**HAND GESTURE RECOGNITION FOR HOME**

**APPLIANCES CONTROL**

**BY**

**ONYENEKE, ANTHONY CHIDUBEM**

**MATRICULATION NUMBER: 200353**

**A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING,**

**FACULTY OF TECHNOLOGY, UNIVERSITY OF IBADAN**

**IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE AWARD OF**

**BACHELOR OF SCIENCE DEGREE IN**

**ELECTRICAL AND ELECTRONIC ENGINEERING OF THE UNIVERSITY OF IBADAN**

**JUNE, 2023**

# CERTIFICATION

This is to certify that the project work upon which this report is based, was carried out by ONYENEKE, ANTHONY CHIDUBEM (Matriculation Number 200353), in the Department of Electrical and Electronic Engineering, Faculty of Technology, University of Ibadan.

-------------------------------------- ---------------------------------

Project Supervisor Date

Dr. O. E. Adetoyi

-------------------------------------- ---------------------------------

Head of Department Date

Dr. O.O. Olakanmi

# DEDICATION

This project is dedicated firstly to the Almighty God who has blessed me with his unending grace, love, strength and knowledge throughout its period of implementation.

To my parents, who remain my pillar of love and support as well as my project supervisor who has guided me throughout the entirety of this project.

# ACKNOWLEDGEMENT

I would like to acknowledge God for the wisdom he gave me in the course of this project. I would also like to acknowledge the guidance and support of my supervisor, Dr. O. E. Adetoyi during the course of this project. She introduced me to deep learning and from then on, I have learned a lot. I also appreciate my course adviser, Dr. Kamil, for his constant help right from 100L days till now. To all my lecturers in the department as well, who taught me most of what I know about Electrical and Electronics Engineering, I say thank you.

I want to acknowledge my loving parents for their constant love and also moral, financial, and emotional support throughout the project. Also, my siblings (Frances, Kenneth, Pascal and Helen), aunties (Chioma and Ejima) and uncles (Tochukwu and Fr. Chibueze) were all of great help and I am so grateful to each of them. Their sacrifice of love is well appreciated.

I would also like to acknowledge the plant manager of Wartsila Ewekoro powerplant, Engr. Ikenna Okpala, for the experience gained during my six-month internship with Wartsila for that played a major role in the completion of this project. I am thankful as well to the entire Wartsila and Nestle staff for making time for the numerous questions I asked which helped me acquire lots of practical knowledge used in the execution of this project.

More so, words will fail me to appreciate all my friends who contributed one way or the other to the success of this project. I also cannot but be immensely grateful to every single one of my classmates (Ohmies) and hostel mates (Irawo and Talent) for their care and support all through our journey together in the University.

Finally, I would like to appreciate Toyin, Kunle, Ehijoe, Peculiar, Joseph, Akhere, Matthew, Matthias and Raji for their time and efforts in making me understand how a Convolutional Neural Network (CNN) works and how a model can be used for control.

Thank you all for everything you did towards the successful completion of this project.

# ABSTRACT

Hand gesture recognition has emerged as a promising approach for enhancing human-computer interaction and providing seamless control over various devices. In this project, a system is proposed for hand gesture recognition specifically designed for home appliances control. The aim is to implement a convolutional neural network (CNN) model on a low power IOT device that accurately recognizes and classifies various hand gestures in real time, which can be used to control home devices.

Convolutional Neural Networks (CNNs) have demonstrated remarkable performance in image classification tasks, making them an ideal choice for hand gesture recognition. Eighteen (18) different hand gestures were gotten from Kaggle and used in training the CNN model. This dataset was carefully annotated and preprocessed to ensure high-quality training data. The trained CNN model was then optimized and deployed on a Raspberry Pi, a popular and affordable single-board computer. The Raspberry Pi serves as an edge device, enabling real-time inference on the captured video stream from a camera. The compact size and low power consumption of the Raspberry Pi make it suitable for deployment in a home environment.

A prototype was produced at the end of this project and it successfully recognizes and classifies hand gestures, allowing users to control home appliances efficiently and intuitively. Through extensive testing, the model achieved high accuracy (99.48%) and robustness, ensuring reliable appliance control even in varying lighting conditions and different hand positions.

The project's outcomes demonstrate the potential of hand gesture recognition technology for enhancing the usability and accessibility of home appliances. The developed system offers a user-friendly and hands-free control mechanism, reducing the dependency on traditional interfaces and promoting a more interactive and convenient user experience.

# TABLE OF CONTENTS

[CERTIFICATION ii](#_Toc139276670)

[DEDICATION iii](#_Toc139276671)

[ACKNOWLEDGEMENT iv](#_Toc139276672)

[ABSTRACT v](#_Toc139276673)

[TABLE OF CONTENTS vi](#_Toc139276674)

[List of Acronymns ix](#_Toc139276675)

[List of Figures xii](#_Toc139276676)

[CHAPTER 1 1](#_Toc139276677)

[INTRODUCTION 1](#_Toc139276678)

[1.1 Background 1](#_Toc139276679)

[1.2 Problem Statement 2](#_Toc139276680)

[1.3 Aim and Objectives 2](#_Toc139276681)

[1.4 Significance of Research 3](#_Toc139276682)

[1.5 Project Layout 4](#_Toc139276683)

[CHAPTER 2 5](#_Toc139276684)

[LITERATURE REVIEW 5](#_Toc139276685)

[CHAPTER 2 5](#_Toc139276686)

[2.1 Evolution of Hand Gesture Recognition 5](#_Toc139276687)

[2.2 Supervised Learning 6](#_Toc139276688)

[2.2.1 Definition 6](#_Toc139276689)

[2.2.2 Working Operation of Supervised Learning 6](#_Toc139276690)

[2.2.3 Supervised Learning Algorithms 7](#_Toc139276691)

[2.3 Review of Convolutional Neural Network 8](#_Toc139276692)

[2.3.1 Introduction 8](#_Toc139276693)

[2.3.2 How Convolutional Neural Networks Work 9](#_Toc139276694)

[2.3.3 Most Commonly used Activation Functions 13](#_Toc139276695)

[2.3.4 Filter Hyperparameters 14](#_Toc139276696)

[2.4 Related Works on Hand Gesture Recognition System using Convolutional Neural Network 15](#_Toc139276697)

[2.5 Home Automation Systems 16](#_Toc139276698)

[2.6 Need for Home Automation 17](#_Toc139276699)

[CHAPTER 3 19](#_Toc139276700)

[METHODOLOGY 19](#_Toc139276701)

[3.1 System Overview 19](#_Toc139276702)

[3.2 Training the CNN and Building the Model 20](#_Toc139276703)

[3.3 Image Data Acquisition (Data Collection) 20](#_Toc139276704)

[3.4 Pre-processing of Image Data 21](#_Toc139276705)

[3.5 Configuration of the CNN Algorithm 22](#_Toc139276706)

[3.6 System Architecture 24](#_Toc139276707)

[3.7 Raspberry Pi Configuration 25](#_Toc139276708)

[3.8 System Construction 28](#_Toc139276709)

[3.8.1 Interfacing Raspberry Pi Camera with the Raspberry Pi 3B+ 28](#_Toc139276710)

[3.8.2 Interfacing Liquid Crystal Display (LCD) with the Raspberry Pi 3B+ 28](#_Toc139276711)

[3.8.3 Interfacing Relay Module with the Raspberry Pi 3B+ 29](#_Toc139276712)

[3.9 Hand Gesture Recognition 30](#_Toc139276713)

[3.10 Control of Devices 31](#_Toc139276714)

[CHAPTER 4 33](#_Toc139276715)

[RESULTS AND DISCUSSION 33](#_Toc139276716)

[4.1 Action Recognition 33](#_Toc139276717)

[4.1.1 Accuracy and Loss 34](#_Toc139276718)

[4.1.2 Recall 37](#_Toc139276719)

[4.1.3 Precision 37](#_Toc139276720)

[4.1.4 F1-Score 38](#_Toc139276721)

[4.2 Hand Gesture Recognition for Appliance Control 39](#_Toc139276722)

[CHAPTER 5 40](#_Toc139276723)

[CONCLUSION AND RECOMMENDATION 40](#_Toc139276724)

[5.1 Conclusion 40](#_Toc139276725)

[5.2 Recommendation 40](#_Toc139276726)

[Appendices 42](#_Toc139276727)

[Appendix I 42](#_Toc139276728)

[Appendix II 46](#_Toc139276729)

[Appendix III 49](#_Toc139276730)

[Appendix IV 56](#_Toc139276731)

[Appendix V 58](#_Toc139276732)

[References 59](#_Toc139276733)

# List of Acronymns

|  |  |  |
| --- | --- | --- |
| **S/N** | **ABBREVIATIONS** | **MEANING** |
| 1. | AC | Alternating Current |
| 2. | AI | Artificial Intelligence |
| 3. | ANN | Artificial Neural Network |
| 4. | AWS | Amazon Web Services |
| 5. | COM | Common |
| 6. | CNN | Convolutional Neural Network |
| 7. | ConvNets | Convolutional Neural Networks |
| 8. | CPU | Central Processing Unit |
| 9. | CSI | Camera Serial Interface |
| 10. | DC | Direct Current |
| 11. | DL | Deep Learning |
| 12. | EMF | Electro-Magnetic Field |
| 13. | GND | Ground |
| 14. | GPIO | General Purpose Input Output |
| 15. | GPU | Graphic Processing Unit |
| 16. | HCI | Human-Computer Interaction |
| 17. | I/O | Input/output |
| 18. | I2C | Inter Integrated Circuit |
| 19. | IDE | Integrated Development Environment |
| 20. | IoT | Internet of Things |
| 21. | IP | Internet Protocol |
| 22. | KNN | K-Nearest Neighbor |
| 23. | LCD | Liquid Crystal Display |
| 24. | LED | Light Emitting Diode |
| 25. | MCU | Micro-Controller Unit |
| 26. | NB | Nota Bene |
| 27. | NC | Normally Closed |
| 28. | NO | Normally Open |
| 29. | RAM | Random Access Memory |
| 30. | ReLU | Rectified Linear Unit |
| 31. | RGB | Red Green Blue |
| 32. | RPi | Raspberry Pi |
| 33. | SCL | Serial Clock |
| 34. | SD Card | Secure Digital Card |
| 35. | SDA | Serial Data |
| 36. | SSH | Secure Shell |
| 37. | SVM | Support Vector Machines |
| 38. | VCC | Voltage Common Collector |
| 39. | ML | Machine Learning |
| 40. | ROI | Region of Interest |

# List of Figures

[Figure 2.1: Chinese sign language using different hand gestures 6](#_Toc135093802)

[Figure 2.2: Convolutional Neural Network Working Operation 10](#_Toc135093803)

[Figure 2.3: Convolution Process 11](#_Toc135093804)

[Figure 2.4: Demonstration of Convolution Process using Values 11](#_Toc135093805)

[Figure 2.5: Max Pooling Operation 12](#_Toc135093806)

[Figure 2.6: Average Pooling Operation 12](#_Toc135093807)

[Figure 2.7: Demonstration of Max Pooling Operation using Values 12](#_Toc135093808)

[Figure 2.8: Fully Connected Layer 13](#_Toc135093809)

[Figure 2.9: Function g(z) for the ReLU Activation Function 13](#_Toc135093810)

[Figure 2.10: Dimensions of a filter 14](#_Toc135093811)

[Figure 2.11: Stride Operation 14](#_Toc135093812)

[Figure 2.12: Zero Padding Modes 15](file:///C:\Users\Hi\Desktop\Onyeneke%20Anthony%20Project.docx#_Toc135093813)

[Figure 2.13: An Automated Home 17](#_Toc135093814)

[Figure 3.1: Block Diagram of the Hand Gesture Recognition System 19](#_Toc135093817)

[Figure 3.2: Training phase procedures 20](#_Toc135093818)

[Figure 3.3: Kaggle Hand Gesture Recognition Dataset 21](#_Toc135093819)

[Figure 3.4: Pre-processing Steps 22](#_Toc135093820)

[Figure 3.5: Training of the model after CNN Configuration 23](#_Toc135093821)

[Figure 3.6: Summary of Model Architecture 24](#_Toc135093822)

[Figure 3.7: Summary of Model Architecture 25](#_Toc135093823)

[Figure 3.8: wpa\_supplicant file for SSH Configuration 26](#_Toc135093824)

[Figure 3.9: Putty Setup for SSH with Raspberry Pi 26](#_Toc135093825)

[Figure 3.10: Login onto Putty Terminal 27](#_Toc135093826)

[Figure 3.11: Putty Terminal 27](#_Toc135093827)

[Figure 3.12: Interfacing Raspberry Pi with Pi Camera via the CSI 28](#_Toc135093828)

[Figure 3.13: Interfacing Raspberry Pi with LCD using an I2C 29](#_Toc135093829)

[Figure 3.14: Interfacing Raspberry Pi with Relay Module via Level Shifter 29](#_Toc135093830)

[Figure 3.15: Flowchart of the Hand Gesture Recognition System 30](#_Toc135093831)

[Figure 3.16: The Hand Gesture Recognition Detection Code 31](#_Toc135093832)

[Figure 3.17: Hand Gestures used for the Prototype 32](#_Toc135093833)

[Figure 3.18: Circuit Diagram for the Circuit of the Hand Gesture Recognition System 32](#_Toc135093834)

[Figure 4.1: Accuracy Score versus Number of Epochs 35](#_Toc135093835)

[Figure 4.2: Loss Score versus Number of Epochs 35](#_Toc135093836)

[Figure 4.3: Confusion Matrix 36](#_Toc135093837)

[Figure 4.4: Classification Report for each Class 36](#_Toc135093838)

[Figure 4.5: Recall versus Hand Gesture Classes 37](#_Toc135093839)

[Figure 4.6: Precision versus Hand Gesture Classes 38](#_Toc135093840)

[Figure 4.7: F1-Score versus Hand Gesture Classes 38](#_Toc135093841)

[Figure 4.8: System recognizing gesture labelled as ‘1’ 39](#_Toc135093842)

[Figure 4.9: System recognizing gesture labelled as ‘5’ 39](#_Toc135093843)

# 

# INTRODUCTION

## Background

The use of convolutional neural network (CNN) in hand gesture recognition for device control in home automation is a rapidly growing area of research, with significant potential to improve the way we interact with technology in our homes. The development of CNN-based hand gesture recognition systems has been driven by the rapid growth of the Internet of Things (IoT), which has led to an increasing number of devices that can be controlled using hand gestures.

Hand gesture recognition systems are employed in a variety of applications, such as the following: human-computer interaction (HCI), medical operation, gesture-based gaming control, control of home appliances, gesture control car driving, communication.

Traditionally, the control of devices in home automation has relied on physical buttons, switches, and remote controls. However, these methods can be cumbersome and inconvenient, requiring the user to locate and manipulate a physical device in order to control the technology in their home. In contrast, hand gesture recognition using CNNs has the potential to provide a more natural and intuitive way to interact with technology in our homes.

The use of CNNs for hand gesture recognition has been extensively studied in a variety of contexts, including computer vision, robotics, and human-computer interaction. However, the application of CNNs to hand gesture recognition for device control in home automation is a relatively new and emerging field. However, there are several challenges associated with using CNNs for hand gesture recognition.

