VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

AN APPROACH OF HAND GESTURES RECOGNITION FOR HOUSEHOLD APPLIANCES ADMINISTRATION USING CONVOLUTIONAL NEURAL NETWORK

By
DANG CHI THINH
ITDSIU20104

A thesis submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Data Science

Ho Chi Minh City, Vietnam
March, 2024

AN APPROACH OF HAND GESTURES RECOGNITION FOR HOUSEHOLD APPLIANCES ADMINISTRATION USING CONVOLUTIONAL NEURAL NETWORK

APPROVED BY ADVISOR	APPROVED BY THESIS COMMITTEE
Assoc. Prof. Nguyen Thi Thuy Loan	Assoc. Prof. Nguyen Van Sinh
	Assoc. Prof. Huynh Kha Tu
	Assoc. Prof. Nguyen Thi Thuy Loan
	Dr. Le Duy Tan

THESIS COMMITTEE
(Whichever applies)

ACKNOWLEDGEMENTS

I want to express my sincere gratitude to all those who have supported me throughout the journey of completing my thesis.

First and foremost, I would like to extend my profound thanks to my instructor, Assoc. Prof. Nguyen Thi Thuy Loan, for her invaluable guidance and expertise. It is my most incredible honor to have completed this thesis under her supervision. Her insightful feedback has always played a vital role in shaping the outcome of this thesis.

I would also like to express my heartfelt appreciation to my family, parents, and sister for your unconditional love and constant encouragement. Your sacrifices and understanding have always been the very first bedrock of my achievements. I am profoundly grateful for your presence and support.

Lastly, I want to acknowledge my friends and soulmates' mental support and contributions. Your advice, with its diverse perspectives, was crucial in my journey to overcoming the difficulties of completing this thesis.

I am honored to offer my sincerest thanks to all those mentioned above. Your efforts, advice, instructions, and contributions have made this thesis possible.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

In the swiftly evolving landscape of machine learning applications, their integration into various facets of human life, notably within household appliances, has witnessed notable growth. Accompanying this proliferation requires practical control tools and methods, which often encounter challenges related to complexity and limited commercial applicability despite the convenience they promise. This thesis endeavors to address this predicament by presenting a novel approach to hand gesture recognition for the administration of household appliances. The methodology aims to overcome the shortcomings of current techniques by offering a lightweight, user-friendly, and expeditious solution. The proposed system leverages state-of-the-art technologies such as MediaPipe, Tensorflow, OpenCV, and a Convolutional Neural Network architecture. By amalgamating these tools, the research seeks to provide an accessible and efficient means of controlling appliances through intuitive hand gestures, thereby enhancing user experience, and circumventing the complexities and commercial limitations inherent in current methods.

CHAPTER 1 – INTRODUCTION

1.1 BACKGROUND

In recent years, there has been a rising demand for developing natural and intuitive methods for controlling household appliances in smart homes. One of the most promising options is using simple and wild hand gestures to interact with those appliances. With the increasing availability of affordable depth-sensing cameras, such as the Microsoft Kinect and the Intel RealSense, hand gesture recognition has become more feasible and accessible. Combining these technologies with machine learning and embedding image classification into smart homes' administrative systems draws much attention from customers and companies.

Several studies have shown that Convolutional Neural Networks (CNNs), one of the main branches of neural networks, can recognize and classify hand gestures for applications such as sign language recognition, human-computer interaction, and virtual reality. According to Cao, CNN is an efficient real-time approach to detecting human poses, including hand gestures and standings, using a depth camera [1]. Moreover, using CNN for hand gestures and fingertip detection, Li has developed a virtual keyboard with high accuracy for AR/VR [2]. These studies demonstrate the effectiveness of CNNs in recognizing hand gestures in various contexts.

1.2 PROBLEM STATEMENT

In the context of smart homes, hand gesture recognition can provide a natural and intuitive interface for controlling household appliances such as televisions, air conditioners, and lights. However, several challenges need to be addressed, including the variations in hand shapes, sizes, orientations and occlusions, and noisy backgrounds. Moreover, the complexity of those methods makes it hard for them to be widely used in commercials. Several approaches have been proposed to overcome these challenges, including hand-crafted feature extraction, deep learning-based methods, and hybrid approaches that combine both.

1.3 OBJECTIVES

This research proposes an approach for hand gesture recognition for household appliance administration using CNNs. This approach addresses the challenges of hand gesture recognition in the context of smart homes. It provides an efficient, lightweight, fast, producer-friendly, and user-friendly method for controlling household appliances. The approach will be evaluated on a dataset of hand gesture samples collected from multiple users, and the results will be compared with existing state-of-the-art methods, demonstrating the efficacy of our approach. Overall, this research seeks to contribute to developing efficient, quick, and friendly methods for smart homes through the use of CNN-based hand gesture recognition.

1.4 THESIS STRUCTURE

This thesis will be divided into five different chapters, including:

Chapter 1 - Introduction: As mentioned above, this chapter focuses on the reason for its topic, the background, and the final goal of this thesis.

Chapter 2 - Literature Review: In this chapter, previous studies by researchers worldwide will be carefully read, analyzed, and gathered as the basic knowledge in this domain. Papers related to this topic are also considered for their strengths and weaknesses.

Chapter 3 - Methodology: This chapter will mainly focus on the methods implied in this thesis, which consist of libraries, algorithms, and theoretical approaches.

Chapter 4 - Implementation: In this chapter, a demonstration with a model, interface, and data storage will be developed as a sample for use in real-world applications.

Chapter 5 - Conclusion and Future Work: Finally, this chapter will gather all the conclusions throughout the thesis and discuss future improvements.

CHAPTER 2 - LITERATURE REVIEW

2.1. OVERVIEW OF CNN

2.1.1. CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Network (CNN) is a famous deep-learning architecture built using living beings' visual perception mechanisms. The first version of CNN was established in 1980 in the name of “recognition” by Kunihiko Fukushima [3]; in his paper, Kunihiko Fukushima used the idea of the responsibility for detecting light in receptive fields by cells in animal visual cortex by Hubel and Wiesel [4].

In 1990, LeCun et al. [5] established LeNet-5, which is considered the first modern structure of CNN. LeNet-5 is a multi-layer neural computer hardware development that can be trained using the backpropagation method to help it memorize and classify handwritten digits. Later, LeCun improved this method using a new learning paradigm called graph transformer networks (GTNs) [6].

Since then, with the help of computer hardware developments, many methods have been introduced with the same purpose of improving CNN. In 2012, to combine the traditional algorithm with the new one, Niu introduced the hybrid classifier using CNN as a feature extractor and SVM as a recognizer [7]. In 2015, Simonyan published a paper discussing how the CNN layer’s depth affects the learning rate as well as the accuracy of the model [8]. In 2018, J. Gu et al. introduced the full hierarchy structure of the CNN model, including the components, the applications, and the related algorithms.

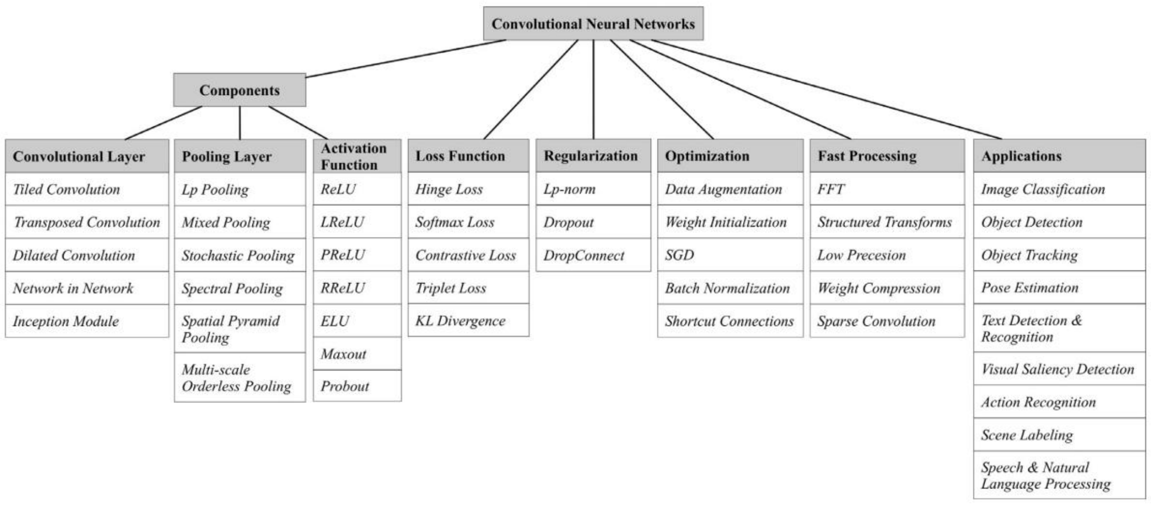


Figure 2.1: Hierarchy structure of CNN

2.1.2. COMPONENTS OF CNN

Although there are many variants of modern CNN structure, they mostly share the same basic components, including the convolutional layers, pooling layers, fully connected layers, and an output layer.

Several convolutional kernels are used to calculate the feature maps in the convolutional layers. To be exact, the kernels reduce each region of neurons in the previous layer and an element-wise nonlinear activation function into a single neuron in the next layer. The final feature map is the combination of many kernels in the network. Following is the formula for the computation:

$$\boldsymbol{z}_{i,j,k}^{l} = {\boldsymbol{w}_{k}^{l}}^{T} \boldsymbol{x}_{i,j}^{l} + \boldsymbol{b}_{k}^{l} \tag{1}$$

where the ${\boldsymbol{w}_{k}^{l}}^{T}$ is the weight vector, $\boldsymbol{b}_{k}^{l}$ is the bias and $\boldsymbol{x}_{i,j}^{l}$ is the input patch with the center (i, j) from the previous layer. Using $\boldsymbol{z}_{i,j,k}^{l}$, an activation function is applied to detect non-linear feature of the neural network:

$$\boldsymbol{a}_{i,j,k}^{l} = \boldsymbol{a}\left(\boldsymbol{z}_{i,j,k}^{l}\right) \tag{2}$$

with $\boldsymbol{a}$ is the non-linear activation function. The widely used functions include *sigmoid, tanh* [9], and *ReLU* function [10], which will be discussed later.

The second basic component is the pooling layer. The main goal of this layer is to reduce the number of weights and, therefore, the computational cost. Otherwise, it also avoids the overfitting probability of the model. A successful pooling layer can extract valuable features and remove irrelevant details from the data [11]. Each of the feature maps is connected to a pooling layer in the formula of:

$$y_{i,j,k}^{l} = pool\left(a_{m,n,k}^{l}\right), \forall(m,n) \in \mathcal{R}_{ij} \qquad (3)$$

in which $pool()$ is the pooling function and $\mathcal{R}_{ij}$ is the region around the center (i,j).

The fully connected layers are optional for the convolutional neural network as they connect all the previous layer's neurons with all the current layer's neurons to find the logical informative features [12]. Depending on the complexity of the problem, adding those layers can adjust the balance of accuracy and time cost for the network [13].

The last layer in a complete convolutional neural network is the output layer, where the layers above are connected to the network outputs, where the results are used to predict the input class. The most common operators in this layer for classification tasks are *softmax regression* [14] and *Support Vector Machine* [15].

2.2. RELATED WORKS

At the dawn of gesture recognition, external equipment such as trackers, sensors, or gloves was used as a support method for tracking human hands [16]. This approach, however, could have been more practical due to its limited availability.

Therefore, scientists developed vision-based methods to reduce the need for physical equipment. Throughout years of research, many algorithms have been tested, from simple classification methods to the complicated world of machine learning.

In 2019, Wu published a paper in which he used a Double-Channel Convolutional Neural Network (DC-CNN) to detect the edge of human hands; this approach achieved an accuracy of 98% [17]. In another method, Chung et al. used skin color detection, morphology, and background subtraction approach combined with their modified AlexNet and VGGNet neural networks to reach an accuracy of 95.6% [18]. Also, using a topology-reserving, self-organizing variant of the neural network, Florez et al. overcome the limitations of the hand-crafted features extraction approach [19]. However, the difficulty of these algorithms is the biggest drawback that needs to be re-calculated.

In the approach of simpler methods, Li et al. applied CNN and Support Vector Machine (SVM) together with a joint bilateral filter and surrounding area filling algorithm [20]. Ong and Bowden also proposed a boosted tree-based classification method for gesture detection and recognition [21]. Although the success rate of this method can reach more than 97%, the data used to test is said to be simple with similar backgrounds [21], which leads to the loss of natural characteristics of the gesture recognition problem. Those combinations are more accessible to implement - which results in a lower accuracy of 90.52%, compared to the above papers - yet remain too complex for wide uses.

