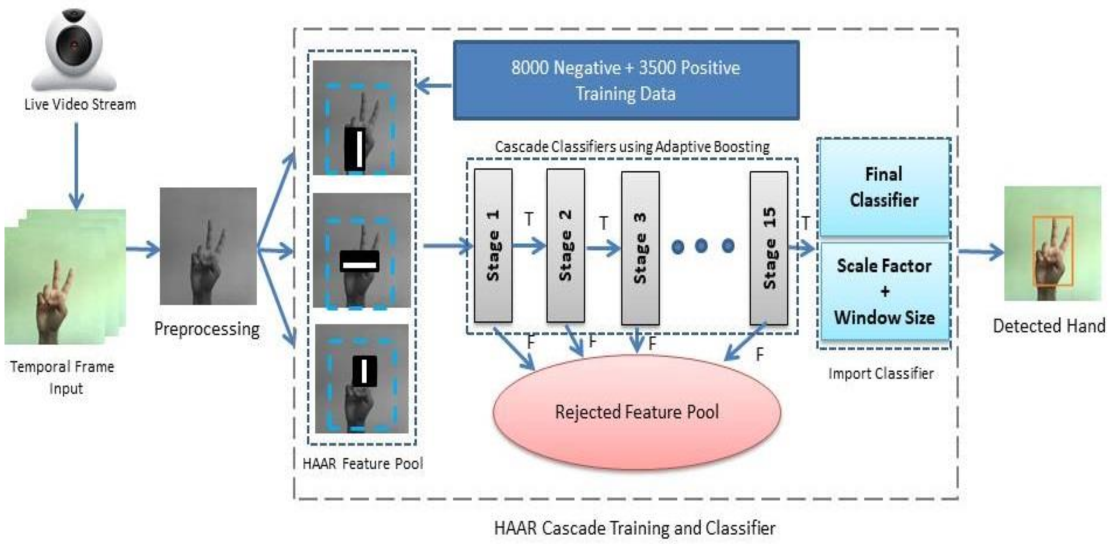
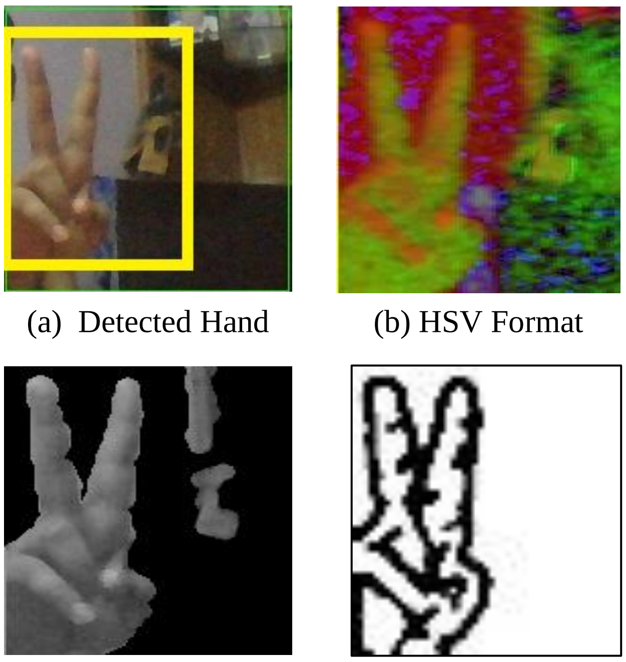


Fig 3.1: Open Hand detection Architecture using HAAR cascades

3.2 Gesture Recognition using Convolutional Neural Network

Hand gesture detection are the building block for connecting Software Applications. So, efficacy of this step is very important for our proposed method. In this work we are detecting 4 classes i.e. *V-shape, Index, Thumb* , *Fist* and *Blank Image* which represents No Class / No Gestures. We trained our 6-layered Convolutional Neural Network for 4 classes of gestures. The details of training the CNN is explained later in section 4. The Temporal Frames which are passed from the last stage of HAAR Classifier is the input to this stage . Here the Image contains the Hand Region cropped out of Original Frame , but we cannot pass this information directly to the Trained CNN Model because it contains background noise .There exists 2 problems which are Addressed Here : (a) Detection of Hand in Noisy Background (b) Detection of Hand in Plain Background.The algorithm to solve the above mentioned issues are discussed below.

To eliminate the background Noise , we implemented a colorspace based Skin Segmentation on Detected Hand Frame to select only the Skin Region . The colorspace used is Hue Saturation Value (HSV) Model since HSV color space is more intuitive to how people experience color than the RGB color space. As hue (H) varies from 0 to 1.0, the corresponding colors vary from red, through yellow, green, cyan, blue, and magenta, back to red. As saturation(S) varies from 0 to 1.0, the corresponding colors (hues) vary from unsaturated (shades of gray) to fully saturated (no white component). As value (V), or brightness, varies from 0 to 1.0, the corresponding colors become increasingly brighter. The input Hand Detected Image is first converted into HSV Colorspace from RGB Colorspace as shown in Fig 3.2(b) .This HSV model filters the skin pixels from the HSV Image. This filtered image is then morphologically eroded and dilated to remove Noise and then Morphologically Opened. This Opening of Mask removes unstable and scattered pixels from background which does not represent skin pixels.


(a) Detected Hand (b) HSV Format

(c) Masked Hand (d) Threshold Image

Fig 3.2 : Represents step by step process of Eliminating the background noise by preprocessing before input to Deep-CNN

After this step the Skin Filtered Hand Image is Masked Out from the Original Temporal Frame as shown in Fig 3.2(c) .

The image is again Preprocessed to detect the Hand in Plain Background as shown in Fig 3.2(d) which is discussed in Details in Section 4.6 and then passed on to The Deep-CNN Model for Classification of Gestures . The Softmax Layer of this Neural Network Model predicts the Gesture in the Input Temporal Frame. The Output of the Softmax Layer CNN Model is converted to One –Hot Array and then Compared with Labelled Gestures to assign a Gesture Label to that Particular Input Temporal Frame which is useful during Mapping of Software Applications . Whenever the Video is Empty the Default Label assigned to the Frame is *No-Gesture* Label . The Code of this is available in Repository [14].

The Architecture of the CNN Model starting from Input Hand Image to Final Labelling of the Gesture is Illustrated in the Diagram as shown in Fig 3.2.