## Problem Statement

One of the main challenges is the high variability of hand gestures and the difficulty in accurately detecting and classifying them. Hand gestures can vary greatly in terms of shape, size, orientation, and movement, making it difficult for CNNs to accurately recognize and interpret them. In addition, the presence of other objects or backgrounds in the scene can interfere with the accuracy of hand gesture recognition.

Another challenge is the need for real-time processing of hand gestures in order to provide a responsive and seamless user experience. In home automation, it is important for hand gestures to be recognized and interpreted quickly in order to control devices in a timely manner. This requires efficient and effective algorithms for recognition of hand gestures using CNN.

Furthermore, the implementation of hand gesture recognition using CNN in home automation systems must take into account the limited computational resources and power constraints of IoT devices. This requires the development of lightweight and efficient CNN architectures that can run on low-power devices without sacrificing accuracy or performance.

In conclusion, the use of CNN for hand gesture recognition in home automation presents several challenges and opportunities. Further research is needed to address the challenges and develop effective and efficient algorithms for hand gesture recognition using CNNs in home automation systems.

## Aim and Objectives

The aim of this project is to implement a convolutional neural network (CNN) on a low power IOT device that accurately recognizes and classifies various hand gestures in real time, which can be used to control home devices.

The realization of this project aim will be attained through these objectives:

1. comparison of CNN architectures to select model with high accuracy and low computational requirement that can detect and classify a wide range of hand gestures in real time;
2. datasets of selected hand gestures will be acquired;
3. evaluation of the performance of the hand gesture recognition system in terms of accuracy, speed, and robustness to variations in hand gestures and background noise;
4. to design and implement a user-friendly interface for the hand gesture recognition system that is easy to use and intuitive. The hand gesture recognition system would be designed using the following python libraries on the Spyder IDE: keras, skleran, OpenCV, numpy, pandas and matplotlib;
5. to integrate the hand gesture recognition system with a home automation system to enable the control of devices using hand gestures. The home automation system would be designed in the following stages as:
6. deployment of model on the raspberry pi;
7. interfacing of camera, liquid crystal display (LCD) and relay module with the controller;
8. infinite looping of hand gesture recognition code on the controller so that the system would keep checking for gestures placed in front of it;
9. connection of socket boxes to the relay module. The socket boxes will provide for manual control of devices while the home automation system would provide for automatic control of devices;
10. document the project methodology, results and conclusions in a final report.

## Significance of Research

By enabling the control of devices using hand gestures, CNN-based hand gesture recognition has the potential to provide a more intuitive and natural user experience, making it easier and more convenient to control the technology in our homes.

One of the main benefits of using CNNs for hand gesture recognition in home automation is the detection and classification of wide ranges of hand gestures with high accuracy. This can enable the control of a wide range of devices using hand gestures, providing a more versatile and flexible user experience. In addition, CNNs are able to process hand gestures in real-time, providing a responsive and seamless user experience.

Another important benefit of using CNNs for hand gesture recognition in home automation is the ability to develop lightweight and efficient algorithms that can run on low-power devices. This is particularly important in the context of the Internet of Things (IoT), where many home automation devices have limited computational resources and power constraints. By using CNNs, it is possible to develop hand gesture recognition systems that can run on low-power IoT devices without sacrificing accuracy or performance.

Overall, the use of convolutional neural network (CNN) for recognition of hand gestures in home automation has the potential to provide a more intuitive, natural, and versatile way to control devices in our homes. This has significant implications for the future of home automation and the way we interact with technology in our daily lives.

## Project Layout

There are five chapters in this project, which are further broken down into sections. Chapter one gives an introduction to the project. It outlines the aim and objectives as well as the problem to be solved. The project's literature review is presented in Chapter two. A thorough review of earlier works that are related to this project is done in this chapter. Chapter three contains the methodology with which this project would be carried out. The experimental process, and results are all discussed in depth in chapter four. The conclusion and suggestions for future works are presented in chapter five.

# 

# LITERATURE REVIEW



## Evolution of Hand Gesture Recognition

Understanding the evolution of the video camera and how it has changed and been applied to various applications requires a solid understanding of its origins. The development of glove-based control interfaces marked the beginning of the history of hand gesture recognition for computer control (Premaratne, 2014). Researchers came to the conclusion that sign language-inspired motions may be used to provide clear instructions for a computer interface. This subsequently changed as more precise infrared cameras, accelerometers and even fiberoptic bend sensors were developed (optical goniometers). Eventually, some of those advancements in glove-based systems made it possible to implement computer vision-based recognition without any sensors on the glove. Hand gesture recognition has progressed from 2D to 3D applications during the past 25 years. Data collection, feature extraction, and hand gesture extraction form the foundation of the 2D tracking technique. The third dimension has been added to hand gesture recognition, which is no longer restricted to physical or digital surfaces. As processing power, camera performance, and computer-vision-style learning algorithms have advanced quickly in recent years to support 3D applications, this issue has attracted significant interest and research. There are two methods through which a machine can identify a hand gesture (Sonia Raheja, 2011):

1. Hand detection techniques: It contains methods for locating hands in captured images following pre-processing, including appearance-based and model-based methods and;
2. Soft computing techniques:  Neural Networks, Fuzzy Systems, Machine Learning, Evolutionary Computation, Probabilistic Reasoning, and their hybrid methods are the main components.

As indicated in Figure 2.1, the user may utilize hand motions that are in the form of Chinese sign language, sign language for the physically challenged, or any other sign language that has been pre-defined in the system by the manufacturer.

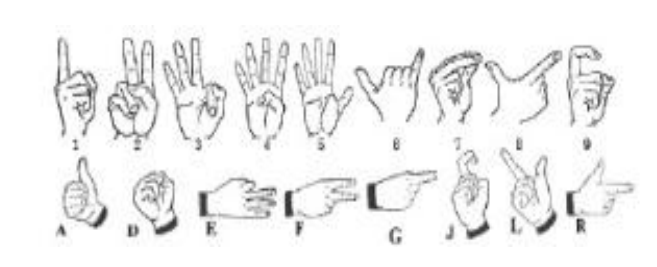


Figure 2.1: Chinese sign language using different hand gestures

## Supervised Learning

### Definition

Supervised learning is a branch of machine learning (ML) and artificial intelligence (AI). It is also known as supervised machine learning. It is distinguished by the way it trains computers to accurately classify data or predict outcomes using labeled datasets. The model modifies its weights as input data is fed into it until the model has been properly fitted, which takes place as part of the cross-validation process. Supervised learning assists enterprises in finding scalable solutions to a number of real-world issues. Applications of supervised learning are in classification of types of flowers, classification of hand gestures etc.

### Working Operation of Supervised Learning

A training set is used in supervised learning to train models to achieve the expected results. This training dataset has both the correct inputs and outputs, enabling the model to develop over time. The loss function serves as a gauge for the algorithm's accuracy, and iterations are made until the error is sufficiently reduced. When using data mining, supervised learning may be divided into two categories of issues: regression and classification.

1. Regression: To comprehend how dependent and independent variables are related, this is used. It is frequently used to produce estimates, including those for a company's sales revenue. Popular regression algorithms include linear regression, logistical regression, and polynomial regression.
2. Classification: To accurately classify test data into different categories, this uses an algorithm. It identifies particular entities in the dataset and makes an effort to determine how those items should be classified or labeled. Linear classifiers, decision trees, k-nearest neighbor, support vector machines (SVM) and random forests are examples of common classification algorithms.

### Supervised Learning Algorithms

Supervised machine learning uses a variety of computation methods and algorithms. The most popular learning techniques are briefly explained below, often calculated using software like R or Python:

1. Naive Bayes: A classification method known as Naive Bayes adopts the idea of Class Conditional Independence from the Bayes Theorem. This means that each predictor has an equal impact on the outcome and that the existence of one feature does not affect the presence of another in the probability of a certain result. Multinomial Nave Bayes, Bernoulli Nave Bayes, and Gaussian Nave Bayes are the three different varieties of Nave Bayes classifiers. This method is mostly applied in spam detection, text classification, and recommendation systems.
2. K-Nearest Neighbor: The KNN algorithm, also referred to as K-nearest neighbor, is a non-parametric algorithm that groups data points according to their proximity and correlation with other pieces of accessible information. This approach makes the assumption that related data points can be discovered close to one another. It then assigns a category based on the most prevalent category or average after attempting to determine the distance between data points, typically by Euclidean distance. Data scientists favor it because of how simple it is to use and how quickly calculations are completed, but as test datasets get larger, processing times are longer, which makes it less desirable for classification jobs. KNN is frequently employed in image recognition and recommendation systems.
3. Neural Network: Neural networks handle training data by simulating the connectivity of the human brain using layers of nodes, which is mostly used for deep learning algorithms. Inputs, weights, a bias (or threshold), and an output make up each node. This "fires" or activates the node, sending data to the following layer in the network, if the output value exceeds a predetermined threshold. This mapping function is learned by neural networks through supervised learning, with gradient descent adjustments made in response to the loss function. We can be sure in the model's accuracy to produce the right answer when the cost function is at or close to zero.
4. Support Vector Machine (SVM): Vladimir Vapnik created the well-known supervised learning model known as the support vector machine, which is used for both data classification and regression. The distance between two classes of data points is at its greatest point on a hyperplane, which is how it is often used to solve classification problems. The decision boundary is a hyperplane that divides the classifications of data points (such as apples versus oranges) on either side of the plane.
5. Random Forest: Another adaptable supervised machine learning technique, random forest is utilized for both classification and regression. The "forest" refers to a set of independent decision trees that are combined to lower variation and produce more precise data predictions.
6. Linear Regression: In order to predict future outcomes, linear regression is frequently employed to determine the relationship between a dependent variable and one or more independent variables. Simple linear regression is used when there is only one independent variable and one dependent variable. It is called multiple linear regression as the number of independent variables rises. It attempts to plot a line of best fit for each type of linear regression, which is determined using the least squares method. When shown on a graph, this line is straight in contrast to other regression models.
7. Logistic Regression: While logistic regression is used when the dependent variable is categorical, or has binary outputs, such as "true" and "false" or "yes" and "no," linear regression is used when the dependent variable is continuous. Despite the fact that both regression models aim to identify the relationships between the data inputs, logistic regression is mostly employed to address binary classification issues, such as spam identification.

## Review of Convolutional Neural Network

### Introduction

Neural networks are a subset of machine learning and are at the core of deep learning algorithms. They are made up of node layers, each of which includes an input layer, one or more hidden layers, and an output layer. Each node has a threshold and weight that are connected to one another. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next tier.

Different neural network types are employed for diverse applications and data types. Recurrent neural networks, for instance, are frequently used for speech and natural language processing, whereas convolutional neural networks (also known as CNNs or ConvNets) are more frequently employed for classification and computer vision applications. Before CNNs, identifying objects in images required the use of laborious, manual feature extraction techniques. Convolutional neural networks, on the other hand, now offer a more scalable method for classifying images and recognizing objects by using matrix multiplication and other concepts from linear algebra to find patterns in images. However, they can be computationally taxing, necessitating the use of graphics processing units (GPUs) when modeling them.

### How Convolutional Neural Networks Work

Convolutional neural networks outperform other neural networks when given inputs such as images, voice, or audio. There are three main categories of layers in CNNs and they are as follows:

1. Convolutional Layer
2. Pooling Layer
3. Fully Connected Layer

A convolutional neural network's first layer is the convolutional layer. The fully-connected layer is the last layer, although the convolutional layers can be followed by several other convolutional layers or pooling layers. The CNN becomes more complicated with each layer, detecting larger areas of the image. Early layers emphasize basic elements like colors and borders. The larger features or shapes of the object are first recognized when the visual data moves through the CNN layers, and eventually the intended object is recognized.

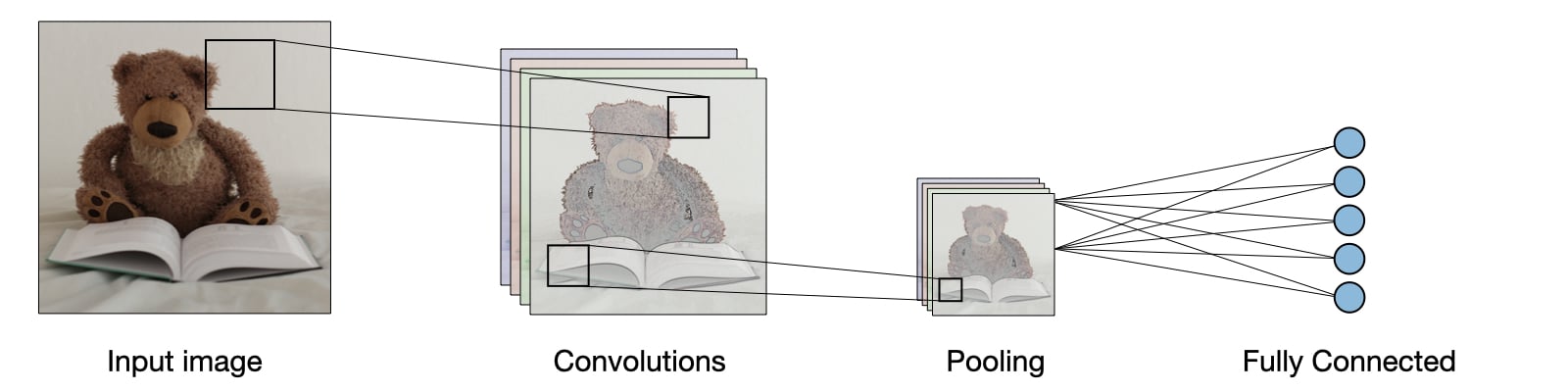


Figure 2.2: Convolutional Neural Network Working Operation

1. Convolutional Layer

The central component of a CNN is the convolutional layer, which is also where the majority of computation takes place. It needs input data, a filter, and a feature map, among other things. The input will be a color image that is composed of a 3D pixel matrix. As a result, the input has three dimensions—height, width, and depth—that are analogous to RGB in an image. Additionally, a feature detector, also referred to as a kernel or filter is present and will move through the image's receptive fields and determine whether the feature is there. The process above is referred to as Convolution. A two-dimensional array of weights serving as the feature detector represents a portion of the image. The filter size, which also controls the size of the receptive field, is normally a 3x3 matrix, however they can vary in size. Following the application of the filter to a portion of the image, the dot product between the input pixels and the filter is determined. The output array is then fed with this dot product. Once the kernel has swept through the entire image, the filter shifts by a stride and repeats the operation. An activation map or feature map is the ultimate result of the series of dot products from the input and the filter. A CNN adds nonlinearity to the model by applying an activation function, such as Rectified Linear Unit (ReLU), to the feature map following each convolution process. As was previously noted, a second convolution layer may come after the first. When this occurs, the CNN's structure may become hierarchical because the later layers will be able to view the pixels in the earlier layers' receptive fields. Figure 2.3 demonstrates the convolution process:

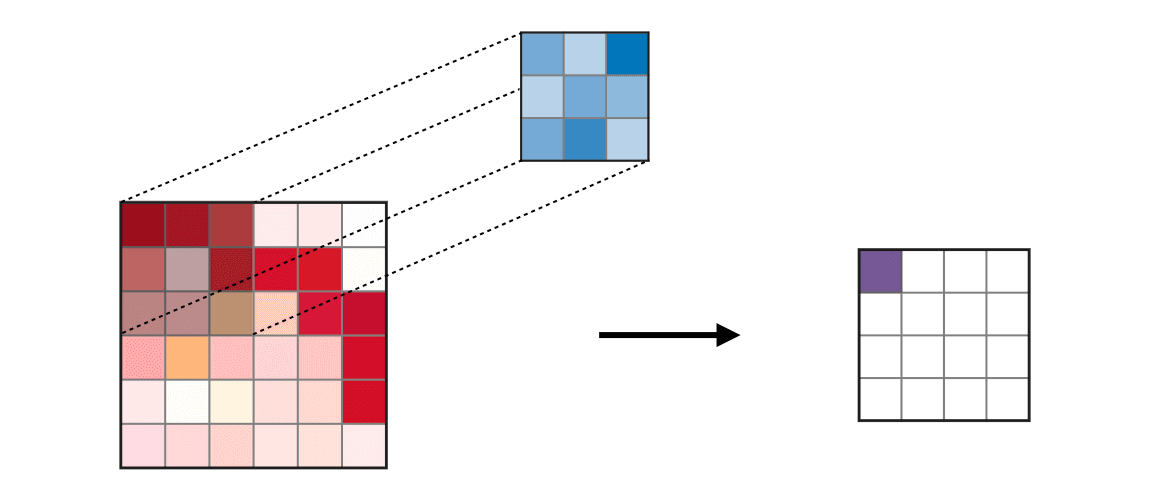


Figure 2.3: Convolution Process

Figure 2.4 uses a 5X5 input matrix 3X3 filter matrix filled with values to demonstrate how the convolutional layer operates.

