



S	
E1	
E2	
<b>For Office Use Only</b>	

**Masters Project Final Report**  
**(MCS)**  
**2019**

<b>Project Title</b>	<b>Sinhala Sign Language to Text Interpreter based on Machine Learning</b>
<b>Student Name</b>	<b>W D T Peiris</b>
<b>Registration No. &amp; Index No.</b>	<b>2017MCS058 17440585</b>
<b>Supervisor's Name</b>	<b>Dr. D. A. S. Atukorale</b>

<b>For Office Use ONLY</b>



# Sinhala Sign Language to Text Interpreter based on Machine Learning

**A dissertation submitted for the Degree of Master of  
Computer Science**

**W. D. T. Peiris  
University of Colombo School of Computing  
2019**



# Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: W. D. T. Peiris  
Registration Number: 2017MCS058  
Index Number: 17440585

---

Signature:

Date: 20/06/2020

This is to certify that this thesis is based on the work of

Mr./Ms.

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:  
Supervisor Name:

---

Signature:

Date:

## Abstract

Sinhala sign language is the primary mode of communication between hearing-impaired Sri Lankans. Their main difficulty in interacting with the general public is that majority of the population do not know how to interpret sign language. This problem in communication and level of comprehension is disadvantageous for the hearing and speaking impaired.

For sign languages with a large user base, there are already many projects in place to alleviate these problems; but since the sign language user community in Sri Lanka is small, there is little effort put into projects and research to come up with solutions for this problem. Therefore, the primary objective of this study was to design and develop a desktop software application that captures video in real-time of a person using Sinhalese fingerspelling sign language, process and identify the gestures based on machine learning and interpret the signed hand gestures in the video and output text to a screen as words. In order to achieve this task, A dataset was created for the Sinhala fingerspelling alphabet to serve as training images for the machine learning process. 27000+ images were obtained to train 27 hand gestures. After careful consideration, Inception, a convolution neural network was selected and trained to interpret the images.

The Graphical User Interface and the underlying code was written in Python with Tensorflow acting as the framework which handled the machine learning component. In addition to that, to achieve the final objective; various methods of image preprocessing, image extraction, skin detail, and background removal techniques were also studied. The project intentionally left out any wearable technology or other 3<sup>rd</sup> party appliances to keep the cost to the user as low as possible. Once application development was complete, the system was evaluated against 6 individuals with each test subject performing 120 tests for hand gesture character recognition resulting in an overall accuracy of 95%.

## Acknowledgement

Special thanks to my supervisor Dr. Ajantha Atukorale for his constant guidance and supervision; and to all the staff at UCSC for their support. A huge thank you to friends, family and everyone else who made exceptions to their busy schedules and volunteered their valuable time to train and test the dataset used for the project.

# Contents

Declaration .....	i
Abstract .....	ii
Acknowledgement .....	iii
List of Figures .....	vi
List of Tables .....	viii
Chapter 1 Introduction .....	1
1.1 Problem Definition .....	1
1.2 Project Objectives .....	2
1.3 Project Scope .....	3
1.4 Summary .....	3
Chapter 2 Literature Review .....	4
2.1 Introduction .....	4
2.2 Development of Sign Language .....	4
2.3 Finger Spelling Alphabet .....	5
2.4 Approaches to Hand Gesture Recognition .....	6
2.5 Sensor Based Gesture Recognition .....	7
2.6 Vision based Gesture Recognition .....	9
Vision based Gesture Recognition – Camera .....	10
2.7 Detection .....	10
2.8 Recognition .....	11
Points of Interest .....	12
Centroids .....	12
Markov Models .....	13
LeNet and Convolution Architecture Models .....	13
2.9 Selecting an Approach .....	15
2.10 Summary .....	16
Chapter 3 Methodology .....	17
3.1 High Level System Design .....	17
3.2 Implementation and Components .....	21
Video Capture .....	21
3.3 Pre-processing .....	23
Region of Interest .....	23
Skin Detection .....	24

Creating a Mask.....	26
3.4 Creating the Dataset .....	29
3.5 Training .....	34
Create Image Lists Function.....	35
Get Image Location Function .....	36
Bottleneck Files .....	37
3.6 Tensorboard.....	40
3.7 Main Processing Unit .....	44
3.8 Transliteration .....	47
How the Module Works .....	50
Graphical User Interface .....	52
Chapter 4 Results and Evaluation .....	55
4.1 Introduction .....	55
4.2 Evaluation Criteria .....	55
4.3 Alphabet Character Evaluation .....	56
4.4 Confidence level Analysis.....	61
4.5 Word Evaluation .....	69
4.6 Summary .....	71
Chapter 5 Conclusion and Future Work .....	72
5.1 Interpreting Results .....	73
5.2 Difficulties and Limitations .....	75
5.3 Future Work .....	75
References.....	77
Appendix.....	80

# List of Figures

Figure 1.3.1 Proposed system .....	3
Figure 2.3.1 Sinhala fingerspelling alphabet .....	5
Figure 2.9.1 Two main recognition techniques .....	15
Figure 3.1.1 High level diagram of the system .....	17
Figure 3.1.2 Creating dataset .....	18
Figure 3.1.3 Tree command for dataset directory .....	19
Figure 3.1.4 Training .....	19
Figure 3.1.5 Setup for the final application .....	20
Figure 3.1.6 Proposed application wireframe mockup .....	21
Figure 3.2.1 0.7MP image of a white surface under ambient conditions .....	22
Figure 3.2.2 5MP image of white surface .....	22
Figure 3.2.3 Video capture with region of interest marked .....	23
Figure 3.3.1 Sliced image .....	24
Figure 3.3.2 Input threshold from user controls the background removal mask .....	25
Figure 3.3.3 Colour space of YCrCb [27] .....	25
Figure 3.3.4 Image in YCrCb colour space .....	26
Figure 3.3.5 Colour space on x-y-z plane[28] .....	26
Figure 3.3.6 Trackbars created in OpenCV show colour space thresholds. ....	27
Figure 3.3.7 Creating the threshold .....	27
Figure 3.3.8 CRn 145, 135 and 125 .....	27
Figure 3.3.9 Largest contour is the hand .....	28
Figure 3.3.10 Finding the largest contour .....	28
Figure 3.3.11 Output from preprocessing .....	29
Figure 3.4.1 Note how all these hand gestures are similar .....	33
Figure 3.5.1 Sample use of train.py .....	35
Figure 3.5.2 Creating image lists .....	35
Figure 3.5.3 Assigning labels .....	36
Figure 3.5.4 Get image location .....	36
Figure 3.5.5 Contents of a bottleneck image file for reference .....	37
Figure 3.5.6 Create_graph function .....	37
Figure 3.5.7 Creating bottleneck files .....	38
Figure 3.5.8 Scalar values for tensorboard .....	38
Figure 3.5.9 Final_ops function .....	39
Figure 3.5.10 Setting up image lists .....	39
Figure 3.5.11 Merge .....	39
Figure 3.5.12 Writing to summary files .....	40
Figure 3.5.13 Writing to graph files .....	40
Figure 3.5.14 Training progress printing to terminal .....	40
Figure 3.6.1 Learning rate .....	41
Figure 3.6.2 Training(orange) and validation(blue) accuracy for lrate= 0.0001 .....	41
Figure 3.6.3 Training(orange) and validation(blue) accuracy for lrate=0.01 .....	41
Figure 3.6.4 Training(orange) and validation(blue) accuracy for lrate= 2 .....	41
Figure 3.6.5 Accuracy of the final dataset .....	42
Figure 3.6.6 Cross entropy .....	42
Figure 3.6.7 Changing biases and weights .....	43
Figure 3.6.8 Distribution of biases .....	43
Figure 3.7.1 Loopspeed .....	44
Figure 3.7.2 GUI has a trackbar to control the i value .....	44



Figure 3.7.3 Conditions within the main processing unit .....	45
Figure 3.7.4 Sequence being constructed .....	46
Figure 3.7.5 Predict function .....	47
Figure 3.7.6 Sess.run.....	47
Figure 3.7.7 Res and max_score .....	47
Figure 3.8.1 Sequence of characters become a word.....	48
Figure 3.8.2 spelling amma with the sinhala sign language .....	49
Figure 3.8.3 A sentence in sign language .....	49
Figure 3.8.4 Cycles through list of vowels .....	51
Figure 3.8.5 List of consonants + hal kireema.....	51
Figure 3.9.1 The GUI.....	52
Figure 3.9.2 Mask under different thresholds .....	53
Figure 3.9.3 Text Speed trackbar .....	53
Figure 3.9.4 Closer view of text output .....	54
Figure 4.3.1 Hand gesture used for Test A .....	59
Figure 4.3.2 Hand gesture used for Test B .....	59
Figure 4.4.1 Changing code to print confidence level .....	61
Figure 4.4.2 Confidence level data .....	62
Figure 4.4.3 Change in confidence level between test scenarios.....	65
Figure 4.4.4 similar gestures .....	66
Figure 4.4.5 Similar gestures .....	67
Figure 4.4.6 Confusion matrix .....	67
Figure 4.4.7 Normalized confusion matrix .....	68
Figure 4.5.1 Sinhala letter frequency .....	69
Figure 4.6.1 Application in use.....	73
Figure 5.1.1 Distorted images from similar gestures .....	74

## List of Tables

Table 3.4.1 List of trained gestures.....	32
Table 3.4.2 Pixels washed out by background light .....	34
Table 3.4.3 A good image doesn't have shadows .....	34
Table 3.8.1 Some gestures, their labels and corresponding text.....	48
Table 3.8.2 Gesture sequence .....	50
Table 4.3.1 Scenario A.....	57
Table 4.3.2 Scenario B.....	58
Table 4.3.3 Summary of results .....	60
Table 4.3.4 Removing low achieving results.....	61
Table 4.4.1 Confidence level data for 'a' .....	63
Table 4.4.2 Confidence level data for other labels .....	64
Table 4.4.3 Confidence level data for label 'j' .....	66
Table 4.5.1 List of phrases to sign .....	69
Table 4.5.2 Results of word phrase test .....	70
Table 4.5.3 Average time for a gesture.....	70

# Chapter 1 Introduction

The Introduction chapter provides an informative preliminary breakdown of the research problem. It defines the research problem, identifies the objectives that must be done to overcome the problem, and the scope in which the project should be carried out.

## 1.1 Problem Definition

Verbal communication and speech are what sets apart humanity from other beings. Over the course of centuries, various human communities have developed innumerable forms of communication. Unfortunately, there are people with disabilities who cannot communicate verbally or who are hearing impaired. Other means of communication, such as sign language and Braille has been developed for their convenience.

In recent years, there has been an increasing tendency to smoothen the discrepancies in communication between languages with varying success. Translation programmes, voice recognition and text-to-speech applications have come a long way since their primitive beginnings. With this new wave of bridging gaps in communication, Sign language recognition software coupled with image processing and Artificial Intelligence, have also seen a surge in interest among the computer science community. Sinhala sign language is the main method of communication for the hearing and speaking impaired community of Sri Lanka. Their main obstacle in interacting with the public is that most of the people do not know how to interpret sign language. This problem in communication is disadvantageous for the hearing and speaking impaired and helps isolate the two groups even further. Developed countries and in languages where there is a large user base, there are many projects in place to lessen these problems; but since the sign language user community in Sri Lanka is much smaller than in other countries, there has not been much a lot of effort put into projects and research to bridge the gap between the two communities.

Sri Lanka's research in this area is still in its preliminary stages and a viable application is not available for use; granted, there have been several studies and projects conducted for Sinhala sign language, but they employ the use of wearable or hand-tracking technology that requires extra equipment; which can sometimes, be expensive.

Therefore, there is the need to develop a simple, easy to use, and inexpensive system that facilitates communication between those who use sign language and those who do not. An ideal application would be where one points a mobile phone camera at a signing person and the language is instantly converted to speech. We're not there yet. This project aims to take a step in that direction by laying the groundwork needed – a software application that can interpret hand gesture signs and convert them to text in real-time.

## 1.2 Project Objectives

The main objective of this project is to design and develop a software application that captures video in real-time of a person using Sinhalese Fingerspelling sign language, processes the frames through a neural network based on machine learning and interpret the signed hand gestures in the video and output text to a screen as words. This will create a system that interprets Sinhala sign language to text thereby facilitating the communication between a hearing-impaired person and someone who does not know sign language. In route to achieving the aforementioned objective, the project will also deal with research on the deaf community, their sign language usage, the evolution of the hand gestures, and involvement and the role of the human interpreter.

Hence, the other objectives that will be covered include:

- Create an openly available centralized image dataset for Sinhala Finger Spelling hand gestures which can be used in future research.
- Research on Sri Lanka's Hearing-Impaired community, the most commonly used sign language system within the community and the difficulties faced when communicating with people who do not understand sign language.
- Observe the different variations and expressions in hand gestures specific to Sinhala sign language.
- Closely study the average rate of communication via sign language; the proposed solution should be able to be on par with the speed of hand gestures and interpret them in real-time.

### 1.3 Project Scope

Design and implement a standalone desktop application that uses a single camera (web camera) to capture and recognize hand gestures of Sinhala Sign Language. The scope of sign language recognition is limited to static gestures of the fingerspelling alphabet.

The application will capture hand gestures as letters and attempt to cluster and recognize them as words with the help of a convolutional neural network. Note that, since the language recognition is limited to static hand gestures, the words that can be recognized are also limited to words that are made up of static hand gestures. In the completed system, User 1 stands in view of the camera and communicates with hand gestures; the camera captures the video stream and processes the captured frames in real-time using a pre-trained convolutional neural network to interpret Sinhala sign language and output their corresponding text to a screen.

The output text on the screen is read by user 2 (who is not proficient in sign language). Video capture, processing and output screen are on the same device e.g. Laptop, Desktop PC. Therefore, in this project, the scope is limited to one device only and the communication takes place in presence of both the signee and the reader.

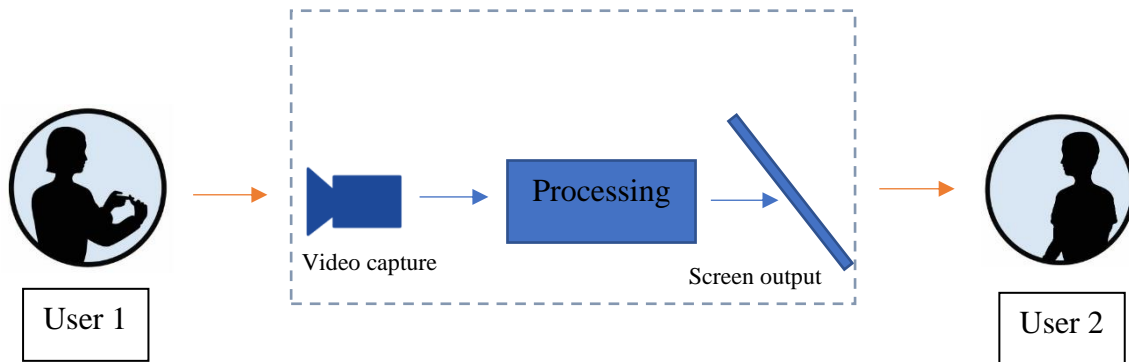


Figure 1.3.1 Proposed system

### 1.4 Summary

The opening chapter discussed the problems faced by the hearing-impaired community, the gaps in communications and justified how a project that attempts to bridge the gap could be beneficial. The next section proposed a solution for this problem and defined a scope under which a project would be carried out.

# Chapter 2 Literature Review

## 2.1 Introduction

This chapter deals with the background study and past research related to the project domain. It starts off with the beginnings of sign language and goes onto describe the development of various methods of fingerspelling. The chapter next describes the two main methods of hand gesture detection: using sensors and vision-based detection.

It then details the different approaches and their results taken by past researchers to tackle the problem of implementing an automated sign language interpreter. Finally, the section goes through approaches by Sri Lankans to attempt to automate the hand gesture recognition process and selects the best approach giving justification on why it is chosen.

## 2.2 Development of Sign Language

The development of language for communication can be considered as one of the most important milestones in human advancement. Linguistics categorize both vocal and sign languages as natural languages meaning that they evolved naturally in humans through use and repetition without conscious planning or premeditation.[1]

It is theorized that after humans developed full-fledged vocal languages, the prominence of hand gestures and signs to communicate took a secondary role and almost all civilizations used vocal languages as their main modes of communication; only when there is an impairment that hindered vocal communication did sign language come into use. Hence, sign language usage is mainly limited to hearing and speaking impaired. Although sign language is used primarily by the deaf and speaking impaired, it is also used by hearing individuals, such as those unable to physically speak, and people having trouble with the spoken language due to a medical condition or those with deaf family members, such as children of deaf adults. Therefore, anywhere communities of deaf people exist, sign languages have developed and evolved over time as a convenient means of communication.

## 2.3 Finger Spelling Alphabet

Eventually, another form of sign language developed called Manually Coded Languages; which uses the signs taken from a natural sign language but used according to the grammar of the spoken language[2]. These manually coded languages directly follow the grammar and syntax of the written form and usually have a specific hand gesture to represent a letter or syllable. Unlike the sign languages that have evolved naturally, these manual codes are the conscious invention of deaf and hearing educators and have been popularized in schools and educated communities.

Manually coded sign language is often referred to as the fingerspelling alphabet. Fingerspelling alphabets, though slower than conventional sign language, is slowly taking prominence within the deaf and hearing-impaired communities since fingerspelling is able to give sign language the same level of vocabulary and meaning to words and expressions as its written language counterpart.

Sinhala sign language is the main method of communication for the hearing and speaking impaired community of Sri Lanka [3]. Natural sign language and more specifically, fingerspelling alphabet, in Sri Lanka is heavily influenced by British sign language hand gestures.



*Figure 2.3.1 Sinhala fingerspelling alphabet*

## Why a Sign Language Interpreter is Needed?

The problem in daily life about communication is disadvantageous for the hearing and speaking impaired and helps isolate the two groups even further. According to the Sri Lanka Federation of the Deaf, there are over three hundred thousand (300,000+) deaf people in Sri Lanka. Moreover, the World Health Organization has revealed that approximately 9% of the population in Sri Lanka has a loss of hearing [4]. In developed countries and in places where have a large user base, there are many projects in place to alleviate these problems; but since the sign language user community in Sri Lanka is much smaller than in other countries, the state of research done on this area is still in its early stages.

However, in recent years, there has been an increasing trend to facilitate the discrepancies in communication with varying success, but these projects for Sinhala sign language interpreters use wearable or hand-tracking technology that requires extra equipment; which can be expensive. Sri Lanka's research in this area is still in its preliminary stages and a viable application is not available for use. Therefore, there is the need to develop a simple, easy to use, and inexpensive system that facilitates communication between those who use sign language and those who do not.

## 2.4 Approaches to Hand Gesture Recognition

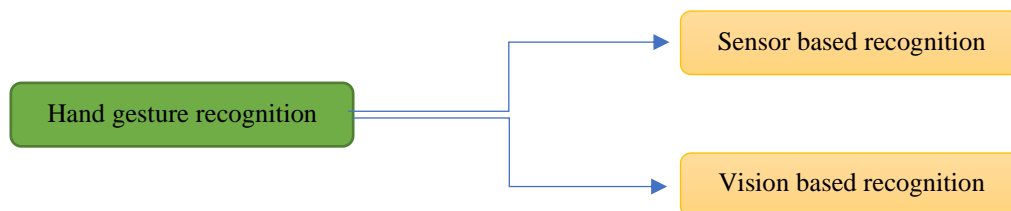
Although the objective of this project is to produce a sign language interpreter. Sign language recognition is inherently bound with hand gesture recognition. Hence, it is important to acknowledge the various methods by which hand gesture recognition is achieved in computer systems. With the advent of microcomputers in the 1970s, the processing power of computers saw a dramatic increase while the size of computers steadily decreased. This increase in processing speed and power allowed people to utilize image processing and recognition algorithms at lower costs rather than depending on large mainframes to process data [5]. Over the past 30 years, there have been many programs and algorithms developed for processing images and extract information from them. Among them, algorithms for facial, voice and (to a lesser extent) gesture recognition have been at the forefront of technological researches.



As mentioned above, facial and voice recognition has come a long way since its inception. There are viable commercial products already available and in use today that recognize faces and interprets voice commands.

Human gesture recognition, on the other hand, is still in its early stages of development and therefore, is still very much an open hurdle and an active area of research. In part due to it still being an open problem, multiple ways of approaching this issue has been suggested. The following sections aim to discuss various approaches by other researchers or projects.

The projects for gesture and in extension sign language recognition, can be divided into two broad categories. A sizable portion of the projects is based on sensors to detect arm or finger movement while the other approach uses vision to detect gestures.



*Figure 2.5.1 two main approaches for hand gesture recognition*

Let us examine the sensor-based approach first:

## 2.5 Sensor Based Gesture Recognition

Although it is not within the scope of this project to include external peripherals to aid in hand gesture recognition, such methods were taken into consideration during the background study for the project and are included here for completeness sake. A popular method of data capture is by using sensors. Different types of sensors are placed in the hand when the user performs the associated gestures and related data is recorded and analysed.

Examples of such projects include the use of a system based on accelerometers[6]. Here, they place accelerometers (coupled with gyro meters for improved accuracy) and with the aid of a transmitter, sends data to a computer that passes data through an OCR (Optical Character Recognition) which in turn deduces the implied gesture depending on the trajectory of the accelerometer movement. Another example of sensor-based gesture recognition is the use of sensory gloves[7]. These gloves have fixed gyro or accelerometers attached to them and work

on similar principles shown above[8]. An interesting example of the use of general-purpose devices been adapted for hand gesture recognition is the use of a Nintendo Wii controller[9].



*Figure 2.5.2 - prototype sensory glove*

When considering the projects that utilize wearable technology, one research project, a paper published in 2016 at the International Conference on Advances in ICT for Emerging Regions stands out, partly due to the fact that it deals with Sinhala Sign Language recognition and also the fact that it combines several sensors input to formulate its results[10].

Here, the authors of the paper make use of Myo Gesture Recognition Arm Band[11]. The combination of gestural data (surface Electromyography) that measures the muscle activity and special data (accelerometer and gyroscope) that measures the hand movements for sign recognition produces a much better combination and makes it easier for a training model to reach its target quickly. The mapping is done by implementing multiple neural networks under supervised machine learning.



*Figure 2.5.3 - MYO Armband*

The authors claim that their method provided a 100% success rate for dependent study and 94.4% accuracy for an independent study of hand gestures. The main disadvantage of using wearable peripherals for gesture recognition is that it hinders hand movement and, in some cases, can be expensive to implement.

## 2.6 Vision based Gesture Recognition

Vision based approach makes use of an image/video capture device to extract visual information for gesture identification. This approach is more complex than measuring hand movements with sensor data because image processing of bare hands against a dynamic background requires multifaceted algorithms to extract information and is, therefore, more sensitive to errors.

The changing background is a large barrier in this approach; therefore, most projects have tried to confront this issue by subtracting the background from the hand and then applying algorithms on the remaining data. An easier route taken by most projects is to use devices such as Microsoft's Kinect or LeapMotion. These modules have built-in capability to track hand movements thereby eliminating the need for low-level data extraction. One such project published in Sensors scientific Journal in 2015[12], uses Microsoft's Kinect sensor and applies a hierarchical conditional random field (CRF) that recognizes hand signs from the hand motions. They obtain a success rate of 90.4%.

Their focus is on tackling three main problems.

1 – differentiating between signed and non-signed patterns in the continuous hand-motion stream.

2 – some signs share similar patterns and 3 – each sign begins and ends in a specific hand shape.

Various other projects also use a similar approach when using Kinect to identify gestures[13].