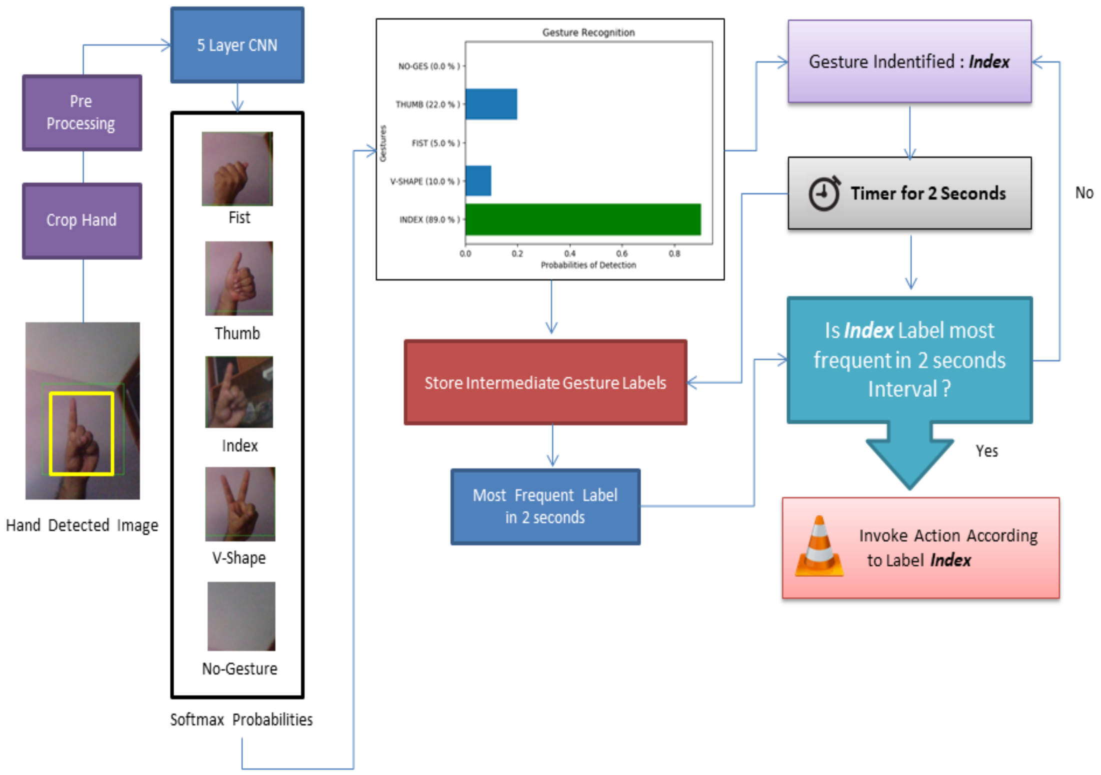


Fig 3.2 : Illustrates the working principle of Gesture Recognition

3.3 Mapping of Software Application to the Labelled Gestures

This is an important Contribution since most of the Existing Models are used either in hand-held devices or Arduino Based Devices by taking help of Foreign Sensors besides the usage of Deep Learning Models. But in this literature we proposed a interesting Mapping of Software Application to the Gestures identified without the usage of any sensors or Hardware and yet it reaches a desirable amount of Accuracy to be commercialized . This step is to make sure Human Gestures can connect and control Desktop Applications .

In this step , we take the Gesture Labelled Temporal Frame and we calculate the Percentage generated by Softmax Layer of the Deep-CNN Model. Since, we are dealing with Computer Applications , the accuracy must be high enough to control the Applications with ease. For this purpose , we deviced an Algorithm for Mapping Gestures to a Software Application which is illustrated in Fig …. .

Table 3.1 : Algorithm for converting gesture to software action

Algorithm

```
1.   Input ← Video Stream with Labelled Temporal Frames
2.   X-Bins ← [ 0 , 0.2 , 0.4 , 0.6 , 0.8 , 1.0 ]
3.   Y-Bins ← [ ‘No-Gesture’ , ‘Thumb’ , ‘Fist’ , ‘V-Shape’ , ‘Index’]
4.   Plot Bar_Graph ( X-Bins , Y-Bins )
5.   Label_count ← [ ] , freq ← 0 # (No Gesture has label 0 )
6.   for each bars in Bar_Graph do:
7.     Max_Score ← max( Softmax Probabilities )
8.     If Max_Score >= 0.7
9.         Timer.sleep(2 , callback ) and Timer.start()
10.        Label_count=Label_count.append( index[Y-Bins(bars) ])
11.   function callback() { freq = mode( Label_count )
12.   If freq >0 and freq = = index(Y-Bins(bars))
13.       Fire Respective Applications
14.    else
15.       Abort and  Label_count= [ ]
```

If a labelled gesture is found in the Temporal Frame , it does not invoke a software immediately , instead , it checks some parameters before invoking the software . As per the above algorithm , at first a Bar Plot is obtained for each Temporal Frame in which Y-axis contains all the Labels and X-Axis contains all the Probability of Detection ranging from 0 to 1 i.e 0% to 100% . For each of these frames we select the Gesture only if the Detection rate is in excess of 70 % . But , the application is not invoked immediately , instead it waits for 2 seconds and scans the Gesture Label during this period and the most frequent gesture between this period is noted down . Then the software corresponding to the frequent gesture in this interval of 2 seconds is invoked .

For this literature , our choice of Application is VLC Media Player , since it is an open-source Media Player currently available in the market and we can easily tweak the settings using code . The table below illustrates the gesture mapping with softwares

Table 3.2 : Algorithm for converting gesture to software action

Gesture	Action
Thumb	Opens VLC Media Player from Menu
V-Shape	Pauses the Current Video if Running
Fist	Plays a Rock Video Song from Memory
Index	Stops Video and Closes VLC Media Player
No-Gesture	Do Nothing

As of Now , we have fixed set of gestures but we can extend this to many gestures and map different kinds of softwares in the future .

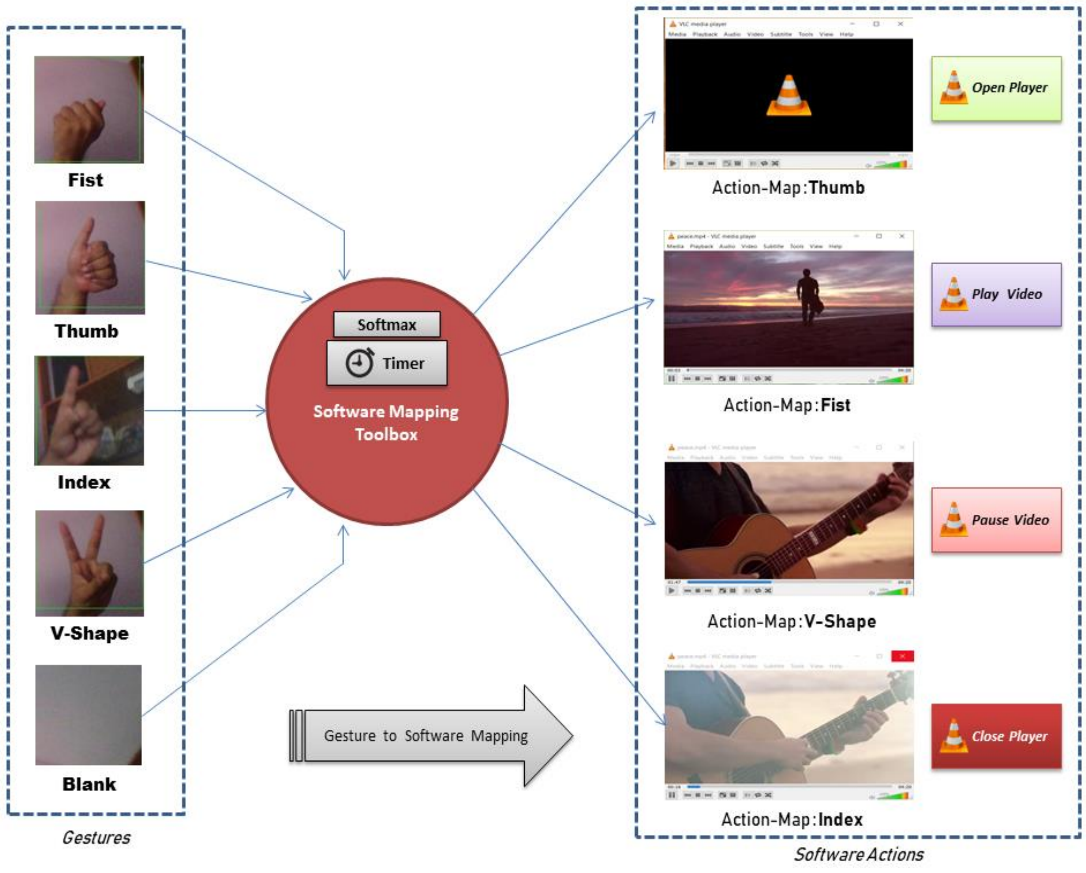


Fig 3.3 : Illustrates the working principle of Software Action Mapping to Gestures

4. EXPERIMENTATION AND RESULTS

As mentioned previously, for each of the stages of hand detection and gesture classification, models were trained using data collected by the authors. In this chapter, the methods of collection and preparation of datasets for each stage is described. Furthermore, the training performed on the dataset in order to achieve hand detection and gesture classification is also described.