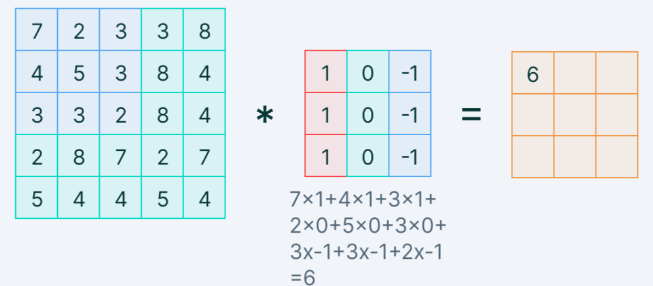


Figure 2.4: Demonstration of Convolution Process using Values

2. Pooling Layer

Down-sampling, sometimes referred to as pooling layers, carries out dimensionality reduction and lowers the number of parameters in the input. The pooling operation sweeps a filter across the entire input similarly to the convolutional layer, with the exception that this filter lacks weights. Instead, the kernel populates the output array by applying an aggregation function to the values in the receptive field. There are essentially two types of pooling:

1. Max pooling: The filter selects the pixel with the highest value to send to the output array as it advances across the input. As a side note, this method is applied more frequently than average pooling.

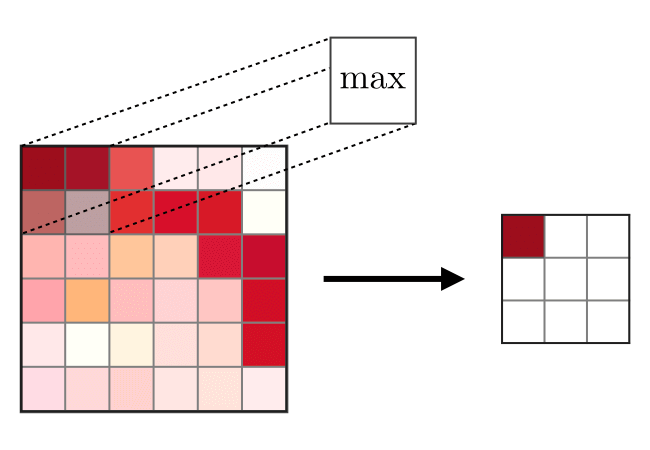


Figure 2.5: Max Pooling Operation

1. Average pooling: The filter calculates the average value inside the receptive field as it passes across the input and sends that value to the output array.

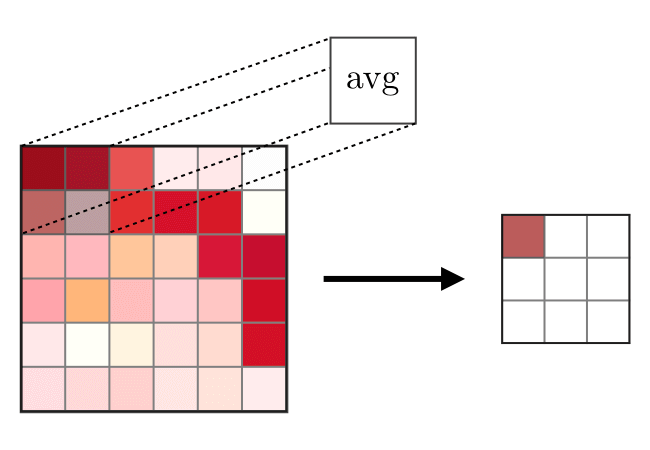


Figure 2.6: Average Pooling Operation

The pooling layer loses a lot of information, but it also offers the CNN a number of advantages. They aid in lessening complexity, increase effectiveness, and reduce the likelihood of overfitting. Figure 2.7 shows a proper example of how a max pooling layer operates:



Figure 2.7: Demonstration of Max Pooling Operation using Values

3. Fully Connected Layer

The full-connected layer is exactly what its name implies. In partially connected layers, pixel values from the input image are not connected directly to the output layer. In contrast, every node in the output layer of the fully-connected layer is directly connected to a node in the layer above it. Based on the features that were retrieved from the preceding layers and their various filters, this layer conducts the classification operation. Fully connected layers typically employ classification activation functions like the SoftMax activation function to categorize inputs appropriately, producing a probability ranging from 0 to 1. Convolutional and pooling layers typically use activation functions like the sigmoid and ReLU.

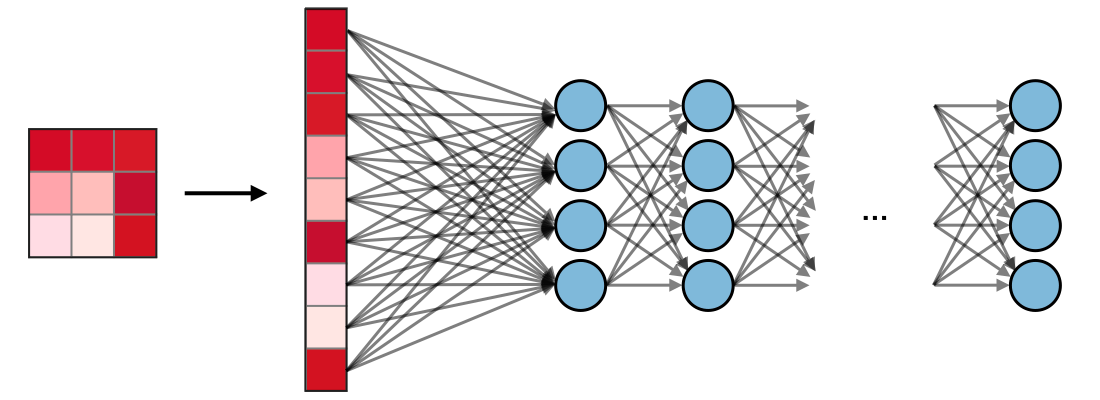


Figure 2.8: Fully Connected Layer

### Most Commonly used Activation Functions

1. Rectified Linear Unit (ReLU): All elements of the volume are activated using a function g called the rectified linear unit layer (ReLU). It seeks to give the network non-linearities. The function is defined as follows:

Equation (2.1)

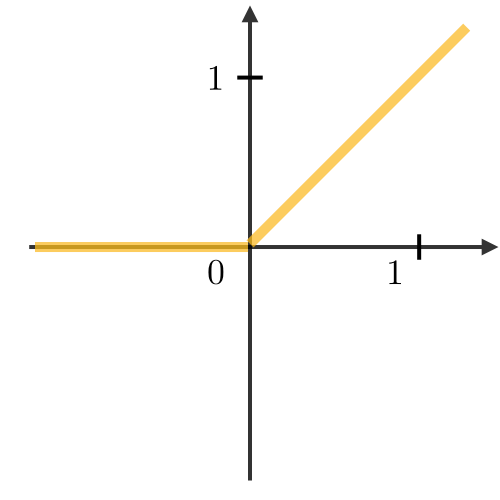


Figure 2.9: Function g(z) for the ReLU Activation Function

1. Softmax: The softmax step can be thought of as a generalized logistic function that receives as input a vector of scores (x ∈ Rn) and produces (through a softmax function at the end of the architecture) a vector of output probability (p ∈ Rn). This is how it is explained:

Equation (2.2)

where

𝜎= Softmax

z =input vector

𝑒𝑧𝑖 =standard exponential function for input class

k = number of classes in multiclass classification

𝑒𝑧𝑗 = standard exponential function for output class

The Softmax layer must have the same number of nodes as the output layer.

### Filter Hyperparameters

Knowing the significance of the filters in the convolution layer's hyperparameters is very needed. Below are 3 essential filter hyperparameters:

1. Dimensions of a filter:A filter of size length, F × breadth, F, applied to an input containing C channels represents a F×F×C volume that performs convolutions on an input of size I×I×C and outputs an activation map of size O×O×1.

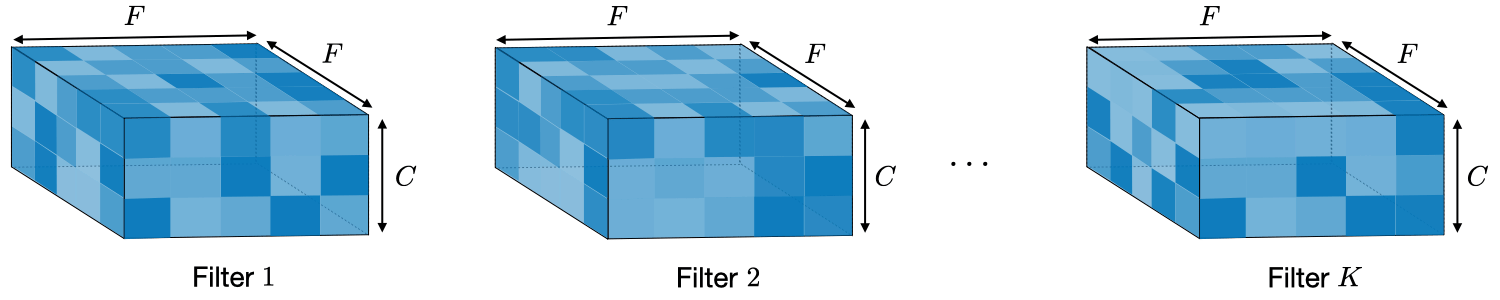


Figure 2.10: Dimensions of a filter

1. Stride: The stride S indicates how many pixels the window advances after each operation for a convolutional or pooling process.

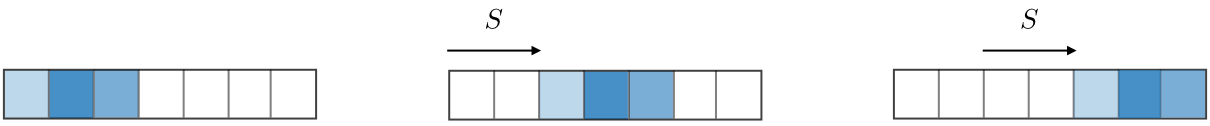
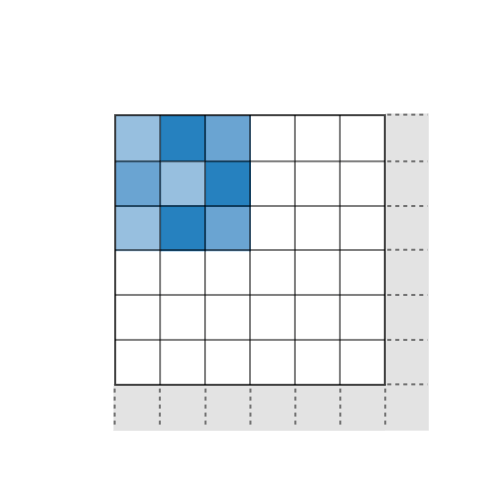
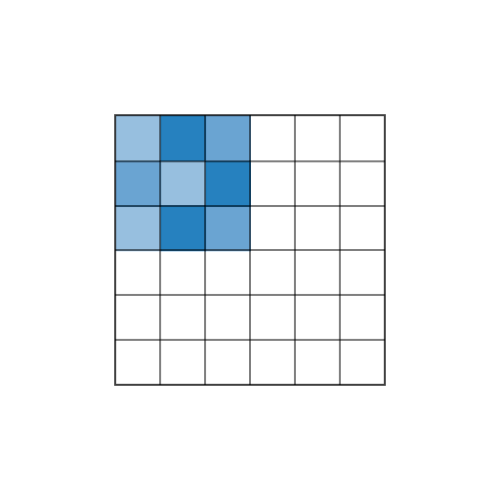


Figure 2.11: Stride Operation

1. Zero-padding: Zero-padding is the practice of padding input borders with P zeroes on each side. One of the three modes in Figure 2.12 can either manually provide this value or set it automatically. Valid, same, and full zero padding modes are represented in (a), (b) and (c) respectively:

(a) (b) (c)

Figure 2.12: Zero Padding Modes



## Related Works on Hand Gesture Recognition System using Convolutional Neural Network

Several innovations concerning use of hand gesture as a medium of interaction with a computer have been developed. In this project, slight improvement has been made with respect to the related works which mostly make use of tedious processes where all the possible features may not be extracted. These related works are as follows:

(Stergiopoulou, 2009) developed the use of artificial neural network (ANN) in hand gesture recognition. For this system, a color segmentation approach was applied in hand detection after filtering. Extraction of the fingers and hand shape and orientation features was then carried out and passed to the ANN. This method achieved an accuracy of 94.05%.

(N. D. Georganas, 2008) proposed the use of haar-like features in gesture detection. Adaboost was the algorithm used to learn the model. The project consists of two levels, a higher level where gestures were detected and a lower one where posture detection was carried out. For the higher level where gesture detection was carried out, a speculative context-free grammar is used. For each input, according to the grammar, a terminal string is generated. Rules are calculated and associated probabilities are generated for each rule. When selecting the rule, the rule with the highest likelihood for the provided string was chosen. Different gestures are associated with different rules hence the gesture associated to the selected rule was returned as the input gesture. This achieved an accuracy of 71.48%.

(H. Huynh, 2013) also went on to develop gesture detection using the ANN. Skin colors were the focal point in the segmentation of images for this system. ANN’s selected features were variation of pixels through scalar description, boundary and cross sections. For training of the ANN, the developed feature vectors were applied. They achieved an accuracy of 98%.

The three extraction processes above are quite tedious and there is a 50% chance that all possible features are not extracted. Therefore, the use of CNN for extraction was brought about as it is a form of automated feature engineering that is less tedious than the other processes above. (S. Gupta, 2015) proposed an algorithm that uses a 3D CNN for hand gesture recognition. Image intensity and challenging depth were the basis of recognition for this system. Using a VIVA dataset, they achieved an accuracy of 77.5%.

(G. Cutipa, 2017) also proposed the use of CNN in gesture recognition using the five invariants scale: noise, translation, background, illumination and rotation. LSP dataset were used in this system and an accuracy of 96.2% was achieved.