A recent project involving Sinhala sign language and the Microsoft Kinect has been published in GSTF Journal on Computing[14]; in it, the researchers track skeleton points such as shoulders, elbows and wrists and try to map them to a gesture dictionary data file.

This project does not deal with finger spelling Sinhala sing language and is able to identify about 10 words based on gestures. Results show 94% success.

Yet other projects employ the use of coloured bands on fingers or fingertips. This greatly improves the success rate and accuracy of the system but also acts as a deterrent for adoption since it requires an extra effort on the users' end to procure armbands or gloves with coloured fingertips. As mentioned earlier, the objective of this project is to produce a simple and easy to use inexpensive system. This calls to do away with external peripherals like armbands, gloves or Kinects. The next section of the chapter focuses on vision-based approaches that use a simple digital camera to extract gesture information.

## Vision based Gesture Recognition – Camera

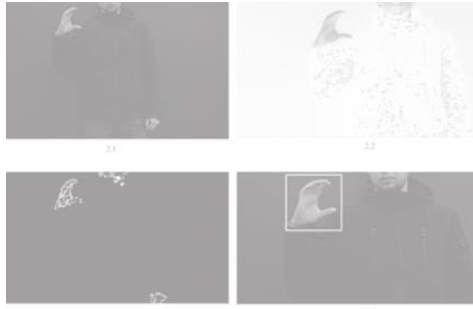
Before examining gesture recognition methods by using a camera alone, let us first identify the high-level stages involved in recognizing a hand gesture. A vision-based hand gesture recognition system can be divided to two components.

1. Detection
2. Recognition

The detection component of the system is responsible for the definition and extraction of visual features of the hands. The recognition layer is responsible for the grouping of spatiotemporal data which is extracted in the detection component and assigning the result to groups associated with a particular class of gesture.

## 2.7 Detection

This is the starting point for all vision-based sign language recognition systems. Detection isolates the region of interest by segmenting off the corresponding image regions before being passed on to the recognition phase.



*Figure 2.6.1 - Extracting region of interest through filters*

There are many ways of extracting this information, but when it comes to hand gesture recognition, almost always, the images are sent through a colour filter akin to skin colour and subtract the background from the region of interest [15].

## 2.8 Recognition

The recognition phase interprets the denotation of the gesture based on posture and position of fingers and hand. Recognition of hand gestures, or signed letters as in this case, becomes difficult as the vocabulary increases.

Template matching is a widely used technique in the recognition stage. In pattern recognition, the obtained image from detection is matched with an already existing template. The template matching process does a pixel-by-pixel comparison of data for the obtained image and of an image in a template. The best matching prototype to the obtained image is considered as the final result. Template matching requires a certain level of threshold and can be highly skewed by the training image data that was used to obtain the template prototype. Therefore, it is crucial to carefully calculate the levels of threshold and variance allowed since this leads to errors when identifying gestures. Template matching can be implemented and combined with different techniques in various ways. Let us consider a few of the more common methods with high success rates.

## Points of Interest

Mekala et al. [16] make use of neural networks for identification and tracking to translate the sign language to a voice/text format. They do this by of Point of Interest (POI) regions or track points on the fingertips and center of the palm. The input goes through Gaussian and Median filters and finally through a Sobel edge detection filter which helps identify the ‘points of interest’ i.e. the fingertips.

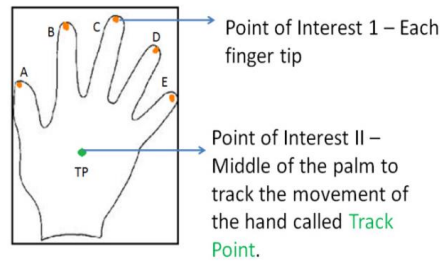


Figure 2.6.2 - Mekala et al. Points of Interest

In the final iteration, 55 fingertip elements are collected. Their main concern is to reduce the computational strain on the system, admittedly, their system uses less memory but in doing so the methodology by which the data is interpreted becomes complex. They claim that they achieved a 100% success rate, but it is unclear if the evaluation is based on independent test data.

## Centroids

Pansare et al. [17] interpret American Sign Language with the use of a web camera. In the first stage, they capture still images and convert them the binary by using a median filter. Edge detection of extracted image using “Sobel” method finds edges using the Sobel approximation to the derivative edges at those points where the gradient of input binary image I is maximum. The feature vector is formed using centroid given by  $(C_x, C_y)$ .

$$(C_x, C_y) = \frac{1}{n} \left( \sum_{t=1}^n x_t, \sum_{t=1}^n y_t \right)$$

Feature matching is done measuring the Euclidian distance between a training image template and input image’s feature vector. The resulting vector that gives the least Euclidean distance is the gesture. Test results show a success of 90% but is limited in scope to static hand gestures.

In the Sri Lankan context, there is a research paper published in 2013 [18] that uses the Centroid method. They simplify the approach even further by dividing the processed image of the hand to four quadrants; then they obtain the centroid of the area covered by the hand in each quadrant and cross-reference it with an existing database of pre-trained data. Test results show a success rate of 92% for static hand gestures.

The centroid method and its variations are straightforward and relatively easy to implement hence it is unsurprising that there are many projects using a variant of this approach. One drawback of the centroid method is that most sign languages are dynamic and require movement, therefore although the Centroid method produces relatively good results there is a tradeoff and higher error rate than more vigorous methods like Markov Models

## Markov Models

Apart from template matching mentioned above, in recent years, Markov Models have seen a surge in popularity in recognition systems. One of the earliest applications of Markov Models in sign language can be found in the IEEE TPAMI Journal of 1998 December issue; Starner and Pentland describe [19] a system to recognize hand sign symbols by using a Markov Model. Up until that point, most projects focused on using a template matching or neural net approach for recognition.

One of the most promising methods of detection is Hidden Markov Models. It proposes an approach by using the Adaboost algorithm to detect the user's hand and a contour-based hand tracker is formed combining condensation and partitioned sampling [20]. This approach is divided into two main sub-processes named as pre-processing and classification process. The classification process uses a model called Discrete Hidden Markov Model (Baum-Welch algorithm is used to train). According to the results provided, the average recognition rate of Hidden Markov Models is better than other traditional methods with a 93 – 97% success rate.

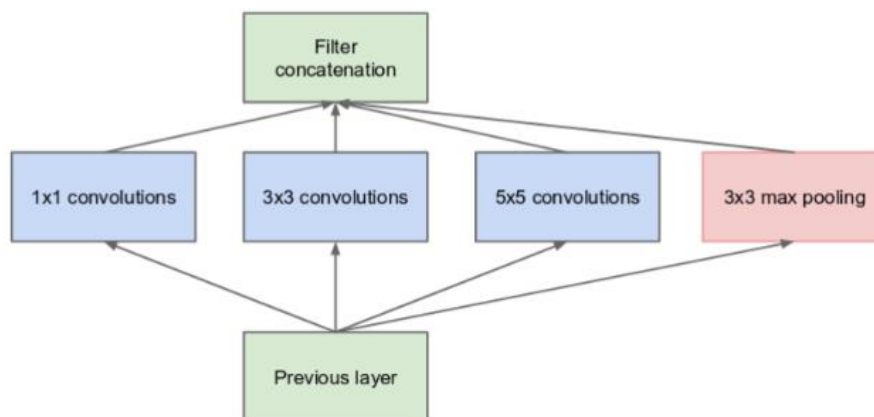
## LeNet and Convolution Architecture Models

LeNet, first introduced in 1998 [21], is a popular image recognition model that is used in commercial products. Although LeNet requires considerable computing power as the image details increase; image pre-processing can help alleviate these hurdles before LeNet is applied. Potkins and Philippovich [22] in their research, come up with a modified version of LeNet-5. The main disadvantage of LeNet-5 is overfitting in some cases and no built-in mechanism to

avoid this. They, therefore, improved the benchmark architecture by adding dropout layers. The new version gives results of 93% whereas earlier it was 87%.

As mentioned above, one major drawback of the earlier convolution neural networks is overfitting, Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. This "fully-connectedness" is what causes overfitting. Inception was a neural network that was introduced much later at the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) in a paper titled "Going deeper with convolutions" [23]. The main argument from the authors of Inception was that regions of interest (relevant parts) inside the image can have large variations in size. Therefore, choosing the right kernel size for the convolution process is important.[24] A large filter is preferred for information that is spread throughout the image, but a smaller filter size may be preferred for more localized details.

Unlike earlier neural networks which stacked layers (making it computationally intensive), Inception was designed to have multiple sized filters at the same level. This makes the network "wide" rather than "deeper".



*Figure 2.6.3 Inception has varying filter sizes for the same layer*

Neural networks are generally computationally expensive. With Inception, (and then improved in later versions) the authors added an extra 1x1 convolution before larger convolutions to reduce the computation cost.

In 2015, versions 2 and 3 of Inception were published. These new iterations improved upon the original by implementing auxiliary classifiers, tackling bottleneck issues. Release of

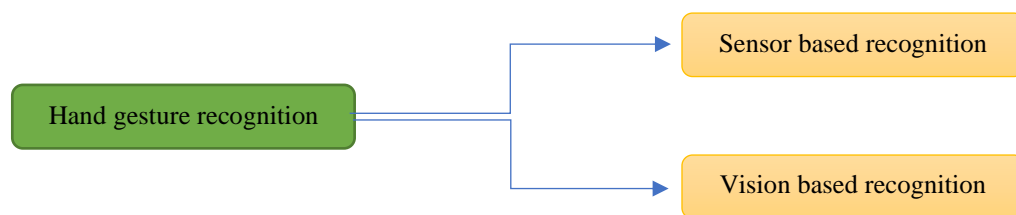


Inception has revived interest in convolution neural networks and is now one of the go to machine learning solutions for researchers.

## 2.9 Selecting an Approach

As defined in the scope of the project, preference was given to do away with any wearable technology and therefore a vision-based approach was selected to identify the gestures.

Even within the vision-based approach – any external peripherals should not be worn by the user. Given that the decision was made to go ahead with a vision-based approach – identifying gestures using a neural network was the best solution.



*Figure 2.9.1 Two main recognition techniques*

Based on the factors mentioned above in section 2.8, and the recent resurgence of convolution neural networks, it was decided to use the 3<sup>rd</sup> version of the Inception model as the training mechanism for this project. Another important point that factored into Inception being selected is that the project intends to use a pure machine learning-based solution to the task.

There have been projects and research conducted in Sri Lanka which attempted to use wearable technology or pure image processing alone to identify hand gestures but there have been no substantial attempts at recognizing Sinhala sign language using Neural Networks. Another point of note is that, if this project sees further development, using a model that is well established and is seeing continuous support from the community is beneficial for the project. One could argue that a large model like Inception would be overkill for a small-scale project like detecting hand gestures. But it is also worth noting that, convolution neural networks have always been CPU intensive and there is no considerable increase in resource consumption for Inception when compared with the other Deep Neural Networks that perform similar tasks.[25]

## 2.10 Summary

Literature Review Chapter went over the approaches taken by different researchers to solve similar problems. It considered the pros and cons of each path and then made the decision to use a vision-based approach to detect the gestures and implement Inception v3 – a convolution neural network to train and interpret the hand gestures.

The decision to use this particular model was based on the fact that limited research had been done in Sri Lanka for similar projects using neural networks, the fact that Inception has been successfully applied for a wide variety of image processing tasks, and partly due to the rekindled interest of neural networks within the image processing community.

# Chapter 3 Methodology

This chapter describes the design architecture and implementation steps of the proposed system. It begins with a high-level system diagram and breaks down each component to sections and then delves into how each component functions with a detailed description at each step.

## 3.1 High Level System Design

As outlined in the *Project Scope* section of the first chapter, the system will have three main application components.

1. Collecting data (images)
2. Training the network
3. Final Application which can interpret the hand signs

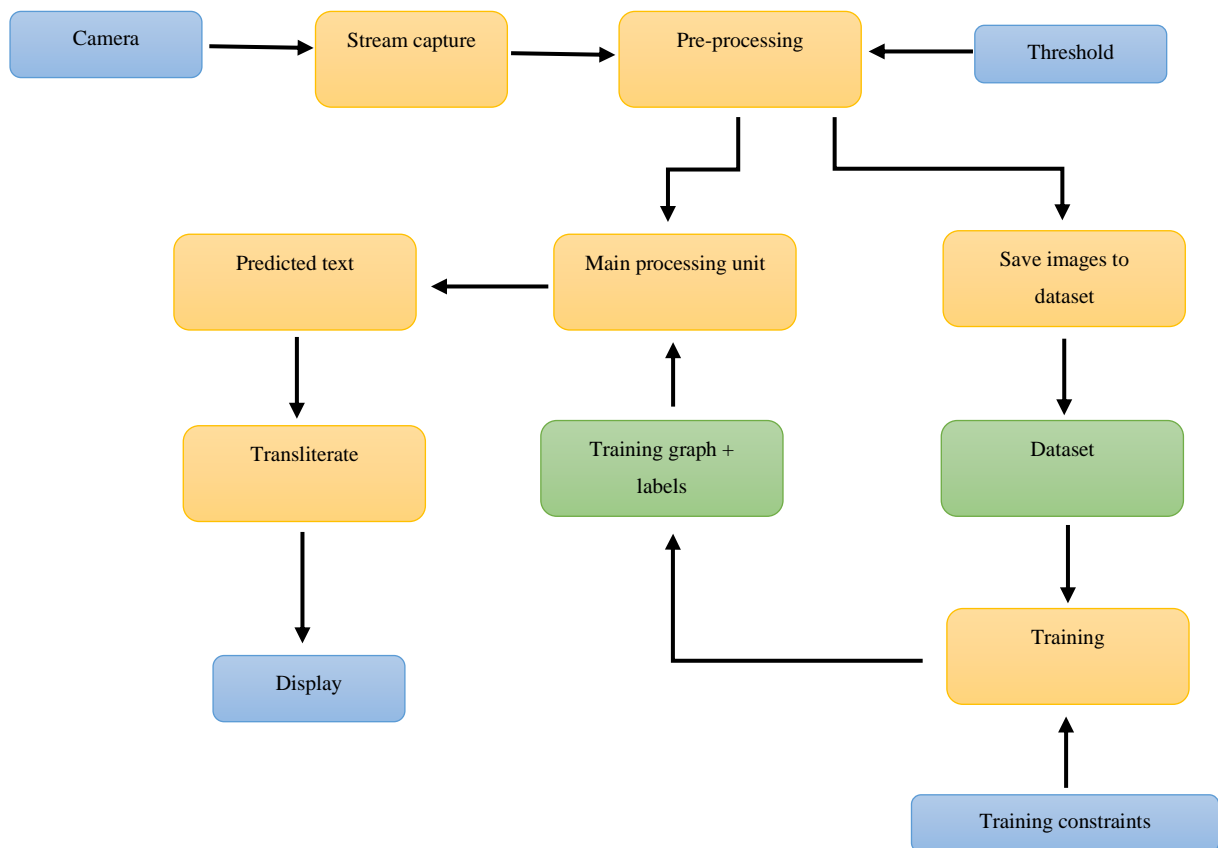


Figure 3.1.1 High level diagram of the system

In the data collecting section of the application, the components shown below come into play. The camera captures the images and creates a frame within the image to slice out a region of interest. The captured image is then processed for skin detection, background removal and smoothing before it is saved in the database.

A detailed explanation of each subcomponent is given later in the chapter.

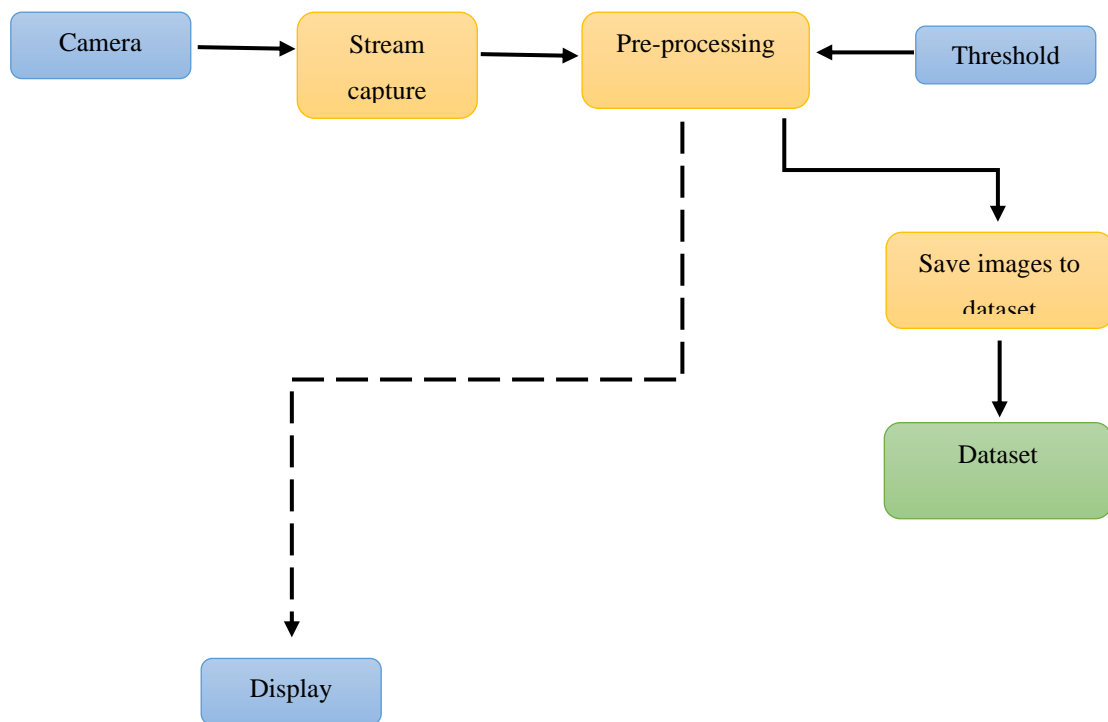


Figure 3.1.2 Creating dataset

Dataset would comprise of a folder structure described as follows: the main directory would be called *dataset* under dataset directory, there would be sub-directories with the labels for each hand gesture.

```
dataset/  
├── a  
├── aa  
├── az  
├── b  
├── delete  
├── dh  
└── e
```

Figure 3.1.3 Tree command for dataset directory

Figure 3.1.3 shows the directory structure. Note that *a*, *aa*, *az*, *b* and *delete* correspond to the labels of the hand gestures. A sub-directory would have about 1000 images for each hand gesture. In the training phase, the images that were saved to the dataset is trained on the inception neural network.

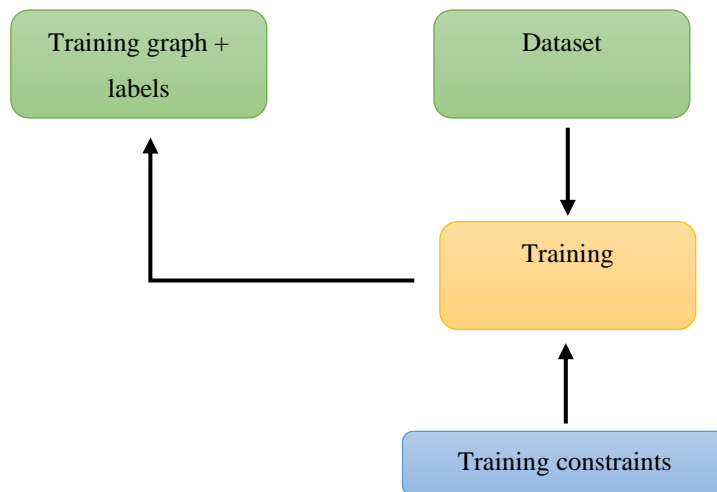


Figure 3.1.4 Training

The training module builds a list of training images from the dataset directory. It analyzes the subfolders in the dataset directory, splits them into stable training, testing, and validation sets, and returns a data structure describing the lists of images for each label and their paths. It also sets the number of iterations i.e. the training steps epoch etc. There is no interface for the training module; output on the terminal would be adequate to show the training progress, validation accuracy and entropy.

The main arguments to run the training programme would be a string path to a directory containing subfolders of images, testing percentage - integer percentage of the images to reserve for tests and validation percentage - integer percentage of images reserved for validation.

A complete list of input arguments is given below:

- image\_dir = Path to folders of labeled images.
- output\_graph = Where to save the trained graph.
- output\_labels = Where to save the trained graph's labels.
- how\_many\_training\_steps = How many training cycles to run before ending.
- learning\_rate = How large a learning rate to use when training.
- testing\_percentage = What percentage of images to use as a test set.
- validation\_percentage = What percentage of images to use as a validation set.
- eval\_step\_interval = How often to evaluate the training results during training and print output to the console.
- bottleneck\_dir = Path to cache bottleneck layer values as files.
- final\_tensor\_name = The name of the output classification layer in the retrained graph.
- model\_dir = path to the inception model
- summaries\_dir = where logs and training summary is saved.

Most of these arguments are self-explanatory, few other parameters such as the option to flip images, crop or change the brightness of images randomly can also be considered. The final application, the one that the end operator uses, should be a GUI. It should have a windowed interface, running a standalone desktop application on a single window. The application should have a video frame showing the input video, a trackbar or slider to adjust threshold values for input and an output showing the Sinhala text.

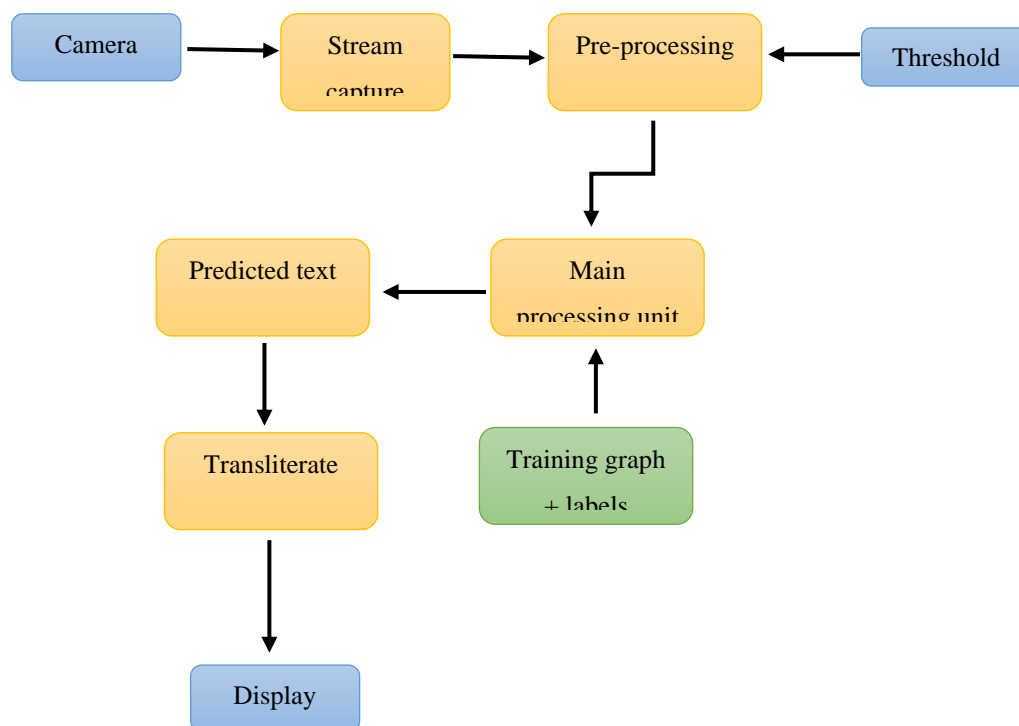


Figure 3.1.5 Setup for the final application

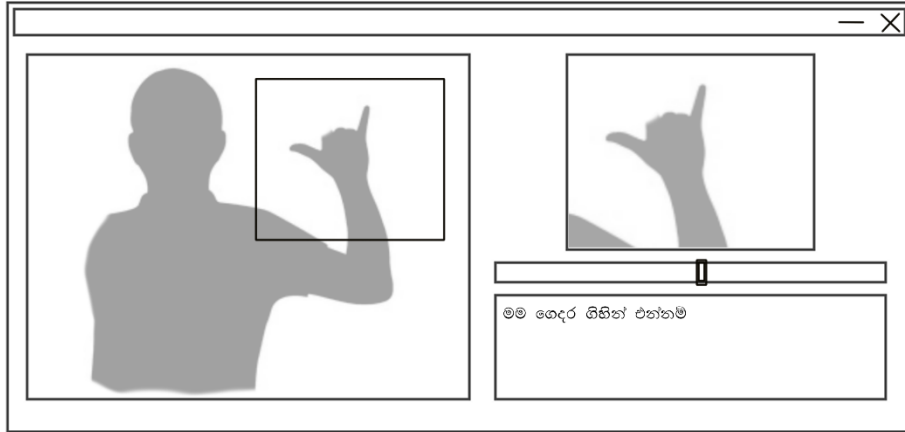


Figure 3.1.6 Proposed application wireframe mockup

Figure 3.1.6 shows how the proposed application would look like after it is complete. The large window on the left shows the web camera input with a guidance area to show that the user must place the hand within that region for the system to interpret the signs.