2.3. RESEARCH METHOD

In this thesis, *MediaPipe* – a library of computer vision created by Google and *Tensorflow* – a library that helps users build models of CNN fast and easily, will be used as the detector of hand gestures. Moreover, *ELU, SELU, ReLU, LReLU, RReLU, SReLU, Sigmoid,* and *Tanh* functions will be implemented and compared to find the appropriate algorithm with the highest accuracy to optimize the output result for use. In terms of the output layer, *Softmax Regression* will be applied to classify the output of previous layers.

2.3.1 MEDIA PIPE

MediaPipe is a framework created by Google Research for other researchers, students, and software developers to construct perception pipelines [22].

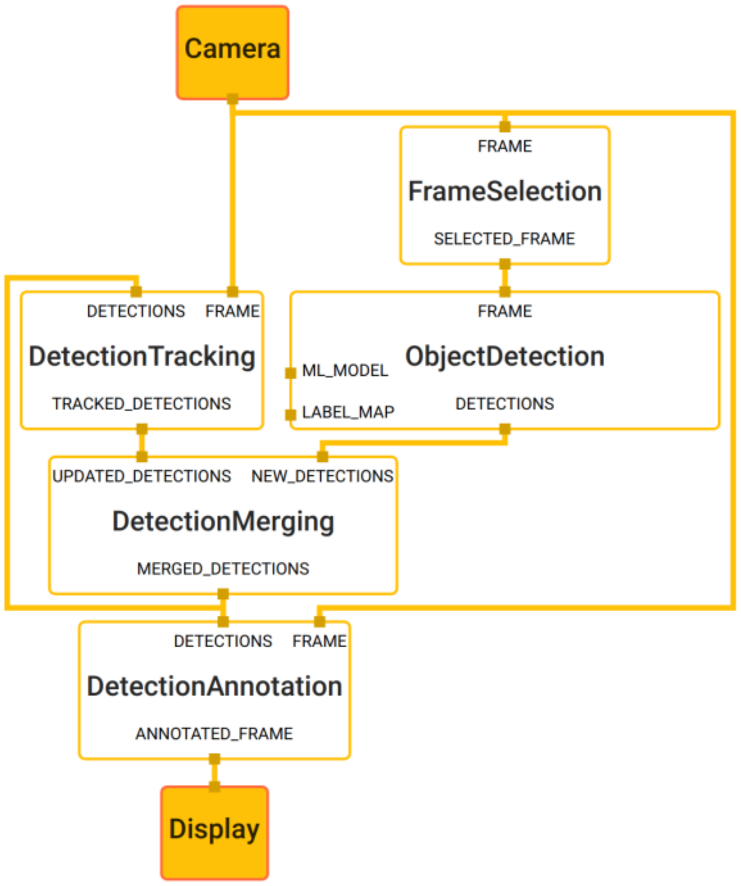


Figure 2.2: Object Detection using MediaPipe library

MediaPipe also allows configurations for these pipelines for incremental improvements and evaluation [22].

The main parts of *MediaPipe* include a framework for sensory inference, tools for evaluating the performance of the perception pipelines, and calculators, which are the reusable inference and processing components [22].

This thesis will focus on the ability of *MediaPipe* to detect and construct a complete collection of landmarks for human hands, track them to create images of traces, and pass the images to the Convolutional Neural Network for classification.

The architecture of the hand-tracking ability of *MediaPipe* includes 2 models working parallelly: a palm detector that locates human hands using an oriented hand bounding box for a full input image and a hand landmark model that inputs the hand bounding box and returns a high-fidelity 2.5D landmarks [23].

For palm detection, *MediaPipe* deploys a single-shot detector. Instead of detecting a full hand, the *MediaPipe* model detects the palm only [23]. Using this strategy, *MediaPipe* can reduce the number of detecting anchors in a model by 3 to 5 times compared to the traditional method of detecting hands with fingers. The reason for this significant improvement is that estimating bounding box of palms is simplier as they are smaller and contain fewer details [23].

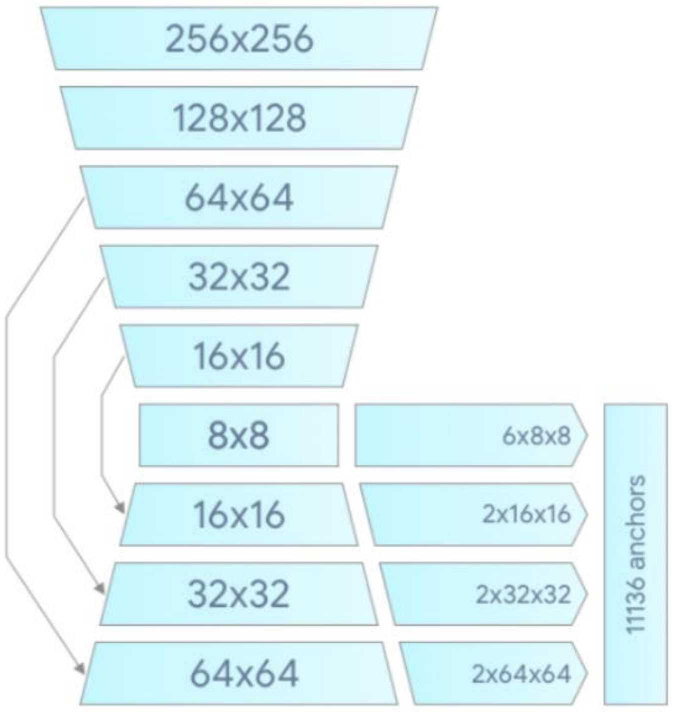
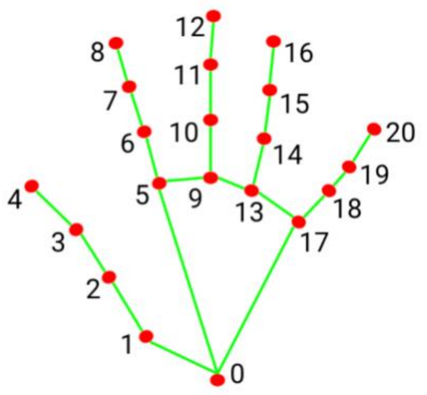


Figure 2.3: Palm detector model architecture in MediaPipe

After hand detection, the hand landmarks model localizes 21 coordinates inside the detected region using regression and returns 3 outputs: 21 landmarks of the hands, including x-position, y-position and the relative depth, a probability variable for checking whether there is hand presence in the image, and a binary variable for checking if the input hand is left or right [23].



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 2.4: List of 21 landmarks on a hand used in MediaPipe

Following is an example of hand detection by *MediaPipe*. The backgrounds of those examples are different, which proves the detecting ability of *MediaPipe*:

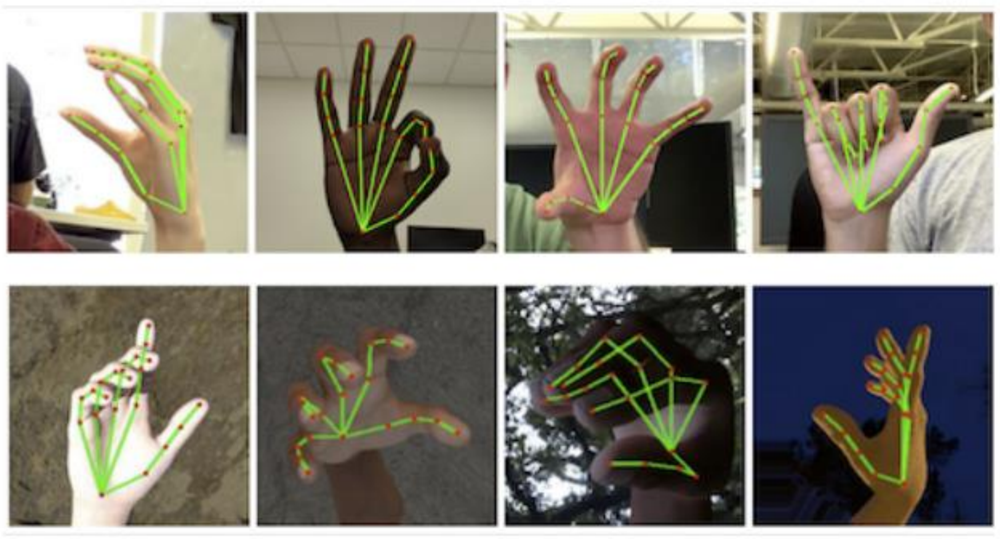


Figure 2.5: Example of hand detection by MediaPipe in different backgrounds

The strongest point of *MediaPipe* is that thanks to the idea of using palm to detect human hands, this library can reduce the cost in both time and space. This breakthrough makes *MediaPipe* light, quick, and efficient enough to be applied to commercial appliances.

2.3.2 TENSORFLOW

TensorFlow is a library developed by Google researchers to assist users in creating ready-to-use, quick, and convenient CNN models by utilizing pre-defined classes of various layers of CNN structure.

Tensorflow is a complete machine-learning platform that includes data preparation, model development, fine-tuning, deployment, and maintenance.

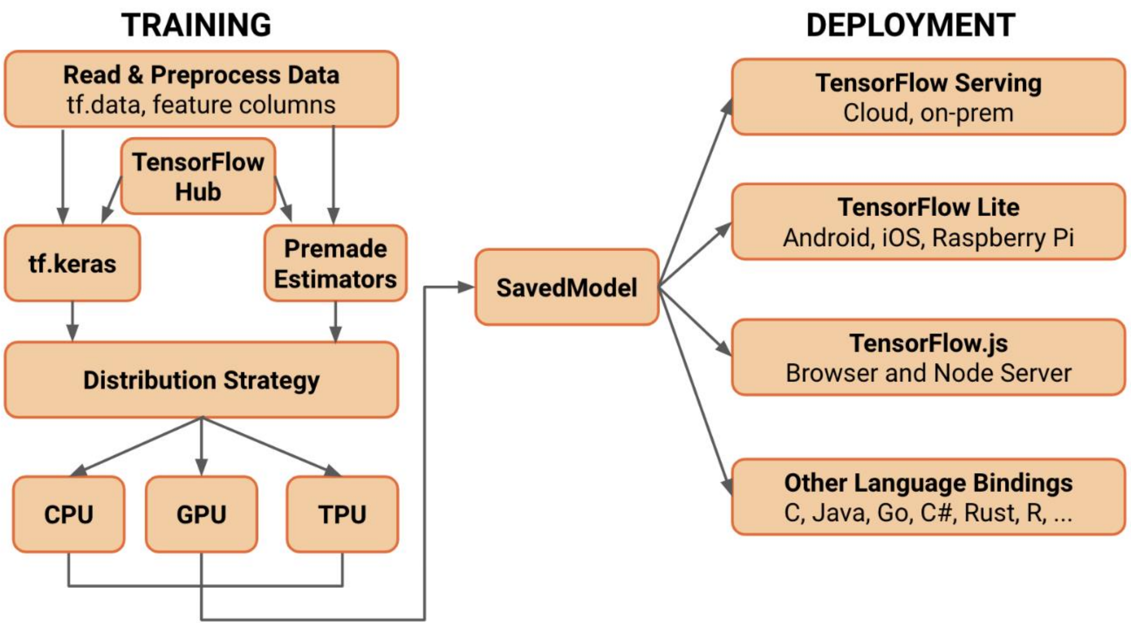
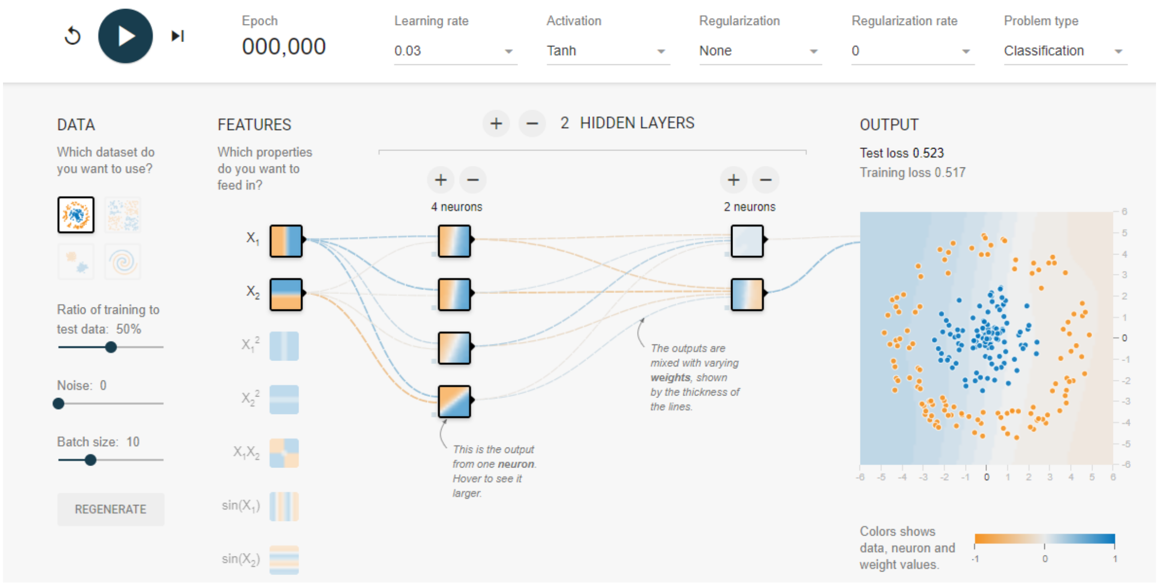


Figure 2.6: Tensorflow core functions structure

In this thesis, Tensorflow will be implemented as the model training and light deployment for further development.