## Home Automation Systems

The term "home automation" describes the automatic and electronic management of functions, activities, and equipment in a home. Additionally, one can simply control the appliances and features in the house via the internet to improve convenience, increase security, and even reduce household expenses. It is a network of hardware, communication, and electronic interfaces that enables the Internet-based integration of commonplace devices. Whether you're at home or thousands of miles away, you can control each gadget from your smartphone or tablet because they all have sensors and WIFI connectivity. No matter where you are, you may use this to turn on the lights, turn on the fridge etc. Home automation is made up of three major components: sensors, controllers, and actuators. Sensors monitor variations in temperature, sunshine, or motion such as hand gestures. Then, according to one’s preferences, home automation systems can change those settings and more. Computers, tablets, and smartphones that are used to send and receive messages regarding the status of automated systems in one’s house are referred to as controllers. The actual mechanism or function of a home automation system is controlled by actuators, which can be light switches, motors, or relays as in the case of this project. They are set up to be activated by a controller's remote instruction.

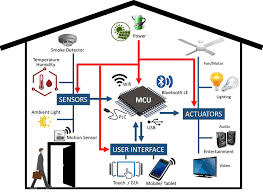


Figure 2.13: An Automated Home

## Need for Home Automation

Systems for home automation provide a range of services and capabilities. The following are some of the most typical features offered by these platforms: monitoring for fires and carbon monoxide, remote control of the lights, thermostat management, appliance management, cameras and home automation security, live video monitoring, alarm mechanisms, alerts through text and email in real-time, integration of a digital personal assistant, no-key entry, voice command activation

A home automation system's main objective is to simplify household operations. Here are a few advantages:

1. Remote access: Using a laptop or smartphone, one may manage his/her house from a distance.
2. Comfort: Making one's house more livable and comfortable through the use of home automation. For instance, adjusting the thermostat to chosen settings in advance to ensure that one’s house is constantly at a pleasant temperature.
3. Convenience: Schedule the automatic activation of devices at specific times, or access their settings remotely from any location with an Internet connection. More essential things can be focused on when one doesn't have to remember to lock the door or turn out the lights.
4. Increased safety: Home automation security features like pressure sensors, carbon monoxide monitors, and smart fire detectors can help shield one's house from tragedy.
5. Energy conservation: Home automation enables more awareness towards power consumption. For instance, reducing the amount of time that lights are left on or lowering the temperature in a room, when empty, to reduce energy usage.

# 

# METHODOLOGY

## System Overview

In this project, the aim was to design a device that detects hand gestures using a raspberry pi which would then interpret the control action to be taken. The raspberry pi observed this through a pi camera module interfaced to it. A CNN model was first trained to recognize certain hand gestures. Figure 3.2 displays the processes carried out in training the model. The CNN model was run by a raspberry pi 3B+ device which then carried out a control action via its GPIO ports as shown in Figure 3.1. The CNN model was then deployed on the raspberry pi. This The overall activity of this system is further explained by figure 3.1.

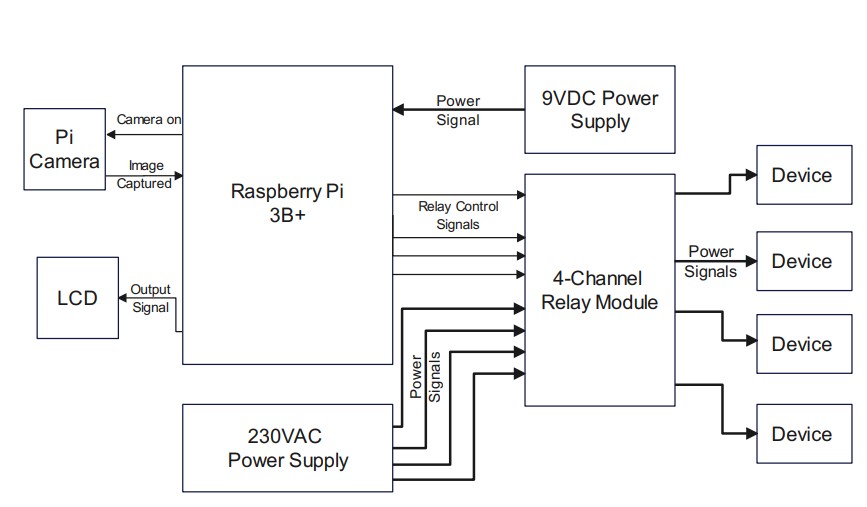


Figure 3.1: Block Diagram of the Hand Gesture Recognition System

## Training the CNN and Building the Model

For the system implementation, the high-level programming python language was used. This programming language possesses sets of different libraries, which were applied in the building of the CNN model, performing image processing and confusion matrix calculation. The IDE used was Spyder. The CNN classifier was created using the Keras library. The confusion matrix was calculated using Sklearn. Image processing was done using the OpenCV library. Array operations was performed using NumPy. The confusion matrix, model accuracy, and loss values were visualized using Matplotlib. The model was trained on a laptop before deployment on the raspberry pi took place. Figure 3.2 illustrates a flowchart of the steps from image acquisition to the training of the CNN model.



Figure 3.2: Training phase procedures

## Image Data Acquisition (Data Collection)

This project made use of the Kaggle hand gesture recognition dataset. There are 21,600 images in this collection with 18 different gestures as shown in Figure 3.3. Typically, 25% of the dataset were used for validating while the majority 75% were used in training. Consequently, there are 900 training and validating images and 300 testing images for each gesture. From Figure 3.3, it is observed that the hand gestures display different hand signs which can be used as forms of communication. Hand signs such as palm opened, palm closed, thumbs up, fist, zero, four, peace and lots of other hand signs. These hand signs represent different labels which the processor will use to identify and classify what type of label a particular hand gesture signifies. These hand signs represent different labels and they are seen in figure 3.3:

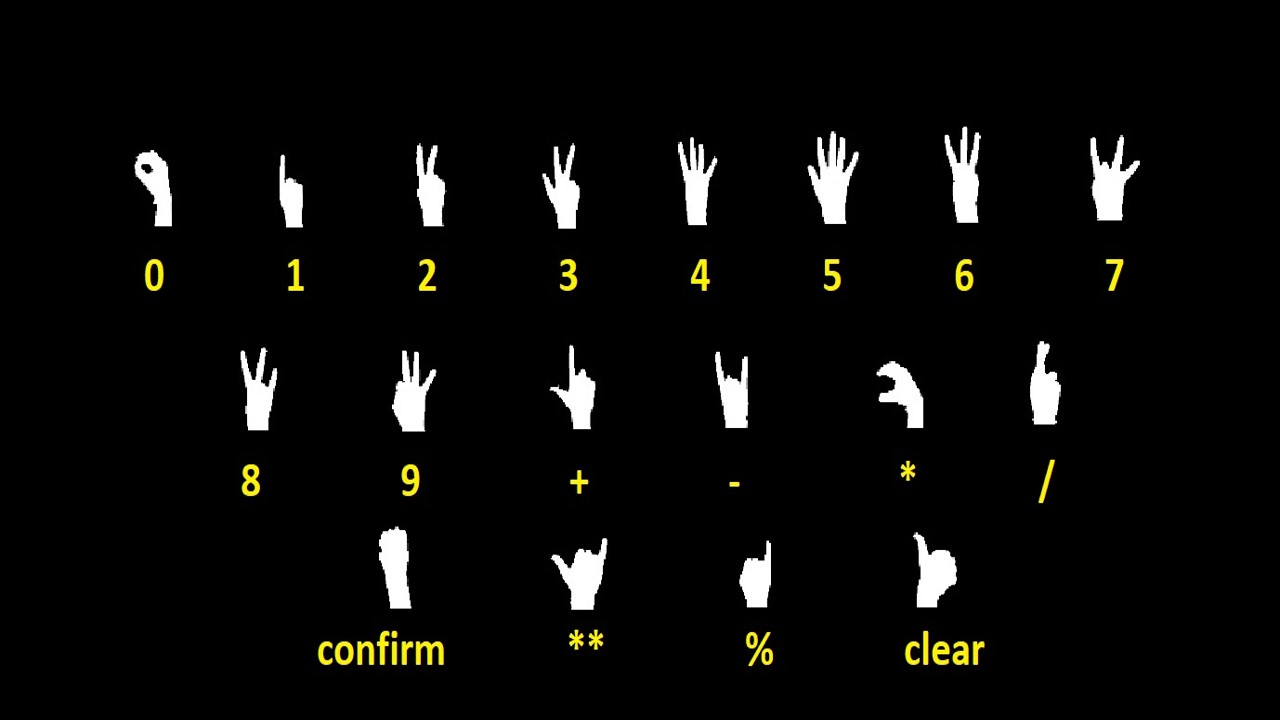


Figure 3.3: Kaggle Hand Gesture Recognition Dataset

## Pre-processing of Image Data

This process was applied to reduce the image processing complexity and also achieve more accuracy and efficiency of the system. In this process, different methods were applied to achieve possible maximum efficiency of the system. Firstly, image background subtraction was applied to the image to allow the system display only the hand’s image. The K-gaussian distribution, which chooses the proper gaussian distribution for each pixel and offers a higher flexibility on various images because of light variations, is the basic foundation on which the background subtraction is based. Only the image of the hand was left once the background had been removed. Grayscale conversion of the image was then done for easier training of the CNN using only one color. The image was then sharpened using morphological erosion filter where the darkest value in a neighborhood is selected. Noise reduction was also carried out using median filter. Resizing of the image to size 64x64 was then done before feeding it to the CNN. The images in the Kaggle dataset passed through these pre-processing steps hence the images can be fed straight into the CNN model after the image data acquisition process.



(a) (b) (c)

Figure 3.4: Pre-processing Steps: (a) Unprocessed image (b) Image after background subtraction (c) Processed grayscale image

## Configuration of the CNN Algorithm

The CNN that was used in this system comprised of 3 major layers which had various sub-layers performing very important roles for the optimum efficiency of the system. These 3 major layers are the input layer in which an activation function is applied, the process layer and an output layer. There are 4 convolutional layers, 2 max-pooling layers, 2 dropout layers, a flattening layer, 2 dense layers, and the output layers that make up the convolutional neural network used in this project to recognize hand gestures. To prevent over-fitting, the network possessed a dropout performance.

With a kernel size of 3x3 and stride of (1,1), the 1st convolutional layer included 32 distinct filters. Rectified Linear Unit (ReLU) was the activation function that was employed in this 1st layer. The 2nd convolutional layer included 64 distinct filters, kernel size of (3,3) and stride of (2,2). These were used to introduce non-linearity as it has been demonstrated that ReLU outperforms other activation functions like sigmoid or tanh in terms of performance. The activation exists at zero threshold. ReLU can be vulnerable during training, although this vulnerability can be mitigated by setting a proper learning rate. The input size must be specified because it is an input layer. The padding mode was set to same. Given that the input shape is 64x64x1, this network will need a grayscale image with a 64x64 size.

The feature maps will be created by this layer and transferred to the following layer. The CNN then has a max pooling layer with a pool size of 2x2 that pulls the maximum value out of a window with a size of 2x2. As the pooling layer only keeps the largest value and discards the remainder, the spatial size of the representation gradually shrinks. The network can better understand the images because of this layer's selection of only the most crucial features. A dropout layer of 0.15 was then applied.

A third convolutional layer follows, with a kernel size of 3x3 and a stride (2, 2) of 64 distinct filters. The last convolutional layer which consisted of kernel size of 3x3 and a preset stride (1,1) of 128 distinct filters was then applied. The activation function used in these layers were also ReLU. Another max pooling layer with a pooling size of 2x2 followed this layer. To prevent the model from over-fitting, the second dropout was then placed after these layers so as to randomly discard 25% of the total number of neurons. The flattening layer receives the output from the preceding layers and flatten it into a vector from a two-dimensional matrix. This layer enables processing of the up to this point-achieved data by the completely connected layers. The activation function for the subsequent layer, the dense layer with 96 nodes, is ReLU.

Finally, 18 nodes, one for each kind of hand gestures, is present in the output layer. The activation function for this layer is the Softmax function, which produces a probabilistic value for each of the classes.

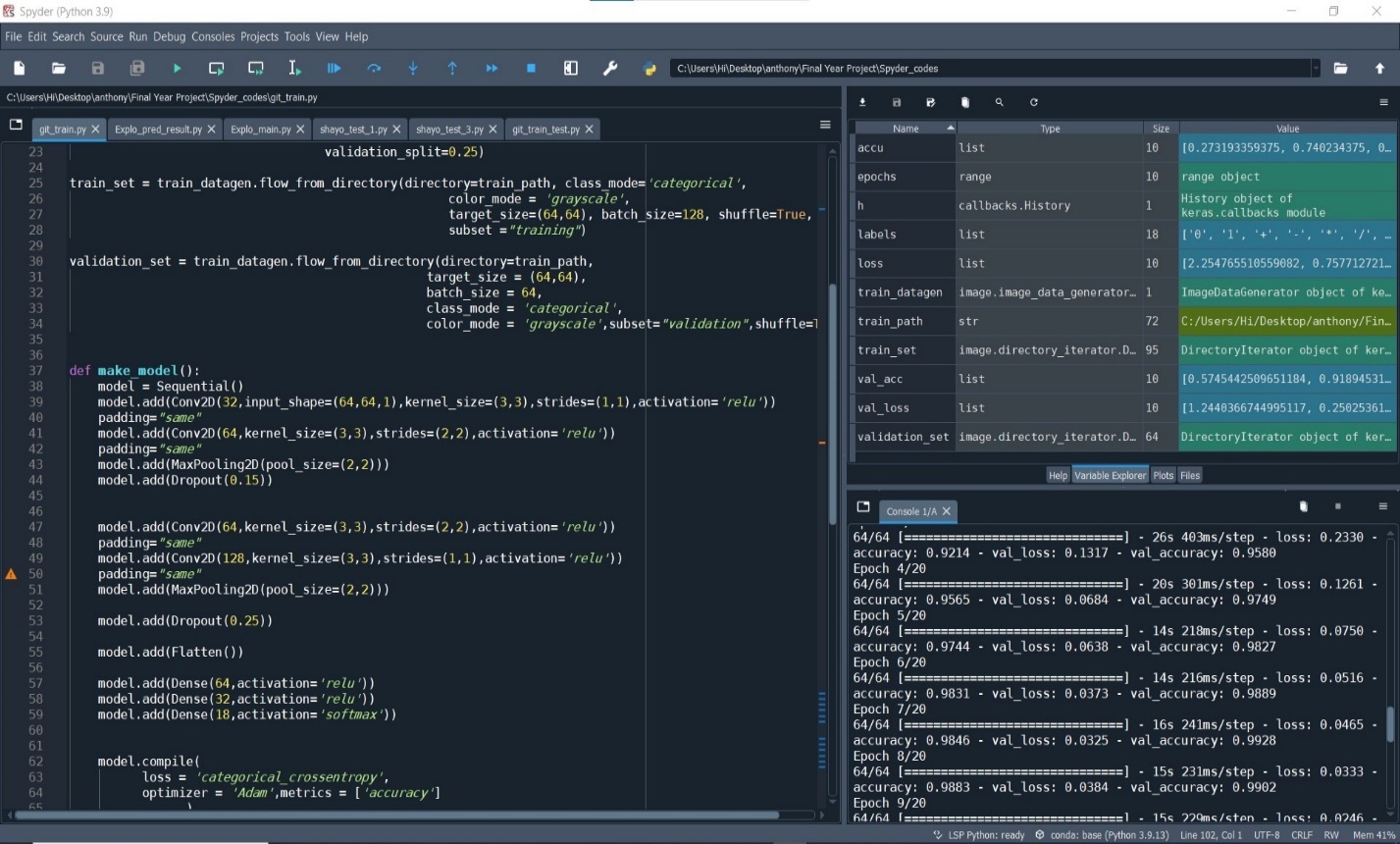


Figure 3.5: Training of the model after CNN Configuration

## System Architecture

A complex architecture was designed for training the model, this architecture was used due to the availability of a wide range of datasets for training the CNN. The architecture is shown in fig 4.1. It consists of 4 convolutional layers, 2 max-pooling layers, 2 dropout layers, a flattening layer, 2 dense layers, and the output layers. The summary of the model architecture is shown in figure 3.6:

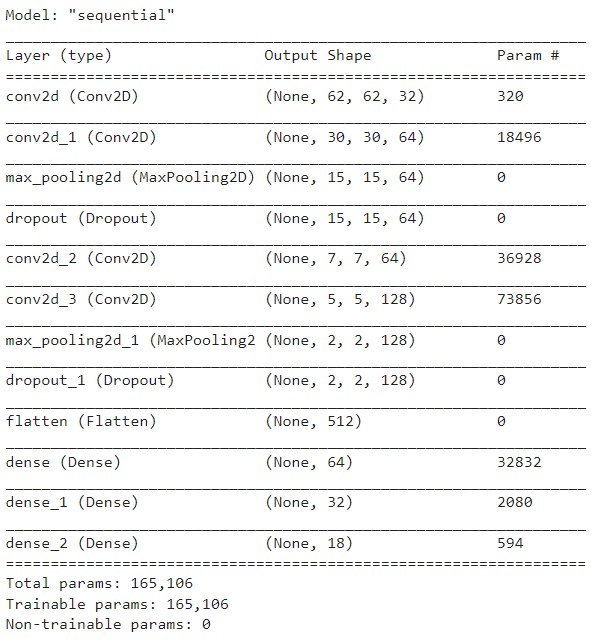


Figure 3.6: Summary of Model Architecture

The last layer is the output layer with 18 neurons which corresponds to the 18 classes to be classified. The total number of parameters trained in the model was 165,106.

The CNN was trained using a Python-based Keras sequential model on Spyder. The dataset was split into train and test sets. 75% of the dataset was used in training the model and 25% of it was used for validation during training.

## Raspberry Pi Configuration

In this project, a Raspberry Pi 3 model B CPU was utilized. On the SD card, the Raspbian Buster operating system was installed, which is based on the Linux kernel. The Raspberry Pi's lack of a screen or other external peripherals makes it necessary to gain access to the CPU's graphical user interface. An Ethernet cable that can be connected to a laptop or a high definition multimedia interface cable that is connected to a screen can be used for this. Another option is to configure a wireless connection using putty and secure shell (SSH). For this project, SSH was used to establish a headless connection between a laptop and a raspberry pi. The Raspberry Pi can connect to a ready-made local area network using SSH. The local area network in this experiment was a mobile hotspot. The configuration file contains crucial settings like the network id and password.

The SD card was taken out of the Raspberry Pi and put into a laptop using a card reader in order to configure SSH on the laptop. As illustrated in Figure 3.7, an empty SSH file without an extension was then created and stored to the SD card.

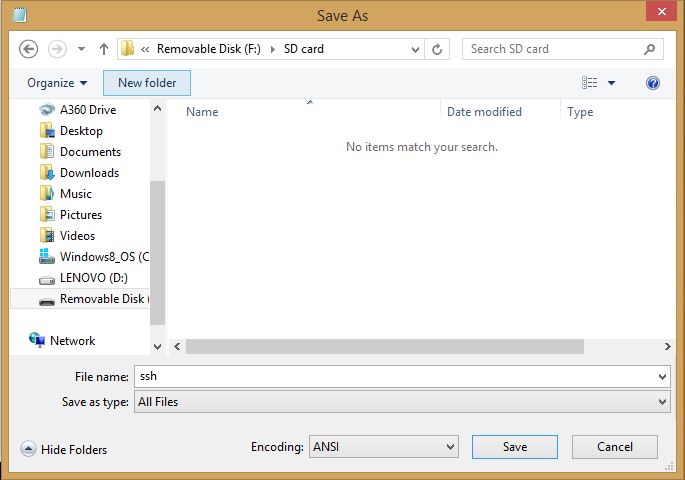


Figure 3.7: Summary of Model Architecture

The wpa\_supplicant.conf file, a second configuration file, is created using a text editor like notepad or notepad++. As seen in Figure 3.8, notepad++ was employed for this project. The local area network name is indicated by the ssid parameter, and the passkey for the local area network is indicated by the psk parameter. Care must be made to type these elements with the right letter case. In order to identify the wpa\_supplicant file as a configuration file, it must also be saved with a .conf extension. Both the configuration file and the empty SSH file were saved to the SD card. After that, the SD card was removed and put back into the Raspberry Pi.

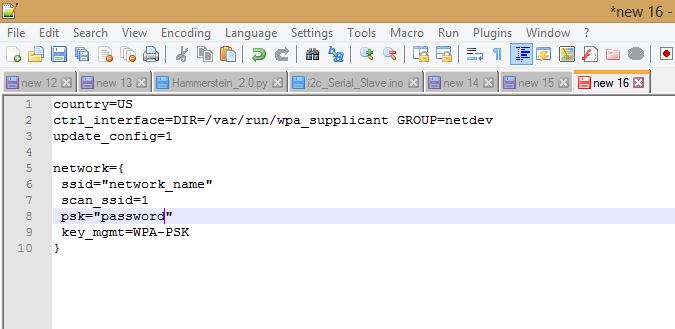


Figure 3.8: wpa\_supplicant file for SSH Configuration

Putty was then run after being installed on the computer. A windows terminal called Putty enables direct communication with Linux hardware. It enables users to connect via SSH from a computer on the same network to the command terminal of Linux computers. The host name or IP address feed received the raspberry pi's IP address. 22 was assigned to the port. The Raspberry Pi's IP address can be found on the bottom of the device or by using an IP scanning program. Figure 3.9 displays the configuration panel for Putty.

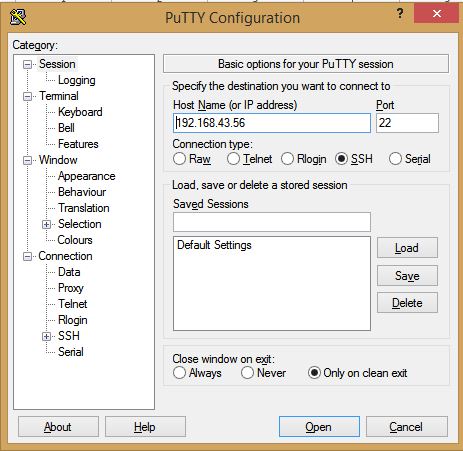


Figure 3.9: Putty Setup for SSH with Raspberry Pi

The Raspberry Pi and mobile hotspot can then be turned on. The mobile hotspot must also be linked to the computer's Wi-Fi. As a result, the raspberry pi and computer are both linked to the same local area network. Thus, communication between the computers is possible. The open button was then clicked as indicated above. The Raspberry Pi asks the user for a username and password as seen in Figure 3.10 after clicking open in Figure 3.9. "pi"  and "raspberry" serve as the username and password, respectively. No character appears on the screen while you type the password, and the pointer stays in the same location as in Figure 3.10.

The manufacturers of the Raspberry Pi and its Raspbian operating system have offered this as a security safeguard.



Figure 3.10: Login onto Putty Terminal

The putty terminal opens after entering the proper credentials. From the terminal, you may access the Raspberry Pi's command line. The terminal is shown in the manner depicted in Figure 3.11.

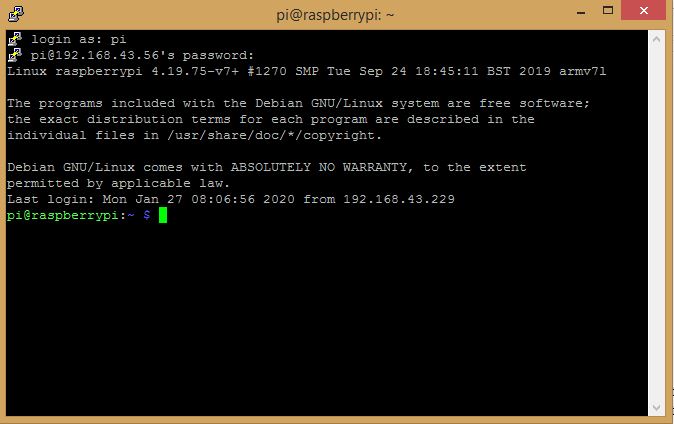


Figure 3.11: Putty Terminal

## System Construction

The raspberry pi needs to be interfaced with certain components before it can perform both the hand gesture recognition and device control operations. This section gives an overview of what components were needed and how they were interfaced with the raspberry pi.

### Interfacing Raspberry Pi Camera with the Raspberry Pi 3B+

The pi Camera module is a camera that can be used to take pictures and high definition video. The Raspberry Pi Board has a Camera Serial Interface (CSI) interface to which the pi Camera module was attached directly through a 15-pin ribbon cable. The python pi camera library was used to access the pi camera after the connection was made. Figure 3.12 shows the interfacing connection between the raspberry pi and pi camera via the CSI.



Figure 3.12: Interfacing Raspberry Pi with Pi Camera via the CSI

### Interfacing Liquid Crystal Display (LCD) with the Raspberry Pi 3B+

The LCD used in this project was interfaced with an inter integrated circuit (I2C). The I2C interface protocol is used to transfer data using only 2 pins, SDA and SCL, making it simpler and easier. For this project, a 16x2 LCD display module that outputs only 4 pins (SDA, SCL, VCC and GND) and has an I2C adapter connected to the 16 pins of a standard LCD display was used. The PCF8574 8-Bit I/O Expander IC in the I2C adapter transforms the parallel data needed by the LCD display module from the I2C signals sent by the raspberry pi. This adapter has a jumper to turn the backlight ON or OFF, which can reduce power usage. It also has an internal potentiometer to change the LCD display module's contrast. The python rpi\_lcd library was used to access the LCD after connection was made. Figure 3.13 shows the interfacing connection between the raspberry pi and LCD via the 4 pins.

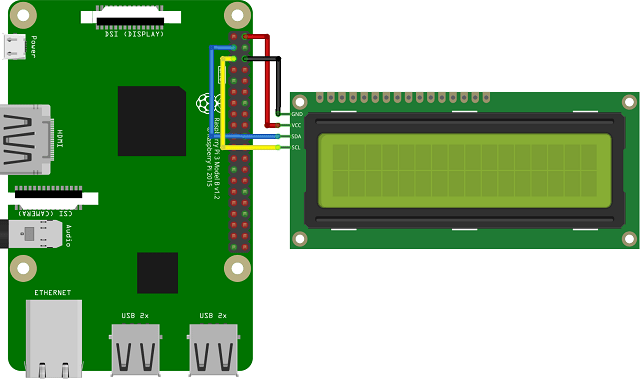


Figure 3.13: Interfacing Raspberry Pi with LCD using an I2C

### Interfacing Relay Module with the Raspberry Pi 3B+

The output from the raspberry pi is 3.3V but the relay module requires 5V to operate. Hence, a level converter or level shifter which would step up the voltage from 3.3V to 5V was used. The figure shows how the raspberry pi was interfaced with the relay module via the level shifter. Also, the RPi library was used to access the GPIO pins of the raspberry pi.

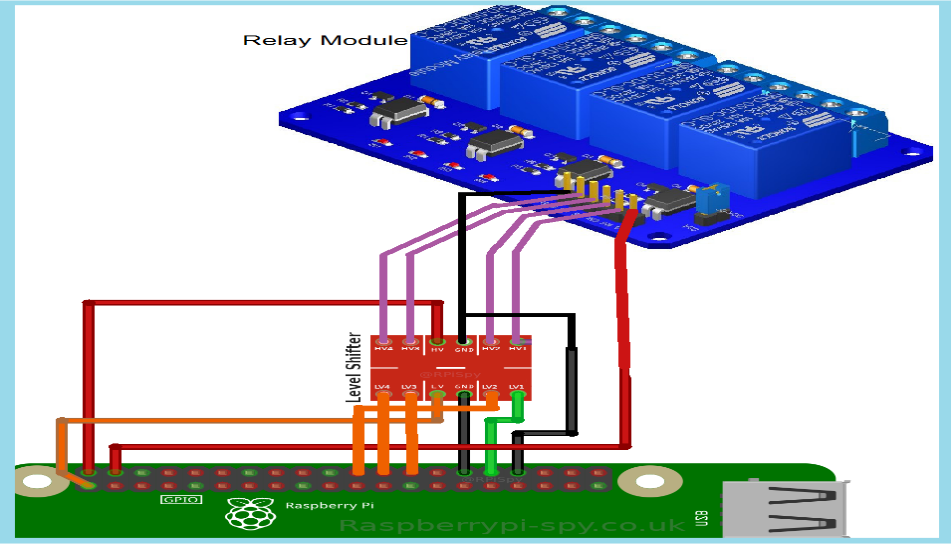


Figure 3.14: Interfacing Raspberry Pi with Relay Module via Level Shifter

## Hand Gesture Recognition

From Figure 3.17, which displays the flowchart of the hand gesture recognition system, it can be observed that the hand gesture recognition process begins once the raspberry pi is turned ON. The pi camera then comes ON and captures images placed in front of it. This image is then forwarded to the trained CNN model which is deployed on the raspberry pi.

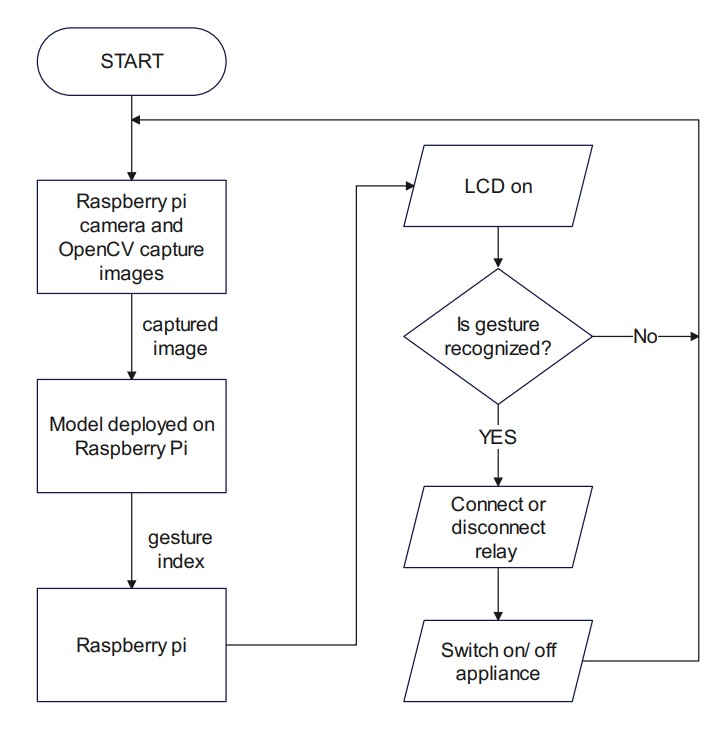


Figure 3.15: Flowchart of the Hand Gesture Recognition System

OpenCV was the library used to capture the hand gestures placed in front of the camera. A function get\_result() receives a gesture index which depends on what hand gesture is displayed and carries out a corresponding action for the respective condition assigned to that gesture. Figure 3.18 displays the detection code while it was running.