The smaller window on the right shows a masked silhouette of the hand with the background removed. It is this area that will be processed by the neural network. There is a slider below the small window, it controls an input threshold – for example, if the room is dark and there is noise in the background, the user can move the slider/trackbar to change the lighting threshold on the mask so that a well-balanced image is sent for processing. Finally, an output text area is displayed below the trackbar which shows the interpreted Sinhala text output.

## 3.2 Implementation and Components

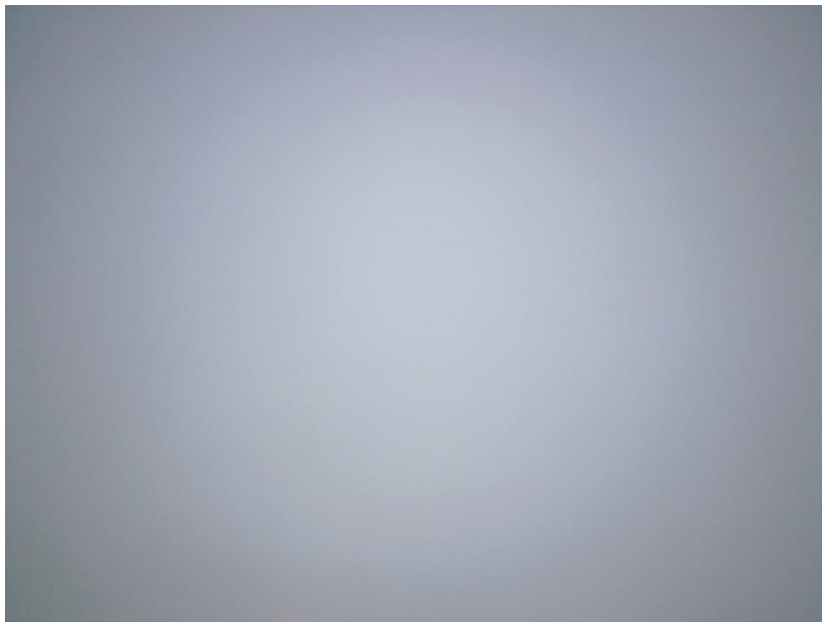
### Video Capture

Early in the project scope and design phase, the decision was made to keep using a low-resolution camera so as to prove that the system is suitable for use without the need for extra equipment or in the future if the system is ported to a mobile device, the system will function without utilizing much system resources. Therefore, an integrated webcam with a resolution of 720×1280 (about 0.7 megapixels) able to capture 30 fps was selected.

The figure below shows an image of a white surface captured by the webcam. Note the amount of noise in the picture. Then compare it with figure 3.2.2, an image of the same surface, captured at a higher resolution under ambient light conditions.



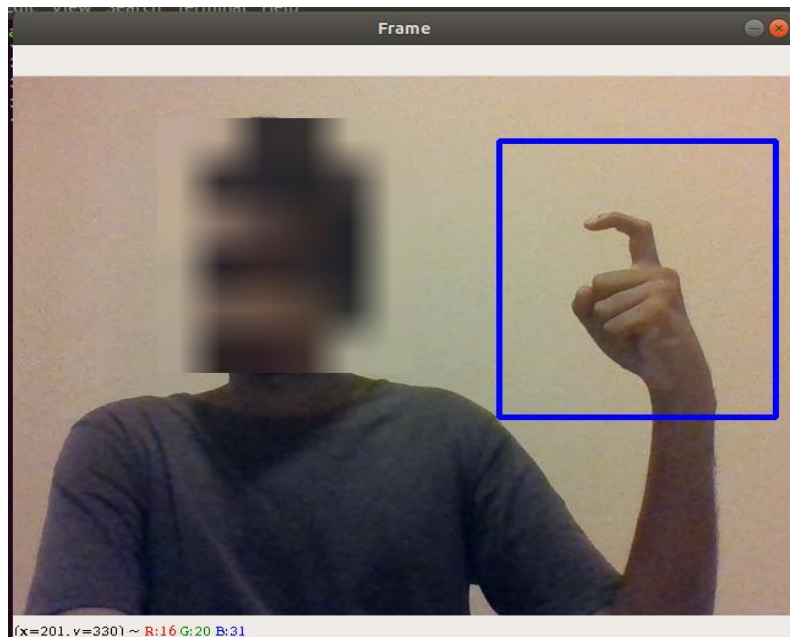
*Figure 3.2.1 0.7MP image of a white surface under ambient conditions*



*Figure 3.2.2 5MP image of white surface*

Also, take note of the white balance of the image. These defects in image capture need to be addressed later in the preprocessing stage. The next step was to use define a region of interest, an area on the video frame the user would have to place their hand for the system to recognize a gesture.





*Figure 3.2.3 Video capture with region of interest marked*

Figure 3.2.3 shows a user making hand gestures within the region of interest. The square on screen is about 300x300 pixels – the preferred input size for Inception training images. The square was drawn through OpenCV and has no impact on image processing other than acting as a guide for the user to show the area to place hand. OpenCV has an inbuilt function to draw shapes on screen:

```
cv2.rectangle(img, (x1, y1), (x2, y2), (255,0,0), 2)
```

### 3.3 Pre-processing

#### Region of Interest

Once the region of interest is sliced and extracted from the video frame. It would look something like the image shown below in figure 3.3.1 – a 3 channel RGB image with 300 by 300 pixels and a colour depth of 24 bits. It is on this image that subsequent image processing techniques are applied.



*Figure 3.3.1 Sliced image*

## Skin Detection

Skin detection is the process of finding skin coloured pixel regions in a captured frame. Skin detection is generally used as a preprocessing step in security cameras, consumer electronics and various other instances where humans need to be identified. A skin classifier transforms a given pixel into an appropriate colour space and then uses a threshold level to match the pixels to the skin area. It groups pixels together and defines a decision boundary for the skin area.

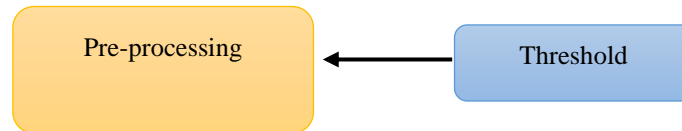
Researchers have found that RGB performs poorly in skin detection, therefore, most researchers suggest the use of other colour spaces such as HSV, HLab or YCrCb [26]. For this project, it was decided to use YCrCb as the colour scheme of choice to detect skin, mainly because it allowed us to control the thresholds for the Y and Cr components. YcrCb was actually developed for digital video and colour television – based on earlier YUV colour space.

In OpenCV, image data can be converted between colour spaces as shown below:

```
img_ycrCb = cv2.cvtColor(img1, cv2.COLOR_BGR2YCR_CB)
```

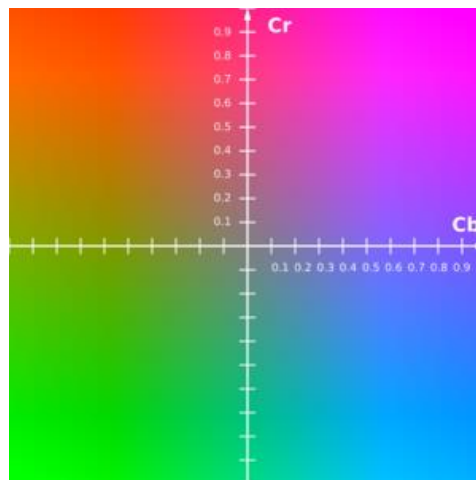
Y in YCrCb stands for Luminance. Luminance or Luma is the brightness or intensity component of colour. It is similar to the grayscale version of an RGB image. Cr and Cb are the red and blue chroma components respectively. By definition, Cb is the blue component relative to the green component and Cr is the red component relative to the green component.

A section of the high-level design diagram is shown below. During training or when the application is being used, the user can change the threshold – i.e. the values of Cr and Cb so that the skin is detected, and the other parts of the image are eliminated.



*Figure 3.3.2 Input threshold from user controls the background removal mask*

As stated earlier, the choice of YCbCr depended on the fact that luminance and Chroma red value can be manipulated easily. As you can see in figure 15, Chroma red lies on the y-axis and a threshold can be set easily where we ask the application to select values for a given range. This is a crucial step in preprocessing because it helps identify various skin tones and the threshold can be adjusted so that all types fair or dark, under different light conditions, can be handled by the application.



*Figure 3.3.3 Colour space of YCrCb [27]*

Figure below shows a sliced image that has been converted to the YCrCb colour space. Note how the skin tone is clearly distinct from the objects in the background.



Figure 3.3.4 Image in YCrCb colour space

Figure shows how the threshold can be changed by controlling one value of the plane – especially for skin tones.

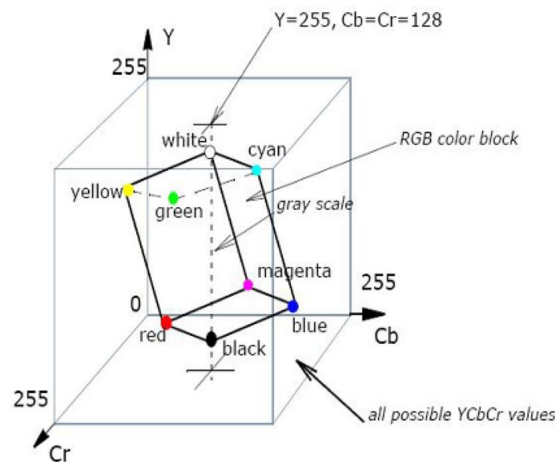


Figure 3.3.5 Colour space on x-y-z plane[28]

## Creating a Mask

In the previous section it was mentioned that one reason for converting RGB to YCbCr colour space was because it allowed us to implement thresholds that identify skin tones easily. The figure below (fig 3.3.6) shows the threshold trackbars created to control the colour space levels to create the mask to identify skin tones.

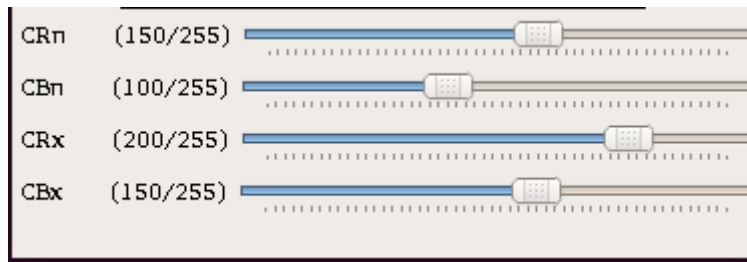


Figure 3.3.6 Trackbars created in OpenCV show colour space thresholds.

CRn, CBn, CRx and CBx represent the following parameters as shown in the code below.

```

16 skin_ycrcb_min = np.array((0, CRn, CBn))
17 skin_ycrcb_max = np.array((255, CRx, CBx))
18 mask = cv2.inRange(blur, skin_ycrcb_min, skin_ycrcb_max)

```

Figure 3.3.7 Creating the threshold

CRn is the most used parameter since it has the Chroma red parameter – the colour closest to Sri Lankan skin tones.

Once a threshold is identified for the skin tones, a contour can be created to show the outlines on the hand. Few examples with varying threshold values for chroma red are shown below:

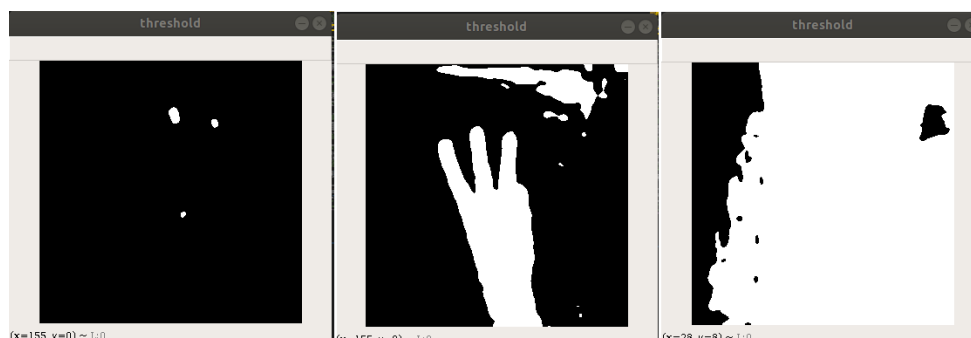


Figure 3.3.8 CRn 145, 135 and 125

Once the optimum threshold values are selected by the user, the image data is passed on to the next section of the preprocessing stage. You'll note that, in the center image (fig 3.3.8), there are some areas outside the hand that the mask has failed to capture even though the user has tried to eliminate the background through changing the thresholds. In figure 3.3.9, there are several white areas. Usually, the area covered by the hand is the largest contour. The next step is to find the largest contour and eliminate the rest.



Figure 3.3.9 Largest contour is the hand

The largest contour by area can be found as follows:

```

30 contours,hierarchy=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
31 areas = [cv2.contourArea(c) for c in contours]
32 max_index = np.argmax(areas)
33 inx=contours[max_index]
34 cv2.drawContours(ctr, inx, -1, (0,255,0), thickness=cv2.FILLED)
35 ctr = cv2.fillPoly(ctr, pts =[inx], color=(255,255,255))

```

Figure 3.3.10 Finding the largest contour

The code snippet above cycles through the contours and fills out the largest contour. There are some in-between steps to apply Gaussian blurs and dilation so that the full area of the hand is selected to our mask.

Once the program discovers the largest contour i.e. the hand, it creates a mask and performs a bitwise and operation on the original image so that the background is eliminated. The final result is given in figure 3.3.11 – it is an 8-bit grayscale image of resolution 299x299. This is the image that is used in training or is sent to the main processing unit for identification.



Figure 3.3.11 Output from preprocessing








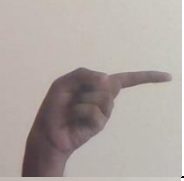

### 3.4 Creating the Dataset

Once the preprocessing system was set up, the next step was to create a dataset to train the neural network. As mentioned earlier, it was decided to capture a 3-channel grayscale image with 300x300 resolution. There are 56 hand gestures in the fingerspelling Sinhala alphabet. This 56 gestures include all the standard letters of the alphabet, the common diacritics and gestures for rarely used diacritics such as *gayanukiththa* (ගයනුකිත්ත) ്, *gaṭa sahita elapili deka* (ගැටය සහිත ඇලපිලි දෙක) ്aa, etc.






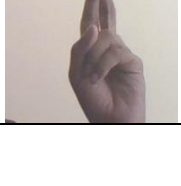
Early in the project, the decision was made to drop the rare and uncommon diacritics from the dataset. The scope of the project also limits the hand signs that need to be recognized by the system to static gestures only. These limitations were imposed because most of the gestures for diacritics and non-static gestures have similar hand poses to those of more commonly used letters. For example, the gesture for *ඛ* and *ඞ* are almost identical. Their only difference is that, *ඞ* performs a swivel motion with the hand as it is being gestured. Therefore, it was decided that, in order to give a better chance for the system to identify *ඛ* and for the fact that, both letters convey the same sound, *ඞ* would not be implemented to the system.

After limiting the dataset to static gestures, 27 different hand gestures remained; hence 28 subdirectories were created under *dataset* directory. 27 hand gestures included gestures for a ‘space’ sign to signal a space between words and a ‘delete’ gesture to signal that the last letter on the screen should be removed. An extra directory was added to show the blank frames where

there are no hand gestures. These 28 directories each have 1000 images from 5 different people totaling close to 28000 images.

Hand gesture	Label	Letter
	a	අ
	aa	ආ
	az	ඇ
	i	ඉ
	u	උ
	e	ඌ
	k	ක
	g	ග
	j	ඊ



	t	တ
	dh	ဋ
	th	ဋာ
	d	ဋီ
	n	ဋာ
	p	ဋာ
	b	ဋီ
	m	ဋီ
	y	ဋာ
	r	ဋာ









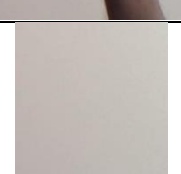
	l	ಲೆ
	w	ಲೈ
	s	ಸಿ
	h	ಹಿ
	nz	ಂ
	ch	ಲೈ
	space	N/A
	delete	N/A
	nothing	N/A

Table 3.4.1 List of trained gestures

Care was taken to make sure that the images were captured from similar devices that are expected to be used when running the application. Understandably, if a high-end camera with

higher resolution was used – it would have produced much more detailed images, but it was decided to capture the training images with the same resolution that Inception Neural Network requires as input rather than perform resizing operations.

Another task that required close attention was the process of capturing images for gestures with similar hand gestures. Looking at figure 3.4.1, it is evident that the gestures for ལྷོ, ལྷོ, ལྷོ and ལྷོ have similar gestures. During the data capture process, the participants made sure that their hand positions and finger locations for these letters were placed correctly.



*Figure 3.4.1 Note how all these hand gestures are similar*

Another important problem when creating the dataset was to avoid turbulent backgrounds. The mask does remove the background, but it is always advisable to capture the frames with a plain background.

For example, the image below has a light bulb in the background, this distorts the pixels on the hand area as well and can have a negative impact on the training data.



*Table 3.4.2 Pixels washed out by background light*

Through trial and error, it was found that a dim plain background with the hand showing minimal shadows yielded the best results.



*Table 3.4.3 A good image doesn't have shadows*

### 3.5 Training

Once dataset was complete, next step in the system implementation process was to train the images. For training, another python script was created which acted as a standalone application independent of the final app. A GUI wasn't created since training is not a common/repeated

task. The user can start the training programme from the terminal by running the `train.py` Python script. A sample use case is given below. Note how several input arguments are applied (the full list of parameters is given in *Chapter 3 – High-Level System Design* section of this report)

```
(venv) [redacted]@pc:~/[redacted]/v12/copyofv12$ python3 train.py \  
> --bottleneck_dir=logs/bottlenecks \  
> --how_many_training_steps=2000 \  
> --model_dir=inception \  
> --summaries_dir=logs/training_summaries/basic \  
> --output_graph=logs/trained_graph.pb \  
> --output_labels=logs/trained_labels.txt \  
> --image_dir=./dataset \  
> --learning_rate=0.1
```

Figure 3.5.1 Sample use of `train.py`

The initial section of the `train.py` defines the functions and sets up the images.

### Create Image Lists Function

The first function creates the image lists for training, validation and testing. It builds a list of training images from the dataset provided, analyzes them accordingly, builds validation and testing sets based on the defined percentages and returns a data structure describing images and their corresponding labels.

```
53 def create_image_lists(image_dir, testing_percentage, validation_percentage):  
54     if not gfile.Exists(image_dir):  
55         print("Image directory '" + image_dir + "' not found.")  
56         return None  
57     result = {}  
58     sub_dirs = [x[0] for x in gfile.Walk(image_dir)]  
59     is_root_dir = True  
60     for sub_dir in sub_dirs:  
61         if is_root_dir:  
62             is_root_dir = False  
63             continue
```

Figure 3.5.2 Creating image lists

As you can see from the snippet above, `create_image_lists()` function requires three arguments; `image_dir` path to the image dataset, `testing_percentage` 10% of the images were used for testing and `validation_percentage` again, 10% were used for validation. As mentioned earlier, the complete dataset contains 1000 images per label and there are 28 labels which brings the total to 28000 images. Next, there is a simple check to verify if there are enough images in the sub directories:

```
70 if len(file_list) < 20:  
71     print('WARNING: Folder has less than 20 images, which may cause issues.')
```

The most important step of the function is the next few lines of code. It cycles through the images in each subdirectory (label) and creates a hash. The hash is used to generate a probability value to decide which category (testing, validation or training) the image should go to.

```
75 for file_name in file_list:
76     base_name = os.path.basename(file_name)
77     hash_name = re.sub(r'_nohash_.*$', '', file_name)
78     hash_name_hashed = hashlib.sha1(compat.as_bytes(hash_name)).hexdigest()
79     percentage_hash = ((int(hash_name_hashed, 16) %
80                        (MAX_NUM_IMAGES_PER_CLASS + 1)) *
81                       (100.0 / MAX_NUM_IMAGES_PER_CLASS))
82     if percentage_hash < validation_percentage:
83         validation_images.append(base_name)
84     elif percentage_hash < (testing_percentage + validation_percentage):
85         testing_images.append(base_name)
86     else:
87         training_images.append(base_name)
88     result[label_name] = {
89         'dir': dir_name,
90         'training': training_images,
91         'testing': testing_images,
92         'validation': validation_images,
93     }
94     return result
```

Figure 3.5.3 Assigning labels

The returned value is a dictionary containing an entry for each label subdirectory, with images split into three categories.