4.1 Collection of raw data for creating images

Five (5) volunteers were asked to perform each of the gestures for thirty (30) seconds. Video footage of these were captured, and from the video footage, frames were extracted to form images for training both of the models, hand detection as well as gesture classification

4.2 Preparation of dataset for HAAR Classifier

For preparation of data for training the HAAR classifier, from the entire collection of images, three (3) images of each gesture were taken. In addition, four (4) images of the open hand (i.e., palm) were taken to increase variations. Therefore, in total sixteen (16) images were taken as positives. Each of these images were converted to grayscale and resized to various sizes, described below, as a parameter.

The negatives were collected from [6] , which is a database of images, which are "organized according to the WordNet hierarchy (currently only the nouns), in which each node of the hierarchy is depicted by hundreds and thousands of images." [6]. The various synets from which the images were collected are: room, house, people. This is based on the assumption that interaction with computers is likely to happen in an indoor environment, so the background is likely to contain rooms, and people. In total, Like the positive images, these images were then converted from RGB to grayscale and resized down from their original size to 100x100 pixels.

Next, each of the positive images were superimposed on the negative images, to create the training images. For this, the opencv_createsamples utility provided by OpenCV 3.1.0 was used. The opencv_createsamples utility takes an image, and super imposes that image on a certain number of other images, and produces a new set of superimposed images, and a log. In this process, the image is superimposed on random locations on the base image and is rotated by random angles. The limit of these rotations, the number of superimposed images to be created can be specified. The parameters used by the authors to create samples are explained in Table 4.1. The interpretations of the parameters may be found in [7]. It is to be noted that this entire process was repeated for each positive image

Table 4.1 : Parameters used for generating samples

Parameter	**Interpretation**	**Value**
maxxangle	Maximum rotation angle in x-direction, in radians	0.5
maxyangle	Maximum rotation angle in y-direction, in radians	0.5
maxzangle	Maximum rotation angle in z-direction, in radians	0.5
maxidev	Maximum intensity deviation of foreground	40
h	Height of sample in pixels	20
w	Width of sample in pixels	200
num	Number of sample to generate	500 (each)

Following the process described above, eight thousand (8000) superimposed images were created. The images were then compiled into a ".vec" file as required by OpenCV 3.1.0. The ".vec" file is a binary format containing the images. The opencv_createsamples utility not only records the position of the superimposed images in annotation files, but also adds the coordinates of the superimposed images to the file name itself. This is explained in Fig 4.2. Accordingly, for the 8000 images generated, each image had the coordinated of the positive hand in the file name, and this served as a label for future training processes. This prepared dataset is available at [16].

4.3 Training a HAAR Cascade Classifier for hand detection

The next step involved the training of the boosted cascade classifier from the from the “.vec” file and the dataset described in the previous section. For this purpose, the utility program opencv_traincascade described in [8] was used. At this point it is worthwhile to point out that several window sizes of positives were used: 20pixels, 30pixels, 40 pixels, and 50 pixels. Moreover, several classifiers were trained for several stages. This was done to conduct a comparative study of the effect of window size, and number of stages on classification accuracy. The opencv_traincascade command can be supplied with various flags and options to control type of features (HAAR or LBP), the boosting algorithm, types of HAAR features. The various parameters, their interpretations are explained in Table 4.2

Table 4.2 - Parameters used for Training Classifier

Parameters	**Interpretation**	**Value Used**
numPos	Number of positive samples to be used in each stage	6000
numNeg	Number of negative samples to be used in each stage	10000
numStages	Number of cascade stages to be trained	Varied from 10-20
bt	Boosting Algorithm: DAB/RAB/GAB	GAB (Gentle AdaBoost)
minHitRate	Minimum desired hit-rate for each stage of classification.	0.995
maxFalseAlarm Rate	Maximum permissible false alarm rate for each stage of classification.	0.05
Mode	Type of HAAR features to use for training: BASIC/CORE/ALL	BASIC (only upright features)

4.4 Testing and evaluation of Cascade Classifier

Using the method described in the preceding section, several classifiers were trained. In this section, a comparative study of the classifiers is drawn. As mentioned in Sections 4.1 and 4.2, various classifiers were trained for various values of:

a. *Window Size*: The size of the positive sample (square, in pixels). Window sizes of 20, 30, 40, and 50 were used.
b. *Number of Stages*: Number of stages for which the classifier was trained. Classifiers were trained for 10, 12, 15, 20 stages.

For testing purposes, a test dataset was generated from positive samples not used for training the classifier. These positive samples were superimposed upon the negatives in the exact same method described in Section 4.1. The total number of superimposed images used in the test dataset was five hundred (500), each with one positive sample in it. For detection of a positive from a image, python’s OpenCV implementation provides with a method called detectMultiscale(). For details, the reader is requested to peruse through [9]. This method takes an image, in this case one image from the test set, a cascade classifier, and some parameters, and returns the list of coordinates of predicted locations of positives. The parameters greatly affect the detection accuracy of the classifier. Therefore, these parameters were varied one at a time, and the effects on performance were observed. The performance metric chosen, was accuracy, as the dataset was not unfairly skewed towards positives or negatives. We define accuracy as the number of true positives detected by the algorithm to the number of positives in the dataset. The parameters varied are explained in Table 4.3.

Table 4.3 : Parameters used for Detectimg Objects

Parameter	**Interpretation**	**Range of Values**
Scale Factor	At each stage, the image is reduced by some factor, and passed to the next stage (more features). This is the scale factor.	[1.01, 1.02, …, 2.20]
Minimum Neighbours	The least number of neighbours that a detected object must have in order to be retained in the next stage	[1, 2, …, 50]
Window Size	Size of positive sample to be detected.	[20, 30, 40, 50]
Number of Stages	Number of stages for which classifier was trained	10, 12, 15, 20

To further explain the effects of Scale Factor, Refer to Fig 4.1 and 4.2. In Fig 4.1, the scale factor value = 1, and in 4.5, value = 10. It is to be noted that there is a stark difference in the number of false positives detected by the classifier. When minNeighbours = 1, just 1 neighbour is enough for the algorithm detectMultiScale to pass on the region as a positive, and it is passed on to subsequent stages. However, when minNeighbours = 10, a much more stringent bound is placed, because 10 such regions must be identified before the region can be classified as positive. As a result, many of the false positives are eliminated through the stages and only promising examples are passed on. Consequently, a much better result is obtained.