The detection code first captures live videos from the pi camera and segments the hand region using a running average background subtraction technique. It also draws contours around the segmented hand and displays thresholded images to the user.

The detection code updates the frames of the default OpenCV window, which shows the captured video stream, with text and images indicating the current state of the hand gesture recognition application. For example, during the initial 30 frames, the window shows the text "Ready to Scan Hand Gesture" at the bottom of the frame.

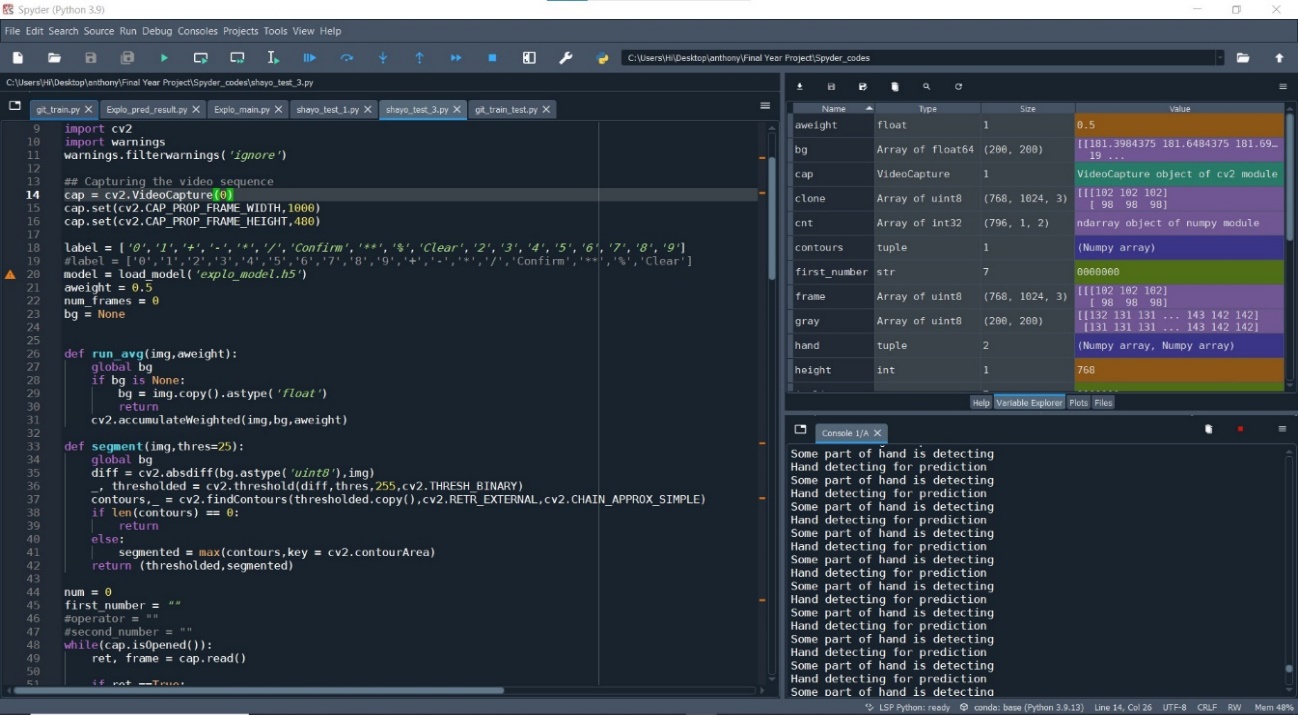


Figure 3.16: The Hand Gesture Recognition Detection Code

## Control of Devices

From Figure 3.19, GPIO6, GPIO12, GPIO13 and GPIO16 of the raspberry pi are connected to transistors Q1, Q2, Q3 and Q4 respectively and are set as HIGH ports. These transistors have a breakdown voltage of 5V and act as switching devices for the relays. When the pi is turned ON and the pi camera captures the image displayed in front of it, the image is sent to the model on the raspberry pi. The model then checks if the image is one of the hand gestures in the dataset. If YES, the raspberry pi would make the GPIO port, associated with the displayed image, to become LOW. Hence, allowing flow of current through the transistor which in turn charges the coil of the relay and allows flow of current to the device. For the prototype, only 4 devices will be controlled. Therefore, only hand gestures 0-7 will give an output to the relay for control of devices as shown in Figure 3.18. If there is a demand for more devices to be controlled, more relays can be installed but only to a maximum of the number of hand gestures trained in the model (20). From Figure 3.8, it can be observed that the integrated interactive circuit (I2C) liquid crystal display (LCD) which was interfaced to the raspberry pi via the serial data (SDA) port and the serial clock (SCL) port. This acts as a feedback to the user, telling the user that the appliance has been switched ON or OFF. A diode acting as a free-wheeling diode was also used in the circuit so as to protect the transistor from getting damaged by high in-rush current after the GPIO port has returned to HIGH.



Figure 3.17: Hand Gestures used for the Prototype

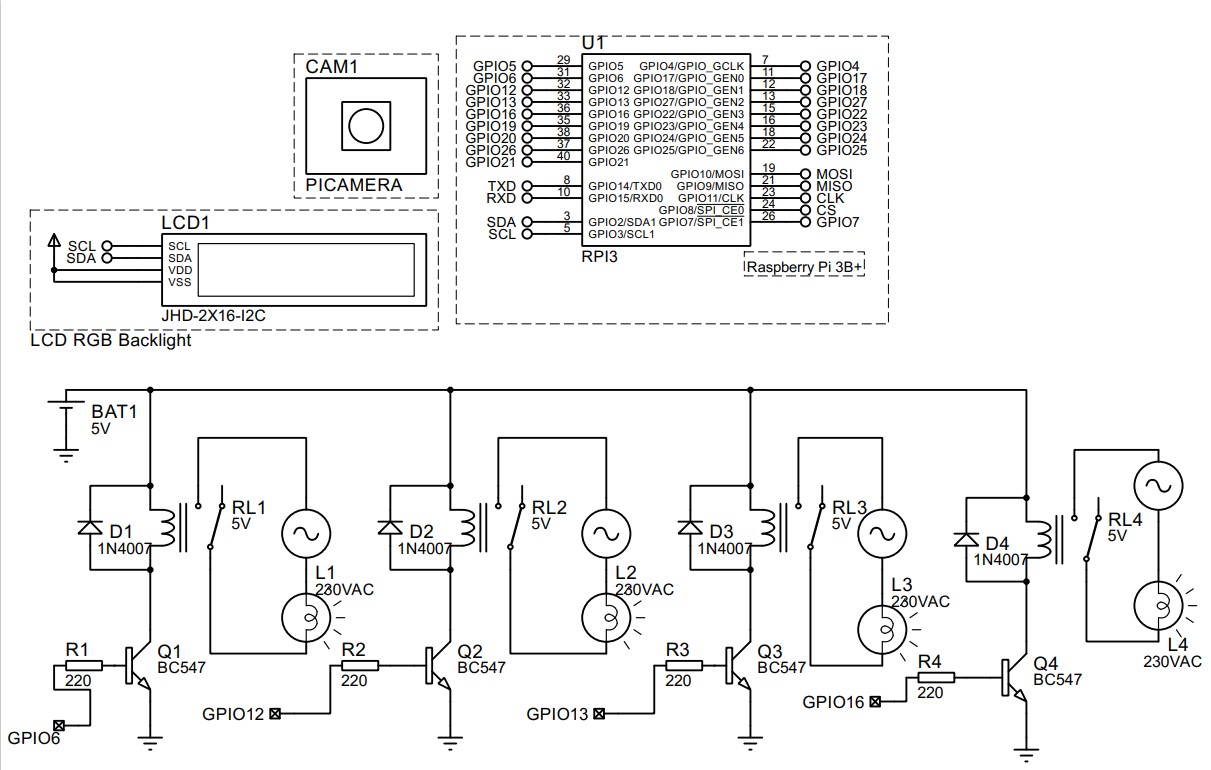


Figure 3.18: Circuit Diagram for the Circuit of the Hand Gesture Recognition System

# 

# RESULTS AND DISCUSSION

## Action Recognition

There are 18 identifiable hand gestures of a human that has been trained. If any of these hand gestures are identified in the captured frame, then the corresponding state of the home appliance will be followed according to the class of the action in Table 4.1:

Table 0‑1: Hand Gestures and their corresponding controlled devices



It was also observed that the number of epochs used to train the model and the number of images for each gesture are both improving the accuracy of the model. Epochs is the total number of times the same data set was used to train the model.

The performance was tested on the test dataset which comprised of 5400 images of the 18 classes used to train the model. This performance test was carried out after the training phase of the model and the approximate time for detection after hand is detected is 0.03 seconds. The metrics used for performance analysis on the model were as follows:

### Accuracy and Loss

Accuracy is a classification metric that includes the percentage of accurate predictions made by a model. A loss function analyzes how effectively the neural network models the training data by comparing the target and predicted output values. Reduction of this difference in output between the predicted and the target during training is essential. The accuracy and loss’ are computed as in Equation 4.1 and 4.2 respectively (Rahman, September 2017):

Equation (4.1)

Cross-entropy is defined as:

Equation (4.2)

where,

is the truth label

is the Softmax probability for the iₜₕ class

Figure 4.2 and figure 4.3 shows the plots of both the accuracy and loss scores against the number of epochs that the training went through. From figures 4.1 and 4.2, it is observed that as the number of epochs increases the accuracy also increases and tends towards 1 while the loss decreases and tends towards 0. This thereby signifies that as the model is continuously trained, it becomes more accurate and loss reduces.

After applying the trained CNN model to the test dataset, the achieved accuracy and loss values were 99.48% and 1.24% respectively which proves that the model is well trained to carry out the hand gesture detection. For the confusion matrix achieved from the dataset, the true labels were plotted against the predicted labels and is displayed on figure 4.3. From the confusion matrix, it is observed that no hand gesture, within the trained set, was recognized wrongly as the model was able to classify the amount of each hand gesture placed in each folder corresponding to their respective gesture index. A table containing the classification report was also printed out on the Spyder console and it is displayed on figure 4.4. From the classification report, high levels of precision, precision and recall were achieved by the model.

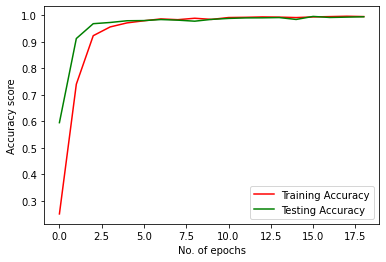


Figure 4.1: Accuracy Score versus Number of Epochs

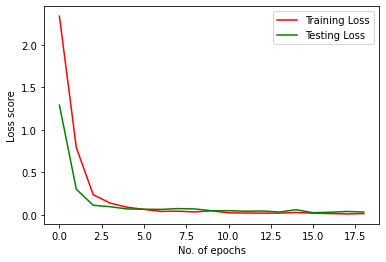


Figure 4.2: Loss Score versus Number of Epochs

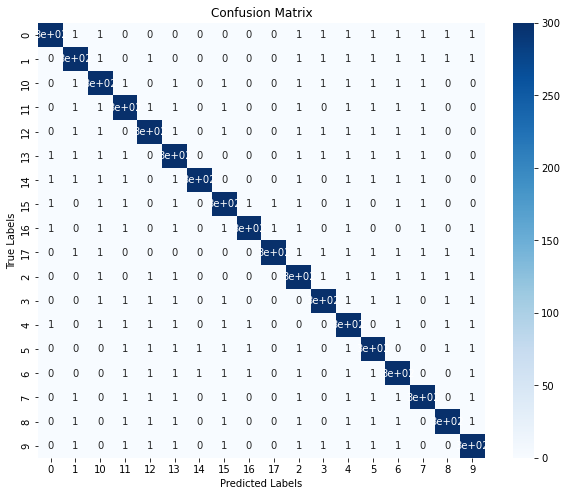


Figure 4.3: Confusion Matrix

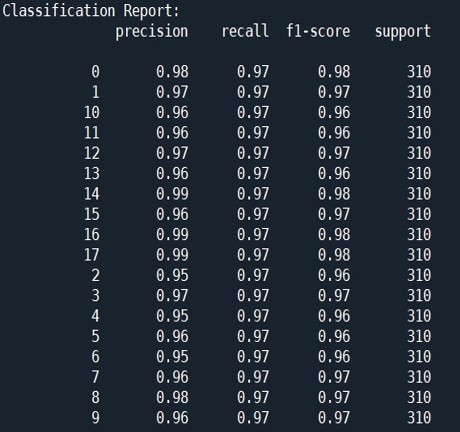


Figure 4.4: Classification Report for each Class

### Recall

This is the percentage of the total number of relevant instances that were retrieved. The recall score (or simply recall) is a classification statistic that measures the proportion of positive class predictions that match the ground truth to all positive samples. Recall, thus, assesses a classifier's capacity to identify positive samples. Figure 4.5 shows a bar chart of the recall values plotted against the different classes of hand gestures.

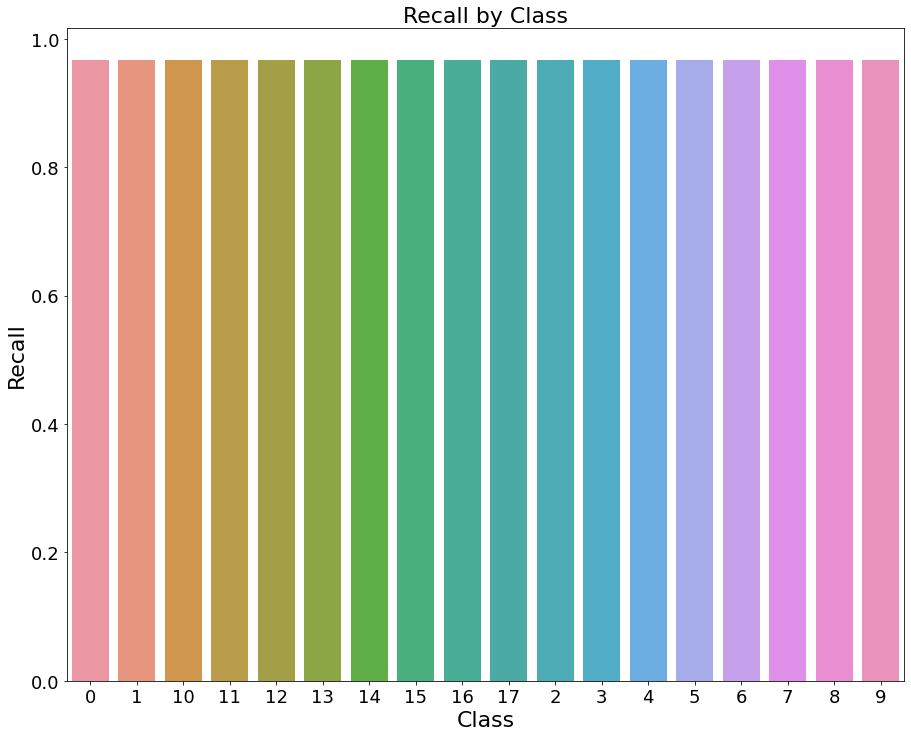


Figure 4.5: Recall versus Hand Gesture Classes

### Precision

It is defined as the percentage of the relevant instances among the instances retrieved. The precision score (or simply precision) measures the proportion of predictions in the Positive class that are confirmed by ground truth to be positive. In other words, precision assesses a classifier's capacity not to classify a negative sample as positive. Figure 4.6 shows a bar chart of the precision values plotted against the different classes of hand gestures.