## Get Image Location Function

This function is used when creating bottleneck files from the images. It inputs the dictionary of training images created for each label, the label names assigned for the images, a category – test, validate or train, and *ind* – index integer offset of the image from the beginning. *ind* helps pinpoint an image in the subdirectory. It returns the file system path to an image.

```
95 def get_img_location(image_lists, labelname, ind, image_dir, cat):
96     label_lists = image_lists[labelname]
97     if cat not in label_lists:
98         tf.logging.fatal('No such cat: %s.', cat)
99     cat_list = label_lists[cat]
100    if not cat_list:
101        tf.logging.fatal('Label %s has no images in the cat %s.', labelname, cat)
102    mod_ind = ind % len(cat_list)
103    base_name = cat_list[mod_ind]
104    sub_dir = label_lists['dir']
105    full_path = os.path.join(image_dir, sub_dir, base_name)
106    return full_path
```

Figure 3.5.4 Get image location

Another function with a similar functionality is the `get_bottleneck_location()` function. It returns the path to the list of bottleneck files created during training.

```

@pc:~/v12/copyofv12/logs/bottlenecks/a$ cat saved_100_img.jpg.txt
0.5442202,0.00736502,0.52097136,0.532896,0.29959294,0.15473026,0.005340468,1.7302022,0.34369114,0.19216232,0.38614058,0.31749254,0.
0.46950296,0.08579703,0.49619517,0.0010494769,0.056313634,0.20615359,0.20431949,0.6998147,0.23135936,0.09393879,0.06044933,0.052748
045757823,0.0863757,0.063577354,0.1616531,1.8512914,0.35511938,0.56396306,0.08079804,0.9624541,1.8972828,0.12239854,0.09207047,0.43
47456124,0.04966002,0.04982276,0.45261353,0.16978782,0.6831471,0.008385545,0.03375749,0.0050619068,0.22264214,0.65543234,0.07566728
105,0.3011474,0.56118244,0.101686254,0.021610573,0.34092718,0.1622705,0.106905416,0.47333753,0.2707823,0.19350037,0.5448604,0.85641
704,0.0924105,0.48959762,0.14456661,0.12760027,0.04226679,0.25127247,0.3879266,0.32413045,0.057897083,0.43668965,0.03718185,0.07037
81484,0.17807364,0.012021416,0.11371928,0.7648723,0.28604823,0.18868364,1.5846158,0.15033981,0.07061878,0.5680906,0.047045693,0.017
5,0.2492748,0.50233513,0.29096276,0.11982454,0.19286226,0.48476523,0.5677535,0.08255747,0.5928304,1.1434137,0.053734645,0.112016425
24,0.32613164,0.16024314,0.1252879,0.25582075,0.19808987,0.18771417,0.27125037,0.0053605754,0.0032241985,0.21201387,0.16436087,0.28

```

Figure 3.5.5 Contents of a bottleneck image file for reference

Next function of note is the `create_graph()` function.

```

127 def create_graph():
128     with tf.Graph().as_default() as graph:
129         model_filename = os.path.join(FLAGS.model_dir, 'classify_image_graph_def.pb')
130         with gfile.FastGFile(model_filename, 'rb') as f:
131             graph_def = tf.GraphDef()
132             graph_def.ParseFromString(f.read())
133             bottleneck_tensor, jpeg_data_tensor, resized_input_tensor = (
134                 tf.import_graph_def(graph_def, name='', return_elements=[
135                     BOTTLENECK_TENSOR_NAME, JPEG_DATA_TENSOR_NAME,
136                     RESIZED_INPUT_TENSOR_NAME]))
137             return graph, bottleneck_tensor, jpeg_data_tensor, resized_input_tensor

```

Figure 3.5.6 Create\_graph function

It reads a saved graph from graph file (`graph_def = tf.GraphDef()`) and returns the graph file for trained Inception network.

## Bottleneck Files

Bottlenecks in a neural network help to reduce the number of channels in the network between layers. Because Inception uses small sized kernels, the feature representation can increase considerably as it passes through each layer.

```

247 def bottleneck(sess, image_lists, labelname, ind, image_dir,
248                cat, bottleneck_dir, jpeg_data_tensor,
249                bottleneck_tensor):
250     label_lists = image_lists[labelname]
251     sub_dir = label_lists['dir']
252     sub_dir_path = os.path.join(bottleneck_dir, sub_dir)
253     ensure_dir_exists(sub_dir_path)
254     b_location = get_b_location(image_lists, labelname, ind,
255                                bottleneck_dir, cat)
256     if not os.path.exists(b_location):
257         create_btlenck(b_location, image_lists, labelname, ind,
258                        image_dir, cat, sess, jpeg_data_tensor,
259                        bottleneck_tensor)
260     with open(b_location, 'r') as b_file:
261         b_string = b_file.read()
262     did_hit_error = False
263     try:
264         b_val = [float(x) for x in b_string.split(',')]
265     except ValueError:
266         print('invalid')
267         did_hit_error = True
268     if did_hit_error:
269         create_btlenck(b_location, image_lists, labelname, ind,
270                        image_dir, cat, sess, jpeg_data_tensor,
271                        bottleneck_tensor)
272     with open(b_location, 'r') as b_file:
273         b_string = b_file.read()
274     b_val = [float(x) for x in b_string.split(',')]
275     return b_val

```

Figure 3.5.7 Creating bottleneck files

It must also be noted that, *train.py* has provisions to include tensorboard charts. Figure 3.5.8 shows the code snippet that defines the training summaries data.

```

541 def summary(var):
542     with tf.name_scope('summaries'):
543         mean = tf.reduce_mean(var)
544         tf.summary.scalar('mean', mean)
545         with tf.name_scope('stddev'):
546             stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
547         tf.summary.scalar('stddev', stddev)
548         tf.summary.scalar('max', tf.reduce_max(var))
549         tf.summary.scalar('min', tf.reduce_min(var))

```

Figure 3.5.8 Scalar values for tensorboard

The last function before *main* begins is the *final\_ops()* function which adds some retrain operations to the top layer and defines variables to hold the weights and set up backward pass which returns the tensors .



```

553 def final_ops(class_count, final_tensor_name, bottleneck_tensor):
554     with tf.name_scope('input'):
555         bottleneck_input = tf.placeholder_with_default(
556             bottleneck_tensor, shape=[None, BOTTLENECK_TENSOR_SIZE],
557             name='BottleneckInputPlaceholder')
558
559         ground_truth_input = tf.placeholder(tf.float32,
560                                           [None, class_count],
561                                           name='GroundTruthInput')
562     layer_name = 'final_training_ops'

```

Figure 3.5.9 Final\_ops function

Now that all the functions needed are defined, let us focus on the major sections of the main function.

First, it sets up the directory paths by calling the functions that were discussed earlier. Next, it sets up the pre-trained graph and sets up the lists of all the images according to category.

```

630 image_lists = create_image_lists(FLAGS.image_dir, FLAGS.testing_percentage,
631                                FLAGS.validation_percentage)
632 class_count = len(image_lists.keys())
633 if class_count == 0:
634     print('No valid folders of images found at ' + FLAGS.image_dir)
635     return -1
636 if class_count == 1:

```

Figure 3.5.10 Setting up image lists

Next section adds new layers that need to be trained; the `add_evaluation_step()` function checks the accuracy and merges all the summaries.

```

661 (train_step, cross_entropy, bottleneck_input, ground_truth_input,
662  final_tensor) = final_ops(len(image_lists.keys()),
663                            FLAGS.final_tensor_name,
664                            bottleneck_tensor)
665
666 evaluation_step, prediction = add_evaluation_step(
667     final_tensor, ground_truth_input)
668
669 merged = tf.summary.merge_all()
670 train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train',
671                                     sess.graph)

```

Figure 3.5.11 Merge

Then there is a for loop which cycles for the number of training steps that were provided as an argument parameter at the beginning (5000 steps). Within the loop, the training step feeds the bottlenecks and labeled train image data to the graph.

```

699     train_summary, _ = sess.run(
700         [merged, train_step],
701         feed_dict={bottleneck_input: train_bottlenecks,
702                   ground_truth_input: train_ground_truth})
703     train_writer.add_summary(train_summary, i)
704

```

Figure 3.5.12 Writing to summary files

Finally, it writes out the trained graph and the labels with the weights.

```

750     output_graph_def = graph_util.convert_variables_to_constants(
751         sess, graph.as_graph_def(), [FLAGS.final_tensor_name])
752     with gfile.GFile(FLAGS.output_graph, 'wb') as f:
753         f.write(output_graph_def.SerializeToString())
754     with gfile.GFile(FLAGS.output_labels, 'w') as f:
755         f.write('\n'.join(image_lists.keys()) + '\n')

```

Figure 3.5.13 Writing to graph files

One more tweak within the loop is that it prints out the progress of the training as it is in progress to the terminal. A sample output is shown below (the complete version of the training output can be found in the Appendix):

```

Step: 0, Train accuracy: 20.0000%, Cross entropy: 3.259764, Validation accuracy: 7.0% (N=100)
Step: 50, Train accuracy: 82.0000%, Cross entropy: 2.387794, Validation accuracy: 83.0% (N=100)
Step: 100, Train accuracy: 94.0000%, Cross entropy: 1.796941, Validation accuracy: 94.0% (N=100)
Step: 150, Train accuracy: 98.0000%, Cross entropy: 1.409656, Validation accuracy: 89.0% (N=100)
Step: 200, Train accuracy: 95.0000%, Cross entropy: 1.158465, Validation accuracy: 95.0% (N=100)
Step: 250, Train accuracy: 93.0000%, Cross entropy: 0.911820, Validation accuracy: 98.0% (N=100)
Step: 300, Train accuracy: 98.0000%, Cross entropy: 0.721095, Validation accuracy: 97.0% (N=100)
Step: 350, Train accuracy: 96.0000%, Cross entropy: 0.693118, Validation accuracy: 96.0% (N=100)
Step: 400, Train accuracy: 96.0000%, Cross entropy: 0.591889, Validation accuracy: 97.0% (N=100)
Step: 450, Train accuracy: 100.0000%, Cross entropy: 0.479280, Validation accuracy: 98.0% (N=100)
Step: 500, Train accuracy: 98.0000%, Cross entropy: 0.478022, Validation accuracy: 94.0% (N=100)
Step: 550, Train accuracy: 99.0000%, Cross entropy: 0.374854, Validation accuracy: 95.0% (N=100)
Step: 600, Train accuracy: 100.0000%, Cross entropy: 0.373101, Validation accuracy: 98.0% (N=100)
Step: 650, Train accuracy: 99.0000%, Cross entropy: 0.308552, Validation accuracy: 100.0% (N=100)
Step: 700, Train accuracy: 97.0000%, Cross entropy: 0.441026, Validation accuracy: 95.0% (N=100)
Step: 750, Train accuracy: 96.0000%, Cross entropy: 0.370755, Validation accuracy: 99.0% (N=100)
Step: 800, Train accuracy: 100.0000%, Cross entropy: 0.309538, Validation accuracy: 98.0% (N=100)
Step: 850, Train accuracy: 98.0000%, Cross entropy: 0.267817, Validation accuracy: 96.0% (N=100)

```

Figure 3.5.14 Training progress printing to terminal

## 3.6 Tensorboard

As mentioned in the design section of this chapter, the training programme accepts several arguments, one of which is the learning rate.

```
766 with tf.name_scope('train'):  
767     optimizer = tf.train.GradientDescentOptimizer(FLAGS.learning_rate)  
768     train_step = optimizer.minimize(cross_entropy_mean)
```

Figure 3.6.1 Learning rate

Several learning rates were tried on a sample of training data before settling on a final decision of 0.001.

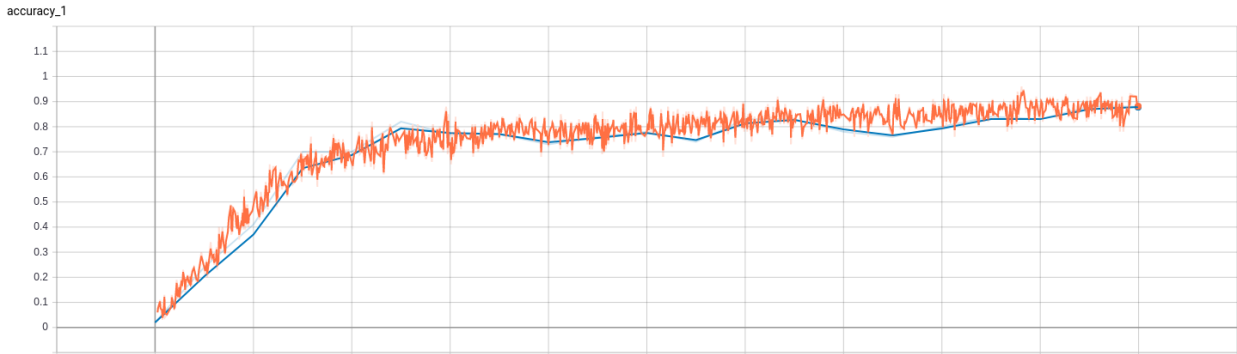


Figure 3.6.2 Training(orange) and validation(blue) accuracy for  $lr=0.0001$

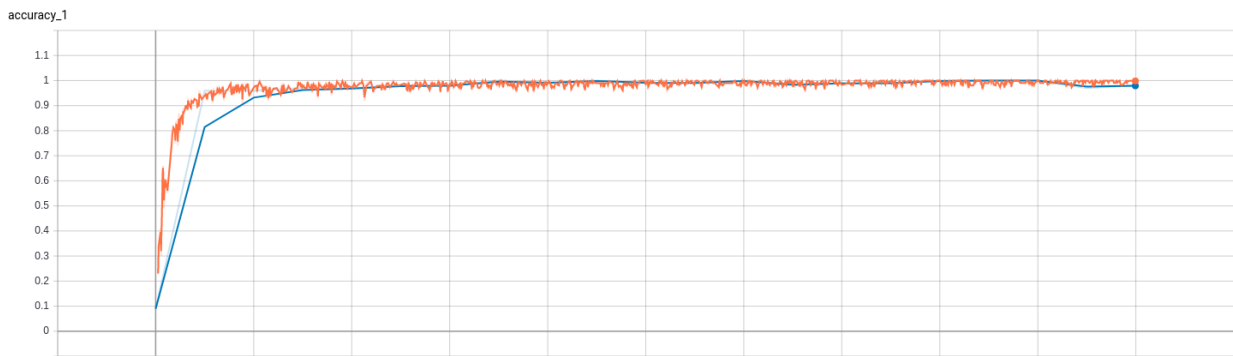


Figure 3.6.3 Training(orange) and validation(blue) accuracy for  $lr=0.01$

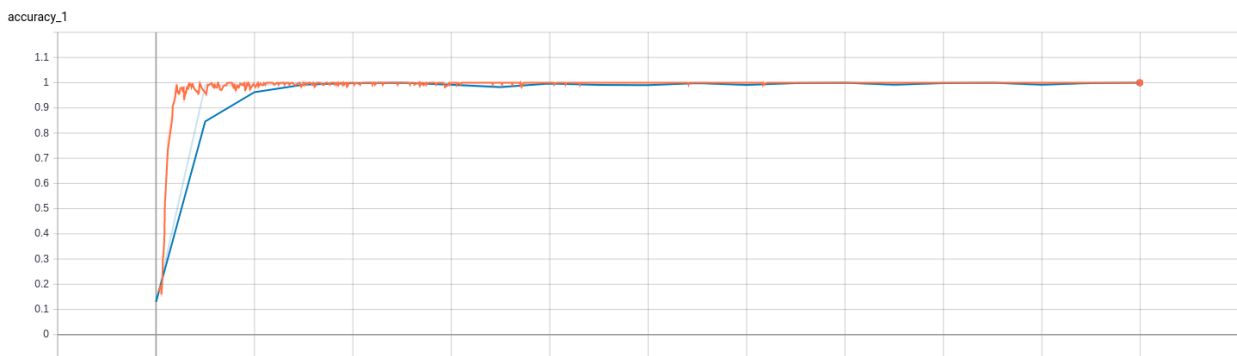


Figure 3.6.4 Training(orange) and validation(blue) accuracy for  $lr=2$

Note how smaller learning rates lead to a lower accuracy. It is important to have a high learning rate but not so high enough that overfitting occurs.

The figure below shows the final training of the dataset running for 5000 epochs. Training accuracy is shown in orange, and the validation accuracy (as the dataset is being trained – with a sample size of 100) was recorded at 50 step intervals, is shown in blue. In a perfect scenario, both curves should line up perfectly on top of each other. Based on figure 3.6.5, training accuracy is slightly below validation accuracy, especially at the beginning, it shows that there is a slight tendency to overfit the labels but given the fact that other learning rates produced lower final accuracy or entropy, it was decided that this iteration of the training settings would best fit the project.

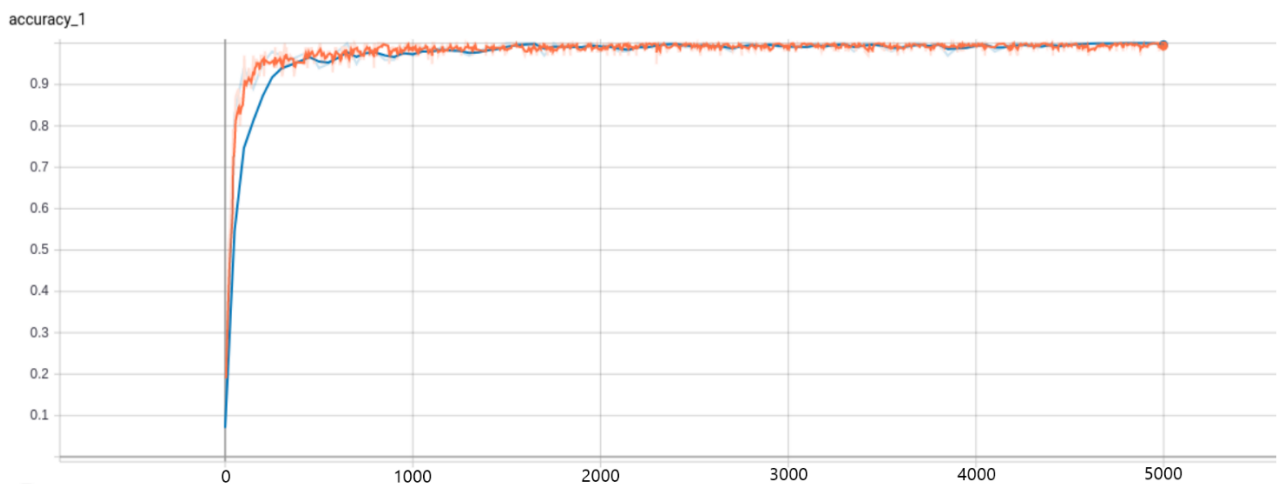


Figure 3.6.5 Accuracy of the final dataset

Figure 3.6.6 shows the entropy as the training progressed. The shape of the curve is as expected, finally settling on an entropy of 0.054638.

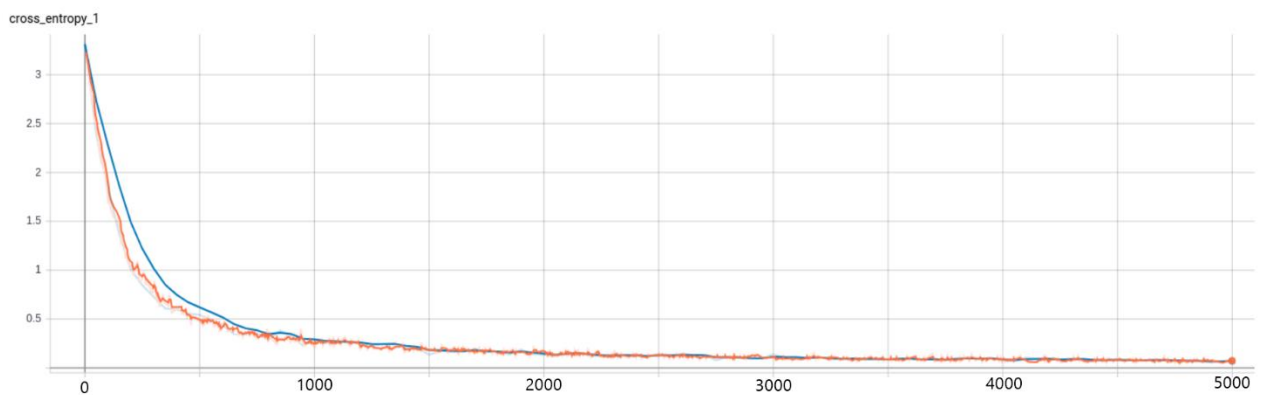


Figure 3.6.6 Cross entropy

The changes in weights and biases were also recorded during the training of the dataset. Note how the changes can still continue – but if continued, this would negatively impact the network

and cause overfitting. Each square on the figures below represents 500 steps. Note how the module converges quickly to a high accuracy; this is possible due to transfer learning. Transfer learning starts with pre-trained weights. Then, the weights are adjusted as the network trains for the new data.

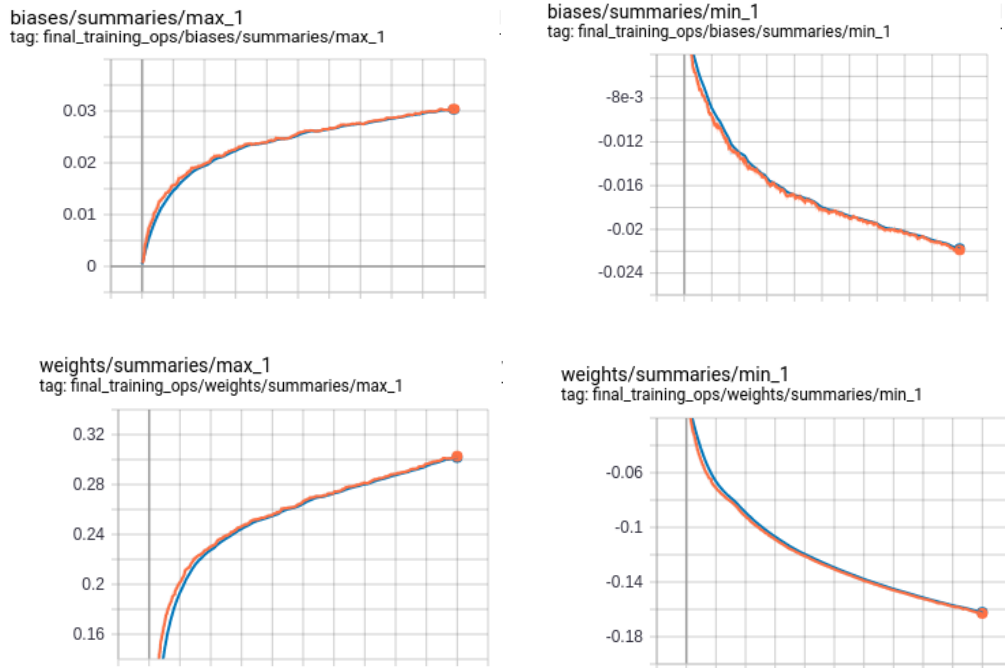


Figure 3.6.7 Changing biases and weights

Figure below shows the spread or the distribution of biases as the training progressed.

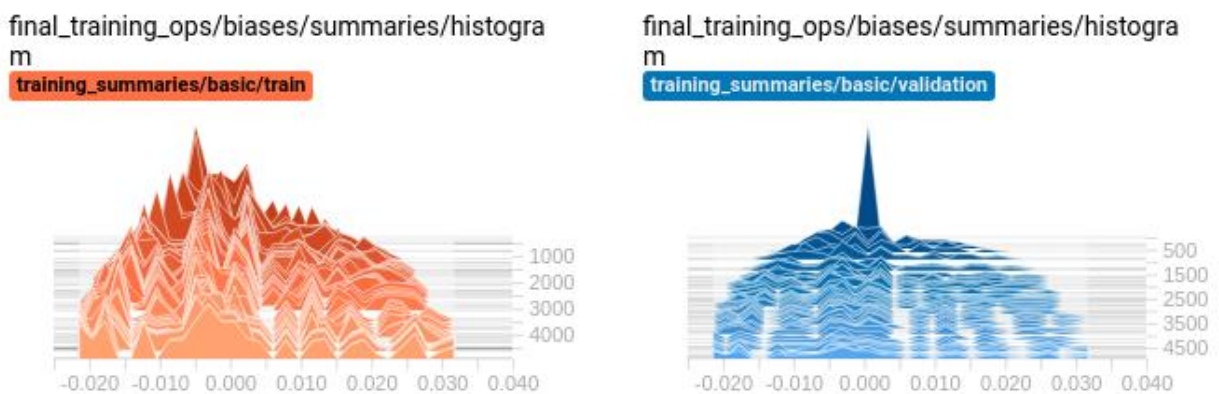


Figure 3.6.8 Distribution of biases

### 3.7 Main Processing Unit

As stated in an earlier chapter, the user needs to pose the gesture for a fraction of a second before it gets registered in the system. Given that all the processing takes place in an infinite while loop (once the application starts) where all the image processing and manipulations take place, the programme captures about 15 video frames every second on average.