Epoch 000,000
Learning rate 0.03
Activation Tanh
Regularization None
Regularization rate 0
Problem type Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 0
Batch size: 10
REGENERATE

FEATURES
Which properties do you want to feed in?

2 HIDDEN LAYERS
4 neurons
2 neurons

This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

OUTPUT
Test loss 0.523
Training loss 0.517

Colors shows data, neuron and weight values.

Figure 2.7: An example of Tensorflow model visualization

2.3.3 OPENCV

OpenCV is an open-source computer vision library that includes over 2500 distinct algorithms. It is managed by a global community of data scientists and routinely updated with cutting-edge methodologies in all aspects of computer vision.

Figure 2.8: Corporations in roles of members of OpenCV community

In this thesis, an interface will be created using OpenCV. The main abilities of this interface are to record the hand landmarks as the training dataset, display the landmarks on the computer or other screens for reference, and display the detected commands from the prediction by the model.

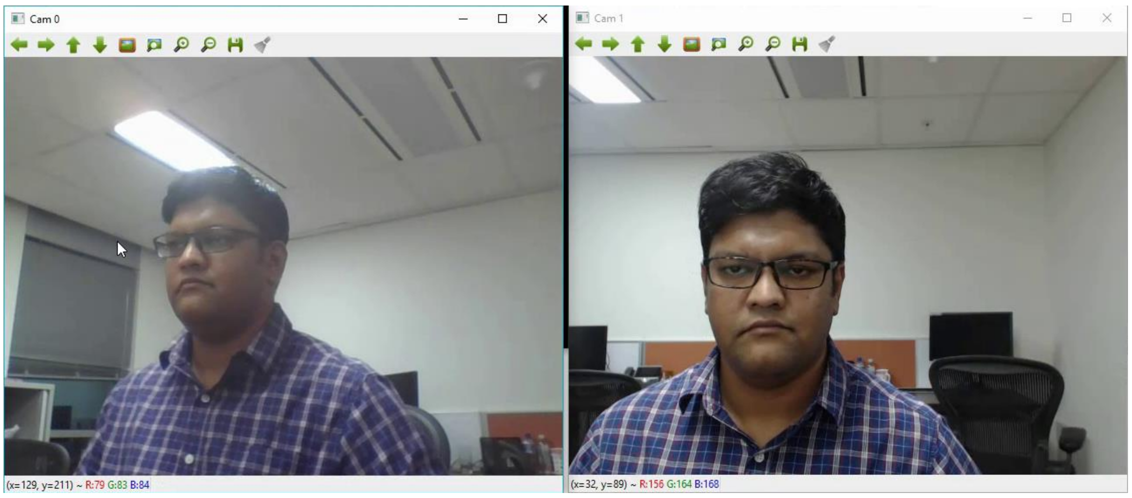

Cam 0
(x=129, y=211) ~ R:79 G:83 B:84
Cam 1
(x=32, y=89) ~ R:156 G:164 B:168

Figure 2.9: An example of OpenCV's ability to capture images using cameras

Further details about implementing OpenCV will be discussed and provided later in the Implementation section.

2.3.4 ACTIVATION FUNCTIONS

This section will discuss different activation functions in the neural networks before being applied to this thesis.

ELU

ELU, or *Exponential Linear Unit*, is an activation function that uses exponential calculation. Unlike *ReLU*, *ELU* have the case of negative, which helps it to draw the mean unit activation closer to 0 [24]. Following is the formula of *ELU*:

$$ELU(x) = \begin{cases} x & if\ x > 0 \\ \alpha(\exp(x) - 1) & if\ x \leq 0 \end{cases} \quad (4)$$

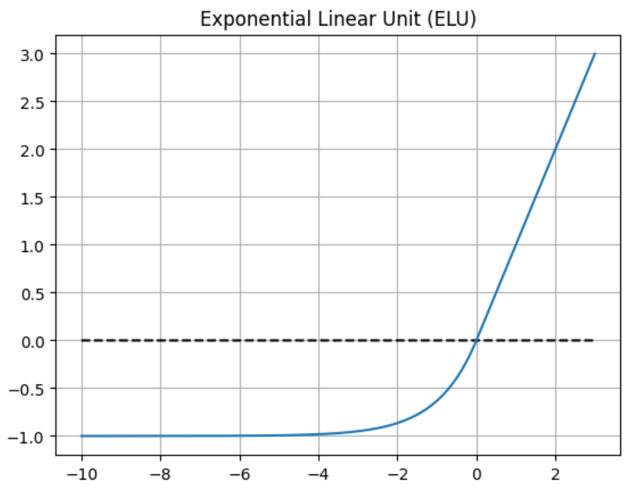


Figure 2.10: Distribution of ELU algorithm

SELU

SELU, or *Scaled Exponential Linear Unit*, is a scaled version of *ELU*, which makes it to be able to self-normalize [25]. Following is the formula of *SELU*:

$$SELU(x) = \lambda \times \begin{cases} x & if\ x > 0 \\ \alpha \times \exp(x) - \alpha & if\ x \leq 0 \end{cases} \quad (5)$$

with $\lambda \approx 1.0507$ and $\alpha \approx 1.6733$

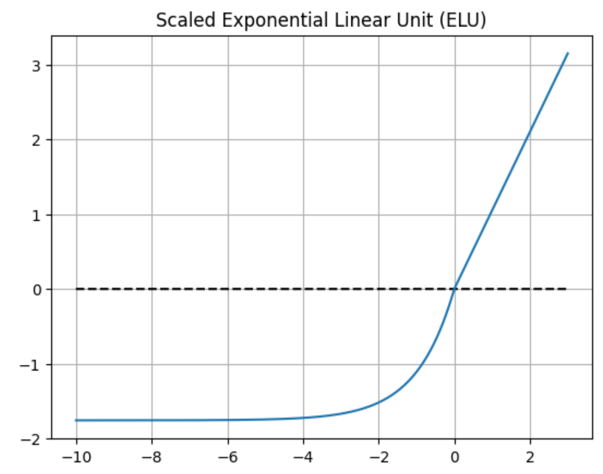


Figure 2.11: Distribution of the SELU algorithm

ReLU

ReLU, or *Rectified Linear Unit*, is one of the most popular activation functions in the Convolutional Neural Network [26]. The concept of *ReLU* is to limit the lower bound of the result to 0, otherwise it returns the input values. It is wisely chosen because of the simple-but-effective characteristic. Following is the formula of *ReLU*:

$$ReLU(x) = \begin{cases} x & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases} \qquad (6)$$

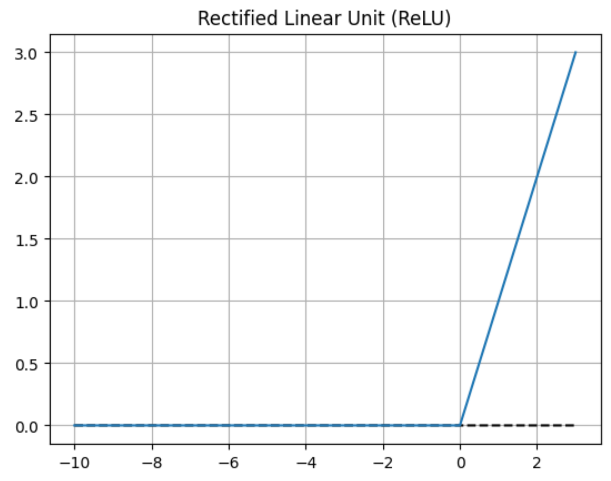


Figure 2.12: Distribution of ReLU algorithm

LReLU

LReLU (*Leaky Rectified Linear Unit*) is a variant of *ReLU*, where a slope α, usually 0.2, is multiplied with the negative values of x [27]. Following is the formula of *LReLU*:

$$LReLU(x) = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases} \tag{7}$$

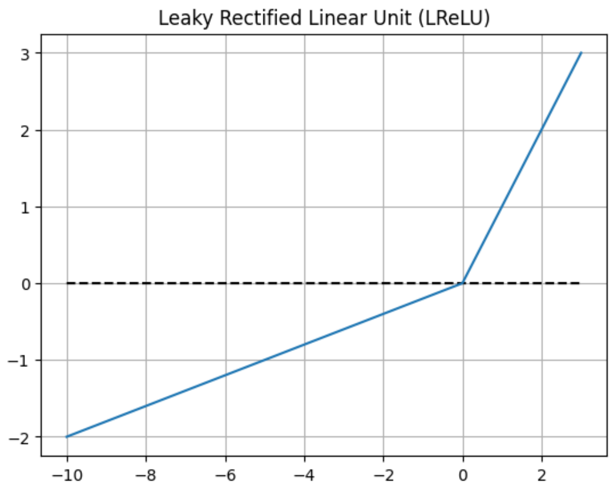


Figure 2.13: Distribution of LReLU algorithm

RReLU

RReLU, or *Randomized Rectified Linear Unit*, is a variant of *ReLU* where a random slope α from a uniform random distribution $U(lower, upper)$ is multiplied with the negative values of x [28]. Following is the formula of *RReLU*:

$$RReLU(x) = \begin{cases} x & if\ x > 0 \\ \alpha x & if\ x \leq 0 \end{cases} \tag{8}$$

with $\alpha = U(lower, upper), lower < upper\ \&\ lower, upper\ \in\ [0,1)$

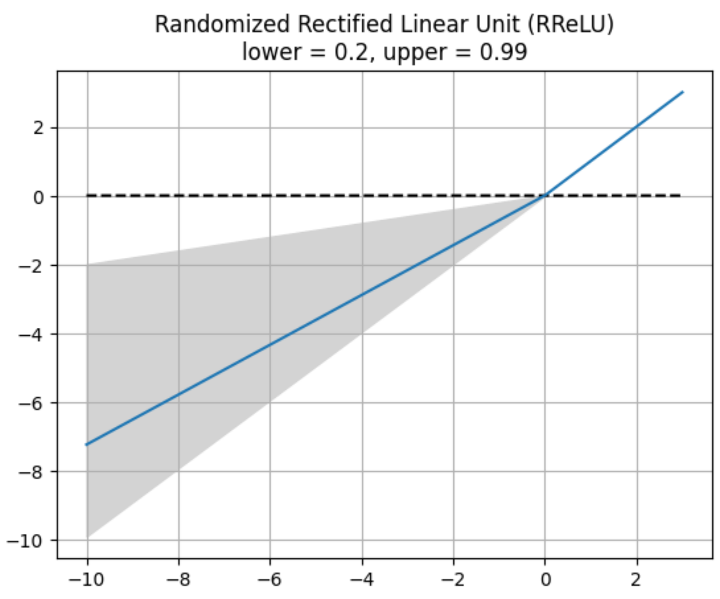


Figure 2.14: Distribution of RReLU algorithm, lower = 0.2, upper = 0.99

SReLU

SReLU, or *S-shaped Rectified Linear Unit*, is a variant of *ReLU*. This function is developed based on Weber-Fechner law and Stevens's power law. The distribution of this function is divided into 3 different sections with 4 parameters: t_l and α_l for the left section, t_r and α_r for the right section [29]. Following is the formula of *SReLU*:

$$SReLU(x) = \begin{cases} t_l + \alpha_l \times (x - t_l) \; if \; x \leq t_l \\ x \; if \; t_l < x < t_r \\ t_r + \alpha_r \times (x - t_r) \; if \; x \geq t_r \end{cases} \tag{9}$$