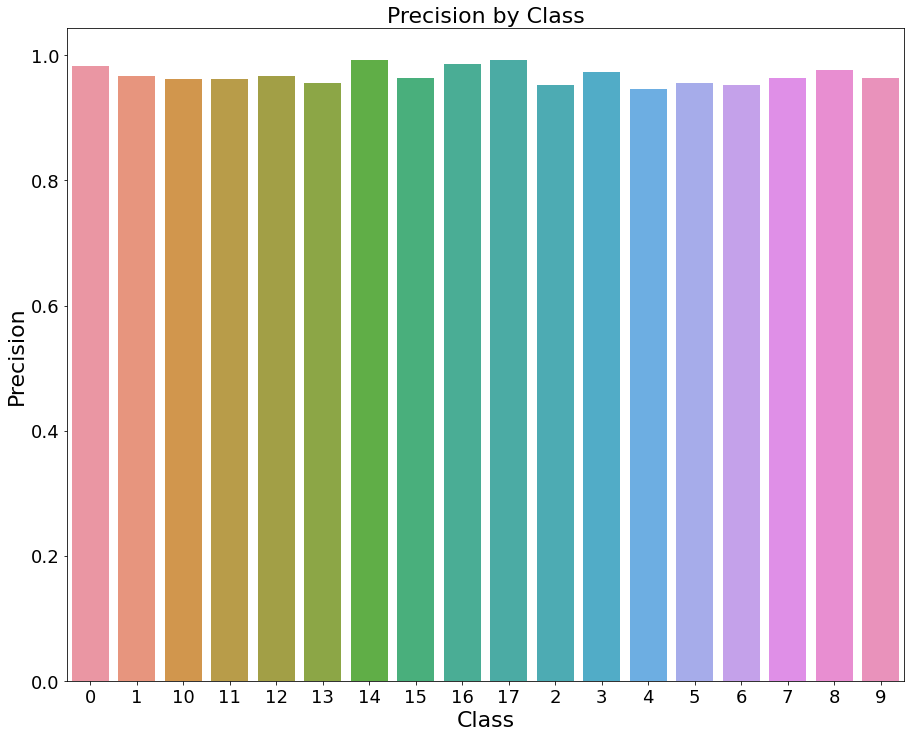


Figure 4.6: Precision versus Hand Gesture Classes

### F1-Score

This is defined as the harmonic mean of the Precision and the Recall. The F1-score tells the accuracy of the classifier in classifying the data points in that particular class compared to all other classes. Figure 4.7 shows a bar chart of its values plotted against the classes of hand gestures.

Equation (4.2)

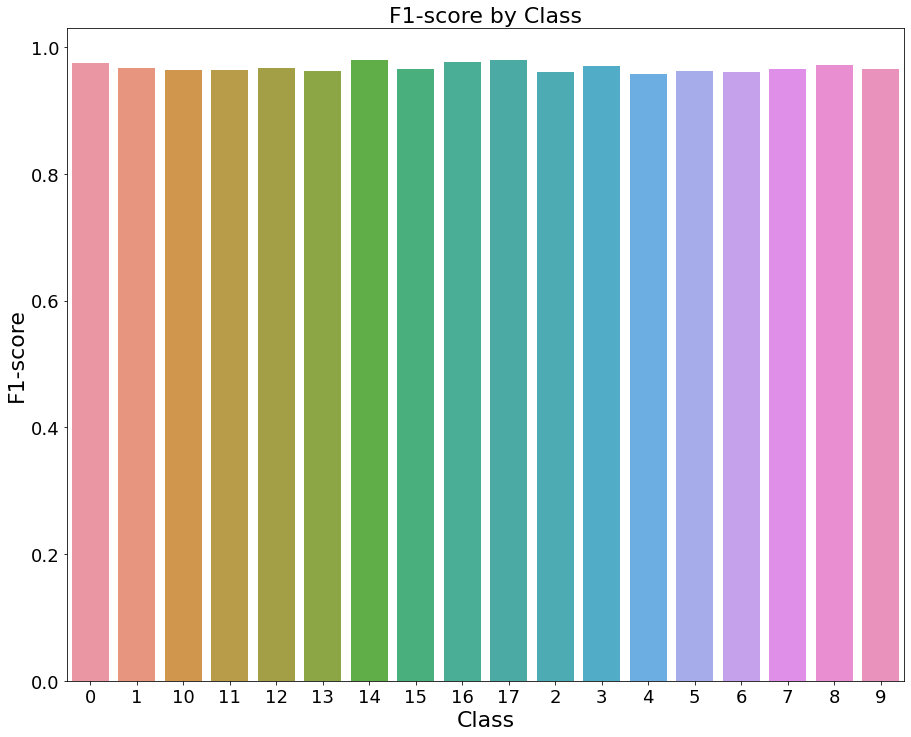


Figure 4.7: F1-Score versus Hand Gesture Classes

## Hand Gesture Recognition for Appliance Control

After detection has been initiated, the pi camera window comes up as seen in figure 4.8. The system selects the initial image placed within the region of interest (ROI) as the background for detection. The square on both figures 4.8 and 4.9, drawn with green lines, is the ROI used for detection. The gesture placed in the ROI is then captured and processed as grayscale image as seen in the gray scale window of both figures 4.8 and 4.9.

From figures 4.8 and 4.9, it can be observed that the gestures assigned to labels 1 and 5 were accurately predicted by the model and hence used for control by the raspberry pi in this project. Therefore, the model can accurately detect and predict what gesture index or label each of the 18 hand gestures represent.

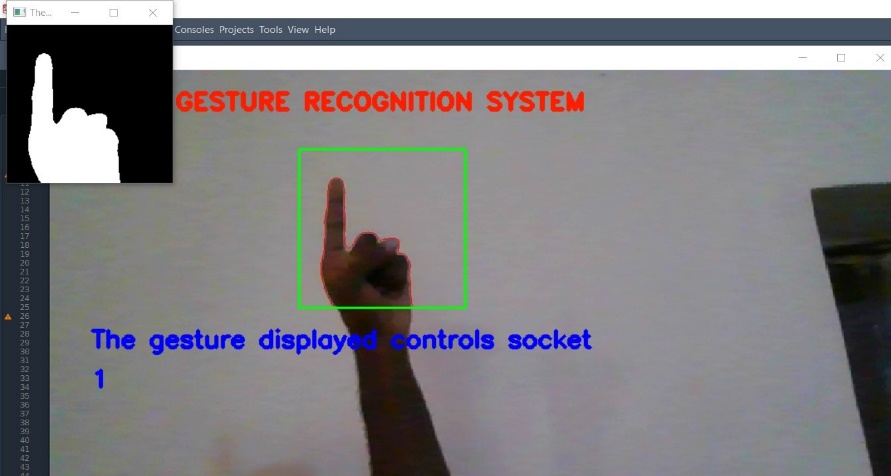


Figure 4.8: System recognizing gesture labelled as ‘1’

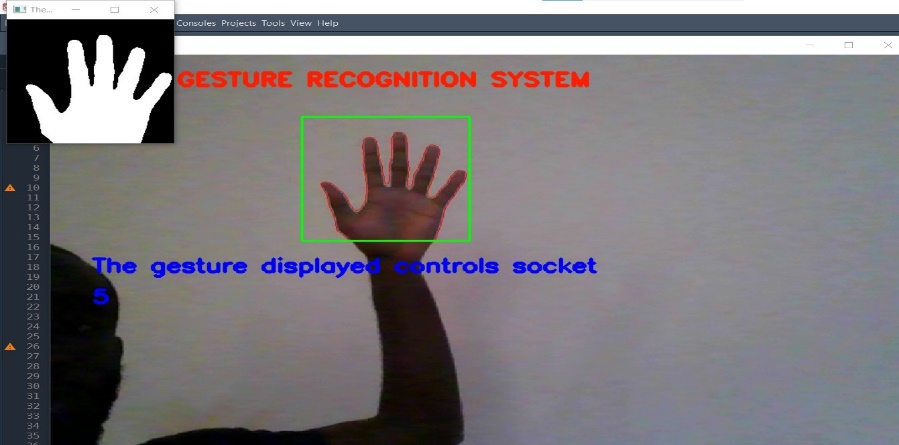


Figure 4.9: System recognizing gesture labelled as ‘5’

# 

# CONCLUSION AND RECOMMENDATION

## Conclusion

In the design and implementation of this system which automates home appliances according to human hand gestures, a raspberry pi 3B+, a raspberry pi camera, a 4-channel relay module, a liquid crystal display and socket boxes were used. A hand gesture recognition model trained on 16,200 images of 18 different hand gestures was designed using a convolutional neural network (CNN) algorithm. The CNN model was developed using Keras and OpenCV to detect and classify the 18 different hand gestures from video frames taken by the raspberry pi. The algorithm achieved an accuracy of 99.56%.

The model was then tested on a test dataset of 5400 images of the 18 hand gestures and it achieved an accuracy of 99.48%. It was observed that as the number of epochs increased, the accuracy of the model kept on increasing. All the training and testing was carried out on a laptop which possessed 32GB RAM and a NVIDIA MX130 graphics card.

The model was then deployed on the raspberry pi and used for detection and classification of hand gestures for control of the socket boxes connected to them. The implementation of this project revealed that a deep learning-based action recognition algorithm can be efficiently deployed on a Raspberry Pi and used for control of home appliances.

## Recommendation

Although the model was highly accurate the following works can be done to further improve the hand gesture recognition system:

1. development of models that require lesser RAM so as to reduce the cost of deployment or application.
2. identification of complicated hand gestures involving motion and two hands.
3. development of a system that can detect gestures in the dark and also perform more complex background subtractions as image processing would fail if background is very similar to skin color and if there is poor lightning.

# Appendices

# Appendix I

**Source code for the Human Gesture Recognition System**

**The CNN algorithm is as follows:**

import matplotlib.pyplot as plt

from keras\_preprocessing.image import ImageDataGenerator

import keras

from keras import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

train\_path = 'C:/Users/Hi/Desktop/anthony/Final Year Project/Training data/train/train'

train\_datagen = ImageDataGenerator(rescale=1./255, rotation\_range = 4,

width\_shift\_range=0.15,

height\_shift\_range = 0.2,

shear\_range=0.3,

fill\_mode='nearest',

validation\_split=0.25)

train\_set = train\_datagen.flow\_from\_directory(directory=train\_path, class\_mode='categorical',

color\_mode = 'grayscale',

target\_size = (64,64), batch\_size=128, shuffle=True,

subset ="training")

validation\_set = train\_datagen.flow\_from\_directory(directory=train\_path,

target\_size = (64,64),

batch\_size = 64,

class\_mode = 'categorical',

color\_mode = 'grayscale', subset="validation", shuffle=True)

def make\_model():

model = Sequential()

model.add(Conv2D(32,input\_shape=(64,64,1),kernel\_size=(3,3),strides=(1,1),activation='relu'))

padding="same"

model.add(Conv2D(64,kernel\_size=(3,3),strides=(2,2),activation='relu'))

padding="same"

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.15))

model.add(Conv2D(64,kernel\_size=(3,3),strides=(2,2),activation='relu'))

padding="same"

model.add(Conv2D(128,kernel\_size=(3,3),strides=(1,1),activation='relu'))

padding="same"

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(64,activation='relu'))

model.add(Dense(32,activation='relu'))

model.add(Dense(18,activation='softmax'))

model.compile(

loss = 'categorical\_crossentropy',

optimizer = 'Adam',metrics = ['accuracy']

)

return model

model = make\_model()

model.summary()

h = model.fit(

train\_set,validation\_data = validation\_set,

epochs=20,steps\_per\_epoch = 64,validation\_steps = 48,

callbacks = [

keras.callbacks.EarlyStopping(monitor='val\_loss',patience=3,mode='auto'),

keras.callbacks.ModelCheckpoint('explo/model\_{val\_loss:.3f}.h5',

save\_best\_only = True,save\_weights\_only=False,

monitor='val\_loss')

]

)

model.save('explo\_model.h5')

#%matplotlib inline

accu= h.history['accuracy']

val\_acc=h.history['val\_accuracy']

loss=h.history['loss']

val\_loss=h.history['val\_loss']

epochs=range(len(accu)) #No. of epochs

plt.plot(epochs,accu,'r',label='Training Accuracy')

plt.plot(epochs,val\_acc,'g',label='Testing Accuracy')

plt.legend()

plt.xlabel('No. of epochs')

plt.ylabel('Accuracy score')

plt.figure()

#Plot training and validation loss per epoch

plt.plot(epochs,loss,'r',label='Training Loss')

plt.plot(epochs,val\_loss,'g',label='Testing Loss')

plt.xlabel('No. of epochs')

plt.ylabel('Loss score')

plt.legend()

plt.show()

train\_set.class\_indices

labels = ['0','1','+','-','\*','/','Confirm','\*\*','%','Clear','2','3','4','5','6','7','8','9']

# Appendix II

**Code for testing the trained CNN model:**

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import keras

from keras\_preprocessing.image import ImageDataGenerator

from sklearn.metrics import classification\_report, confusion\_matrix

# Load the saved model

model = keras.models.load\_model('explo\_model.h5')

# Create a test set generator

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_set = test\_datagen.flow\_from\_directory(directory='C:/Users/Hi/Desktop/anthony/Final Year Project/Testing data/test/test',

target\_size=(64, 64),

batch\_size=1,

color\_mode='grayscale',

class\_mode='categorical',

shuffle=False)

# Generate predictions for the test set

Y\_pred = model.predict\_generator(test\_set, steps=test\_set.samples)

# Convert predictions from probabilities to class labels

y\_pred = np.argmax(Y\_pred, axis=1)

# Get true labels from the test set generator

y\_true = test\_set.classes

# Get the class names

class\_names = list(test\_set.class\_indices.keys())

# Print the classification report

print('Classification Report:')

print(classification\_report(y\_true, y\_pred, target\_names=class\_names))

# Plot the confusion matrix

plt.figure(figsize=(10,8))

sns.heatmap(cm, annot=True, cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)

plt.title('Confusion Matrix')

plt.xlabel('Predicted Labels')

plt.ylabel('True Labels')

plt.show()

# Plot the precision, recall and f1-score

report = classification\_report(y\_true, y\_pred, target\_names=class\_names, output\_dict=True)

data = {'Precision': [], 'Recall': [], 'F1-score': []}

for label in class\_names:

data['Precision'].append(report[label]['precision'])

data['Recall'].append(report[label]['recall'])

data['F1-score'].append(report[label]['f1-score'])

plt.figure(figsize=(10,8))

sns.barplot(x=class\_names, y=data['Precision'])

plt.title('Precision by Class')

plt.xlabel('Class')

plt.ylabel('Precision')

plt.show()

plt.figure(figsize=(10,8))

sns.barplot(x=class\_names, y=data['Recall'])

plt.title('Recall by Class')

plt.xlabel('Class')

plt.ylabel('Recall')

plt.show()

plt.figure(figsize=(10,8))

sns.barplot(x=class\_names, y=data['F1-score'])

plt.title('F1-score by Class')

plt.xlabel('Class')

plt.ylabel('F1-score')

plt.show()