The 15 frames per second average depends on the processing power of the computer the application is being run on. Out of the frames captured, the programme was designed so that every  $i^{\text{th}}$  frame is sent over to the neural network for processing.

```
215 loopspeed = SPEED.get()
216
217 if i == loopspeed:
218     res_tmp, score, predictions = predict(image_data)
219     res = res_tmp
220     i = 0
221     if mem == res:
222         consecutive += 1
223     else:
```

Figure 3.7.1 Loopspeed

In the code snippet above, *predict()* function is fired only when *i* is equal to *loopspeed*. The programme goes through *i* iterations until image data is sent to the neural network for a prediction. The *i* value can be changed through the GUI interface, but it was found experimentally that the optimum value for *i* is seven.



Figure 3.7.2 GUI has a trackbar to control the *i* value

Decreasing *i* causes the programme to run faster but loses accuracy and increasing *i* makes the user hold the gesture much longer and it starts to lose its real-time potential. *SPEED.get()* function shown in figure 3.7.1 is handled through *Tkinter* a Python library for Graphical User Interfaces.

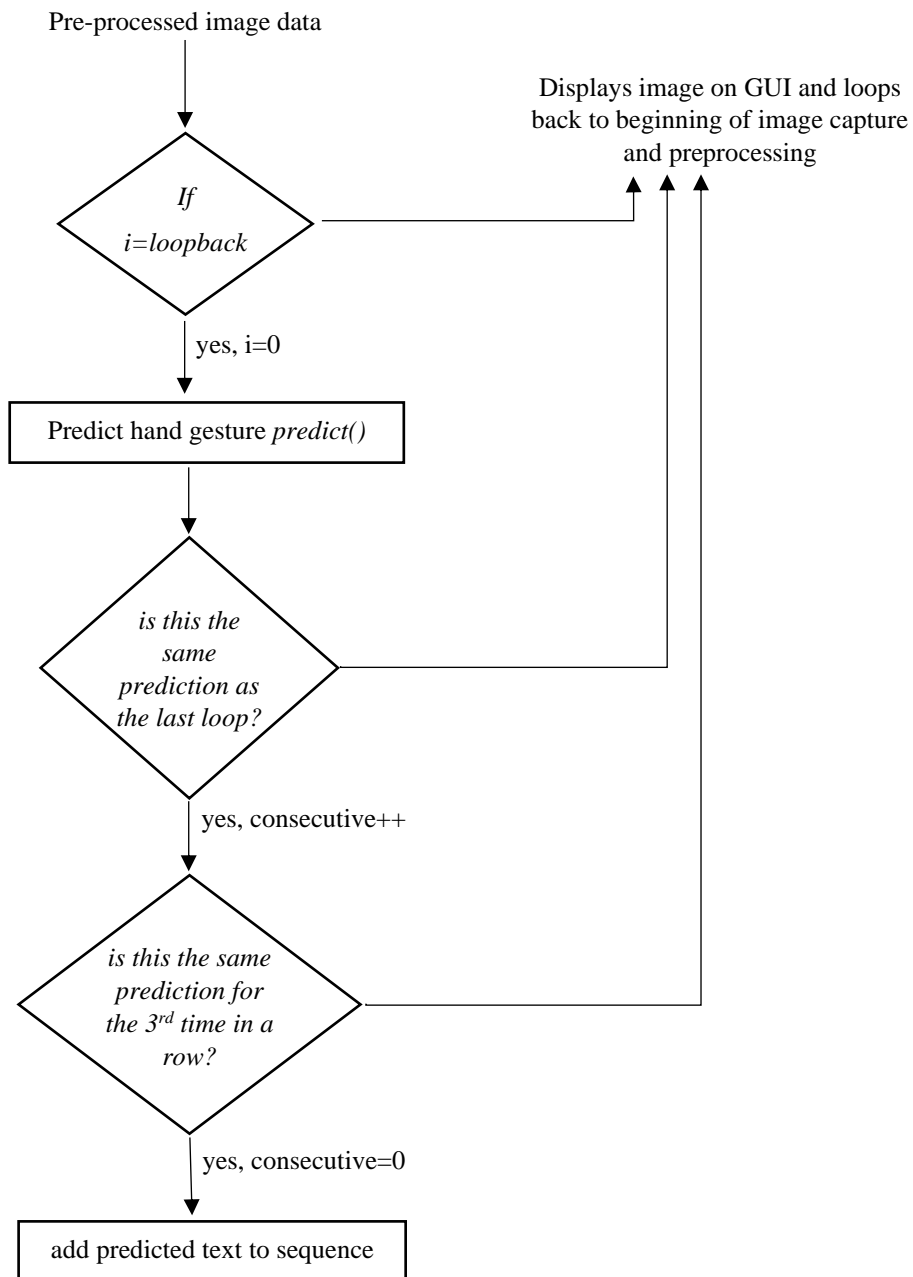


Figure 3.7.3 Conditions within the main processing unit

Full if-else statement for the above code snippet is shown below. Image data is passed on to the *predict()* function and it returns a variable, *res\_temp* and *score*.

*res\_temp* holds the predicted character and *score* is the probability or confidence level with which the Inception neural network can identify *res\_temp*. *mem* is predicted output in the previous iteration and note that the variable *consecutive* needs to be equal to 3 for the *res* variable (predicated character) to be added to the sequence. Therefore, the system must identify that the user is holding the same hand gesture for three consecutive iterations within the main

processing loop before it gets registered in the sequence and displayed to the screen. This extra layer of protection has been added to minimize errors and slow down. Therefore, overall, there are two separate controllers that regulate the speed at which gestures are identified.

I - The first controller checks every  $i^{\text{th}}$  image frame into the prediction function – this can be controlled by the user through a trackbar on the GUI. It controls the overall speed of identifying the gestures.

II - The second controller is fixed. It adds a checkpoint so that a letter is added to the text on the screen only if the prediction algorithm has identified the user is holding the same gesture for three consecutive frames. These three frames are image frames that the earlier  $i^{\text{th}}$  frame check has allowed in.

Hence, if the loopback value is 7, and the programme captures 15 frames per second, a user will have to hold the same gesture for a minimum of 1 second before the gesture is displayed on screen. *sequence* variable stores the string of characters that the user signs in a session.

```
217 |         if i == loopspeed:
218 |             res_tmp, score, predictions = predict(image_data)
219 |             res = res_tmp
220 |             i = 0
221 |             if mem == res:
222 |                 consecutive += 1
223 |             else:
224 |                 consecutive = 0
225 |             if consecutive == 3 and res not in ['nothing']:
226 |                 if res == 'space':
227 |                     sequence += ' '
228 |                 elif res == 'delete':
229 |                     sequence = sequence[:-1]
230 |                 else:
231 |                     if res=="az":
232 |                         res="A"
233 |                     sequence += res
```

Figure 3.7.4 Sequence being constructed

Also note that there are provisions for space and delete within the system so that if a user makes a mistake, they can remove the last character in the sequence by holding the hand gesture for delete. Similarly, there exists a space hand gesture with which a user is able to add spaces between the characters when forming sentences.

*predict()*, called in the 2<sup>nd</sup> line of the code snippet above, is a simple straightforward python function that inputs image data and returns a predicted gesture and the confidence level.



```

44 def predict(image_data):
45     predictions=sess.run(softmax_tensor,{'DecodeJpeg/contents:0': image_data})
46     top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]
47
48     max_score = 0.0
49     res = ''
50     for node_id in top_k:
51         human_string = label_lines[node_id]
52         score = predictions[0][node_id]
53         if score > max_score:
54             max_score = score
55             res = human_string
56     return res, max_score

```

Figure 3.7.5 Predict function

*sess.run* within the *predict* function is defined as follows:

```

72 with tf.Session() as sess:
73     softmax_tensor = sess.graph.get_tensor_by_name('final_result:0')

```

Figure 3.7.6 Sess.run

*tf.session* inputs the image data to the graph and gets the prediction. The function returns a list of predictions. *predictions* is then sorted to get the best predicted gesture label. The returned predicted gesture label is added to the *sequence* and the confidence level – *max\_score* – is sent to the GUI for display. Figure 3.7.7 shows how the *res* (predicted label) and *max\_score* (confidence level) are shown on the display. In this example, it shows that the user is holding the ක් (k, see table 3.3.1) gesture with a probability of 0.84.

A detailed explanation of the GUI and display is given in the Graphical User Interface section of this chapter.

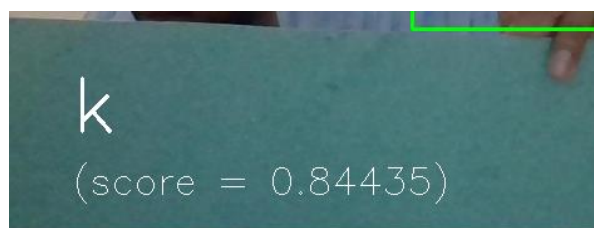


Figure 3.7.7 Res and max\_score

## 3.8 Transliteration

Transliteration is the process of converting letters of one script to another by swapping letters with similar phonetic sounds. During the design process, it was decided that the Sinhala hand gestures would be mapped or labeled to the letters of the English alphabet. For example, the

Sinhala ‘අ’ is trained and labeled as ‘a’, while ‘ආ’ would be labeled in the neural network as ‘aa’ and ‘ඛ’ would be ‘b’ and so on.

As shown in the figure below, the main processing unit would identify the image through a neural network and create a prediction. These predictions are collected in a variable called the *sequence*. At each iteration, the sequence is passed to a transliteration submodule to convert it to Sinhala.

```
sinhala = transliterate.convert(sequence)
```

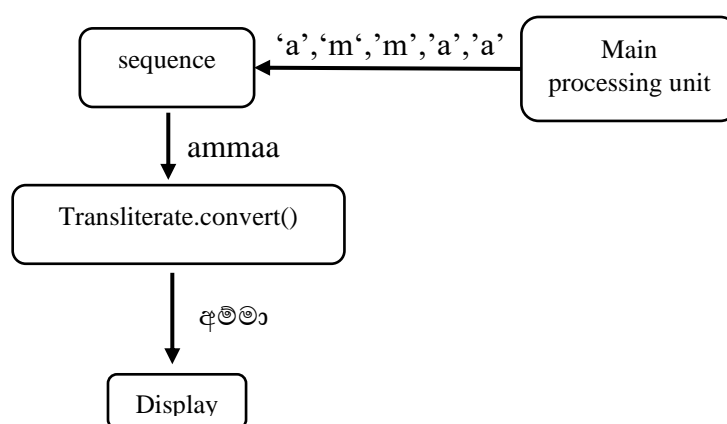


Figure 3.8.1 Sequence of characters become a word

Table 3.8.1 illustrates how some of the labels are mapped to Sinhala letters.

Hand gesture	label	letter
	a	අ
	aa	ආ
	az	ඈ
	e	ඉ
	m	ඛ

Table 3.8.1 Some gestures, their labels and corresponding text

Sinhala fingerspelling alphabet has been designed so that, to sign a particular letter, one might have to combine the letter with certain other letter hand signs to get the proper end result. For example, to sign අමමා the hand gestures shown in figure 3.8.2 should be made. Note how ම and ආ gestures are combined to produce the මා sound.



Figure 3.8.2 spelling amma with the sinhala sign language

Hence, as shown in figure 3.8.2, once the user poses the hand gestures, the system captures them – predicts the labels, which would be ‘ammaa’ given that අ → a, ම → m and ආ → aa. Once the sequence ammaa is predicted by the network according to the hand gestures, the transliteration component kicks in and converts it to the appropriate word අමමා displayed on the output.

To illustrate another example, let’s say the user wants to spell out the sentence මම ගෙදර යනවා. Figure 3.8.3 shows one would achieve this.

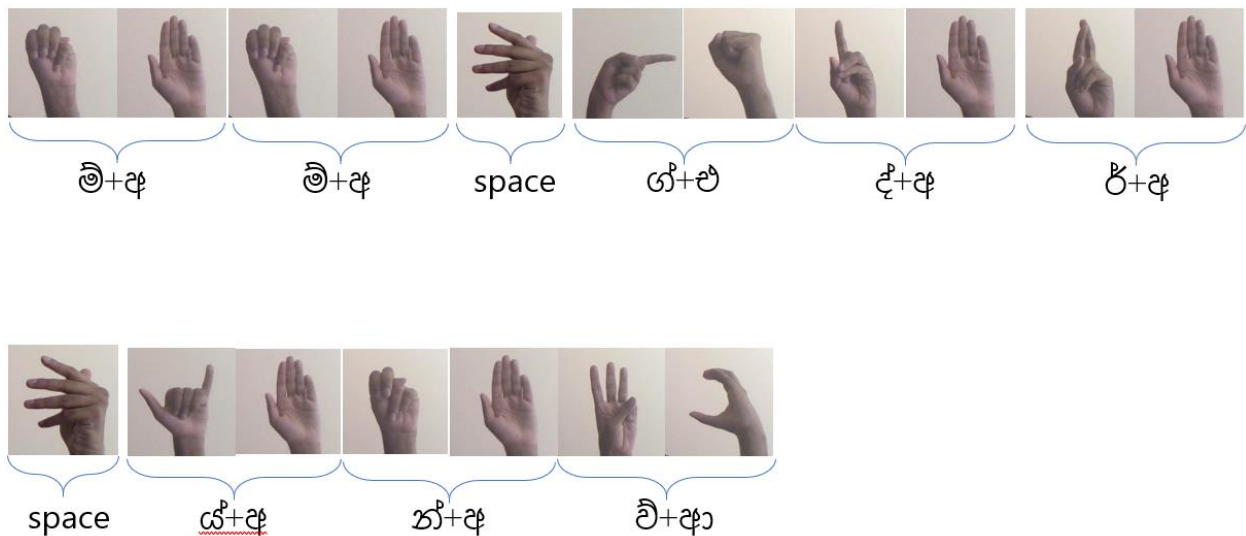


Figure 3.8.3 A sentence in sign language

The programme would read the first gesture - ම, identify the image through the neural network and assign a label *m*. The next gesture is අ. The label would now be *ma*. Similarly, the programme would identify the next two images. Now the label would read *mama*. At each iteration, the labels are sent through a transliteration module. Table 3.8.2 shows how the transliteration works at each iteration.












gesture	Label sequence	Text seen on screen
	m	මි
	ma	මා
	mam	මම
	mama	මම
	Mama[space]	මම
	mama[space]g	මම ගී
	mama[space]ge	මම ගෙ
	mama[space]ged	මම ගෙදී
	mama[space]geda	මම ගෙද
	mama[space]gedar	මම ගෙදර්
	mama[space]gedara	මම ගෙදර

Table 3.8.2 Gesture sequence

Note how when the identified label sequence is *mama[space]g* and the user pose the next gesture as ඵ, the transliteration changes it to ගෙ, instead of ගීඵ. This is achieved through the transliteration submodule.

### How the Module Works

Transliteration module consists of predefined lists. Each list holds characters in English and Sinhala in a particular order. For instance, take the list *vowels*.

```
vowels= ["a", "i", "e", "u"...
```

It has a corresponding list called *vowelsUni* which contains the Sinhalese equivalent of the phonetic English sound. For example, aa is mapped to අආ.

```
vowelsUni= ["අ", "ඉ", "එ", "උ"...
```

When an 'a' appears in the sequence, it is replaced to අආ by a for-loop that cycles through the vowel list.

```

49     for i in range(len_vowels):
50         text = text.replace(vowels[i], vowelsUni[i])

```

Figure 3.8.4 Cycles through list of vowels

Similarly, there several other lists which contain various Sinhala diatrics, special cases and constants. Let's examine a slightly complicated replacement:

```

44     for i in range(len_consonants):
45         text = text.replace(consonants[i], consonantsUni[i]+'ආ')

```

Figure 3.8.5 List of consonants + hal kireema

In the snippet above, the *hal kireema* is added to constants. As mentioned earlier, similar to *vowels* list, there is a consonant list as well.

```

consonants= ["p", "b", "m", "y...
consonantsUni= ["ආ", "ඞ", "ඞ", "ඞ...

```

Once the sequence goes through all the for-loops, it should be converted to Sinhala text. The text is then returned as variable *Sinhala*.

```

sinhala = transliterate.convert(sequence)

```

## Graphical User Interface

Figure 3.9.1 shows the Graphical User Interface for the developed application. On the left it shows the image captured by the web camera. There is a green square overlay which guides the user to place the hand within the area for capture.

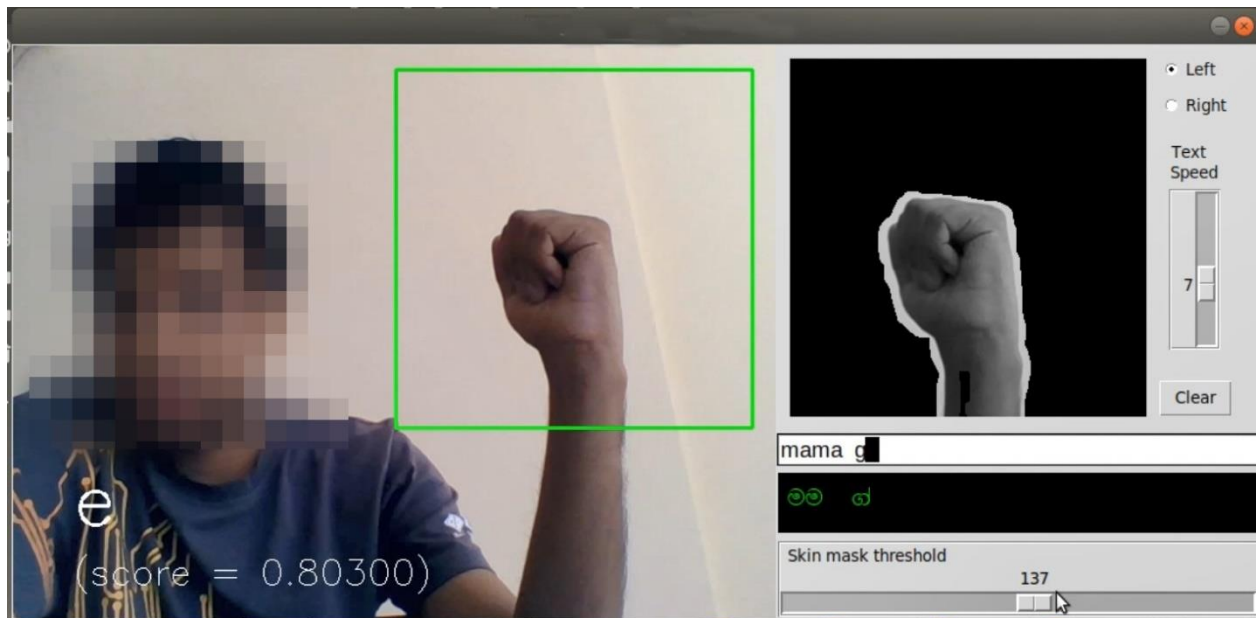


Figure 0.1 The GUI

On the left bottom section of the image you can see an 'e' and a 'score'. 'e' represents the label of the hand gesture that was just identified by the neural network. Note that, as mentioned in the Main Processing Unit section of this chapter, the neural network needs to predict the same gesture three times in a row for it to appear in the final text. The score is self-explanatory, it shows the confidence level of the predicted gesture out of one. The label and scores are available for information purposes. On the right, there is another frame with a mask applied to it. It shows the image that is fed into the neural network for predictions.

The user can change the threshold on the mask by moving skin mask threshold slider at the bottom. Note the differences between figure 3.9.2 – the slider was changed from 138 to 145. In the center image (in figure 3.9.2), the system cannot identify the proper hand gesture because a proper image is not fed to the neural network. It is the user's responsibility to make sure that the skin threshold masks the background and shows the hand gesture properly.

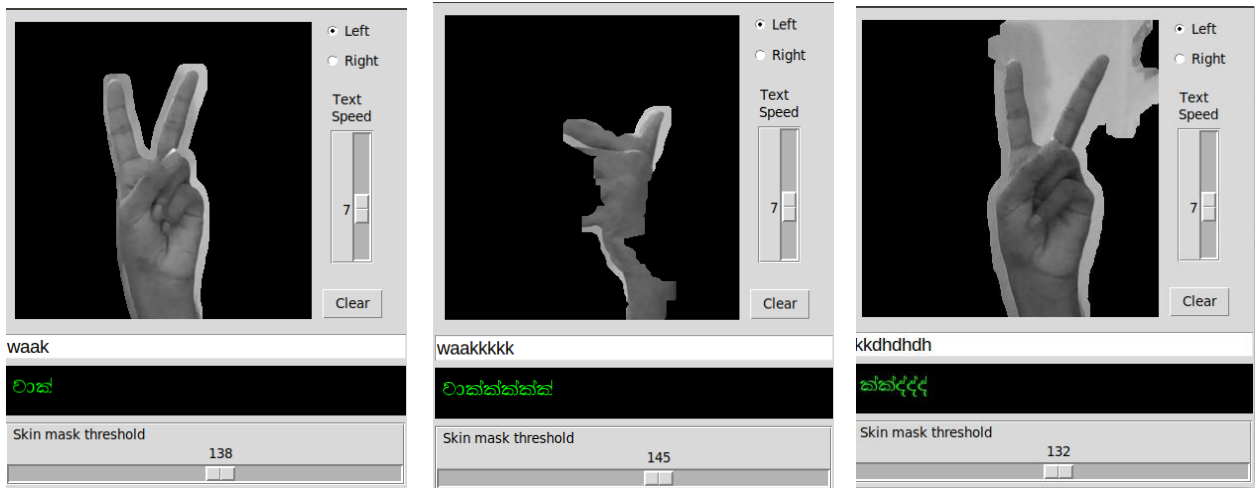


Figure 0.2 Mask under different thresholds

The radio buttons left and right indicate that the application can be used by both left and right-handers. Selecting the preferred hand flips the image from the webcam so that users can use either the left hand or the right to make hand gestures. The vertical slider shown in figure 3.9.3 controls the speed at which letters are added to the final text screen.



Figure 0.3 Text Speed trackbar

Increasing the value of the slider makes the system recognize gestures faster but can lead to errors if the system snaps a picture while the gesture is being changed. The text areas below the masked image frame show sequence of labels returned from the neural network before and after transliteration.



Figure 0.4 Closer view of text output

In figure 3.9.4, you can see that the user is halfway through trying to gesture the sentence මම දර මනවා. The user has already gestured *mama ge* and is now gesturing ද.

As you can see, the white text area shows the predicted labels of the text before transliteration. Although the white text area is not strictly required to be displayed on the end user's display, it can act as a guiding step when fingerspelling Sinhala gestures. The darker window with green text shows the final version of the text in Sinhala. While there is a hand gesture that imitates the backspace key (deletes the last signed letter from the text sequence) the user can also press the *Clear* button to reset the text boxes and erase all letters on screen to start over.



## Chapter 4 Results and Evaluation

### 4.1 Introduction

This chapter explains the evaluation stage of the developed system. It goes through the evaluation criteria, justifies the choice of criteria, selection of test subjects, presents the results and finally performs a result analysis. Whenever a project or research is conducted, it is important to test the final solution against the proposed objectives because the science community does not accept solutions/theories that have not been tested. Therefore, it is important to come up with a proper evaluation strategy in such a way that it doesn't skew or influence the test results in favour of the proposition.

### 4.2 Evaluation Criteria

The objective of this project was to recognize the gestures of the Sinhala fingerspelling alphabet and in extension to identify and form text/words on screen. In order to achieve this task, the system was trained to identify 28 labels (27gestures + nothing label when there is no gesture) and then transliterate those labels to Sinhala text. Therefore, two tests are needed; one to test the accuracy of labels identified and the other to construct words/sentences.