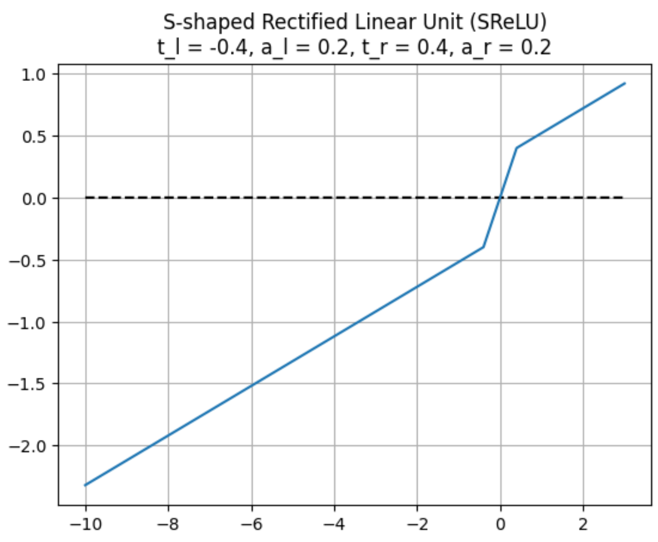


Figure 2.15: Distribution of SReLU algorithm, $t_l = -0.4, a_l = 0.2, t_r = 0.4, a_r = 0.2$

Sigmoid

Sigmoid function is an activation function with the ability to transform any value in the domain $(-\infty, \infty)$ to a number in the range of $[0,1]$ [30]. Following is the formula of *Sigmoid* function:

$$sigmoid(x) = \frac{1}{1-\exp(x)} \tag{10}$$

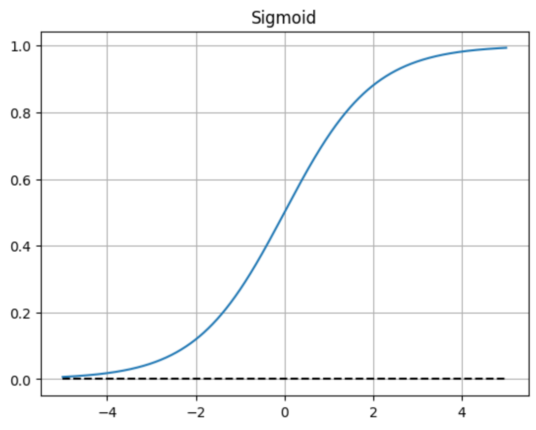


Figure 2.16: Distribution of Sigmoid function

Tanh

Tanh, or *Hyperbolic Tangent* function is an activation function that uses the formula of $tanh$ for calculating the activation threshold. *Tanh* has the output of $[-1, 1]$ for all input $x \in (-\infty, \infty)$ [31]. Following is the formula of *Tanh* function:

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{11}$$

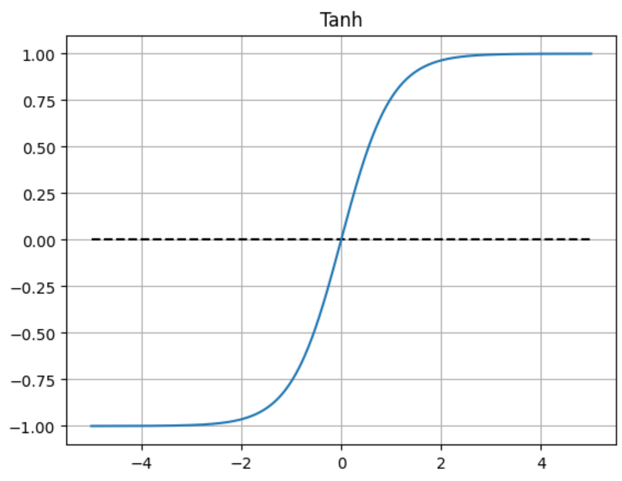


Figure 2.17: Distribution of Tanh function

Softmax Regression

Unlike the mentioned functions, *Softmax Regression* is not an activation function but an output classification function. Given a map of input from the previous layer, *Softmax Regression* will result in the class with the highest likelihood [32]. Following is the formula of *Softmax Regression* function:

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{C} \exp(x_j)}, \forall i = 1, 2, \dots, C \tag{12}$$

with C is the number of output classes.

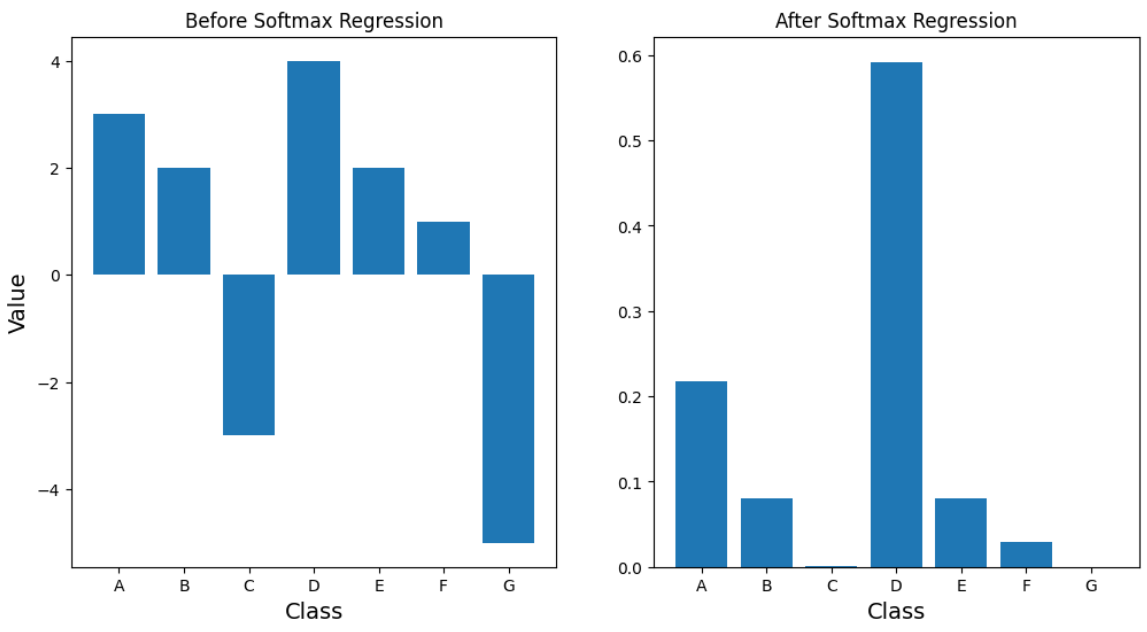


Figure 2.18: Values of classes before and after applying Softmax Regression

CHAPTER 3 – METHODOLOGY

3.1 DATA INFORMATION

The dataset used in this thesis is a collection of vectors with a length of 43. The first value is the label of this vector, followed by 42 additional values, separated into 21 pairs of longitude and latitude of the images, indicating the identified hand landmarks.

To provide the customization ability to this thesis, all of the vectors in this dataset is created and recorded by the users. Not only the labels and commands for each action, the actions themselves can also be customized in users' preferences.

Currently, the dataset used as the demo for the training model in this project consists of 5 actions, including thumb up, thumb down, open, close, and point, with more than 500 vectors for each action.

3.2 GENERAL PIPELINE

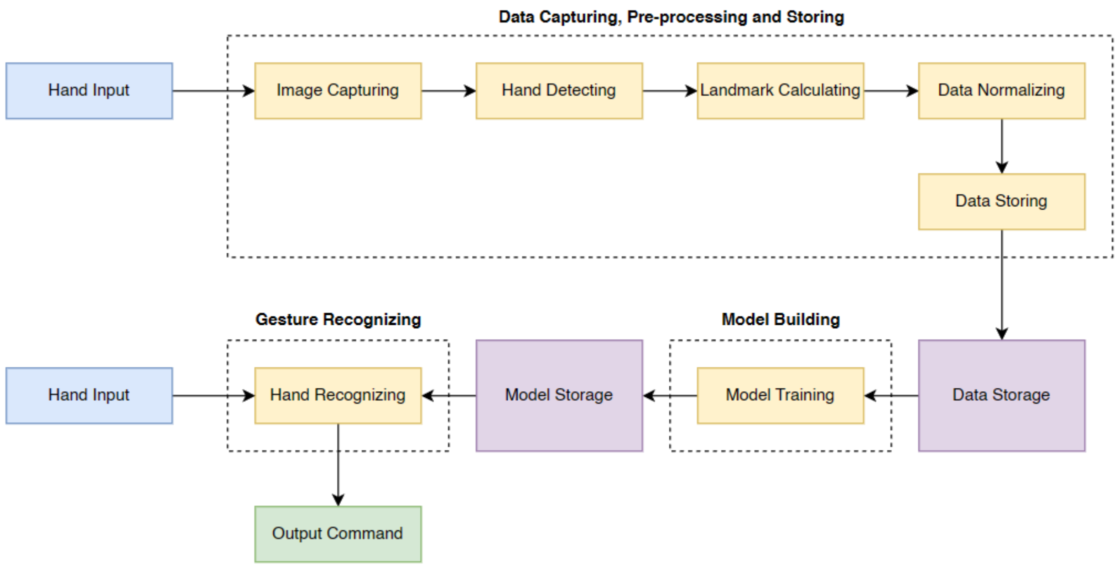


Figure 3.1: General Pipeline of Thesis Application

This thesis application is divided into 3 different sections. Each section has its own responsibilities for data capturing and handling, model building and gesture recognizing.

For data capturing and handling, this section includes capturing the images from users' cameras, detecting whether any hand appearances, calculating the landmark positions for hand skeletal structures as well as normalizing and storing them into a pre-defined storage for data.

For model building section, this is where the CNN model is created with the training data from previous steps. Also, the performance of this model is recorded for any further references. The model is then saved into its separate storage.

For the gesture recognizing section, an interface for user to interact with is built with the purpose of recognizing, classifying users' gestures and returning the result in form of user-defined commands.

Detailed information for those 3 stages are being discussed later in the next part.

3.3 DATA CAPTURING AND PRE-PROCESSING

3.3.1 DATA FEATURES

In this dataset, 2546 vectors of float numbers with the length of each vector of 43 will be preprocessed and brought into the model built with Tensorflow as the training data. Each vector is recorded by users' order and pre-formatted before being written into the dataset. It is ready-to-use without any other pre-processing steps required.

Following is an example of an image with landmarks in the dataset:

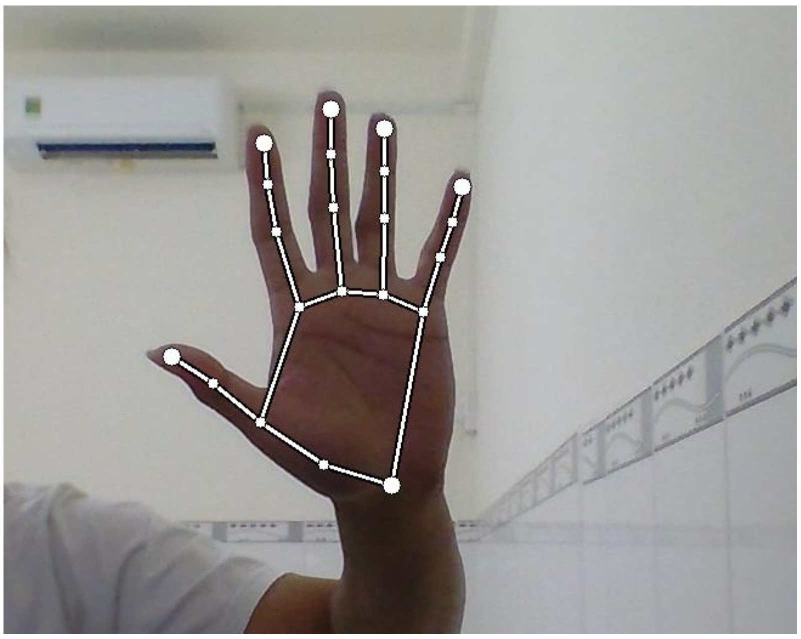
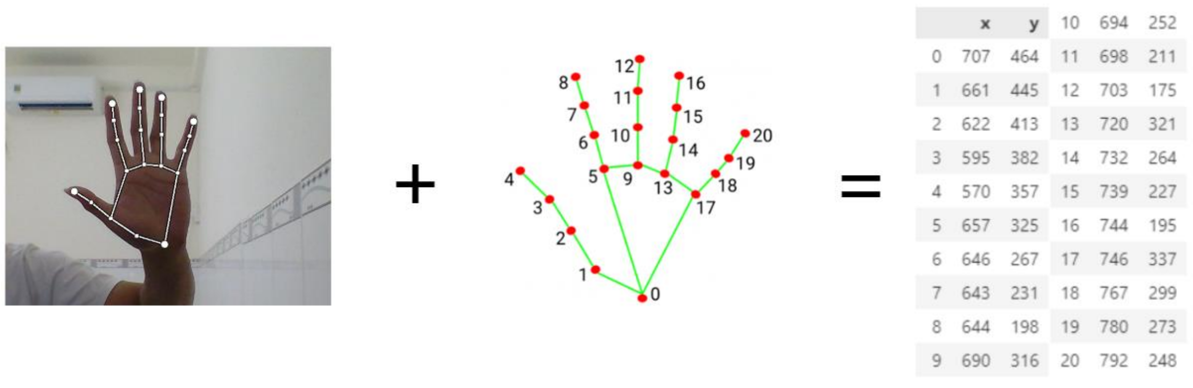


Figure 3.2: An example of hand landmarks recorded in the dataset

From those images, longitudes and latitudes of the hands are going to be extracted and processed in order to be ready for training and testing.