# Appendix III

**The detection code for the Hand gesture recognition system is as follows:**

from relay\_lcd\_func import \*

import cv2

import warnings

warnings.filterwarnings('ignore')

## Capturing the video sequence

cap = cv2.VideoCapture(0)

cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH,1000)

cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT,480)

label = ['0','1','+','-','\*','/','Confirm','\*\*','%','Clear','2','3','4','5','6','7','8','9']

model = load\_model('explo\_model.h5')

aweight = 0.5

num\_frames = 0

bg = None

def run\_avg(img,aweight):

global bg

if bg is None:

bg = img.copy().astype('float')

return

cv2.accumulateWeighted(img,bg,aweight)

def segment(img,thres=25):

global bg

diff = cv2.absdiff(bg.astype('uint8'),img)

\_, thresholded = cv2.threshold(diff,thres,255,cv2.THRESH\_BINARY)

contours,\_ = cv2.findContours(thresholded.copy(),cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_SIMPLE)

if len(contours) == 0:

return

else:

segmented = max(contours,key = cv2.contourArea)

return (thresholded,segmented)

num = 0

first\_number = ""

#operator = ""

#second\_number = ""

while(cap.isOpened()):

ret, frame = cap.read()

if ret ==True:

frame = cv2.flip(frame, 1)

clone = frame.copy()

(height, width) = frame.shape[:2]

roi = frame[100:300, 300:500]

gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (7, 7), 0)

if num\_frames < 30:

run\_avg(gray, aweight)

else:

hand = segment(gray)

if hand is not None:

(thresholded, segmented) = hand

num = num + 1

cv2.drawContours(clone, [segmented + (300, 100)], -1, (0, 0, 255))

cv2.imshow("Thesholded", thresholded)

print("Some part of hand is detecting")

contours, \_= cv2.findContours(thresholded,cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_NONE)

## "Calculator Ready" printing for 3 Secs

if num < 90:

cv2.putText(clone, 'Ready to Scan Hand Gesture', (50, 400), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 3)

## Enter the first Number printing for 2 Secs

elif num > 90 and num < 150:

cv2.putText(clone, 'Control the appliance using their respective hand gesture', (50, 400), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 3)

## Confirmation of first number for 2 Secs

elif num > 481 and num < 540:

cv2.putText(clone, "Confirmed", (50, 300), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 3)

wor = "The gesture index is " + first\_number

cv2.putText(clone, wor, (50, 400), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 3)

elif num > 600:

res = get\_result(first\_number)

in\_line = str(res)

cv2.putText(clone, "The gesture displayed controls socket", (50, 350), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 3)

cv2.putText(clone,in\_line,(50,400),cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,0,0),3)

# sleep(5)

elif num > 900:

cap.release()

cv2.destroyAllWindows()

for cnt in contours:

if cv2.contourArea(cnt) > 5000:

print("Hand detecting for prediction")

## Inputing the first number for 12 Secs and 2 Secs for each character

if num > 150 and num < 481:

cv2.putText(clone, first\_number, (50, 400), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 0), 3)

if num % 60 == 0:

pred = get\_prediction(thresholded)

if pred != "Confirm" and pred != "Clear":

first\_number = first\_number + pred

elif pred == "Clear":

num = 91

first\_number = ""

else:

num = 481

elif num > 600:

if num % 60 == 0:

pred = get\_prediction(thresholded)

if pred == "Clear":

num = 0

first\_number = ""

#operator = ""

#second\_number = ""

cv2.rectangle(clone, (300, 100), (500, 300), (0, 255, 0), 2)

cv2.putText(clone, "Gesture Controlled Calculator", (50, 50), cv2.FONT\_HERSHEY\_DUPLEX, 1, (0, 30, 255), 3)

num\_frames += 1

cv2.imshow('frame', clone)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

else:

break

cap.release()

print(first\_number)

cv2.waitKey()

cv2.destroyAllWindows()

**for the GPIO pins**:

from rpi\_lcd import LCD

## setting up the lcd object

lcd = LCD()

lcd.text("Setting Up...", 1)

import RPi.GPIO as GPIO

from numpy import array

from time import sleep

# set the GPIO mode

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

# Set the pins that are to be controlled as outputs

GPIO.setup(6, GPIO.OUT)

GPIO.setup(12, GPIO.OUT)

GPIO.setup(13, GPIO.OUT)

GPIO.setup(16, GPIO.OUT)

GPIO.setup(19, GPIO.OUT)

GPIO.setup(20, GPIO.OUT)

GPIO.setup(21, GPIO.OUT)

GPIO.setup(26, GPIO.OUT)

# The relay module is active LOW

turn\_off\_socket\_1 = lambda: (GPIO.output(6, GPIO.HIGH), lcd.text("socket 1 is turned off", 1))

turn\_on\_socket\_1 = lambda: (GPIO.output(6, GPIO.LOW), lcd.text("socket 1 is turned on", 1))

turn\_off\_socket\_2 = lambda: (GPIO.output(12, GPIO.HIGH), lcd.text("socket 2 is turned off", 1))

turn\_on\_socket\_2 = lambda: (GPIO.output(12, GPIO.LOW), lcd.text("socket 2 is turned on", 1))

turn\_off\_socket\_3 = lambda: (GPIO.output(13, GPIO.HIGH), lcd.text("socket 3 is turned off", 1))

turn\_on\_socket\_3 = lambda: (GPIO.output(13, GPIO.LOW), lcd.text("socket 3 is turned on", 1))

turn\_off\_socket\_4 = lambda: (GPIO.output(16, GPIO.HIGH), lcd.text("socket 4 is turned off", 1))

turn\_on\_socket\_4 = lambda: (GPIO.output(16, GPIO.LOW), lcd.text("socket 4 is turned on", 1))

turn\_off\_socket\_5 = lambda: (GPIO.output(19, GPIO.HIGH), lcd.text("socket 5 is turned off", 1))

turn\_on\_socket\_5 = lambda: (GPIO.output(19, GPIO.LOW), lcd.text("socket 5 is turned on", 1))

turn\_off\_socket\_6 = lambda: (GPIO.output(20, GPIO.HIGH), lcd.text("socket 6 is turned off", 1))

turn\_on\_socket\_6 = lambda: (GPIO.output(20, GPIO.LOW), lcd.text("socket 6 is turned on", 1))

turn\_off\_socket\_7 = lambda: (GPIO.output(21, GPIO.HIGH), lcd.text("socket 7 is turned off", 1))

turn\_on\_socket\_7 = lambda: (GPIO.output(21, GPIO.LOW), lcd.text("socket 7 is turned on", 1))

turn\_off\_socket\_8 = lambda: (GPIO.output(26, GPIO.HIGH), lcd.text("socket 8 is turned off", 1))

turn\_on\_socket\_8 = lambda: (GPIO.output(26, GPIO.LOW), lcd.text("socket 8 is turned on", 1))

no\_action = lambda: lcd.text("No action.", 1)

clear\_screen = lambda: lcd.clear()

actuate\_dict = {

0 : ("0", turn\_on\_socket\_1), 1 : ("1", turn\_off\_socket\_1),

10 : ("2", turn\_on\_socket\_2), 11 : ("3", turn\_off\_socket\_2),

12 : ("4", turn\_on\_socket\_3), 13 : ("5", turn\_off\_socket\_3),

14 : ("6", turn\_on\_socket\_4), 15 : ("7", turn\_off\_socket\_4),

16 : ("8", turn\_on\_socket\_5), 17 : ("9", turn\_off\_socket\_5),

2 : ("10", turn\_on\_socket\_6), 3 : ("11", turn\_off\_socket\_6),

4 : ("12", turn\_on\_socket\_7), 5 : ("13", turn\_off\_socket\_7),

6 : ("14", no\_action), 7 : ("15", turn\_on\_socket\_8),

8 : ("16", turn\_off\_socket\_8), 9 : ("17", clear\_screen),

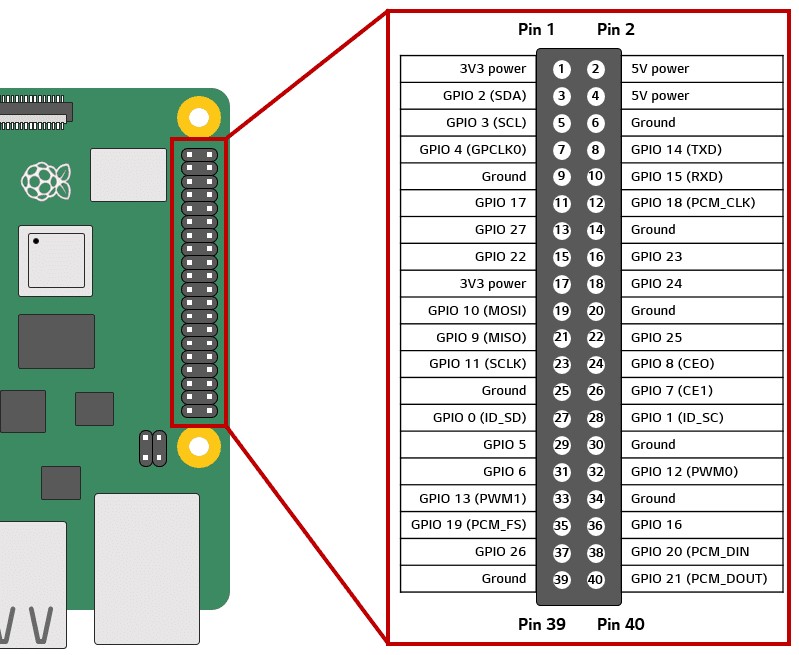
}

label = ['0','1','+','-','\*','/','Confirm','\*\*','%','Clear','2','3','4','5','6','7','8','9']

#label = ['0','1','2','3','4','5','6','7','8','9','+','-','\*','/','Confirm','\*\*','%','Clear']

# Appendix IV

**COMPONENTS USED IN THE DESIGN OF THE SYSTEM**



Raspberry Pi GPIO Pins

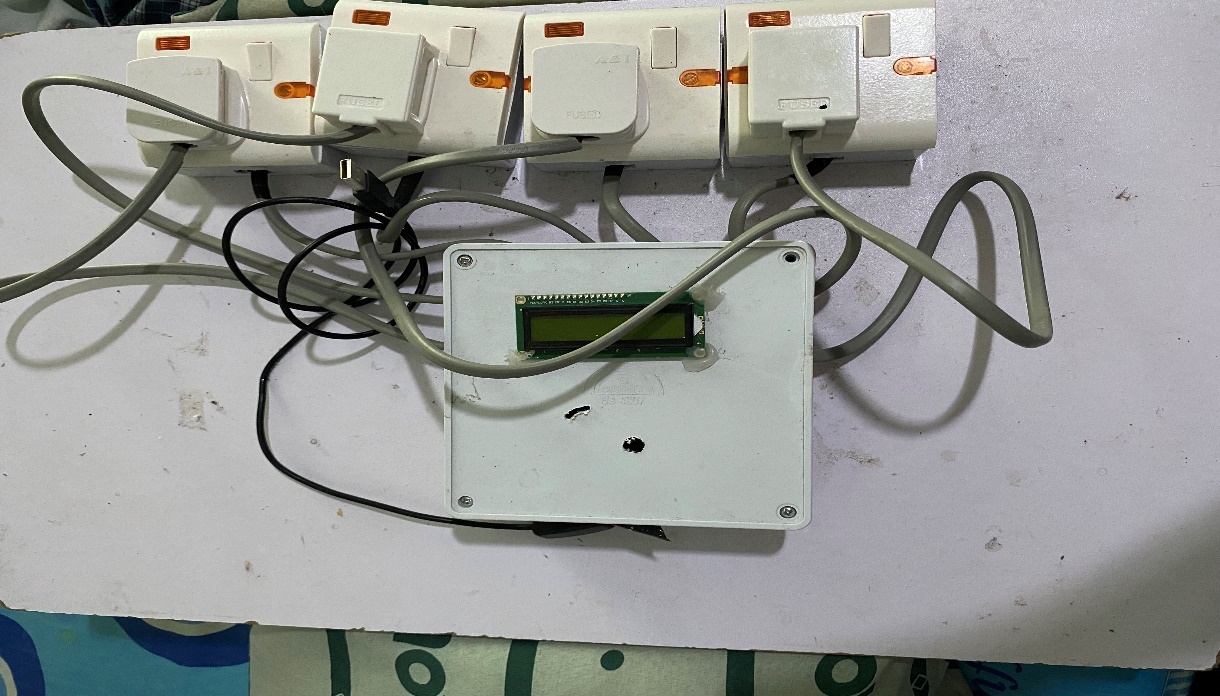


Raspberry Pi 3B Pi Camera Dupont Lines

****  

LCD Relay Module

# Appendix V

****

****

**Assembled Prototype**

# References

A. Krizhevsky, I. S. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958.

Abhinav, A. V. (November, 2019). *Sign Language to Text and Speech Conversion.* Allahabad, India: Computer Science and Engineering Department, MOTILAL NEHRU NATIONAL INSTITUTE OF TECHNOLOGY, ALLAHABAD, PRAYAGRAJ.

Dunne, N. A. (1997). On the pairing of the SoftMax activation and cross-entropy penalty functions and the derivation of the SoftMax activation function. *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, vol. 181. Citeseer, 1997, p. 185.

G. Cutipa, R. L. (2017). Application of convolutional neural networks for static hand gestures recognition under different invariant features. *Electrical Engineering and Computing (INTERCON)*, pp. 1–4.

H. Huynh, J. M.-N. (2013). Static hand gesture recognition using artificial neural network. *Journal of Image and Graphics*, vol. 1, no. 1, pp. 34–38.

Karl Andersson, R. U. (2014). Static Hand Gesture Recognition using Convolutional Neural Network with Data Augmentation. *Joint 2019 8th International Conference on Informatics, Electronics & Vision (ICIEV)* , 324-329.

N. D. Georganas, E. M. (2008). Hand gesture recognition using haar-like features and a stochastic context-free grammar. *IEEE transactions on instrumentation and measurement*, vol. 57, no. 8, pp. 1562–1571.

Premaratne, P. (2014). Human Computer Interaction Using Hand Gestures. *Cognitive Science and Technology*, 978-981.

Rahman, A. A. ( September 2017). Python-based Raspberry Pi for Hand Gesture Recognition. *International Journal of Computer Applications (0975 – 8887)*, Volume 173 – No.4.

S. Gupta, P. M. (2015). Hand gesture recognition with 3d convolutional neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp 1-7.

Sonia Raheja, A. C. (2011). A Survey on Hand Gesture Recognition in Context of Soft Computing. *Advanced Computing*, 46-55.

Stergiopoulou, N. P. (2009). Hand gesture recognition using a neural network shape fitting technique. *Engineering Applications of Artificial Intelligence*, vol. 22, no. 8, pp. 1141–1158.

Y. Bengio, X. G. (2011). Deep sparse rectifier neural networks. *artificial intelligence and statistics*, pp. 315–323.