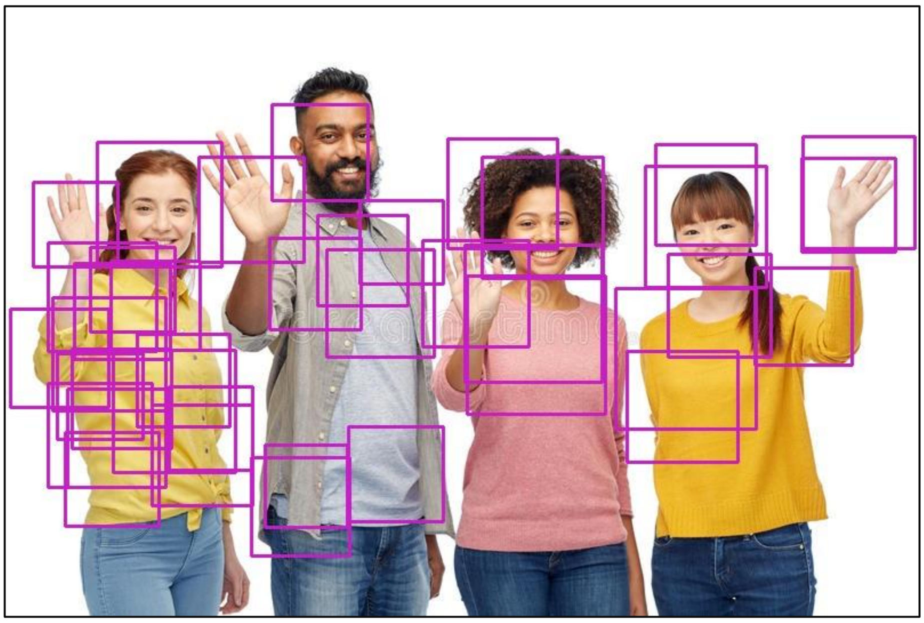


Fig. 4.1: Hand detection with minNeighbours = 1

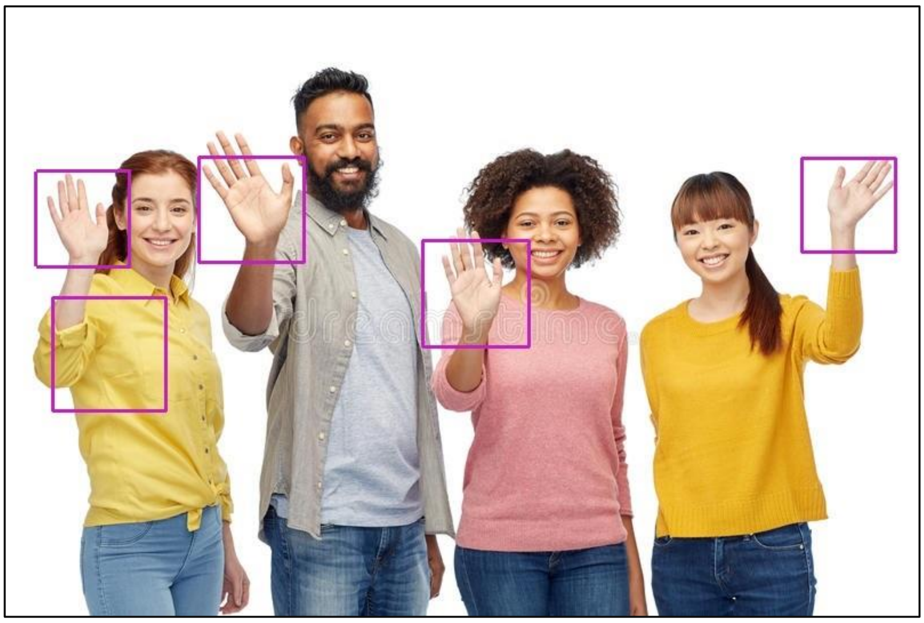


Fig. 4.2: Hand detection with minNeighbours = 10

Finally, the algorithm used to calculate accuracy on the test-set described above is explained in Table 4.4. For testing each parameter in Table 4.3, each of the parameter was varied, keeping the others constant, and accuracy calculated on various values of the parameter, per Table 4.3. The corresponding plots generated in Fig 4.3 through 4.5.

Table. 4.4: Evaluation and graph-plotting algorithm

```
1.    size ← 500
2.    params ← scale, neighbours, stages, window size
3.    vals ← range of values (refer Fig 4.3)
4.    load cascade file
5.    for each val in vals do:
6.    count ← 0
7.    for each image in test-set do:
8.    detectMultiScale(image, params)
9.    get predicted region for img
10.   compare predicted regions and labels (in image name)
11.   if predicted region covers ≥ 70% of actual region, count it as correct
12.   accuracy ← correct predictions / size
13.   plot accuracy vs. params.
```

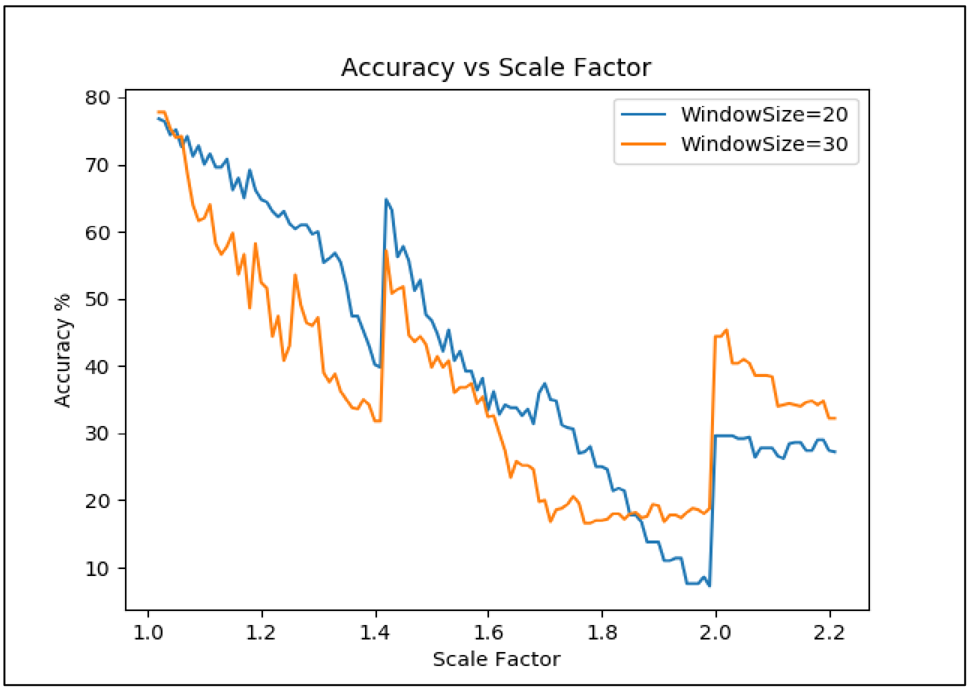


Fig. 4.3: Plot of accuracy vs. Scale factor for both window sizes 20px and 30px

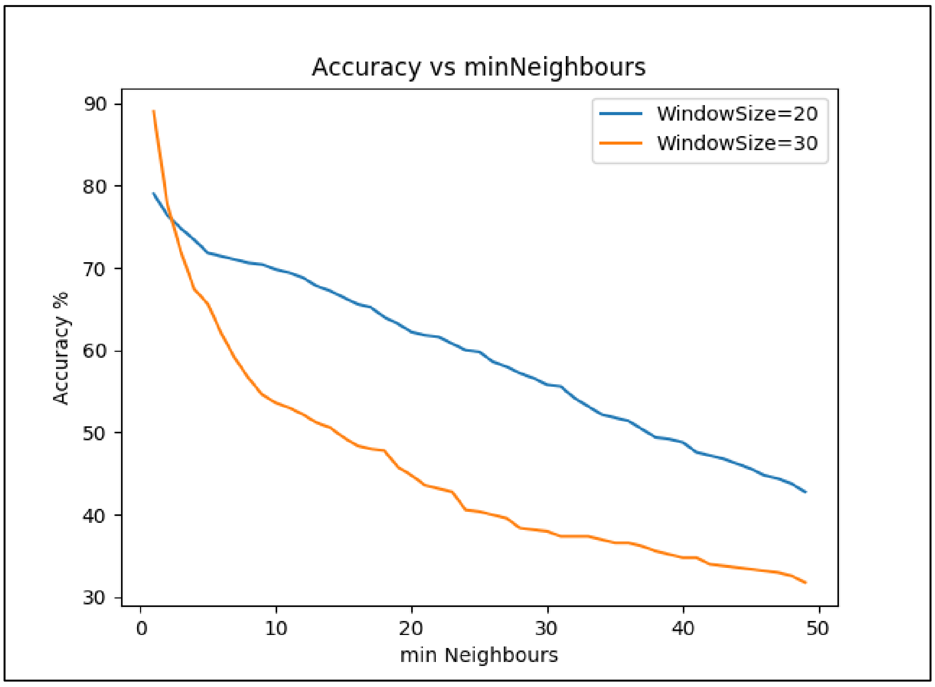


Fig. 4.4: Plot of accuracy vs. min neighbours for both window sizes 20px and 30px