3.3.2 DATA CAPTURING

For each white “dot” as shown above, the location of them are captured and put into a list, which results in a set of longitudes and latitudes of all 21 dots on users’ hand. The numbers recorded are the distances from the top left corner of the camera to the respective location in the unit of pixels. With those positions, a skeleton structure of users’ hand is recorded accurately:



	x	y
0	707	464
1	661	445
2	622	413
3	595	382
4	570	357
5	657	325
6	646	267
7	643	231
8	644	198
9	690	316
10	694	252
11	698	211
12	703	175
13	720	321
14	732	264
15	739	227
16	744	195
17	746	337
18	767	299
19	780	273
20	792	248

Figure 3.3: Combination of image and landmarks map for position

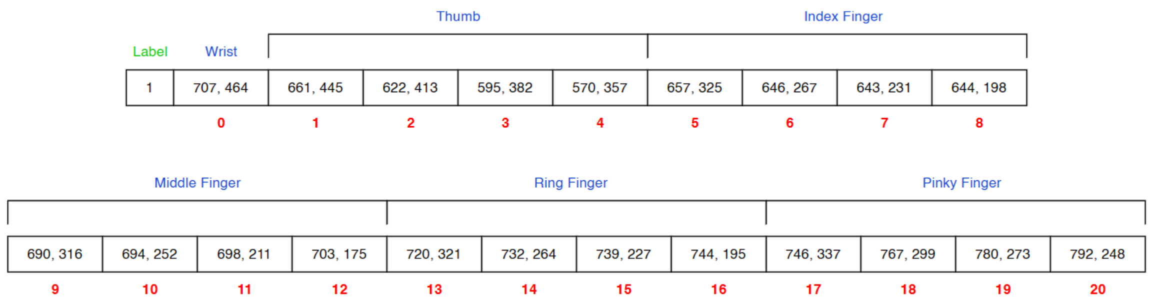
This capturing process is done with the help of MediaPipe as this library provides a high-accurate and quick capturing function.

After capturing, this set of longitudes and latitudes is then transformed into a vector of number and is ready for next step.

3.3.3 DATA NORMALIZING

Although the gestures may be the same, different locations of hand may lead to an enormous number of possibilities for the same action. Moreover, as those landmark positions are in form of pixel location, those big number may lead to high computational cost. As a result, a method of normalizing is applied to solve the above problems.

With this method of normalizing, all the longitudes and latitudes are converted to be in the range from -1 to 1, which reduces the cost for computing significantly.



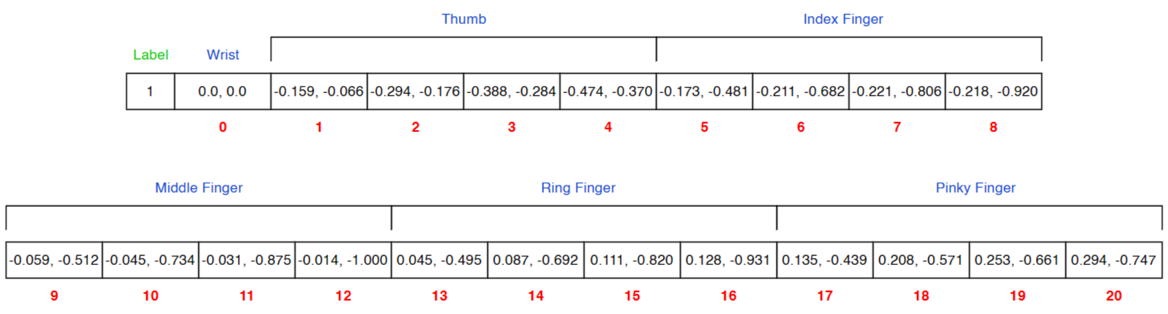
Label	Wrist	Thumb				Index Finger			
1	707, 464	661, 445	622, 413	595, 382	570, 357	657, 325	646, 267	643, 231	644, 198
	0	1	2	3	4	5	6	7	8

Middle Finger				Ring Finger				Pinky Finger			
690, 316	694, 252	698, 211	703, 175	720, 321	732, 264	739, 227	744, 195	746, 337	767, 299	780, 273	792, 248
9	10	11	12	13	14	15	16	17	18	19	20

Figure 3.4: Positions of landmarks in previous example – before processing

This method of normalization is based on the maximum distance between the root location – in this case is the wrist of user – to other position on the hand. Each of the position is then divided by this maximum value, therefore guaranteed to have normalized value between -1 and 1.

The biggest advantage of this method is that, no matter where users' hand is, the longitudes and latitudes of those landmarks will always in the desired range of value, which can help increase the overall performance of the Neural Network model later then.



Label	Wrist	Thumb				Index Finger			
1	0.0, 0.0	-0.159, -0.066	-0.294, -0.176	-0.388, -0.284	-0.474, -0.370	-0.173, -0.481	-0.211, -0.682	-0.221, -0.806	-0.218, -0.920
	0	1	2	3	4	5	6	7	8

Middle Finger				Ring Finger				Pinky Finger			
-0.059, -0.512	-0.045, -0.734	-0.031, -0.875	-0.014, -1.000	0.045, -0.495	0.087, -0.692	0.111, -0.820	0.128, -0.931	0.135, -0.439	0.208, -0.571	0.253, -0.661	0.294, -0.747
9	10	11	12	13	14	15	16	17	18	19	20

Figure 3.5: Positions of landmarks in previous example – after processing

3.3.4 DATA SPLITTING

As a common knowledge, in order to build a CNN model, or any models in general, 3 different segments of data are required. Those are the training, validating and testing dataset.

In this thesis application, the dataset is split with the rule of 7:1:2, which means 70% of the dataset is used for training, 10% of the dataset for validating and other 20% for testing. By using this ratio, the patterns of the general dataset have the higher chance of appearing in the training dataset, while the testing dataset also guarantees the fairness of performance.

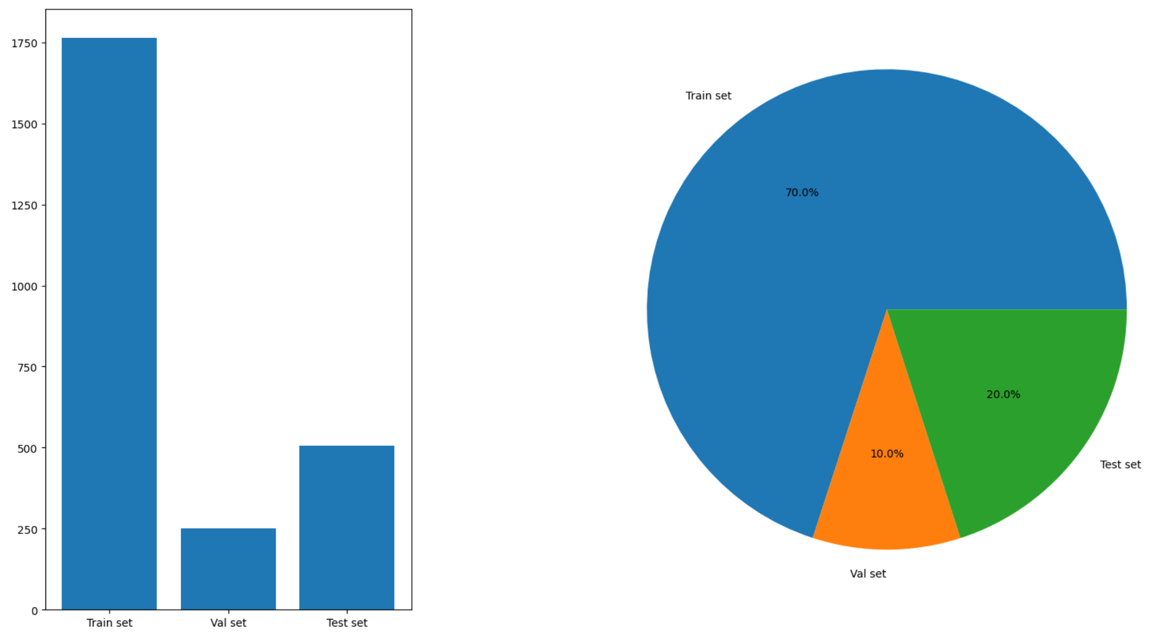


Figure 3.6: Sizes of training, validating and testing sections of the dataset

This splitting section is automatically done using the *train_test_split* method from the *Scikit-learn* library. This is a function that is widely used to divide training, validating and testing dataset for models as with the stratified strategy in splitting, the overall distribution of labels remains unchanged.

3.4 MODEL BUILDING

In this section, a complete model is created with the main purpose is to classify the input in form of human gestures into the right label for further usage.

As mentioned above, a Convolutional Neural Network (CNN) model structure is defined and used to fit the training data.

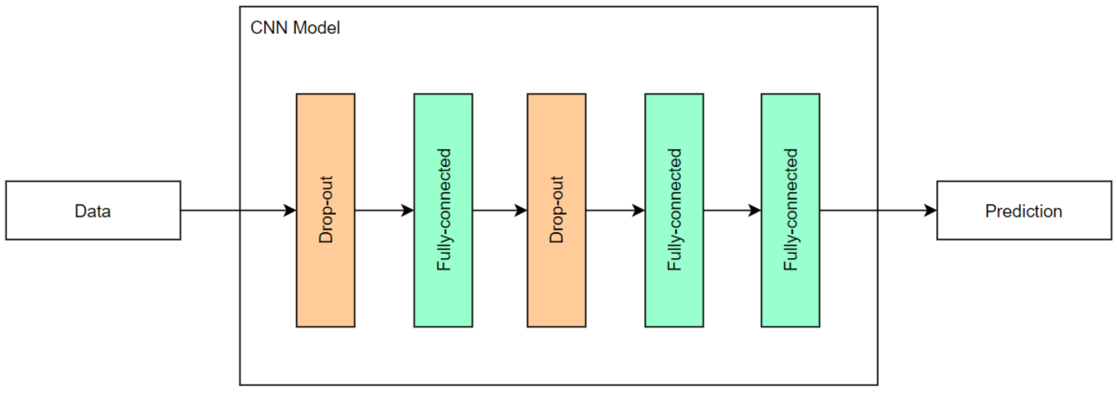


Figure 3.7: Structure of CNN model for thesis application

Different structures and types of layers were experienced in order to find out the structure with greatest performance. After those experiments, the final model consists of 5 layers, including 2 drop-out layers and 3 dense (fully-connected) layers. Although this model is relatively simple, its performance is out of expectation with all the metrics for accuracy/precision/bias and others.

The performance of this model will be carefully discussed in details in the Implementation section.

3.5 GESTURE RECOGNIZING

This final section of thesis application is where users interact with the model. As mentioned above, this section has the main goal of detecting the appearance of hands using the cameras as well as predicting the label for the detected gesture.

For the model used in this section, it is the result of the previous section of model building, therefore, the accuracy of this model is guaranteed with low error and bias.

Also, both the interface and model are well-optimized in order to reduce the complexity of running, from model loading to predicting. With those improvements, this section may achieve all the requirements for a lightweight-but-accurate model for hand gesture recognition.

CHAPTER 4 – IMPLEMENTATION

4.1 DATA CAPTURING

Instead of utilizing a prepackaged dataset downloaded from the Internet, the model in this thesis is trained using the users' data. This strategy ensures flexibility while protecting users' privacy by forcing users to enter their training data but not sending any information out of their data storage.

To construct this dataset, OpenCV will develop a user interface in the form of a camera for recording, pre-processing, and storing data. The overall steps of this procedure are:

First of all, users open the logging mode from the interface and prepare the gestures of their preference. By following this action, a skeletal structure of their hand may appear on the interface.

Next, users press one of the ten number keys on their keyboard. Each number stands for a specific action only. For each action, a following label and command will also need to be updated for the application to refer.

Then, MediaPipe will capture the skeletal structure from the interface and transform into 21 landmark locations. Behind the scenes, those locations are processed into the correct format for consistency with normalization as mentioned in the previous chapter.

After pre-processing, those locations are written down into the data storage as a new line in form of a vector. Now the actions from users are recorded and ready to be used to train, validate, and test the model.