Next important feature would be the test subjects. People have different hand sizes; hence 6 test subjects were selected so that there was an even spread on age and gender. These 6 subjects were not used to obtain images for the training dataset. The initial plan was to have a higher sample size so that more test data can be collected but due to distancing and travel restrictions imposed during the testing and evaluation phase of the project, the sample size was limited to 6 individuals. Finally, light also plays a very important role when identifying hand gestures. The dataset includes hand gestures that are well lit. It would be interesting to see how well the system fares when the ambient light in the captured images change.

To summarize, there are two main tests: test letters and sentences/words. These two tests in turn need to be performed under various light conditions by an independent test group.

Single label/ letter	Words/sentences
ම, ද, න, ආ, etc.	ගෙදර, සියලු මිනිසුන් නිදහස්ව උපත ලබා ඇත, etc.

### 4.3 Alphabet Character Evaluation

The test environment was set up so that the user was holding the hand at about 0.75 m from the web camera. The background behind the posed hand gesture was a plain wall with no features. The following table shows the results for 6 individuals. Each test subject held the gesture for 20 attempts. A gesture was considered as a success if the intended gesture's corresponding letter appeared on the screen and a failure if any other letter was interpreted.

Therefore, each person attempted the same gesture 10 times giving a total of 60 gestures per label/letter. It took about 2 minutes to cycle through the 27 gestures once. So, it took around 25 - 30 minutes for a person to cycle through the hand gesture alphabet 10 times.

***Test scenario A – a well-lit room with a fluorescent bulb. White hue. Minimal shadows on the hand***

***Test scenario B – room with ambient daylight coming in through a window, since the light source forms a side of the room, the hand gestures were illuminated from a side and formed shadows on the skin.***

The table below shows the test results for the 6 test subjects for each gesture in an artificially lit environment. The room was illuminated with fluorescent light with a white hue (similar to where the training image dataset was collected).

Gesture	Number of attempts	Number of successful attempts	Number of failed attempts	Success rate in %
අ	60	60	0	100
ආ	60	59	0	100
ඇ	60	60	0	100
ඈ	60	58	2	96.7
ඉ	60	60	0	100
ඊ	60	59	1	98.3
උ	60	60	0	100
ඌ	60	59	1	98.3
ඍ	60	56	4	93.3
ඎ	60	59	1	98.3
ඏ	60	57	3	95
ඐ	60	45	15	75
එ	60	59	1	98.3
ඒ	60	49	11	81.7
ඓ	60	60	0	100
ඔ	60	59	1	98.3
ඕ	60	47	13	78.3
ඖ	60	58	2	96.7
඗	60	58	2	96.7
඘	60	60	0	100
඙	60	60	0	100
ඛ	60	58	2	96.7
ඛ	60	60	0	100
ං	60	60	0	100
ච	60	60	0	100
[space]	60	58	2	96.7
[delete]	60	60	0	100
Total	1620	1558	61	96.2333

Table 4.3.1 Scenario A

It is also worth mentioning that the 20 gestures were divided as 10 gesture with the left hand and 10 with the right.

The same procedure was repeated in an environment with ambient daylight (not direct sunlight). The results are given below:

Gesture	Number of attempts	Number of successful attempts	Number of failed attempts	Success rate in %
අ	60	60	0	100
ආ	60	60	0	100
ඇ	60	59	1	98.3
ඉ	60	57	3	95
උ	60	59	1	98.3
එ	60	57	3	95
ඞ	60	60	0	100
ඟ	60	57	3	95
ච	60	54	6	90
ඊ	60	59	1	98.3
උ	60	56	4	93.3
ඌ	60	43	17	71.7
ඍ	60	58	2	96.7
ඎ	60	39	21	65
ඏ	60	59	1	98.3
ඐ	60	58	2	96.7
එ	60	45	15	75
ඒ	60	56	4	93.3
උ	60	55	5	91.7
ඌ	60	60	0	100
ඍ	60	59	1	98.3
ඎ	60	55	5	91.7
ඏ	60	58	2	96.7
ඐ	60	60	0	100
එ	60	60	0	100
[space]	60	58	2	96.7
[delete]	60	58	2	96.7
Total	1680	1579	101	93.76667

Table 4.3.2 Scenario B

Before we analyze the results, it must be emphasized that the accuracy of the system was measured on the final output from the system.

In the methodology chapter of this report, it was stated that for a character to appear on the output screen, it must have been predicted by the neural network for three consecutive cycles. Hence, there may have been instances (even in situations where the accuracy is 100%) the

system predicts  $\varphi$  in cycle one,  $\vartheta$  in cycle two, and then records  $\varphi$  in the next three cycles, it will still be output as  $\varphi$ , hence will be a successful attempt.

It is worth noting that most of the characters reach a 100% accuracy rate. The accuracy rate of the gesture attempts in a well-lit room where the hand received balanced illumination from all directions is 96.4% and the accuracy rate for gestures in natural daylight is 93.6%.

For comparison, the below image shows a hand gesture in the two testing situations:



*Figure 4.3.1 Hand gesture used for Test A*



*Figure 4.3.2 Hand gesture used for Test B*

Figure 4.3.2, the hand is illuminated from one direction only and causes shadows on the skin.

Table below shows the summary of the test results combined:

Gesture	Number of attempts	A	B	total - success	A	B	total - fail	Success rate in %
අ	120	60	60	120	0	0	0	1
ආ	120	60	60	120	0	0	0	1
ඇ	120	60	59	119	0	1	1	0.99167
ඉ	120	58	57	115	2	3	5	0.95833
උ	120	60	59	119	0	1	1	0.99167
එ	120	59	57	116	1	3	4	0.96667
ක	120	60	60	120	0	0	0	1
ග	120	59	57	116	1	3	4	0.96667
ජ	120	56	54	110	4	6	10	0.91667
ච	120	59	59	118	1	1	2	0.98333
ඳ	120	57	56	113	3	4	7	0.94167
න	120	45	43	88	15	17	32	0.73333
ඩ	120	59	58	117	1	2	3	0.975
න්	120	49	39	88	11	21	32	0.73333
ප	120	60	59	119	0	1	1	0.99167
බ	120	59	58	117	1	2	3	0.975
ම	120	47	45	92	13	15	28	0.76667
ය	120	58	56	114	2	4	6	0.95
ර	120	58	55	113	2	5	7	0.94167
ල	120	60	60	120	0	0	0	1
ච්	120	60	59	119	0	1	1	0.99167
ජ්	120	58	55	113	2	5	7	0.94167
ඞ	120	60	58	118	0	2	2	0.98333
ං	120	60	60	120	0	0	0	1
ච්	120	60	60	120	0	0	0	1
[space]	120	58	58	116	2	2	4	0.96667
[delete]	120	60	58	118	0	2	2	0.98333
<b>Total</b>	<b>3240</b>	<b>1559</b>	<b>1519</b>	<b>3078</b>	<b>61</b>	<b>101</b>	<b>162</b>	<b><u>0.95</u></b>

Table 4.3.3 Summary of results

The gestures for 𐀀, 𐀁 and 𐀂 have a considerably low success rate than the rest, removing these three gestures from the test results yields:

	Accuracy of the complete dataset		Accuracy when 𐀀, 𐀁 and 𐀂 are removed from the results	
	Well-lit environment	Ambient daylight	Well-lit environment	Ambient daylight
Average	96.23457	93.76543	98.47222	96.66667
Median	98.33333	96.66667	99.16667	96.66667
Std. Deviation	6.755697	8.934622	1.898173	2.989515

*Table 4.3.4 Removing low achieving results*

The overall accuracy of the system was 95%.

Before moving on to the next section of the analysis. It must also be mentioned that 10% of the dataset was also used to test the accuracy of training as soon as the training ended. Based on the training dataset, the system has an accuracy of 99.6%.

#### 4.4 Confidence level Analysis

As shown in figure 4.4.1, the system calculates a probability, or the confidence of the prediction made. During testing, the confidence level of the output was also stored in a separate file.

```

235 if num<28:
236     print("test: "+str(num))
237     top_k = predictions[0].argsort()[-len(predictions[0]):][::-1]
238
239     for node_id in top_k:
240         wewere = label_lines[node_id]
241         sc = predictions[0][node_id]
242         print('%s %.5f' % (wewere, sc))
243         print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")
244         num=num+1

```

*Figure 4.4.1 Changing code to print confidence level*

The application code was modified as shown above so it would print out the confidence level of the labels for each hand gesture frame captured. After testing was complete, the confidence values were collected in a spreadsheet for further evaluation and analysis. Full spreadsheet consists of 27 sheets with each sheet containing 3240 data points giving a total of 90720 values for the entire testing process. The list is quite extensive and important sections of the data collected are included in the appendix section of this report.

Figure 4.4.2 shows a part of the spreadsheet for the data collected for the label a or gesture  $\varphi$ .

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
		a	aa	az	b	oh	d	delete	dh	e	g	h	i	j	k	l	
2	1	0.23843	0.00207	0.0338	0.06833	0.0027	0.061	0.00313	0.0046	0.01977	0.00034	0.00258	0.00176	0.23315	0.00325	0.005	
3	2	0.51983	0.00228	0.00593	0.10239	0.00109	0.0134	0.00146	0.00148	0.015	0.00078	0.00609	0.00031	0.03511	0.0072	0.00342	
4	3	0.37989	0.0045	0.00922	0.09125	0.00192	0.00664	0.01002	0.00099	0.03016	0.00081	0.02116	0.00457	0.04101	0.00263	0.00844	
5	4	0.81128	0.00019	0.00456	0.04747	0.00034	0.01423	0.00104	0.00032	0.00095	0.00026	0.00186	0.00032	0.01092	0.00052	0.00129	
6	5	0.60824	0.00126	0.01046	0.04996	0.00272	0.0039	0.00465	0.00106	0.00765	0.00036	0.01041	0.00137	0.02484	0.00237	0.00753	
7	6	0.88667	0.00008	0.00138	0.03379	0.00009	0.00399	0.00042	0.00075	0.0023	0.00022	0.0001	0.00027	0.00382	0.00032	0.00027	
8	7	0.90674	0.00005	0.00057	0.02795	0.00007	0.0012	0.00219	0.00009	0.00064	0.00002	0.00049	0.00005	0.0027	0.00009	0.00042	
9	8	0.798	0.00065	0.00401	0.01487	0.00183	0.00306	0.00302	0.0002	0.00157	0.00021	0.00161	0.00109	0.00662	0.00047	0.00642	
10	9	0.63536	0.00098	0.00822	0.03294	0.00238	0.00841	0.00382	0.00053	0.00815	0.00044	0.00183	0.00225	0.01979	0.00081	0.00313	
11	10	0.76733	0.00044	0.00311	0.04427	0.00122	0.00587	0.00801	0.00033	0.00102	0.00009	0.00075	0.00065	0.00583	0.00019	0.00196	
12	1	0.81993	0.00019	0.00191	0.01946	0.00025	0.0029	0.00621	0.00016	0.00243	0.00005	0.00037	0.00029	0.00811	0.00005	0.00063	
13	2	0.73582	0.0001	0.00168	0.00872	0.00024	0.00338	0.01641	0.00008	0.00132	0.00004	0.00023	0.00013	0.00325	0.00003	0.00042	
14	3	0.9058	0.00004	0.00048	0.02936	0.00005	0.00079	0.00149	0.00005	0.00121	0.00001	0.00004	0.00008	0.00347	0.00002	0.00016	
15	4	0.92602	0.00009	0.0006	0.02616	0.0001	0.00099	0.00254	0.0001	0.00087	0.00001	0.0001	0.00008	0.00651	0.00005	0.00096	
16	5	0.97059	0.00002	0.00057	0.01371	0.00002	0.00163	0.00049	0.00006	0.00035	0	0.00001	0.00002	0.00117	0.00001	0.0001	
17	6	0.8956	0.00003	0.0004	0.00971	0.00011	0.00116	0.00379	0.00001	0.00044	0.00001	0.00009	0.00004	0.00182	0.00001	0.00019	
18	7	0.88725	0.00004	0.0004	0.03345	0.00008	0.00082	0.00205	0.00015	0.00146	0.00001	0.00002	0.00007	0.00589	0.00002	0.00022	
19	8	0.91182	0.00005	0.00033	0.01725	0.00003	0.00059	0.00199	0.00003	0.00069	0.00002	0.00009	0.00007	0.00441	0.00004	0.00027	
20	9	0.90407	0.00007	0.00142	0.01629	0.0001	0.00905	0.00122	0.00021	0.00166	0.00002	0.00006	0.0001	0.02226	0.00004	0.00168	
21	10	0.87391	0.0001	0.00149	0.03668	0.00009	0.00405	0.00208	0.0001	0.00097	0.00002	0.00003	0.00015	0.00721	0.00004	0.0006	
22	1	0.73403	0.00005	0.00326	0.02186	0.0002	0.00316	0.00029	0.0001	0.00015	0.00008	0.00031	0.00007	0.00019	0.00008	0.00009	
23	2	0.94421	0.00001	0.0002	0.03493	0.00002	0.00017	0.00009	0.00004	0.00004	0.00001	0.00004	0.00001	0.00004	0.00003	0.00001	
24	3	0.89021	0.00004	0.00323	0.08014	0.00009	0.00088	0.00007	0.00007	0.00025	0.00001	0.00008	0.00005	0.0002	0.00011	0.00011	
25	4	0.9559	0.00001	0.00061	0.0184	0.00007	0.00018	0.00008	0.00005	0.00022	0.00003	0.00003	0.00003	0.00009	0.00006	0.00002	
26	5	0.91173	0.00006	0.00077	0.07194	0.00004	0.00032	0.00009	0.00022	0.00039	0.00002	0.00007	0.00003	0.00005	0.00012	0.00005	
27	6	0.96359	0.00001	0.00022	0.01761	0.00007	0.0001	0.00024	0.00004	0.00011	0.00002	0.00004	0.00003	0.00005	0.00005	0.00002	
28	7	0.96168	0	0.00006	0.02096	0.00001	0.00007	0.00006	0.00002	0.00007	0.00001	0	0.00001	0.00001	0.00001	0	
29	8	0.9839	0	0.00034	0.01203	0.00001	0.00064	0.00003	0.00004	0.00001	0.00001	0.00001	0	0.00006	0.00001	0.00003	
30	9	0.45152	0.00095	0.00102	0.32665	0.00024	0.00136	0.00438	0.00095	0.00095	0.00025	0.00359	0.001	0.0004	0.00064	0.00035	
31	10	0.77338	0.00014	0.00041	0.09614	0.00008	0.00092	0.00188	0.00022	0.00035	0.00004	0.00032	0.00011	0.00042	0.00009	0.00009	
32	1	0.94119	0.00001	0.00011	0.03793	0.00001	0.0001	0.0001	0.00001	0.00003	0	0.00003	0.00001	0.00007	0.00002	0.00002	
33	2	0.95147	0.00002	0.00133	0.01301	0.00005	0.00214	0.00016	0.00006	0.00008	0.00003	0.00012	0.00003	0.00017	0.00006	0.00016	
34	3	0.90038	0.00005	0.00212	0.06101	0.00007	0.00184	0.00027	0.00023	0.00029	0.00003	0.00005	0.00005	0.00017	0.00004	0.00008	
35	4	0.61521	0.00019	0.00235	0.1901	0.00009	0.00054	0.00132	0.00013	0.00127	0.00002	0.00029	0.00028	0.00059	0.00008	0.0001	
36	5	0.63101	0.00023	0.00093	0.18209	0.00014	0.00028	0.00151	0.00015	0.00042	0.00002	0.00036	0.0001	0.0002	0.00011	0.00007	
37	6	0.77892	0.00016	0.0035	0.08212	0.00012	0.00531	0.00097	0.00398	0.0028	0.00015	0.00044	0.00031	0.00194	0.00068	0.00055	
38	7	0.88977	0.00003	0.00508	0.05029	0.0002	0.005	0.00011	0.00058	0.00022	0.00003	0.00008	0.00009	0.00066	0.00026	0.00013	
39	8	0.9694	0.00003	0.0013	0.01018	0.00011	0.00102	0.00021	0.00012	0.00009	0.00001	0.00012	0.00003	0.00031	0.00009	0.00029	
40	9	0.98643	0	0.00005	0.00915	0.00001	0.00007	0.00006	0.00001	0.00001	0	0.00002	0	0.00003	0.00001	0.00002	
41	10	0.93055	0.00003	0.00437	0.02507	0.00008	0.00741	0.00012	0.0005	0.00066	0.00002	0.00004	0.00006	0.0062	0.00022	0.00081	
42	1	0.85408	0.00018	0.0002	0.08834	0.00009	0.00061	0.00375	0.00031	0.00071	0.00004	0.00039	0.00022	0.00128	0.00018	0.00065	
43	2	0.94152	0.00009	0.00016	0.03241	0.00009	0.00034	0.00103	0.00011	0.00035	0.00002	0.00029	0.00006	0.00051	0.00009	0.00028	

Figure 4.4.2 Confidence level data

Recall that (refer table 4.3.3) hand gesture  $\varphi$  received a 100% accuracy rate for both testing environments. Let us take a closer look at their predicted probability levels.



Table 4.4.1 shows a section of the results for 2 test subjects with 10 trials (full result sheet has been attached in the Appendix section of this report).

$\varphi$	Well-lit environment	Ambient daylight
1	0.73403	0.92985
2	0.94421	0.41314
3	0.89021	0.63036
4	0.9559	0.49279
5	0.91173	0.67728
6	0.96359	0.74348
7	0.96168	0.72028
8	0.9839	0.80346
9	0.45152	0.47433
10	0.77338	0.6473
1	0.94119	0.60143
2	0.95147	0.60754
3	0.90038	0.63586
4	0.61521	0.50082
5	0.63101	0.73256
6	0.77892	0.74868
7	0.88977	0.43602
8	0.9694	0.48957
9	0.98643	0.61305
10	0.93055	0.45372

Table 4.4.1 Confidence level data for 'a'

In test scenario A, when trial 1 gives a confidence level of 0.73403, this means that the neural network predicts that the hand gesture in the captured frame is an  $\varphi$  with a probability of 0.73. Note how, although both scenarios yield the same accuracy rate, the probability of test scenario A is generally higher than that of B, the room illuminated with ambient daylight.

The following table (4.4.2) shows a summary of the results for various gestures under the two testing conditions.

Average confidence level is calculated based on 60 trials for each gesture under each scenario and median is included within brackets to get an idea of the skew.

label	Scenario A	Scenario B
a	(0.88999) 0.816693	(0.621705) 0.617576
aa	(0.879015) 0.850812	(0.943045) 0.939192
az	(0.93244) 0.896735	(0.974445) 0.933362
b	(0.88012) 0.810612	(0.633485) 0.64153
ch	(0.755935) 0.745422	(0.83252) 0.809287
d	(0.87682) 0.839078	(0.737105) 0.666603
dh	(0.64103) 0.66874	(0.60305) 0.576401
e	(0.86892) 0.780689	(0.722605) 0.69205
g	(0.93919) 0.903865	(0.895285) 0.868847
h	(0.903085) 0.877452	(0.893685) 0.895393
j	(0.69634)	(0.26165)
	0.663489	0.272398
k	(0.772815)	(0.645565)
	0.722163	0.631054
u	(0.91052)	(0.835785)
	0.860502	0.792225

*Table 4.4.2 Confidence level data for other labels*

Except for a few gestures, the general trend is that scenario A has a higher probability of predicting the correct gesture. The chart below shows that test scenario A gives an average confidence level of 0.789 while scenario B gives a confidence level of only 0.71.

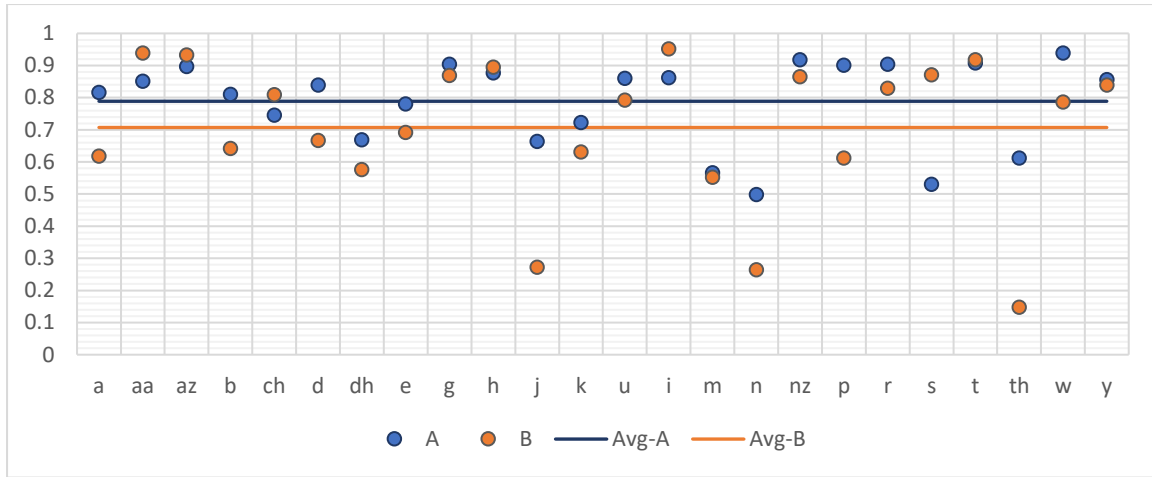


Figure 4.4.3 Change in confidence level between test scenarios

While the confidence levels vary between the two testing scenarios, that does not necessarily mean that inaccurate gestures were predicted. To illustrate this, let's extract the 8<sup>th</sup> attempt of a test subject trying to pose for the gesture label *j* ( $\mathcal{E}$ ).

	a	aa	az	b	ch	d	dh	e	g	h	i	j	k	l	m	n	nz	p	r	s	t	th	w	y				
8	0.0182	0.00491	0.0059	0.01098	0.03578	0.04033	0.04475	0.01022	0.00821	0.00201	0.00294	0.0062	0.21189	0.0481	0.10992	0.01529	0.02385	0.00908	0.00624	0.01667	0.00155	0.02767	0.01624	0.02058	0.05844	0.01341	0.02081	0.20984
9	0.06459	0.00324	0.00714	0.02109	0.01189	0.0858	0.04627	0.00863	0.00307	0.00263	0.00793	0.00319	0.26479	0.009	0.09672	0.06142	0.02014	0.02803	0.00099	0.00591	0.00347	0.03699	0.0209	0.01968	0.1388	0.00577	0.00608	0.01588

The data for the attempt is shown below:

ජ්	<b>j</b>	0.21189
ය්	<b>y</b>	0.20984
ල්	<b>l</b>	0.10992
ත්	<b>th</b>	0.05844
ක්	<b>k</b>	0.0481
[delete]	<b>delete</b>	0.04475
ඩ්	<b>d</b>	0.04033
ච්	<b>ch</b>	0.03578
ස්	<b>s</b>	0.02767
න්	<b>n</b>	0.02385
ව්	<b>w</b>	0.02081
ට්	<b>t</b>	0.02058
අ	<b>a</b>	0.0182
ප්	<b>p</b>	0.01667
[space]	<b>space</b>	0.01624
ම්	<b>m</b>	0.01529
උ	<b>u</b>	0.01341
බ්	<b>b</b>	0.01098
ද්	<b>dh</b>	0.01022
[nothing]	<b>nothing</b>	0.00908
එ	<b>e</b>	0.00821
ං	<b>nz</b>	0.00624
ඉ	<b>i</b>	0.0062
ඇ	<b>az</b>	0.0059
ආ	<b>aa</b>	0.00491
හ්	<b>h</b>	0.00294
ග්	<b>g</b>	0.00201
ර්	<b>r</b>	0.00155

Table 4.4.3 Confidence level data for label 'j'

The system predicts that the user is holding a pose ජ් with a probability of 0.212 – which can seem as if it is low and almost surpassed by the next best-predicted pose of ය් which is 0.209



A normalized confusion matrix was also prepared but it doesn't convey the data properly since the disparity between maximum and minimum is too large.