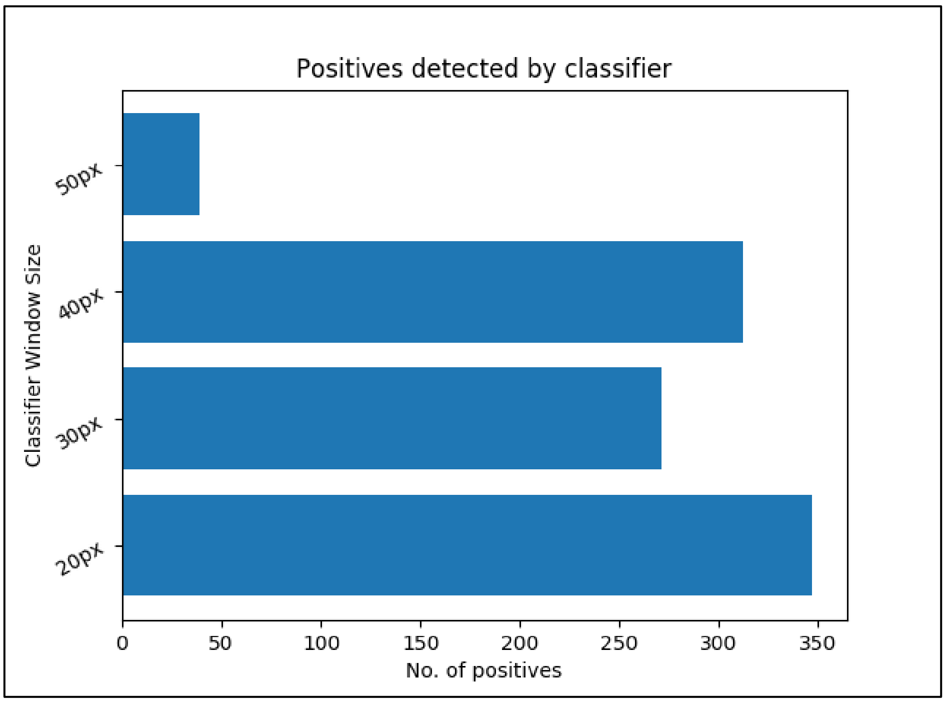


Fig. 4.5: Plot of hits vs. window sizes for different window sizes

From the above results, it was clear that the window size of 20x20 was the most promising choice, so it was used for all further work. An additional advantage of using smaller window size was that the number of false positives is reduced substantially, as the window size decreased. For the values of minimum neighbours and scale factor, values in the range 1-9 and 1.0-1.1 respectively were used. Also, as the window size increases, so does the feature space, and the time taken and resources consumed to train the classifier rises significantly.

4.5 Data augmentation for Convolutional Neural Network

As mentioned in Section 4.1, frames extracted from videos of volunteers performing the four gestures were used as training images. For training the CNN, approximately four hundred (400) images of each gesture. These were then manually labelled, with the file names containing the label, and an indexing number for general use. To the above set of images, another four hundred (400) images containing no gestures were added, following the [10].

This dataset was augmented to increase variety [11] and quantity, and also to take into account various lighting conditions, positioning and various spatial orientation of samples which were not present in the original training data, but could well be present in the real data. To augment the data, the following operations were performed: (Also refer to Fig 4.10)

a. *Linear translation*: Randomly selected images were translated by varying random amounts in the range [-20, +20] pixels on the x-axis, or y-axis, or both.
b. *Rotation*: Randomly selected images were rotated in the range [-20, +20] degrees, the rotation center was randomly selected
c. *Illumination*: The intensity of some randomly selected images were varied by [-40%, 40%]
d. *Noise*: Some randomly selected images hand Gaussian noise injected into them.

After the data augmentation processes described above, the size of the dataset was increased to ten thousand (10,000) images.

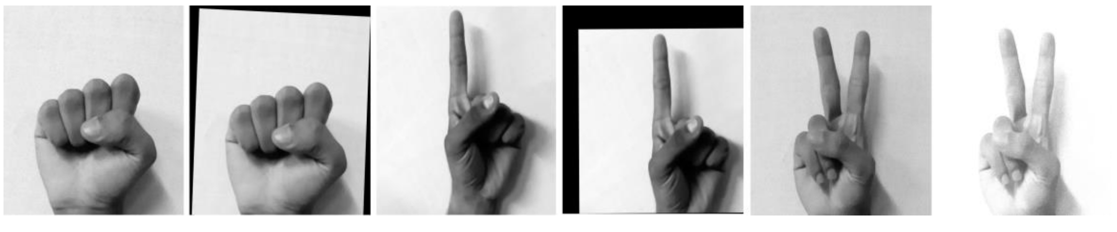


Fig. 4.6: Rotation, Translation, and intensity deviations on images

4.6 Data pre-processing for Convolutional Neural Network

To reduce the feature space, some image pre-processing was performed on the images. Another reason behind performing such image processing operations was to remove the background from the images, keeping only the basic features of the hand. Accordingly, each of the images after the processing in Section 4.4, was subject to the following operations:

a. *Cropped to square aspect ratio*: Each of the images was cropped to a square portion to remove any inconsistency that may arise because of different sizes of images.

b. *Converted to grayscale*: Each image was originally a 3 channel (RGB) image, was converted to a 1 channel grayscale image.to reduce convolution operations and no of activation maps.

c. *Gaussian Blur*: Some portions of the images have no features at all, (i.e., totally white), whereas some are having the hand portions. Thus, the values of the features vary diversely. Gaussian Blur can be thought of as a mean normalization over all pixel values. In addition, Gaussian Blur also reduces noise and detail in images.

d. *Adaptive Thresholding*: The principal purpose of thresholding is to remove the background from the image. For every pixel if it falls within a range, the pixel value is retained, otherwise it is replaced to remove it.

e. *Erosion and dilation*: Erosion is applied to remove unnecessary detail and noise from the image. However, erosion also thins boundary of the image, by removing pixels from the image boundaries. To counteract this, the image is dilated, where the boundaries are again regenerated, and any gaps created by erosion are bridged.

f. *Median Blur*: The above operations unfortunately, introduce unnecessary specks and blemishes into the image, which are removed by median blur.

Refer to Fig 4.11 for a clearer idea. In Fig 4.11, one image from the dataset has been chosen at random and the results of operations *a.* through *f.* are displayed.