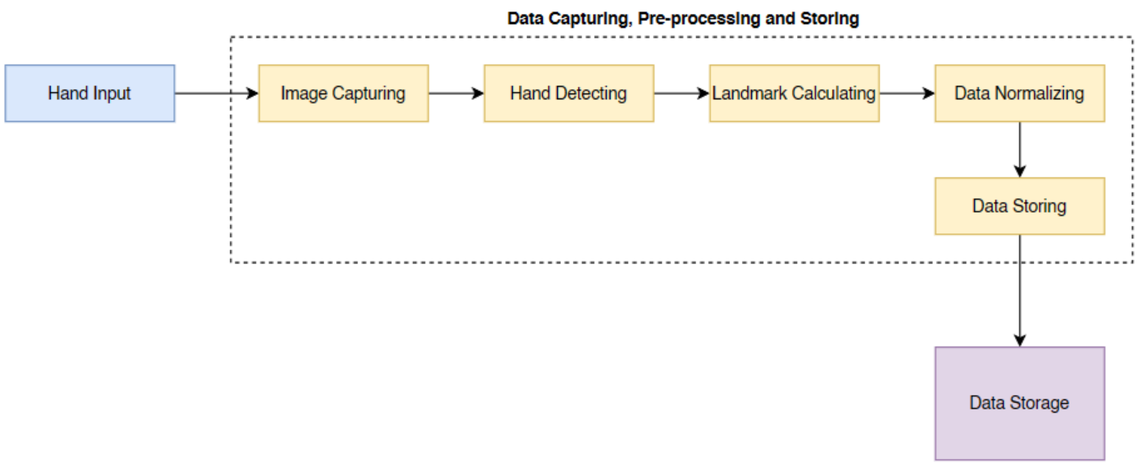


Figure 4.1: Procedure of capturing new data from user

For the detailed process, the above steps are done with the following actions:

Users open the data record mode and prepare their hands to ensure that their hands are on the camera. Using OpenCV, this thesis develops an interface that allows users to view themselves on screen. Additionally, users prepare their hands to assist the interface in detecting their hands and preparing for the subsequent stages.

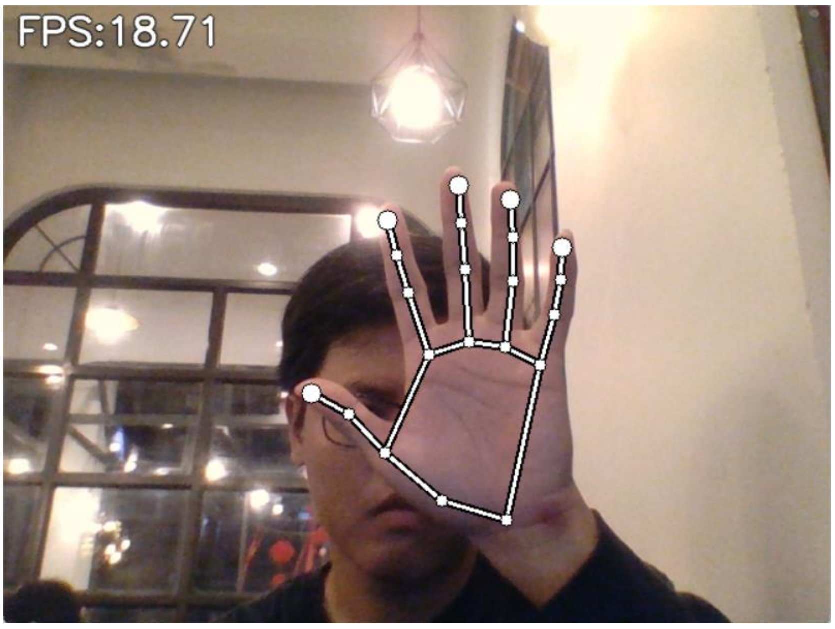


Figure 4.2: An example of the interface detecting users' hand.

Users shape their hand in any shape of their preference. Users can shape their hand in any shape of their liking, after pre-processing and training, each of their action will be the command to call an action of the appliances.

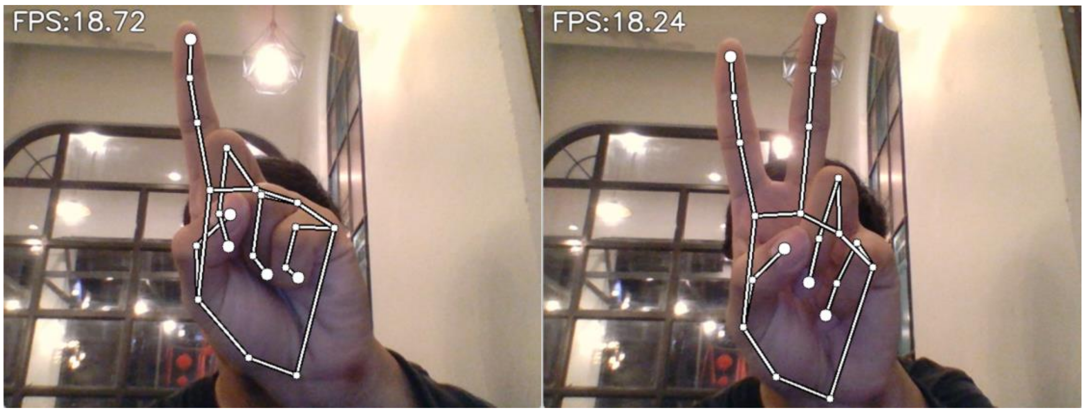
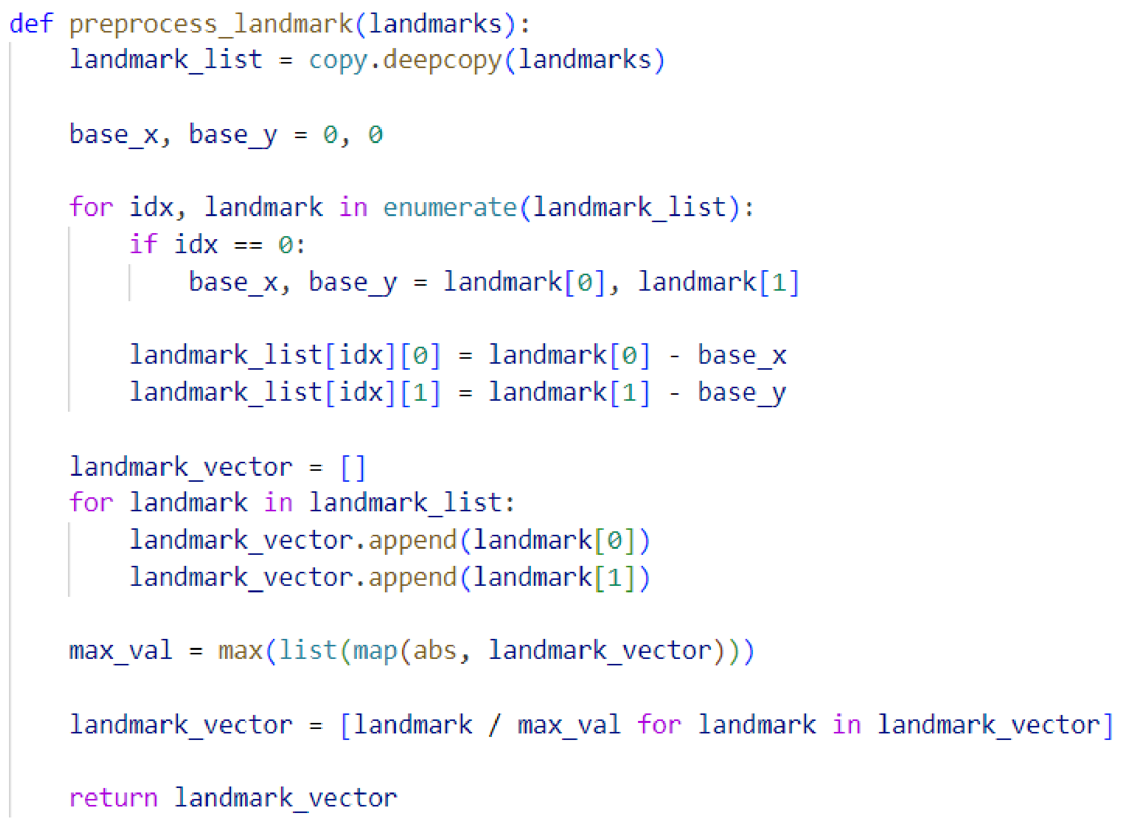


Figure 4.3: Examples of shapes from the user

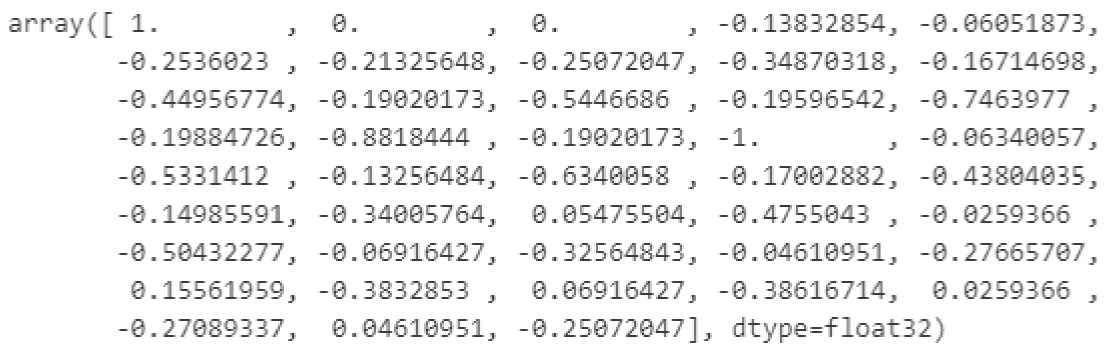
Users press one of the ten number keys on the keyboard. Each number stands for a specific action only. Once users have shaped their hand, they may press the number keys on the keyboard from 0 to 9 to record the position of landmarks from the hand in the form of a vector with 43 values, including the key pushed and the longitude and latitude of 21 landmarks. Each number will be linked to an action using a mapping file in the data store.

Their actions are recorded and pre-processed. After recording the location vectors of the landmarks, they will be normalized. Let the palm of the user's hand be the base position. The positions of additional landmarks are determined by the difference in longitude and latitude from the palm. Then, all values in the landmark vector are divided by the greatest determined distance. These are normalized distances, which will be added to the dataset for training. Following is the code to pre-process the landmark records:



```python
def preprocess_landmark(landmarks):
    landmark_list = copy.deepcopy(landmarks)

    base_x, base_y = 0, 0

    for idx, landmark in enumerate(landmark_list):
        if idx == 0:
            base_x, base_y = landmark[0], landmark[1]

        landmark_list[idx][0] = landmark[0] - base_x
        landmark_list[idx][1] = landmark[1] - base_y

    landmark_vector = []
    for landmark in landmark_list:
        landmark_vector.append(landmark[0])
        landmark_vector.append(landmark[1])

    max_val = max(list(map(abs, landmark_vector)))

    landmark_vector = [landmark / max_val for landmark in landmark_vector]

    return landmark_vector
```

Figure 4.4: Code of Landmarks Normalization

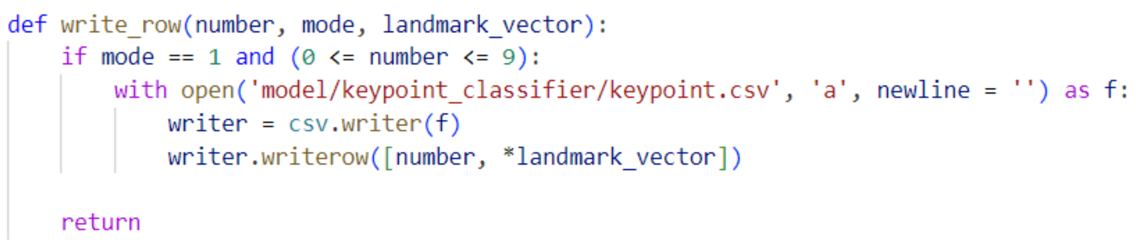


```
array([ 1.        ,  0.        ,  0.        , -0.13832854, -0.06051873,
       -0.2536023 , -0.21325648, -0.25072047, -0.34870318, -0.16714698,
       -0.44956774, -0.19020173, -0.5446686 , -0.19596542, -0.7463977 ,
       -0.19884726, -0.8818444 , -0.19020173, -1.        , -0.06340057,
       -0.5331412 , -0.13256484, -0.6340058 , -0.17002882, -0.43804035,
       -0.14985591, -0.34005764,  0.05475504, -0.4755043 , -0.0259366 ,
       -0.50432277, -0.06916427, -0.32564843, -0.04610951, -0.27665707,
        0.15561959, -0.3832853 ,  0.06916427, -0.38616714,  0.0259366 ,
       -0.27089337,  0.04610951, -0.25072047], dtype=float32)
```

Figure 4.5: An Example of Landmark Vector after pre-processing

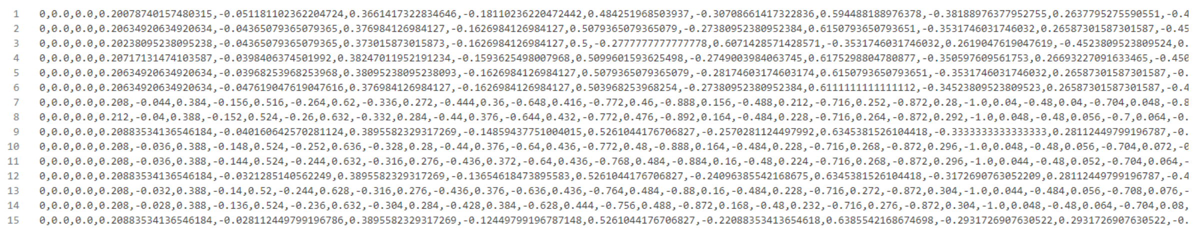
After finishing the pre-processing step, the landmark vector is written down as a new line in the data storage file, where all the records are stored, along with the number for its label.