Figure 4.4.7 Normalized confusion matrix

## 4.5 Word Evaluation

The most frequently used letter in English is the letter ‘e’ if we select 100 characters from English words at random – the letter ‘e’ would be there in 12 of them. In Sinhala, the most commonly used letter is ය. [29]

#	character		occurrence rate	accumulated frequency
1	ය	(=ya)	4.4434%	4.4434%
2	ව	(=va)	4.1540%	8.5975%
⋮	⋮		⋮	⋮
274	ඒ	(=f)	0.0161%	99.0472%
275	පා	(=pau)	0.0155%	99.0627%

Figure 4.5.1 Sinhala letter frequency

Note how the Sinhala letter frequencies are much lower than that of English (partly because Sinhala has a larger alphabet). Considering the frequency of occurrence of the letters and the limited number of gestures available, the participants were given words and phrases to sign through the system. The main goal of this test was to measure how well the users adapted to the system. The time the users took to sign a set of words correctly was measured. If the test subjects made a mistake, they could go back and erase those characters from the text by posing the delete’ sign but this would make them slower and therefore increase the time to complete the test.

Users were given words and phrases from a Sinhala schoolbook and asked to gesture them correctly – the list of phrases is given below.

Phrase/sentence	The number of gestures required to sign the phrase (including space)
අපි පාඩම් කරමු	16
අර බලන්න මල් හරි ලස්සනයි	29
සෙල්ලම් ගෙදරක් හදලා	21
අලුත් අවුරුදු කැවිලි	19
ඇතෙක් බරට රත්තරං	21

Table 4.5.1 List of phrases to sign

Each user (there were 3) was asked to sign the same phrase 3 times. Results are given below:

Phrase/sentence	Gestures required	Test user 1 (time in s)	Test user 2 (time in s)	Test user 3 (time in s)
අපි පාඩම් කරමු	16	62 54 67	60 58 57	70 65 60
අර බලන්න මල් හරි ලස්සනයි	29	108 105 112	104 118 115	106 114 112
සෙල්ලම් ගෙදරක් හදලා	21	79 71 68	75 70 77	72 68 70
අලුත් අවුරුදු කැවිලි	19	60 65 61	57 67 57	55 58 59
ඇතෙක් බරට රත්තරං	21	75 71 73	78 71 76	79 77 78

Table 4.5.2 Results of word phrase test

Phrase/sentence	Average time to make a successful gesture
අපි පාඩම් කරමු	3.84
අර බලන්න මල් හරි ලස්සනයි	3.81
සෙල්ලම් ගෙදරක් හදලා	3.43
අලුත් අවුරුදු කැවිලි	3.15
ඇතෙක් බරට රත්තරං	3.59

Table 4.5.3 Average time for a gesture

Table 4.5.3 shows the average time taken by the system to register a successful gesture. Note that the timer started at the beginning user starts posing and stopped only after the correct



complete phrase had appeared on the screen. This means that there may have been cases where the user was holding the ‘delete’ gesture because the system identified a wrong character.

However, the system takes about 3.56 seconds per gesture which means the user can convey about 17 gestures per minute. This time can be improved by changing the *text speed* slider on the GUI, once a user is comfortable with the system, the user may be able to push the system to 20 characters per minute.

## 4.6 Summary

Results and Evaluation chapter showed that the Design implemented in the Methodology chapter was successful and functioning. It showed the accuracy levels and statistics which can be used in the next chapter to come to conclusions about the overall state of the project.

## Chapter 5 Conclusion and Future Work

Main objective of this project was to design and develop a software application that captures video in real-time of a person using Sinhalese Fingerspelling sign language, processes the frames through a neural network based on machine learning and interpret the signed hand gestures in the video and output text to a screen as words. To achieve this goal, a considerable amount of time was spent in researching about the deaf community in Sri Lanka, their sign language usage, the evolution of the hand gestures and involvement, and the role of the human interpreters.

Languages, hand movements and gestures of other sign languages were also studied – especially American sign language and the Arabic sign language which have similar gestures to the Sinhala fingerspelling alphabet. Although there have been several attempts at using image processing to interpret Sinhala sign language all these methods contained only 10-15 hand gestures and used wearable technology such as Kinect, armbands or coloured gloves. This project took the status quo a few steps further by expanding the interpreted gestures to include 27 gestures and added the ability to form complete sentences while all other previous Sinhala interpretation systems had character recognition and rudimentary word clustering. A dataset was also created for machine learning in order to train the system. The dataset included 27000+ images. Inception neural network was selected in part due to the resurgence of interest in deep learning in the last decade and for its versatility and accuracy in image processing. Another motivation for selecting a neural network was that previous Sinhala sign language projects hadn't utilized its potential and try out how well Inception fares given that Sinhala has a larger pool size of hand gestures.

The Python based application developed uses the web camera of a PC or laptop to capture video stream.

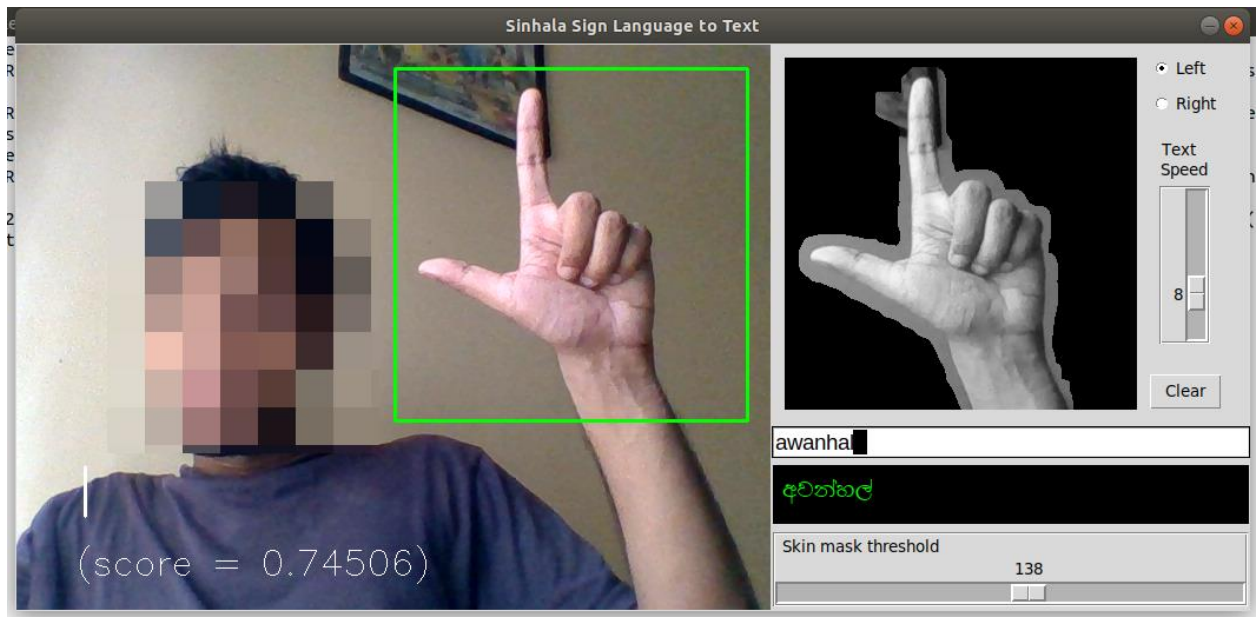


Figure 4.6.1 Application in use

A screen recording of the system being used can be seen via the link:

<https://youtu.be/hMzOWYjWFzI>

Overall, the results showed a high accuracy of 95%. The recognition accuracy could have been higher if it wasn't for the gestures ඔ, ඤ and ඞ which scored a combined rate of around 70% due to those gestures being similar to each other thereby resulting in lowered predictability of those gestures.

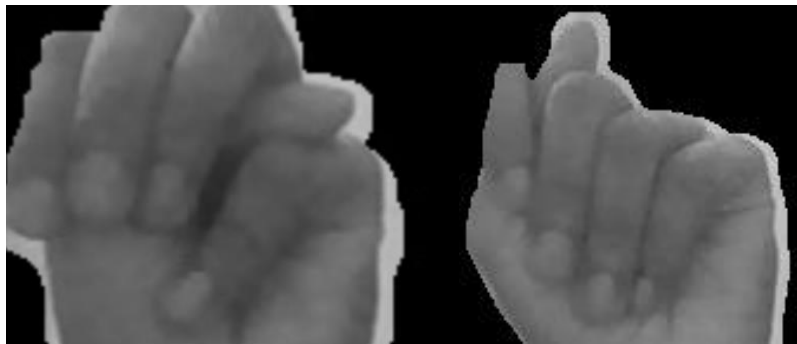
The project reached its objectives on what it set out to do. From its inception, the project was aimed at improving the current research/projects done in this area based on Sinhala sign language. This project expanded the scope by including more hand gestures, using Inception through machine learning and removing expensive wearable equipment.

## 5.1 Interpreting Results

The system achieved an overall accuracy of 95% for character recognition. It has an average speed of 17 signs per minute (including diacritics – note that, most characters comprise of 2 hand gestures).

There was no observable difference in accuracy when posing hand signs with the left or right hands – this is likely due to the fact that the training programme was designed to flip images horizontally during training.

Distinct hand gestures achieved very high success rates while gestures that had similar finger placement or hand shape fared poorly. One reason for this is that Sinhala has a larger character set than most other languages and therefore has similar gestures with subtle variations. Another reason is that the camera used for training and testing has a very low resolution and pixel density (0.7 MP), hence the image and in extension the features of the cropped area containing light and dark features of the skin are of poor quality; the system cannot extract and interpret these results correctly.



*Figure 5.1.1 Distorted images from similar gestures*

Note how in the figure above, the finger positions are almost the same except for the position of the thumb, these two gestures represent two characters. Results show that they have a low accuracy rate. It is likely that on most occasions, the system doesn't pick up on the small variations.

The way a person holds a hand gesture also matters for the accuracy of the results. Most gestures such as ජ, ඨ or ජ still managed to achieve high accuracy even though test subjects held slight variations of the intended hand gesture (changes in the angle of fingers, hand rotated slightly, etc.) but for ambiguous signs such as ඹ or ඞ, a slight variation amplified the error and lead to incorrect predictions.

The system separates the skin tones from the background and creates a mask around the hand that cuts off any pixels from the background that interferes with the prediction. A plain background was used for testing. As long as the background does not have any object similar in colour to the user's skin tone, the system can mask them out. It was noted that there is a clear difference in the success rate of gesture recognition when the lighting changes. Reproducing the lighting conditions that were there when images were captured for the training dataset

yields the best results. Changing the lighting changes how the hand is illuminated – this changes the shadows on the skin which in turn affects the success probability of the gesture prediction.

Among the factors outlined, the issue with lighting can be considered the biggest deterrent to a successful prediction. If the hand is illuminated too brightly, the image gets washed out and pixel information for skin areas is redundant. On the other hand, if the image is too dark, the web camera shifts the white balance automatically can cause a shift in the pixel values.

## 5.2 Difficulties and Limitations

As mentioned above, the biggest limitation of the system was that it wasn't very adaptive to changing light conditions which caused a 2% drop in accuracy when testing environments were changed. There was a counter mechanism in place – the trackbar – to change the skin mask threshold but it only changed the threshold of values once the image had already been captured and therefore was limited in what it could do. It is not possible to change the white balance or any settings on the web camera since a generic camera found in laptops was used. One way to mitigate this issue is to illuminate the room well or train the image dataset under various lighting conditions.

Another unforeseen problem encountered during testing is that, when the web camera operates for a long time, the area around the sensors heat up and cause a slight distortion (shift in red pixel values) of the images which can have a negative effect on the predictions, especially since the red shift can cause skin colour to change on the image. Using a higher-grade web camera would have solved this issue but since the project scope was specifically defined to use a simple web camera – it was decided to not make any changes to the equipment.

## 5.3 Future Work

The scope of the project was covered; however, there are few key areas which can be improved further. The current system identifies 27 static gestures, this can be expanded further to include detection of hand gestures with motion.

From its inception, the idea was to build a foundation or a baseline from which future projects can build on. The current application is a standalone desktop app based on Python; in the future, this can (and should) move to a web-based application so that two parties can communicate over the Internet. Taking another step forward, the application can be deployed on a phone and the text component eliminated altogether and be replaced by a voice recognition technology. Hence, the hand gestures would be converted to speech in real-time.

## References

- [1] “The History of Sign Language,” *Study.com*. <https://study.com/academy/lesson/the-history-of-sign-language.html> (accessed Jan. 21, 2020).
- [2] “Manually Coded Language and Alternate Sign Systems · Deaf: Cultures and Communication, 1600 to the Present · Online Exhibits@Yale.” <http://exhibits.library.yale.edu/exhibits/show/deafculture/manualsystems> (accessed Oct. 28, 2019).
- [3] Rohana Special School, *An introduction to Sri Lankan Sign Language*. Welegoda, Matara, Sri Lanka: Rohana Special School, 2007.
- [4] “Sinhala Sign Language the main communication mode for the Deaf in Sri Lanka | Daily FT.”
- [5] R. A. Allan, *A History of the Personal Computer: The People and the Technology*. Allan Pub., 2001.
- [6] M. Popa, “Hand gesture recognition based on accelerometer sensors,” in *The 7th International Conference on Networked Computing and Advanced Information Management*, Jun. 2011, pp. 115–120.
- [7] K. Grifantini, “Open-Source Data Glove,” *MIT Technology Review*. <https://www.technologyreview.com/s/414021/open-source-data-glove/> (accessed Oct. 28, 2019).
- [8] C. S. Ingulkar and A. N. Gaikwad, “Hand Data Glove : A wearable real time device for human computer Interaction,” 2013.
- [9] T. Schlömer, B. Poppinga, N. Henze, and S. Boll, “Gesture Recognition with a Wii Controller,” in *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction*, New York, NY, USA, 2008, pp. 11–14, doi: 10.1145/1347390.1347395.
- [10] A. L. P. Madushanka, R. G. D. C. Senevirathne, L. M. H. Wijesekara, S. M. K. D. Arunatilake, and K. D. Sandaruwan, “Framework for Sinhala Sign Language recognition and translation using a wearable armband,” in *2016 Sixteenth International Conference on*

*Advances in ICT for Emerging Regions (ICTer)*, Sep. 2016, pp. 49–57, doi: 10.1109/ICTER.2016.7829898.

- [11] “Myo Armband – Control Your Tech Devices Using Hand Gestures.” <http://worldofnovelty.com/myo-armband-control-your-tech-devices-using-hand-gestures/> (accessed Oct. 28, 2019).
- [12] H.-D. Yang, “Sign language recognition with the Kinect sensor based on conditional random fields,” *Sensors*, vol. 15, no. 1, pp. 135–147, Dec. 2014, doi: 10.3390/s150100135.
- [13] Z. Ren, J. Yuan, J. Meng, and Z. Zhang, “Robust Part-Based Hand Gesture Recognition Using Kinect Sensor,” *IEEE Trans. Multimed.*, vol. 15, no. 5, pp. 1110–1120, Aug. 2013, doi: 10.1109/TMM.2013.2246148.
- [14] P. Fernando and P. Wimalaratne, “Sign Language Translation Approach to Sinhalese Language,” *GSTF J. Comput. JoC*, vol. 5, Sep. 2016, doi: 10.7603/s40601-016-0009-8.
- [15] M. Cote, P. Payeur, and G. Comeau, “Comparative Study of Adaptive Segmentation Techniques for Gesture Analysis in Unconstrained Environments,” in *Proceedings of the 2006 IEEE International Workshop on Imaging Systems and Techniques (IST 2006)*, Apr. 2006, pp. 28–33, doi: 10.1109/IST.2006.1650770.
- [16] P. Mekala, Y. Gao, J. Fan, and A. Davari, “Real-time sign language recognition based on neural network architecture,” in *2011 IEEE 43rd Southeastern Symposium on System Theory*, Mar. 2011, pp. 195–199, doi: 10.1109/SSST.2011.5753805.
- [17] J. Pansare, S. Gawande, and M. Ingle, “Real-Time Static Hand Gesture Recognition for American Sign Language (ASL) in Complex Background,” *J. Signal Inf. Process.*, vol. 03, pp. 364–367, Jan. 2012, doi: 10.4236/jsip.2012.33047.
- [18] H. C. M. Herath, W.A.L.V.Kumari, W. A. P. B. Senevirathne, and M. Dissanayake, “IMAGE BASED SIGN LANGUAGE RECOGNITION SYSTEM FOR SINHALA SIGN LANGUAGE,” Apr. 2013.
- [19] T. Starner and A. Pentland, “Real-Time American Sign Language Recognition Using Desk and Wearable Computer Based Video,” *IEEE Trans. PATTERN Anal. Mach. Intell.*, vol. 20, no. 12, p. 5, 1998.



- [20] X. Wang, M. Xia, H. Cai, Y. Gao, and C. Cattani, “Hidden-Markov-Models-Based Dynamic Hand Gesture Recognition,” *Mathematical Problems in Engineering*, 2012. <https://www.hindawi.com/journals/mpe/2012/986134/> (accessed Oct. 12, 2019).
- [21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” in *Proceedings of the Ieee*, 1998, pp. 2278–2324.
- [22] O. Potkin and A. Philippovich, *Hand gestures detection, tracking and classification using Convolutional Neural Network*. 2019.
- [23] C. Szegedy *et al.*, “Going Deeper with Convolutions,” *ArXiv14094842 Cs*, Sep. 2014, Accessed: Jun. 21, 2020. [Online]. Available: <http://arxiv.org/abs/1409.4842>.
- [24] B. Raj, “A Simple Guide to the Versions of the Inception Network,” *Medium*, May 30, 2018. <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202> (accessed May 30, 2020).
- [25] V. S. S. Kandarpa, L. Bonta, A. B, P. K. Baruah, and S. Sankara, “Evaluating Training Time of Inception-v3 and Resnet-50,101 Models using TensorFlow across CPU and GPU,” Mar. 2018, pp. 1964–1968, doi: 10.1109/ICECA.2018.8474878.
- [26] C. ennehar Bencheriet, B. Oudjani, and H. Tebbikh, “An Appropriate Color space to Improve Human Skin Detection,” *INFOCOMP-J. Comput. Sci.*, vol. 9, pp. 1–10, Dec. 2010.
- [27] “YCbCr,” *Wikipedia*. Mar. 22, 2020, Accessed: Jan. 21, 2020. [Online]. Available: <https://en.wikipedia.org/w/index.php?title=YCbCr&oldid=946790958>.
- [28] “Color Models.” [https://scc.ustc.edu.cn/zlsc/sugon/intel/ipp/ipp\\_manual/IPPI/ippi\\_ch6/ch6\\_color\\_models.htm](https://scc.ustc.edu.cn/zlsc/sugon/intel/ipp/ipp_manual/IPPI/ippi_ch6/ch6_color_models.htm) (accessed Jan. 21, 2020).
- [29] S. Goonetilleke, Y. Hayashi, Y. Itoh, and F. Kishino, “An Efficient and User-friendly Sinhala Input Method Based on Phonetic Transcription,” *J. Nat. Lang. Process.*, vol. 14, pp. 147–166, Jan. 2007, doi: 10.5715/jnlp.14.5\_147.

## Appendix

Training output to terminal:

```
Step: 0, Train accuracy: 20.0000%, Cross entropy: 3.259764, Validation accuracy: 7.0% (N=100)
Step: 50, Train accuracy: 82.0000%, Cross entropy: 2.387794, Validation accuracy: 83.0% (N=100)
Step: 100, Train accuracy: 94.0000%, Cross entropy: 1.796941, Validation accuracy: 94.0% (N=100)
Step: 150, Train accuracy: 98.0000%, Cross entropy: 1.409656, Validation accuracy: 89.0% (N=100)
Step: 200, Train accuracy: 95.0000%, Cross entropy: 1.158465, Validation accuracy: 95.0% (N=100)
Step: 250, Train accuracy: 93.0000%, Cross entropy: 0.911820, Validation accuracy: 98.0% (N=100)
Step: 300, Train accuracy: 98.0000%, Cross entropy: 0.721095, Validation accuracy: 97.0% (N=100)
Step: 350, Train accuracy: 96.0000%, Cross entropy: 0.693118, Validation accuracy: 96.0% (N=100)
Step: 400, Train accuracy: 96.0000%, Cross entropy: 0.591889, Validation accuracy: 97.0% (N=100)
Step: 450, Train accuracy: 100.0000%, Cross entropy: 0.479280, Validation accuracy: 98.0% (N=100)
Step: 500, Train accuracy: 98.0000%, Cross entropy: 0.478022, Validation accuracy: 94.0% (N=100)
Step: 550, Train accuracy: 99.0000%, Cross entropy: 0.374854, Validation accuracy: 95.0% (N=100)
Step: 600, Train accuracy: 100.0000%, Cross entropy: 0.373101, Validation accuracy: 98.0% (N=100)
Step: 650, Train accuracy: 99.0000%, Cross entropy: 0.308552, Validation accuracy: 100.0% (N=100)
Step: 700, Train accuracy: 97.0000%, Cross entropy: 0.441026, Validation accuracy: 95.0% (N=100)
Step: 750, Train accuracy: 96.0000%, Cross entropy: 0.370755, Validation accuracy: 99.0% (N=100)
Step: 800, Train accuracy: 100.0000%, Cross entropy: 0.309538, Validation accuracy: 98.0% (N=100)
Step: 850, Train accuracy: 98.0000%, Cross entropy: 0.307917, Validation accuracy: 96.0% (N=100)
Step: 900, Train accuracy: 98.0000%, Cross entropy: 0.296353, Validation accuracy: 96.0% (N=100)
Step: 950, Train accuracy: 99.0000%, Cross entropy: 0.262363, Validation accuracy: 99.0% (N=100)
Step: 1000, Train accuracy: 100.0000%, Cross entropy: 0.257949, Validation accuracy: 97.0% (N=100)
Step: 1050, Train accuracy: 99.0000%, Cross entropy: 0.209571, Validation accuracy: 99.0% (N=100)
Step: 1100, Train accuracy: 98.0000%, Cross entropy: 0.288907, Validation accuracy: 98.0% (N=100)
Step: 1150, Train accuracy: 98.0000%, Cross entropy: 0.266357, Validation accuracy: 99.0% (N=100)
Step: 1200, Train accuracy: 100.0000%, Cross entropy: 0.200622, Validation accuracy: 98.0% (N=100)
Step: 1250, Train accuracy: 98.0000%, Cross entropy: 0.219140, Validation accuracy: 98.0% (N=100)
Step: 1300, Train accuracy: 99.0000%, Cross entropy: 0.199692, Validation accuracy: 97.0% (N=100)
Step: 1350, Train accuracy: 100.0000%, Cross entropy: 0.170689, Validation accuracy: 98.0% (N=100)
Step: 1400, Train accuracy: 99.0000%, Cross entropy: 0.187579, Validation accuracy: 99.0% (N=100)
Step: 1450, Train accuracy: 98.0000%, Cross entropy: 0.170079, Validation accuracy: 99.0% (N=100)
Step: 1500, Train accuracy: 99.0000%, Cross entropy: 0.159585, Validation accuracy: 100.0% (N=100)
Step: 1550, Train accuracy: 100.0000%, Cross entropy: 0.144855, Validation accuracy: 100.0% (N=100)
Step: 1600, Train accuracy: 98.0000%, Cross entropy: 0.192853, Validation accuracy: 100.0% (N=100)
Step: 1650, Train accuracy: 100.0000%, Cross entropy: 0.168094, Validation accuracy: 100.0% (N=100)
Step: 1700, Train accuracy: 99.0000%, Cross entropy: 0.182677, Validation accuracy: 97.0% (N=100)
Step: 1750, Train accuracy: 98.0000%, Cross entropy: 0.156971, Validation accuracy: 100.0% (N=100)
Step: 1800, Train accuracy: 100.0000%, Cross entropy: 0.151367, Validation accuracy: 99.0% (N=100)
Step: 1850, Train accuracy: 100.0000%, Cross entropy: 0.116089, Validation accuracy: 99.0% (N=100)
Step: 1900, Train accuracy: 99.0000%, Cross entropy: 0.135607, Validation accuracy: 99.0% (N=100)
Step: 1950, Train accuracy: 100.0000%, Cross entropy: 0.143743, Validation accuracy: 100.0% (N=100)
Step: 2000, Train accuracy: 99.0000%, Cross entropy: 0.158373, Validation accuracy: 99.0% (N=100)
Step: 2050, Train accuracy: 100.0000%, Cross entropy: 0.151496, Validation accuracy: 99.0% (N=100)
Step: 2100, Train accuracy: 100.0000%, Cross entropy: 0.130377, Validation accuracy: 98.0% (N=100)
Step: 2150, Train accuracy: 100.0000%, Cross entropy: 0.120119, Validation accuracy: 98.0% (N=100)
Step: 2200, Train accuracy: 99.0000%, Cross entropy: 0.130551, Validation accuracy: 100.0% (N=100)
Step: 2250, Train accuracy: 99.0000%, Cross entropy: 0.151680, Validation accuracy: 99.0% (N=100)
Step: 2300, Train accuracy: 100.0000%, Cross entropy: 0.086753, Validation accuracy: 100.0% (N=100)
Step: 2350, Train accuracy: 100.0000%, Cross entropy: 0.131505, Validation accuracy: 100.0% (N=100)
Step: 2400, Train accuracy: 100.0000%, Cross entropy: 0.104784, Validation accuracy: 100.0% (N=100)
Step: 2450, Train accuracy: 100.0000%, Cross entropy: 0.098050, Validation accuracy: 99.0% (N=100)
```