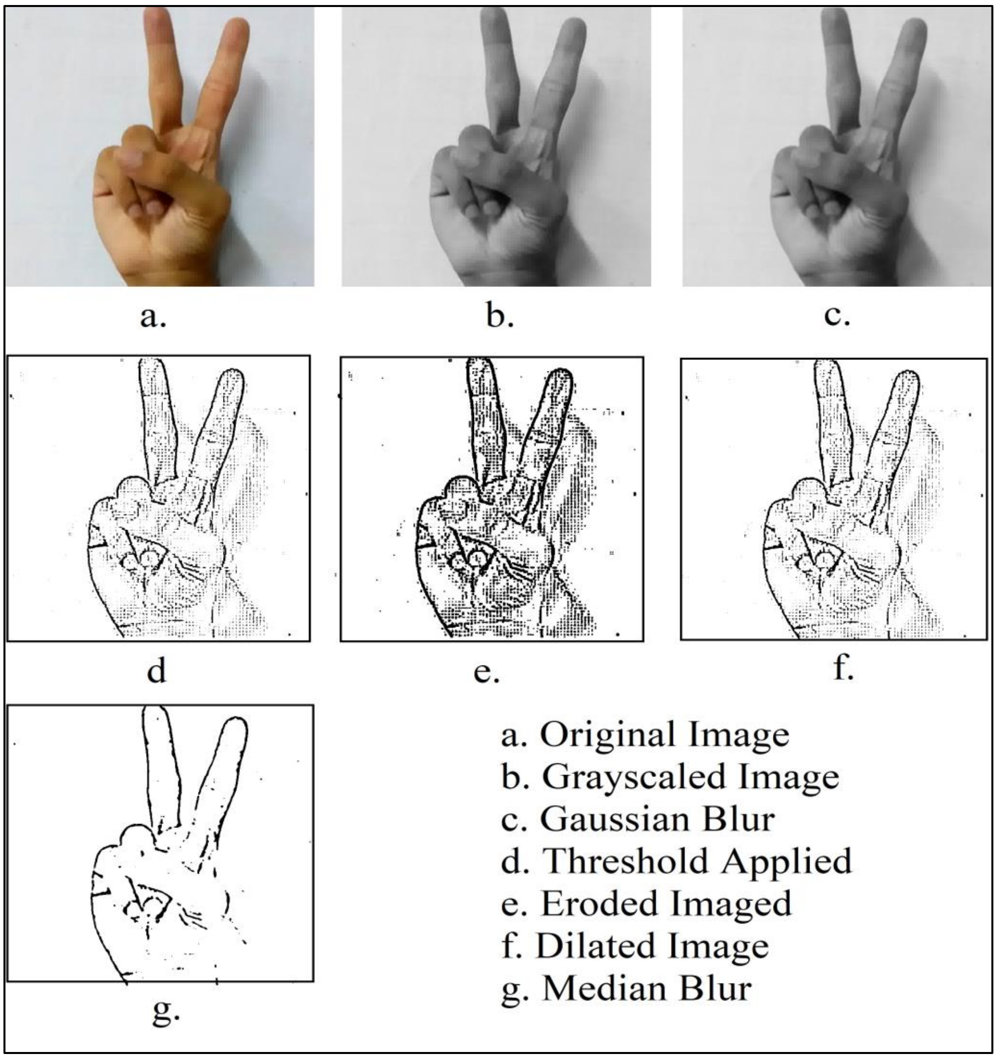


Fig. 4.7: Grayscaling, Gaussian Blur, Thresholding, Erosion, Dilation and Median Blur

Thus, each image in the augmented data set is subject to the above processing, and the images such as *g.* are used for training the Convolutional Neural Network. Before this is done, however, each image is also resized to 64x64 pixels. This is to reduce the processing time. Most state-of-the art algorithms use 32x32 pixels, but since the image has been stripped off most of its features, a larger size is chosen for our purposes.

4.7 Architecture of Convolutional Neural Network and Training

The architecture of the CNN used to classify gestures was partly influenced by a literature survey, as described in Chapter 1, and partly as a result of experimentation with different architectures. The final architecture is as follows:

Fig. 4.5: Architecture of CNN

No.	Layer	Kernel Size	Feature Map/Stride	Activation	Dropout
1.	Convolutional	5x5	32	ReLU	--
	Normalisation	--	--	Local Response	--
2.	Convolutional	5x5	64	ReLU	--
	Normalisation	--	--	Local Response	--
	Max Pooling	2x2	2	--	--
3.	Convolutional	5x5	128	ReLU	--
	Max Pooling	2x2	2	--	--
	Normalisation	--	--	Local Response	--
4.	Fully Connected	--	1024	ReLU	0.5
5.	Fully Connected	--	512	ReLU	0.5
6.	Softmax	--	5	Softmax	--

Various other hyper-parameters used were:

a. Learning Rate = $1.0\text{x}10^{-3}$
b. Image Size = 64x64 pixels
c. Number of epochs = 50
d. Loss function in Classification Layer: Categorical Cross-Entropy [12]
e. Train:Validation Split: 4:1
f. Optimiser: Adam
g. Batch Size = 300
h. Steps Per Epoch = 50

Figure 4.8 shows the architecture of the CNN as depicted by Tensorboard [13].

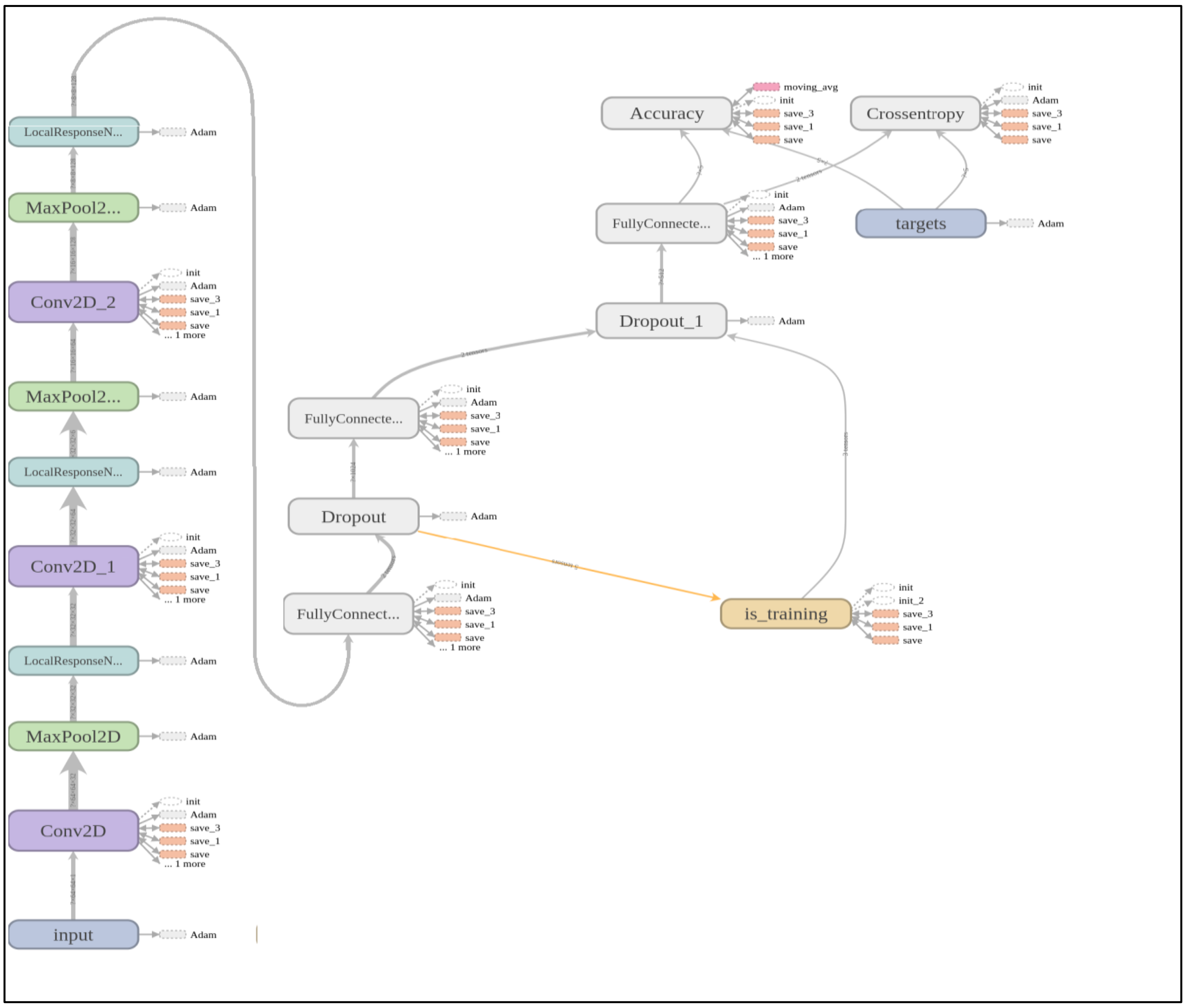


Fig. 4.8: Architecture of CNN as a computational graph

The following Architecture was Trained using 10,000 Images as Mentioned in previous Section. The Model was trained using Batch Size of 300 , with Number of Epochs as 50 and Steps per Epoch was set to 50 . The Time taken to Train the above mentioned Architecture was 4-5 Hours . The Model was trained on a Intel Core i5 Processor @ 2.3 GHz with 8 GB DDR3 Ram and 2GB Dedicated NVIDIA GEFORCE GPU 940M Graphics Processor. The Training was continued until the Validation and Total Loss got stagnant.