Following is the code to write the record into the storage file:



```python
def write_row(number, mode, landmark_vector):
    if mode == 1 and (0 <= number <= 9):
        with open('model/keypoint_classifier/keypoint.csv', 'a', newline = '') as f:
            writer = csv.writer(f)
            writer.writerow([number, *landmark_vector])

    return
```

Figure 4.6: Code to Write the Landmark Vector to Data Storage



```
0,0.0,0.0,0.20078740157480315,-0.05118110236220472 4,0.3661417322834646,-0.18110236220472442,0.484251968503937,-0.30708661417322836,0.594488188976378,-0.38188976377952755,0.2637795275590551,-0.4
0,0.0,0.0,0.20634920634920634,-0.04365079365079365,0.376984126984127,-0.1626984126984127,0.5079365079365079,-0.27380952380952384,0.6150793650793651,-0.3531746031746032,0.26587301587301587,-0.45
0,0.0,0.0,0.20238095238095238,-0.04365079365079365,0.373015873015873,-0.1626984126984127,0.5,-0.2777777777777778,0.6071428571428571,-0.3531746031746032,0.2619047619047619,-0.4523809523809524,0.
0,0.0,0.0,0.20717131474103587,-0.0398406374501992,0.38247011952191234,-0.1593625498007968,0.5099601593625498,-0.2749003984063745,0.6175298804780877,-0.350597609561753,0.26693227091633465,-0.450
0,0.0,0.0,0.20634920634920634,-0.03968253968253968,0.38095238095238093,-0.1626984126984127,0.5079365079365079,-0.28174603174603174,0.6150793650793651,-0.3531746031746032,0.26587301587301587,-0.
0,0.0,0.0,0.20634920634920634,-0.047619047619047616,0.376984126984127,-0.1626984126984127,0.503968253968254,-0.27380952380952384,0.6111111111111112,-0.34523809523809523,0.26587301587301587,-0.4
0,0.0,0.0,0.208,-0.044,0.384,-0.156,0.516,-0.264,0.62,-0.336,0.272,-0.444,0.36,-0.648,0.416,-0.772,0.46,-0.888,0.156,-0.488,0.212,-0.716,0.252,-0.872,0.28,-1.0,0.04,-0.48,0.04,-0.704,0.048,-0.8
0,0.0,0.0,0.212,-0.04,0.388,-0.152,0.524,-0.26,0.632,-0.332,0.284,-0.44,0.376,-0.644,0.432,-0.772,0.476,-0.892,0.164,-0.484,0.228,-0.716,0.264,-0.872,0.292,-1.0,0.048,-0.48,0.056,-0.7,0.064,-0.
0,0.0,0.0,0.20883534136546184,-0.040160642570281124,0.3895582329317269,-0.14859437751004015,0.5261044176706827,-0.2570281124497992,0.6345381526104418,-0.3333333333333333,0.28112449799196787,-0.
0,0.0,0.0,0.208,-0.036,0.388,-0.148,0.524,-0.252,0.636,-0.328,0.28,-0.44,0.376,-0.64,0.436,-0.772,0.48,-0.888,0.164,-0.484,0.228,-0.716,0.268,-0.872,0.296,-1.0,0.048,-0.48,0.056,-0.704,0.072,-0
0,0.0,0.0,0.208,-0.036,0.388,-0.144,0.524,-0.244,0.632,-0.316,0.276,-0.436,0.372,-0.64,0.436,-0.768,0.484,-0.884,0.16,-0.48,0.224,-0.716,0.268,-0.872,0.296,-1.0,0.044,-0.48,0.052,-0.704,0.064,-
0,0.0,0.0,0.20883534136546184,-0.0321285140562249,0.3895582329317269,-0.13654618473895583,0.5261044176706827,-0.24096385542168675,0.6345381526104418,-0.3172690763052209,0.28112449799196787,-0.4
0,0.0,0.0,0.208,-0.032,0.388,-0.14,0.52,-0.244,0.628,-0.316,0.276,-0.436,0.376,-0.636,0.436,-0.764,0.484,-0.88,0.16,-0.484,0.228,-0.716,0.272,-0.872,0.304,-1.0,0.044,-0.484,0.056,-0.708,0.076,-
0,0.0,0.0,0.208,-0.028,0.388,-0.136,0.524,-0.236,0.632,-0.304,0.284,-0.428,0.384,-0.628,0.444,-0.756,0.488,-0.872,0.168,-0.48,0.232,-0.716,0.276,-0.872,0.304,-1.0,0.048,-0.48,0.064,-0.704,0.08,
0,0.0,0.0,0.20883534136546184,-0.028112449799196786,0.3895582329317269,-0.12449799196787148,0.5261044176706827,-0.22088353413654618,0.6385542168674698,-0.2931726907630522,0.2931726907630522,-0.
```

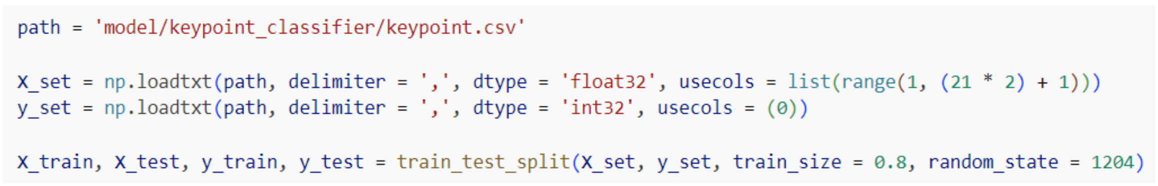
Figure 4.7: A Part of Data Storage

4.2 MODEL TRAINING AND STORING

In this section, a CNN model for classification is created by using the data recorded, pre-processed, and stored after the above steps, along with TensorFlow – a library for training neural networks.

In this section, the steps and the structure of this CNN model are discussed in detail:

Data loading and splitting: for loading the data from the storage, *loadtxt* function from *numpy* is implemented as this function will help read the data without any creation or requirements of columns. Also, loadtxt helps users choose the element positions they want to read instead of loading everything, which can decrease the memory needed as well as increase the read speed. For splitting the data into training and testing sets, *train_test_split* function from *scikit-learn* is chosen as this function is easy to use and provides some useful parameters such as *train_size* or *random_seed*.

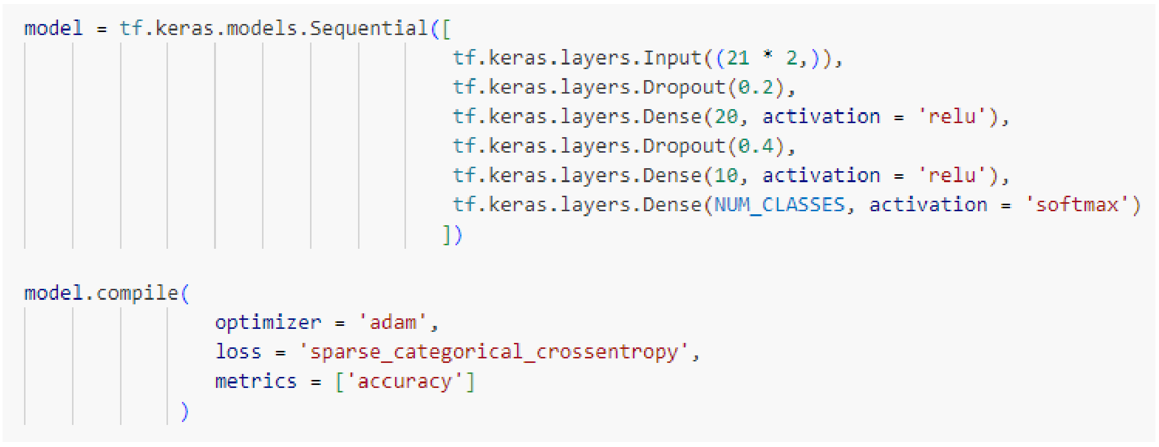


```python
path = 'model/keypoint_classifier/keypoint.csv'

X_set = np.loadtxt(path, delimiter = ',', dtype = 'float32', usecols = list(range(1, (21 * 2) + 1)))
y_set = np.loadtxt(path, delimiter = ',', dtype = 'int32', usecols = (0))

X_train, X_test, y_train, y_test = train_test_split(X_set, y_set, train_size = 0.8, random_state = 1204)
```

Figure 4.8: Code of Data Loading and Splitting

Model building: As the most important part of the total thesis, this step will define a structure, train, and validate the CNN model for classification any input actions from users. With the knowledge in the previous section, a complete structure for this model is mentioned in the previous chapter.

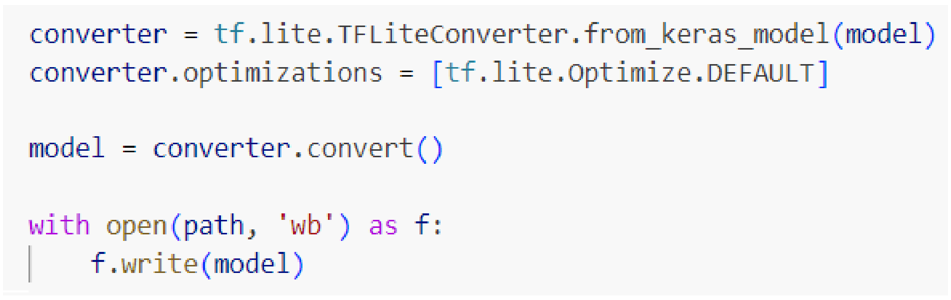
Also, this model uses the Adam algorithm as the optimizer and sparse categorical cross entropy as the loss function for measuring model performance.



```
model = tf.keras.models.Sequential([
                                    tf.keras.layers.Input((21 * 2,)),
                                    tf.keras.layers.Dropout(0.2),
                                    tf.keras.layers.Dense(20, activation = 'relu'),
                                    tf.keras.layers.Dropout(0.4),
                                    tf.keras.layers.Dense(10, activation = 'relu'),
                                    tf.keras.layers.Dense(NUM_CLASSES, activation = 'softmax')
                                   ])

model.compile(
                optimizer = 'adam',
                loss = 'sparse_categorical_crossentropy',
                metrics = ['accuracy']
              )
```

Figure 4.9: Code of Model Building

Model storing: TensorFlow Lite is implemented as the converter to save the model for later usage to reduce the storage required and make the model faster. Following is the code to save the model to the storage:



```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]

model = converter.convert()

with open(path, 'wb') as f:
    f.write(model)
```

Figure 4.10: Code of TensorFlow Lite to save the model as a file.

4.3 INTERFACE CREATING

In this section, the user interface is created using *MediaPipe* and *OpenCV*. This interface is a camera-implemented application for interacting with users. As mentioned above, the main purpose of this application is to detect users' hands, display the landmarks' location, record the location for pre-processing, and display the forecast result for users.

This application's front end consists of the camera's main area, the skeleton of the detected hand, which contains the landmarks as the nodes and the lines connecting two relative nodes together. Also, with any action the model classifies, the respective label will be displayed for the user to re-check if necessary.