Step: 2500, Train accuracy: 98.0000%, Cross entropy: 0.150031, Validation accuracy: 99.0% (N=100)  
Step: 2550, Train accuracy: 99.0000%, Cross entropy: 0.108533, Validation accuracy: 100.0% (N=100)  
Step: 2600, Train accuracy: 100.0000%, Cross entropy: 0.115816, Validation accuracy: 99.0% (N=100)  
Step: 2650, Train accuracy: 100.0000%, Cross entropy: 0.130321, Validation accuracy: 99.0% (N=100)  
Step: 2700, Train accuracy: 100.0000%, Cross entropy: 0.101081, Validation accuracy: 98.0% (N=100)  
Step: 2750, Train accuracy: 100.0000%, Cross entropy: 0.126274, Validation accuracy: 100.0% (N=100)  
Step: 2800, Train accuracy: 99.0000%, Cross entropy: 0.099647, Validation accuracy: 100.0% (N=100)  
Step: 2850, Train accuracy: 96.0000%, Cross entropy: 0.127849, Validation accuracy: 99.0% (N=100)  
Step: 2900, Train accuracy: 98.0000%, Cross entropy: 0.125995, Validation accuracy: 100.0% (N=100)  
Step: 2950, Train accuracy: 100.0000%, Cross entropy: 0.089759, Validation accuracy: 99.0% (N=100)  
Step: 3000, Train accuracy: 100.0000%, Cross entropy: 0.093581, Validation accuracy: 99.0% (N=100)  
Step: 3050, Train accuracy: 100.0000%, Cross entropy: 0.079845, Validation accuracy: 99.0% (N=100)  
Step: 3100, Train accuracy: 100.0000%, Cross entropy: 0.079974, Validation accuracy: 99.0% (N=100)  
Step: 3150, Train accuracy: 100.0000%, Cross entropy: 0.068420, Validation accuracy: 100.0% (N=100)  
Step: 3200, Train accuracy: 100.0000%, Cross entropy: 0.098370, Validation accuracy: 99.0% (N=100)  
Step: 3250, Train accuracy: 100.0000%, Cross entropy: 0.085931, Validation accuracy: 100.0% (N=100)  
Step: 3300, Train accuracy: 100.0000%, Cross entropy: 0.100596, Validation accuracy: 100.0% (N=100)  
Step: 3350, Train accuracy: 99.0000%, Cross entropy: 0.096771, Validation accuracy: 98.0% (N=100)  
Step: 3400, Train accuracy: 100.0000%, Cross entropy: 0.084662, Validation accuracy: 100.0% (N=100)  
Step: 3450, Train accuracy: 100.0000%, Cross entropy: 0.097025, Validation accuracy: 100.0% (N=100)  
Step: 3500, Train accuracy: 100.0000%, Cross entropy: 0.081127, Validation accuracy: 99.0% (N=100)  
Step: 3550, Train accuracy: 100.0000%, Cross entropy: 0.080972, Validation accuracy: 98.0% (N=100)  
Step: 3600, Train accuracy: 99.0000%, Cross entropy: 0.107662, Validation accuracy: 99.0% (N=100)  
Step: 3650, Train accuracy: 100.0000%, Cross entropy: 0.090089, Validation accuracy: 100.0% (N=100)  
Step: 3700, Train accuracy: 99.0000%, Cross entropy: 0.070255, Validation accuracy: 100.0% (N=100)  
Step: 3750, Train accuracy: 100.0000%, Cross entropy: 0.074071, Validation accuracy: 99.0% (N=100)  
Step: 3800, Train accuracy: 100.0000%, Cross entropy: 0.078690, Validation accuracy: 100.0% (N=100)  
Step: 3850, Train accuracy: 100.0000%, Cross entropy: 0.080332, Validation accuracy: 97.0% (N=100)  
Step: 3900, Train accuracy: 100.0000%, Cross entropy: 0.070877, Validation accuracy: 99.0% (N=100)  
Step: 3950, Train accuracy: 99.0000%, Cross entropy: 0.100446, Validation accuracy: 99.0% (N=100)  
Step: 4000, Train accuracy: 99.0000%, Cross entropy: 0.098909, Validation accuracy: 100.0% (N=100)  
Step: 4050, Train accuracy: 100.0000%, Cross entropy: 0.075109, Validation accuracy: 100.0% (N=100)  
Step: 4100, Train accuracy: 100.0000%, Cross entropy: 0.072808, Validation accuracy: 98.0% (N=100)  
Step: 4150, Train accuracy: 99.0000%, Cross entropy: 0.088094, Validation accuracy: 99.0% (N=100)  
Step: 4200, Train accuracy: 100.0000%, Cross entropy: 0.054614, Validation accuracy: 100.0% (N=100)  
Step: 4250, Train accuracy: 99.0000%, Cross entropy: 0.069433, Validation accuracy: 100.0% (N=100)  
Step: 4300, Train accuracy: 100.0000%, Cross entropy: 0.077343, Validation accuracy: 99.0% (N=100)  
Step: 4350, Train accuracy: 100.0000%, Cross entropy: 0.057777, Validation accuracy: 99.0% (N=100)  
Step: 4400, Train accuracy: 100.0000%, Cross entropy: 0.097548, Validation accuracy: 100.0% (N=100)  
Step: 4450, Train accuracy: 98.0000%, Cross entropy: 0.098741, Validation accuracy: 99.0% (N=100)  
Step: 4500, Train accuracy: 100.0000%, Cross entropy: 0.060563, Validation accuracy: 100.0% (N=100)  
Step: 4550, Train accuracy: 99.0000%, Cross entropy: 0.086611, Validation accuracy: 100.0% (N=100)  
Step: 4600, Train accuracy: 100.0000%, Cross entropy: 0.066047, Validation accuracy: 100.0% (N=100)  
Step: 4650, Train accuracy: 100.0000%, Cross entropy: 0.045480, Validation accuracy: 100.0% (N=100)  
Step: 4700, Train accuracy: 97.0000%, Cross entropy: 0.120958, Validation accuracy: 100.0% (N=100)  
Step: 4750, Train accuracy: 100.0000%, Cross entropy: 0.049715, Validation accuracy: 100.0% (N=100)  
Step: 4800, Train accuracy: 100.0000%, Cross entropy: 0.052880, Validation accuracy: 100.0% (N=100)  
Step: 4850, Train accuracy: 99.0000%, Cross entropy: 0.107312, Validation accuracy: 100.0% (N=100)  
Step: 4900, Train accuracy: 100.0000%, Cross entropy: 0.080551, Validation accuracy: 100.0% (N=100)  
Step: 4950, Train accuracy: 98.0000%, Cross entropy: 0.073019, Validation accuracy: 100.0% (N=100)  
Step: 4999, Train accuracy: 100.0000%, Cross entropy: 0.054638, Validation accuracy: 99.0% (N=100)  
Final test accuracy = 99.6% (N=2908)

Confusion Matrix for test results:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
	අ	ආ	ඇ	ඉ	උ	එ	ඞ	ඟ	ඞ	ච	ඳ	ඞ	ඪ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ	ඞ
අ	120															2											
ආ		120		2																							
ඇ			119	1						1																	
ඉ				115		2																					
උ					119		4																			2	
එ				2		116				2																	
ඞ							120																				
ඟ								116																			
ඞ									110																		
ච					2					118																	
ඳ											113	3							7								
ඞ												88	4					4				1					
ඪ									1	6		117															
ඞ											18		88				28					3					
ඞ															119								2				
ඞ																	117				1						
ඞ												6	28					92				3					
ඞ									9										114							2	
ඞ																				113							
ඞ					1																120						
ඞ			1																			119					
ඞ												8				1							113				
ඞ															1								118			2	
ඞ																								120			
ඞ																									120		
[space]																										116	
[delete]																		2									118

		a	aa	az	b	ch	d	delete	dh	e	g	h	i	j	k	l	m	n	nothing
test-1 scene A	1	0.29849	0.00207	0.0338	0.06833	0.0027	0.061	0.00313	0.0046	0.01977	0.00034	0.00258	0.00176	0.23315	0.00325	0.005	0.00135	0.07123	0.01214
	2	0.51983	0.00228	0.00593	0.10239	0.00109	0.0134	0.00146	0.00148	0.015	0.00078	0.00609	0.0031	0.03511	0.0072	0.00342	0.00133	0.02155	0.07864
	3	0.37989	0.0045	0.00922	0.09125	0.00192	0.00664	0.01002	0.00099	0.03016	0.00081	0.02116	0.00457	0.04101	0.00263	0.00844	0.0021	0.04272	0.15035
	4	0.81128	0.00019	0.00456	0.04747	0.00034	0.01423	0.00104	0.00032	0.00095	0.00026	0.00186	0.00032	0.01092	0.00052	0.00129	0.00105	0.04653	0.0197
	5	0.60824	0.00126	0.01046	0.04996	0.00272	0.0099	0.00465	0.00106	0.00765	0.00036	0.01041	0.00137	0.02484	0.00237	0.00753	0.00078	0.0369	0.08943
	6	0.88667	0.00008	0.00138	0.03379	0.00009	0.00399	0.00042	0.00075	0.0023	0.00022	0.0001	0.00027	0.00382	0.00032	0.00027	0.00082	0.0393	0.00187
	7	0.90674	0.00005	0.00057	0.02795	0.00007	0.0012	0.00219	0.00009	0.00064	0.00002	0.00049	0.00005	0.0027	0.00009	0.00042	0.00314	0.02448	0.0023
	8	0.798	0.00065	0.00401	0.01487	0.00183	0.00306	0.00902	0.0002	0.00157	0.00021	0.00161	0.00109	0.00662	0.00047	0.00642	0.00073	0.03271	0.00709
	9	0.69596	0.00098	0.00822	0.03294	0.00238	0.00841	0.00982	0.00053	0.00815	0.00044	0.00183	0.00225	0.01979	0.00081	0.00313	0.00198	0.07172	0.01676
	10	0.76733	0.00044	0.00311	0.04427	0.00122	0.00587	0.00801	0.00033	0.00102	0.00009	0.00075	0.00065	0.00583	0.00019	0.00196	0.00307	0.06274	0.00339
test-2 scene A	1	0.81993	0.00019	0.00191	0.01946	0.00025	0.0029	0.00621	0.00016	0.00243	0.00005	0.00037	0.00029	0.00811	0.00005	0.00063	0.00092	0.05094	0.00301
	2	0.73582	0.0001	0.00168	0.00872	0.00024	0.00338	0.01641	0.00008	0.00132	0.00004	0.00023	0.00013	0.00325	0.00003	0.00042	0.00958	0.07804	0.00222
	3	0.9058	0.00004	0.00048	0.02936	0.00005	0.00079	0.00149	0.00005	0.00121	0.00001	0.00004	0.00008	0.00347	0.00002	0.00016	0.00029	0.02325	0.00125
	4	0.92602	0.00009	0.0006	0.02616	0.0001	0.00099	0.00254	0.0001	0.00087	0.00001	0.0001	0.00008	0.00651	0.00005	0.00096	0.00034	0.00614	0.00104
	5	0.97059	0.00002	0.00057	0.01371	0.00002	0.00163	0.00049	0.00006	0.00035	0	0.00001	0.00002	0.00117	0.00001	0.0001	0.00041	0.00285	0.00047
	6	0.8956	0.00003	0.0004	0.00971	0.00011	0.00116	0.00379	0.00001	0.00044	0.00001	0.00009	0.00004	0.00182	0.00001	0.00019	0.00093	0.02624	0.00074
	7	0.88725	0.00004	0.0004	0.03345	0.00008	0.00082	0.00205	0.00015	0.00146	0.00001	0.00002	0.00007	0.00589	0.00002	0.00022	0.00037	0.04261	0.00057
	8	0.91182	0.00005	0.00033	0.01725	0.00003	0.00059	0.00199	0.00003	0.00069	0.00002	0.00009	0.00007	0.00441	0.00004	0.00027	0.00078	0.02633	0.00085
	9	0.90407	0.00007	0.00142	0.01629	0.0001	0.00905	0.00122	0.00021	0.00166	0.00002	0.00006	0.0001	0.02226	0.00004	0.00168	0.00036	0.01134	0.0024
	10	0.87391	0.0001	0.00149	0.03668	0.00009	0.00405	0.00208	0.0001	0.00097	0.00002	0.00003	0.00015	0.00721	0.00004	0.0006	0.00081	0.04452	0.00141
test-3 scene A	1	0.73403	0.00005	0.00326	0.02186	0.0002	0.00316	0.00029	0.0001	0.00015	0.00008	0.00031	0.00007	0.00019	0.00008	0.00009	0.00331	0.1934	0.00105
	2	0.94421	0.00001	0.0002	0.03493	0.00002	0.00017	0.00009	0.00004	0.00004	0.00001	0.00004	0.00001	0.00004	0.00003	0.00001	0.00066	0.00711	0.00036
	3	0.89021	0.00004	0.00323	0.08014	0.00009	0.00088	0.00007	0.00007	0.00025	0.00001	0.00008	0.00005	0.0002	0.00011	0.00011	0.00035	0.01172	0.00069
	4	0.9559	0.00001	0.00061	0.0184	0.00007	0.00018	0.00008	0.00005	0.00022	0.00003	0.00003	0.00003	0.00009	0.00006	0.00002	0.0002	0.01139	0.00031
	5	0.91173	0.00006	0.00077	0.07194	0.00004	0.00032	0.00009	0.00022	0.00039	0.00002	0.00007	0.00003	0.00005	0.00012	0.00005	0.00053	0.00236	0.00059
	6	0.96359	0.00001	0.00022	0.01761	0.00007	0.0001	0.00024	0.00004	0.00011	0.00002	0.00004	0.00003	0.00005	0.00005	0.00002	0.00026	0.00285	0.00025
	7	0.96168	0	0.00006	0.02096	0.00001	0.00007	0.00006	0.00002	0.00007	0.00001	0	0.00001	0.00001	0.00001	0	0.00019	0.00958	0.00004
	8	0.9839	0	0.00034	0.01203	0.00001	0.00064	0.00003	0.00004	0.00001	0.00001	0.00001	0	0.00006	0.00001	0.00003	0.00006	0.0006	0.00015
	9	0.45152	0.00095	0.00102	0.32665	0.00024	0.00136	0.00438	0.00095	0.00095	0.00025	0.00359	0.001	0.0004	0.00064	0.00035	0.00629	0.02199	0.00418
	10	0.77338	0.00014	0.00041	0.09614	0.00008	0.00092	0.00188	0.00022	0.00035	0.00004	0.00032	0.00011	0.00042	0.00009	0.00009	0.00363	0.06376	0.00082
test-4 scene A	1	0.94119	0.00001	0.00011	0.03793	0.00001	0.0001	0.0001	0.00001	0.00003	0	0.00003	0.00001	0.00007	0.00002	0.00002	0.00016	0.00544	0.00015
	2	0.95147	0.00002	0.00133	0.01301	0.00005	0.00214	0.00016	0.00006	0.00008	0.00003	0.00012	0.00003	0.00017	0.00006	0.00016	0.00135	0.00862	0.00063
	3	0.90038	0.00005	0.00212	0.06101	0.00007	0.00184	0.00027	0.00023	0.00029	0.00003	0.00005	0.00005	0.00017	0.00004	0.00008	0.00065	0.00943	0.00065
	4	0.61521	0.00019	0.00235	0.1901	0.00009	0.00054	0.00132	0.00013	0.00127	0.00002	0.00029	0.00028	0.00059	0.00008	0.0001	0.00148	0.0789	0.00087
	5	0.63101	0.00023	0.00093	0.18209	0.00014	0.00028	0.00151	0.00015	0.00042	0.00002	0.00036	0.0001	0.0002	0.00011	0.00007	0.00216	0.05329	0.00049
	6	0.77892	0.00016	0.0035	0.08212	0.00012	0.00591	0.00097	0.00398	0.0028	0.00015	0.00044	0.00031	0.00194	0.00068	0.00055	0.00252	0.03233	0.00187
	7	0.88977	0.00003	0.00508	0.05029	0.0002	0.005	0.00011	0.00058	0.00022	0.00003	0.00008	0.00009	0.00066	0.00026	0.00013	0.00027	0.02878	0.00079
	8	0.9694	0.00003	0.0013	0.01018	0.00011	0.00102	0.00021	0.00012	0.00009	0.00001	0.00012	0.00003	0.00031	0.00009	0.00029	0.0001	0.0054	0.00059
	9	0.98643	0	0.00005	0.00915	0.00001	0.00007	0.00006	0.00001	0.00001	0	0.00002	0	0.00003	0.00001	0.00002	0.00004	0.0008	0.0001
	10	0.93055	0.00003	0.00437	0.02507	0.00008	0.00741	0.00012	0.0005	0.00066	0.00002	0.00004	0.00006	0.0062	0.00022	0.00081	0.00008	0.00854	0.00177
test-5 scene A	1	0.85408	0.00018	0.0002	0.08834	0.00009	0.00061	0.00375	0.00031	0.00071	0.00004	0.00039	0.00022	0.00128	0.00018	0.00065	0.00157	0.00264	0.00186
	2	0.94152	0.00009	0.00016	0.03241	0.00009	0.00034	0.00103	0.00011	0.00035	0.00002	0.00029	0.00006	0.00051	0.00009	0.00028	0.00035	0.00187	0.00101
	3	0.93115	0.00009	0.00014	0.0406	0.00006	0.00037	0.00063	0.00008	0.00025	0.00003	0.00046	0.00019	0.00031	0.00012	0.00017	0.00087	0.00117	0.00124
	4	0.87876	0.00017	0.00031	0.0416	0.00011	0.00047	0.00337	0.00013	0.00027	0.00006	0.00068	0.00011	0.00018	0.00007	0.00019	0.00208	0.00222	0.00274
	5	0.91023	0.00007	0.00042	0.06084	0.00006	0.00024	0.00065	0.00005	0.00017	0.00003	0.00043	0.00012	0.00015	0.00006	0.00009	0.00069	0.0006	0.00094

	6	0.9296	0.00012	0.00026	0.03009	0.00013	0.0004	0.00142	0.0002	0.00027	0.00006	0.00081	0.00009	0.00016	0.00016	0.00015	0.00134	0.00228	0.00142	
	7	0.89456	0.00007	0.00026	0.04376	0.00004	0.00025	0.00128	0.00008	0.00021	0.00002	0.00049	0.00009	0.00011	0.00007	0.00007	0.00113	0.00171	0.00132	
	8	0.67069	0.00023	0.00041	0.05979	0.00012	0.00045	0.00242	0.00009	0.00043	0.00009	0.00235	0.00024	0.00015	0.00016	0.0001	0.00605	0.00426	0.00331	
	9	0.77935	0.00038	0.00072	0.04044	0.00019	0.00053	0.00533	0.00013	0.00081	0.00012	0.00167	0.00047	0.00032	0.0002	0.00011	0.01104	0.00395	0.00286	
	10	0.77515	0.00006	0.00011	0.1369	0.00004	0.00013	0.00252	0.00008	0.00092	0.00002	0.0001	0.00009	0.0002	0.00006	0.00005	0.00225	0.00418	0.00078	
	test-6 scene A	1	0.96541	0.00001	0.00004	0.02009	0.00001	0.0001	0.00061	0.00004	0.00017	0.00001	0.00003	0.00003	0.00015	0.00002	0.00003	0.00031	0.00118	0.00024
		2	0.91846	0.00001	0.00006	0.02087	0.00001	0.00004	0.00024	0.00001	0.00013	0	0.00003	0.00002	0.00003	0.00002	0.00001	0.00039	0.00094	0.00027
		3	0.907	0.0001	0.0001	0.01661	0.00006	0.00029	0.00507	0.00005	0.00037	0.00005	0.00051	0.00007	0.00009	0.00007	0.00016	0.00436	0.00216	0.00103
		4	0.96808	0.00001	0.00004	0.0111	0.00002	0.0001	0.00039	0.00001	0.00008	0.00001	0.00004	0.00002	0.00002	0.00002	0.00001	0.00093	0.00132	0.0002
		5	0.97567	0.00002	0.00009	0.00957	0.00004	0.00016	0.00042	0.00003	0.00016	0.00001	0.00002	0.00003	0.00008	0.00002	0.00002	0.00035	0.00187	0.00024
6		0.96629	0.00001	0.00008	0.01193	0.00003	0.00006	0.00055	0.00005	0.00017	0.00002	0.00003	0.00005	0.00003	0.00002	0.00001	0.00121	0.00143	0.00018	
7		0.96277	0.00002	0.00007	0.01283	0.00002	0.00007	0.00058	0.00003	0.00017	0.00001	0.00004	0.00003	0.00004	0.00001	0.00001	0.00094	0.00179	0.00024	
8		0.96706	0.00001	0.00005	0.00594	0.00002	0.00006	0.00062	0.00002	0.00015	0.00001	0.00004	0.00002	0.00004	0.00002	0.00001	0.00117	0.00213	0.00017	
9		0.91334	0.00001	0.00007	0.0634	0.00001	0.00005	0.00018	0.00002	0.00022	0.00001	0.00003	0.00004	0.00004	0.00002	0.00001	0.00026	0.00146	0.00021	
10		0.95044	0.00001	0.00007	0.03177	0.00002	0.00008	0.00021	0.00003	0.00012	0.00001	0.00002	0.00002	0.00004	0.00002	0.00002	0.00019	0.00083	0.00023	