4.8 Testing and evaluation of Convolutional Neural Network

In this section, a comparative study is drawn on the CNN showing the effects of reducing certain layers, and that of number of epochs on classification accuracy. For this purpose, a new test data set was created with images previously not used. These images were treated in the exact same way, described in Section 4.5. Five hundred (500) total images were used to create a test data set. The following performance metrics were used:[13]

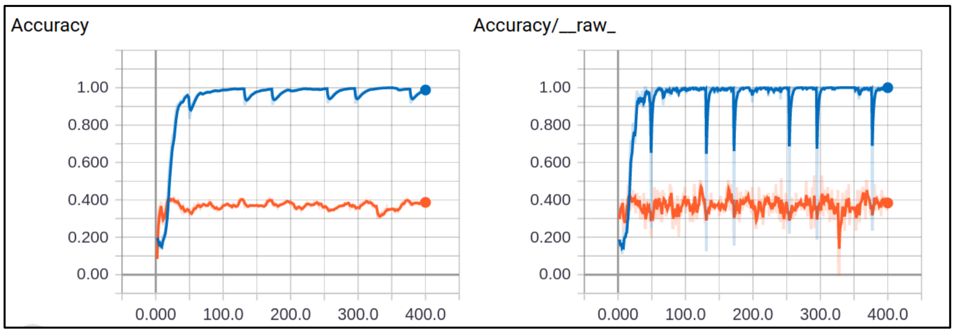
a. Accuracy: The ratio of correctly classified samples to the actual number of samples

b. Precision: $\frac{True\ positives}{True\ positives+False\ Positives}$

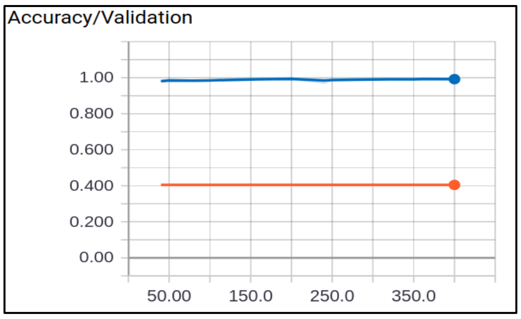
c. Recall: $\frac{True\ positives}{True\ positives+False\ Negatives}$

d. F1-Score: $2 * \left(\frac{Precision*Recall}{Precision+Recall}\right)$

To calculate accuracy, a script as run 5 times over the test set and each time, the accuracy was calculated over each run, and the mean accuracy reported.



a b

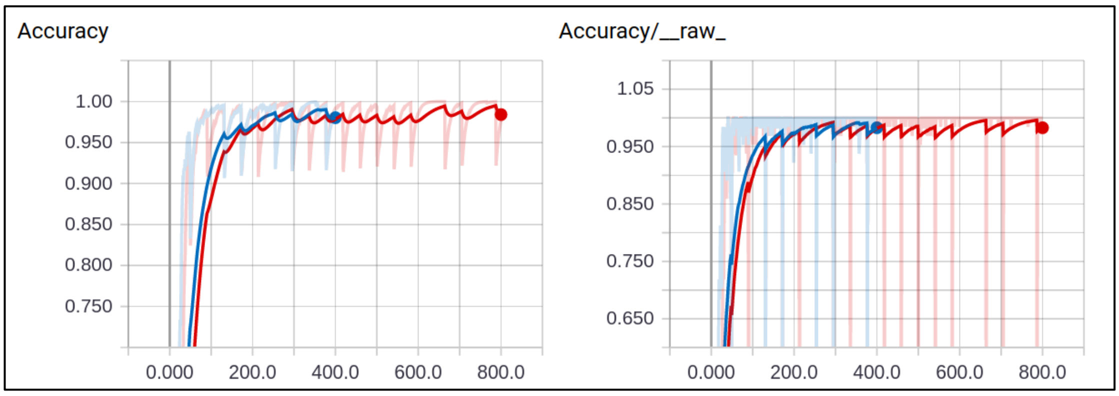


c

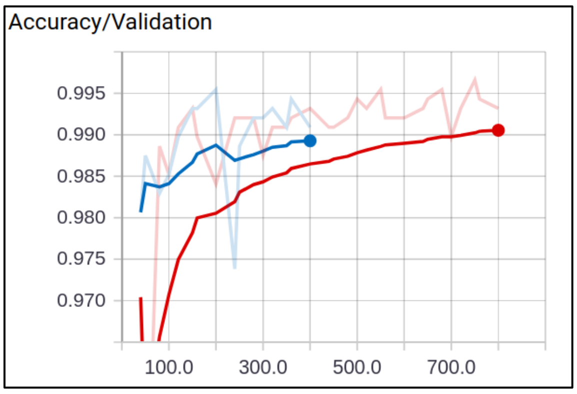
Fig. 4.9: Training and Validation Accuracy, comparison with 1 and 3

Fig 4.9 shows a comparison between a CNN with just 1 convolutional Layer versus the architecture described by Table 4.5. Clearly the addition of more convolutional layers contributes to a marked increase in training accuracy.

Figure 4.10 shows a comparison between training accuracy of CNN's having the same architecture but trained for different epochs (10 and 20). Obviously, a classifier trained for 10 epochs had better performance than one trained for 10 epochs.



a. Training accuracy vs. iterations: Blue: 10 epochs, Red: 20



b. Validation Accuracy vs. iterations: Blue: 10 epochs, Red: 20

Fig. 4.10: Training and Validation accuracy 10, and 20 epochs

Therefore, the final classifier that was chosen was the architecture in Fig 4.8, and it was trained for 20 epochs using the data set previously explained. The results are shown in Table. 4.7.

Table 4.7: Performance Report

	Precision	**Recall**	**F1-score**	**Support**
1.	0.00	0.00	0.00	103
2.	0.99	1.00	1.00	107
3.	0.83	0.07	0.13	73
4.	0.31	0.81	0.45	94
5.	1.00	1.00	1.00	123
Average	0.64	0.62	0.56	500
Average Accuracy after 5 runs = 0.712				

The proposed architecture was also compared against standard small sized Models given in the literature of Orlando et al. [15] . We considered the Architectures Arq1 , Arq3 as given in the Paper[15] which are described below .

Table 4.8: Details of Architectures for Comparison

Arq 1		*Arq 3*	
Type	**Kernel**	**Type**	**Kernel**
Convolution	10 x 10	Convolution	36 x 36
Max Pooling	8 x 8	Max Pooling	5 x 5
Convolution	5 x 5	Convolution	7 x 7
Max Pooling	10 x 10	Fully Connected	---
Fully Connected	---	Softmax	4
Softmax	4		

The Models are evaluated on the basis of 4 classes, since these models were originally designed for detection of 3 classes namely: Open Hand , Close Hand and No Class , but we used all the 4 Gestures we used to find the Efficacy of this Model .