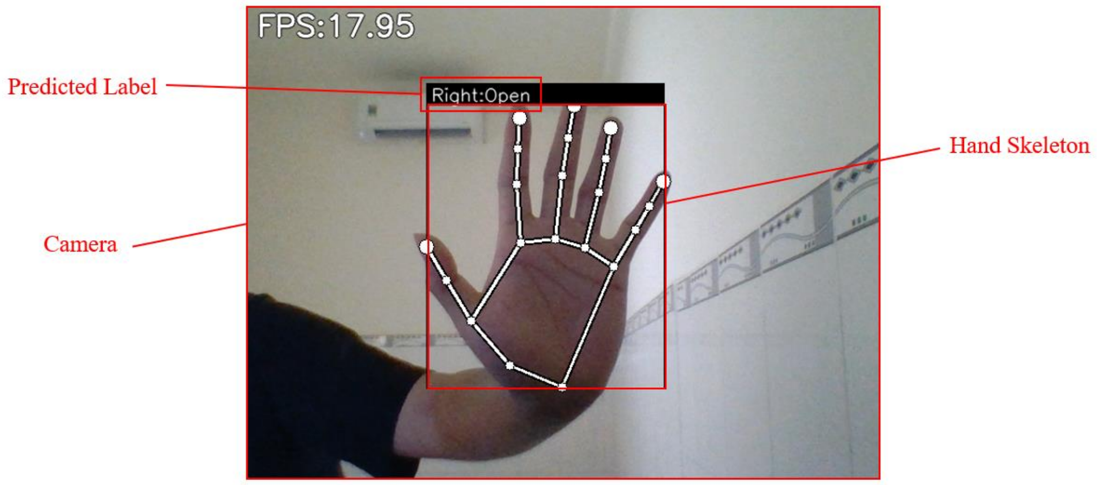


Figure 4.11: Front-end of the Interface

4.4 THESIS SAMPLE

In this section, some examples of the application from this thesis will be introduced, from the main screen to sample detections, to prove the possibility of this application for further development.

Figure 4.12: Application Sample - Main Interface

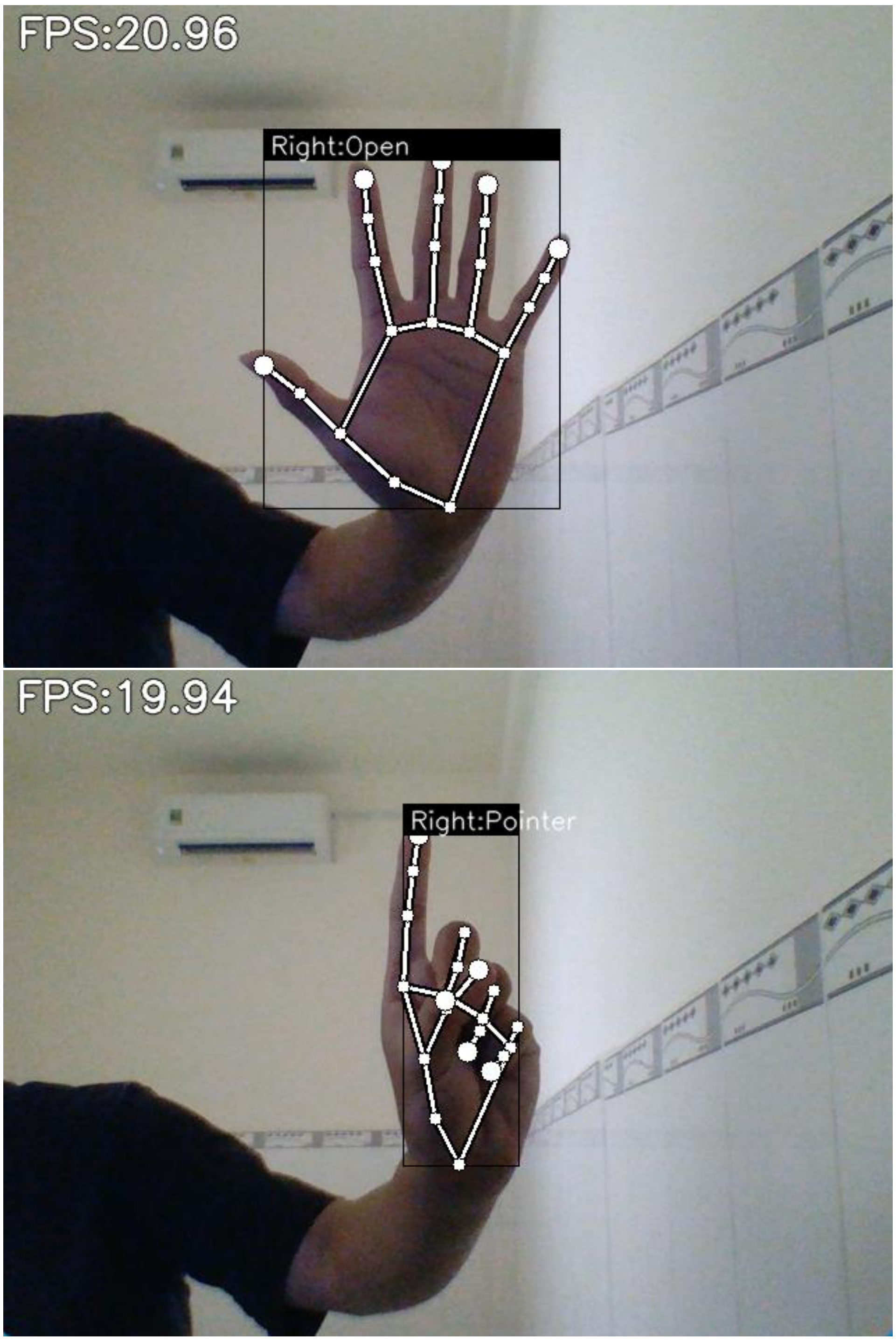


Figure 4.13: Application Sample - Hand Detected and Classified

CHAPTER 5

– CONCLUSION AND FUTURE WORKS

5.1 CONCLUSION

By completing the application, the main goal of this thesis is to create a user-friendly, lightweight, and high-speed application using users' hands to command household applications.

This application proves the possibility of creating a high-quality method for hand gestures recognition with the help of pre-built Python libraries as OpenCV, MediaPipe or TensorFlow. Together with those functions, the user-defined logics and commands also help customizing the model based on the usage and interest of users. Also, the side product of this thesis, which is the customized dataset can be used for other applications and references.

On the other hand, this application still remains some drawbacks that yet to solve. The most important problem is that, given the approximately same actions, the model are not able to distinguish between them and, as a result, some mistakes may appear. Furthermore, the requirement that users need to define the combination of action-command is worth to notice. Currently, all those commands are created and stored directly on data storage, which requires customers to have knowledge about programming in order to change, replace, remove or add more actions.

To my perspective, this thesis also provides good chance to sharpen as well as learn more about any state-of-the-art approaches for the field of gestures classification, or computer vision in general and neural network model family. Moreover, this thesis also provides an opportunity to practice structuring data, including how to pre-process and store in the most efficient way.

5.2 FUTURE WORKS

As this project is a demonstration yet to be completed, this project needs some other steps to become a usable application that can be developed in the future.

First of all, the diversity of actions and commands in the data storage can be extended with the purpose of providing more and more choices for users to customize their needs.

Also, the links between cameras around the house and the computer/application/tool for sharing the trained model can be set up for widen the network of working places of this tool around users' house.

Lastly, further development will also focus on how to link those commands with real household appliances. Although this is the main function of this thesis, this particular field requires more research and development in IOT in order to turn this thesis into real-world applications.

REFERENCES

1. Cao, Z., Simon, T., Wei, S., & Sheikh, Y. (2017). Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields. https://doi.org/10.1109/cvpr.2017.143
2. Li, Y., Lee, T., Kim, J., & Lee, H. (2021). CNN-Based Real-time Hand and Fingertip Recognition for the Design of a Virtual Keyboard. In 2021, 36th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC). https://doi.org/10.1109/itc-cscc52171.2021.9501471
3. Fukushima, K., & Miyake, S. (1982). Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition. In Lecture notes in biomathematics (pp. 267–285). Springer Nature. https://doi.org/10.1007/978-3-642-46466-9_18
4. Hubel, D. H., & Wiesel, T. N. (1968). Receptive fields and functional architecture of monkey striate cortex. The Journal of Physiology, 195(1), 215–243. https://doi.org/10.1113/jphysiol.1968.sp008455
5. LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. A., Hubbard, W., & Jackel, L. D. (1989). Handwritten Digit Recognition with a Back-Propagation Network. In Neural Information Processing Systems (Vol. 2, pp. 396–404). https://proceedings.neurips.cc/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf
6. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324. https://doi.org/10.1109/5.726791
7. Niu, X., & Suen, C. Y. (2012). A novel hybrid CNN–SVM classifier for recognizing handwritten digits. Pattern Recognition, 45(4), 1318–1325. https://doi.org/10.1016/j.patcog.2011.09.021
8. Simonyan, K. (2014, September 4). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv.org. https://doi.org/10.48550/arXiv.1409.1556
9. LeCun, Y., Bottou, L., Orr, G., & Müller, K. (2012). Efficient BackProp. In Lecture Notes in Computer Science (pp. 9–48). Springer Science+Business Media. https://doi.org/10.1007/978-3-642-35289-8_3
10. Qian, S., Liu, H., Liu, C., Wu, S., & Wong, H. (2018). Adaptive activation functions in convolutional neural networks. Neurocomputing, 272, 204–212. https://doi.org/10.1016/j.neucom.2017.06.070

11. Gholamalinezhad, H., Khosravi, H,. (2020). Pooling Methods in Deep Neural Networks, a Review. https://doi.org/10.48550/arXiv.2009.07485
12. Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2013). Improving neural networks by preventing co-adaptation of feature detectors. https://doi.org/10.48550/arXiv.1207.0580
13. Lin, M., Chen, Q. & Yan, S. (2013), Network In Network. https://doi.org/10.48550/arXiv.1312.4400
14. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y
15. Tang, Y. (2015). Deep Learning using Linear Support Vector Machines. https://doi.org/10.48550/arXiv.1306.0239
16. Huang, T. S., Wu, Y., & Lin, J. C. (2003). 3D model-based visual hand tracking. https://doi.org/10.1109/icme.2002.1035929
17. Wu, X. (2019). A hand gesture recognition algorithm based on DC-CNN. Multimedia Tools and Applications, 79(13–14), 9193–9205. https://doi.org/10.1007/s11042-019-7193-4
18. Chung, H., Chung, Y., & Tsai, W. (2019). An Efficient Hand Gesture Recognition System Based on Deep CNN. https://doi.org/10.1109/icit.2019.8755038
19. Florez, F., Garcia, J. M. V., Garcia, J. M. V., & Hernandez, A. C. (2003). Hand gesture recognition following the dynamics of a topology-preserving network. https://doi.org/10.1109/afgr.2002.1004173
20. Li, G., Tang, H. Y., Ma, J., Kong, J., Jiang, G., Jiang, D., Tao, B., Xu, S., & Liu, H. (2017). Hand gesture recognition based on convolution neural network. Cluster Computing, 22(S2), 2719–2729. https://doi.org/10.1007/s10586-017-1435-x
21. Ong, E., & Bowden, R. (2004). A boosted classifier tree for hand shape detection. https://doi.org/10.1109/afgr.2004.1301646
22. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C., Yong, M.G., Lee, J., Chang, W., Hua, W., Georg, M., Grundmann, M. (2019). MediaPipe: A Framework for Building Perception Pipelines. https://doi.org/10.48550/arXiv.1906.08172

23. Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C., Grundmann, M. (2020). MediaPipe Hands: On-device Real-time Hand Tracking. https://doi.org/10.48550/arXiv.2006.10214
24. Clevert, D., Unterthiner, T., Hochreiter, S. (2015). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). https://doi.org/10.48550/arXiv.1511.07289
25. Huang, Z., Ng, T., Liu, L., Mason, H., Zhuang & X., Liu, D (2020). SNDCNN: Self-Normalizing Deep CNNs with Scaled Exponential Linear Units for Speech Recognition. IEEE Conference Publication | IEEE Xplore. https://doi.org/10.1109/ICASSP40776.2020.9053973
26. Agarap, A.F. (2018). Deep Learning using Rectified Linear Units (ReLU). https://doi.org/10.48550/arXiv.1803.08375
27. Castaneda, G., Morris, P., & Khoshgoftaar, T. M. (2020). Evaluating The Number of Trainable Parameters on Deep Maxout and LReLU Networks for Visual Recognition. https://doi.org/10.1109/icmla51294.2020.00072
28. Xu, B., Wang, N., Chen, T. & Li, M. (2015). Empirical Evaluation of Rectified Activations in Convolutional Network. https://doi.org/10.48550/arXiv.1505.00853
29. Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J. & Yan, S. (2015). Deep Learning with S-shaped Rectified Linear Activation Units. https://doi.org/10.48550/arXiv.1512.07030
30. Ito, Y. (1991). Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. Neural Networks, 4(3), 385–394. https://doi.org/10.1016/0893-6080(91)90075-g
31. Zamanlooy, B., Mirhassani, M. (2013). Efficient VLSI Implementation of Neural Networks With Hyperbolic Tangent Activation Function. https://doi.org/10.1109/TVLSI.2012.2232321
32. Vu, T. (2017, February 17). Bài 13: Softmax Regression. Tiep Vu's Blog. https://machinelearningcoban.com/2017/02/17/softmax/