The training of the Above Models was done using same Dataset which was used for our Model. We ran the Models till 20 Epochs. After Training the Models, the Models were tested against Live Video Stream of Web Camera, which acted as the Test Dataset. The Test Video was run for 10 Seconds i.e. 3000 Frames since Real Time Web Cameras are mostly 30 fps. Hence the architecture of Simulation was similar to all the 3 models.

We calculated the Accuracy of the Models using the Metrics mentioned in this section. Since we were dealing with Software, our main concern was Accuracy.

So we found out the Accuracy Metric of the Models Arq1, Arq3 and Proposed Model which is illustrated in Table 4.9

Table 4.9: Comparative Study on Performance of Architectures

Architecture	**No of Frames**	**Epochs**	**Accuracy**
Proposed Method	3000	20	**71.24 %**
Arq 1	3000	20	32.66 %
Arq 3	3000	20	61.35 %

The comparative Study of the Architectures shows that the Proposed Method works best for 5 gestures with a Mixture of Open Hand, Close Hand and No Gesture , However Arq3 performs decently to obtain 2nd best spot in terms of Accuracy inspite of having the Smallest Architecture among the 3 used whereas Arq1 performed Poorly mainly because of the Kernel used and less number of Feature Maps , hence it could not extract enough Features to Distinguish one gesture from another .

5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

We have built the software which can detect four static hand gestures from a video feed, and take actions for a specific application, vlc media player. A detected closed fist causes the player to play rock-music categorized videos, thumbs-up causes the application to start shaped gesture puts the player on pause and show of an index from the user closes the vlc media player. Importantly, the application is for a single user at a time.

5.2 Future Work

Our work has been limited only to static gestures. However, it can be extended to dynamic gestures which can easily be fit in a two-dimensional frame by looking for consecutive frames from a video feed instead of only one frame before labeling the gesture of the detected hand in the frame. For example, a swipe-left gesture can be detected by checking for the changing positions of the detected hand (with all five open fingers) in closely consecutive frames from right to left. Also, in order minimize the presently running window, an index finger moving from top-right to down-left can be used. The software can easily confuse this minimize gesture with a show of only index finger gesture, and to avoid the such situations, the software should wait for a few seconds before taking the action to see if there is a change in the position of the hand in the given frame. This Proposed Gesture Detection Model is currently working in Approximately 15 fps and since we are also planning on making the software available for all the applications available on any device , our future goal is to make this model Realtime Interaction with Software .Currently , the software will be running for a specific application, like the vlc media player . Also, the gestures will perform more generalized actions, like show of index finger will not only close vlc media player but any presently running application/s. Speed analysis of the dynamic gestures need to be done. If the gesture is performed too fast, it should be discarded, because it's likely to be a reflexive action rather than a gesture interaction on behalf of the user. Interacting with software using gesture in purely the user's choice and he can run/stop the software according to his/her discretion.

References

1. Z. Ren, J. Yuan, J. Meng, and Z. Zhang, *“Robust Part-Based Hand Gesture Recognition Using Kinect Sensor”*, vol. 15, no. 5, pp. 1–11, 2013.

2. Q. Z. Sheng, *“Online Human Gesture Recognition Area Chair: Max Mühlhäuser”*, pp. 23–32, 2013.

3. M. Nied, *“Hand Body Language Gesture Recognition Based on Signals from Specialized Glove and Machine”*, vol. 3203, no. c, pp. 1–10, 2016.

4. J. Umverslty, *“A Hand Gesture Recognition Technique from Real Time Video”*, no. May, pp. 21–23, 2015.

5. M. A. Simão, P. Neto, and O. Gibaru, *“Unsupervised Gesture Segmentation by Motion Detection of a Real-Time Data Stream”*, vol. 3203, no. c, pp. 1–9, 2016.

6. ImageNet: http://image-net.org by 2016 Stanford Vision Lab, Stanford University, Princeton University.

7. Leidart, Daniel. “Ubuntu Manuals.” Ubuntu Manpage: Ubertooth-Btle - Bluetooth Low Energy (BLE) Sniffing and More, Ubuntu, 3 July 2010, manpages.ubuntu.com/manpages/trusty/man1/opencv_createsamples.1.html.

8. OpenCV Cascade Classifier Training: https://docs.opencv.org/3.4/dc/d88/tutorial_traincascade.html

9. Jones, Viola. “Cascade Classification¶.” OpenCV: Image Thresholding, OpenCV, 4 May 2003, docs.opencv.org/3.0-beta/modules/objdetect/doc/cascade_classification.html?highlight=detectmultiscale.

10. Some reference

11. J. Wang and L. Perez, *“The Effectiveness of Data Augmentation in Image Classification using Deep Learning”*

12. R. Murugan, *“Implementation of Deep Convolutional Neural Network in Multi-class Categorical Image Classification”*, arXiv.org:1801.01397v1

13. C. Goutte, and E. Gaussier, *“A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation”*, Xerox Research Centre, 6 Chemin de Maupertuis, Meylan, France.

14. S. Nag, “*Gesture Recognition And Software Mapping*” https://github.com/sauradip/Gesture-Recognition-And-Software-Mapping, 14-May-2018. [Online]. Available:https://github.com/sauradip/Gesture-Recognition-And-Software-Mapping. [Accessed: 14-May-2018].

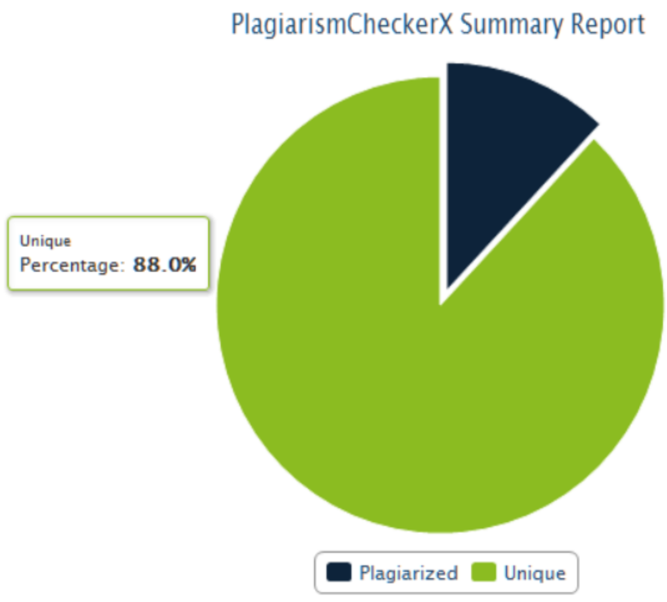
15. J. Orlando, P. Arenas, and R. J. Moreno, “*CONVOLUTIONAL NEURAL NETWORK ARCHITECTURE FOR HAND GESTURE RECOGNITION*,” pp. 1–4, 2017.

16. : Ganguly, Pallab Kumar. “*Hand Classifer Dataset*.” GitHub, 2 Sept. 2017, github.com/pallabganguly/opcv-project.

17. A. Krizevsky, *“ImageNet Classification with Deep Convolutional Neural Networks*”, University of Toronto, https://www.cs.toronto.edu/~kriz/imagenet_classification_with_deep_convolutional.pdf

Plagiarism Report

The Report has been checked by a Standard Plagiarism checker software ***Plagiarism Checker – X (Version 6.06).*** The entire report starting from Introduction up till references is passed a Word document in the software. The software has reported **88% Unique** content which confirms the authenticity and originality of the Literature written by the Authors. The detailed report is shown below.



Date	Wednesday, May 16, 2018
Words	1043 Plagiarized Words / Total 9063 Words
Sources	More than 184 Sources Identified.
Remarks	Low Plagiarism Detected – Your Document needs Optional Improvement.

Fig : Reflects the Authenticity Check Results done using Plagiarism Checker X Version 6.06