Automatic
Control &
Systems
Engineering.

Robotic Arm Control Using Gesture and Speech Recognition with Deep Learning

Sai Karthik Kagolanu

September, 2024

Supervisor: Dr. Payam Soulatiantork

A dissertation submitted in partial fulfilment of the requirements for the degree of MSc Robotics

ABSTRACT

This report presents the development of gesture- and speech-based recognition algorithms designed to control a robotic arm. The dissertation focusses on integrating deep learning techniques to accurately detect hand gestures and spoken commands. This enables an intuitive and natural interaction with the robotic system. The gesture recognition model based on Convolutional Neural Network (CNN) architecture achieved a classification accuracy of 96.98%. The speech recognition model, also based on a convolutional neural network (CNN), achieved a classification accuracy of 95.03%.

The system was trained and evaluated by adding self-generated data to a publicly available dataset on American Sign Language for gesture recognition. Self-generated data was added to a publicly available dataset for digits 0-9 in English for speech recognition. This ensured the system captured the variations in gestures and speech patterns across different users. These recognised commands were then used to control a robotic hand, demonstrating the system's practical application. Although the results are promising, limitations such as time, computational, and hardware constraints were identified.

Future work could focus on employing Mel-Frequency Cepstral Coefficients (MFCCs)-based Recurrent Neural Networks (RNNs) for speech recognition, deploying speech recognition to control the robotic hand, and creating a finger and hand mapping algorithm for gesture recognition.

INDIVIDUAL CONTRIBUTION

I was responsible for designing and implementing the control algorithm for the robotic arm. My role involved researching existing control methods, developing a new hybrid control approach, and coding the algorithms in MATLAB and Arduino.

I designed and developed code for executing gesture recognition and speech recognition using the Deep Learning Toolkit using MATLAB. I also enhanced the algorithms using various parameters and manual hyper-parameter tuning. I created the servo control code in the Arduino IDE to perform certain gestures on the Youbionic Handy Lite based on the received classification output. I created a bridge between MATLAB and robotic hand using a serial port connection for communication.

I also conducted a series of simulations to test the algorithm's effectiveness under various conditions. One of the challenges I faced was optimising the algorithm to efficiently classify on the available dataset, which I overcame by introducing a self-generated dataset. My work significantly improved the algorithm's precision, directly contributing to the project's overall success.

ACKNOWLEDGEMENTS

I would like to express my deepest and most sincere gratitude to my project supervisor, Dr. Payam Soulatiantork, for his unwavering guidance, insightful feedback, and continuous support throughout this dissertation project. His involvement is the single biggest factor for driving this dissertation project to a conclusion. I am also thankful to Dr. Simon Pope for his weekly presentations and workshops on Robotics Project and Dissertation. I would like to extend my thanks Dr. Sean Anderson who oversaw our Deep Learning module and gave us in-depth explanation and examples to execute Deep Learning algorithm on MATLAB and Python.

I would like to acknowledge ChatGPT for helping me brainstorm and refine ideas for my dissertation. Scopus with its wide literature search capabilities was quite useful in facilitating the literature review process. I would like to appreciate draw.io for providing a free and user-friendly flowchart designer. Mathworks also deserves appreciation for their tutorials on using Deep Learning Toolkit for gesture and speech recognition.

A special thanks goes to my family and friends for their encouragement and understanding during the challenging times of this journey. I am highly appreciative of the Department of Automatic Control and Systems Engineering for having such talented and supportive staff and for all the resources they made available. Finally, I gratefully acknowledge the the resources made available by The University of Sheffield, without which this dissertation would not have been possible.

Contents

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Problem Definition 2

1.3 Aims and Objectives 3

1.4 Overview of the Report 4

1.5 Project Management 5

1.5.1 Project Planning 5

1.5.2 Gantt Chart 6

1.5.3 Progress Overview and risks 6

2 Literature Review **8**

2.1 Introduction 8

2.2 Robotic Arm Design and Control 8

2.3 Gesture-Controlled Robotic Arms 9

2.4 Arduino-Based Robotic Arms 9

2.5 Computer Vision and Advanced Control Techniques 9

2.6 Applications and Future Directions 10

2.7 Conclusion 10

3 Methodology **11**

3.1 Research Design 11

3.1.1 Experimental Setup 12

3.1.2 Algorithm Development 12

3.1.3 Integration and Testing 12

3.2 Data Collection Methods 13

3.3 Tools and Techniques . . . 13
3.3.1 Software Tools . . . 13
3.3.2 Hardware Tools . . . 14
3.3.3 Techniques . . . 15
3.3.4 Training Options . . . 16
3.4 Data Analysis . . . 16
3.5 Rationale . . . 17
3.5.1 Deep Learning . . . 17
3.5.2 MATLAB . . . 17
3.5.3 Youbionic Handy Lite . . . 18
3.5.4 Performance Metrics . . . 18

4 Results and Discussion **19**
4.1 Gesture Recognition Results and Discussion . . . 19
4.1.1 Baseline Model Performance Analysis . . . 19
4.1.2 Final Model Performance Analysis . . . 22
4.2 Speech Recognition Results and Discussion . . . 25
4.2.1 Baseline Model Performance Analysis . . . 25
4.2.2 Final Model Performance Analysis . . . 28
4.3 Integrated System Performance and Discussion . . . 32
4.3.1 Control of the Robotic Arm . . . 32
4.3.2 Overall System Results . . . 33
4.4 Summary of Findings . . . 34

5 Conclusion and Future Work **35**
5.1 Conclusion . . . 35
5.2 Limitations . . . 35
5.3 Future Work . . . 35

REFERENCES **37**

6 Appendix **40**
6.1 Video Links . . . 40
6.2 Training options . . . 41

6.3 MATLAB Code Listings for Gesture Recognition 41
6.3.1 Data Acquisition using webcam 41
6.3.2 Gesture Recognition Algorithm training 43
6.3.3 Testing Gesture Recognition Algorithm using Webcam 47
6.4 MATLAB Code Listings for Speech Recognition 49
6.4.1 Data Acquisition using microphone 49
6.4.2 Preprocessing audio files to generate spectrograms 51
6.4.3 Speech Recognition Algorithm training 55
6.5 Arduino Code for Servo Control 58

List of Figures

1.1 Gantt Chart for Project Timeline 6

3.1 System Architecture Flowchart 11
3.2 Process Flowchart 14

4.1 Training result for baseline gesture recognition algorithm 21
4.2 Confusion Matrix for baseline gesture recognition algorithm 21
4.3 Training result for final gesture recognition algorithm 24
4.4 Confusion matrix for final gesture recognition algorithm 24
4.5 Training result for baseline speech-speech recognition algorithm 27
4.6 Confusion Matrix for baseline Speech Recognition algorithm 27
4.7 Training result for final Speech Recognition algorithm 30
4.8 Confusion matrix for final speech recognition algorithm 30
4.9 Testing the deep learning-based control system on gesture 5 33
4.10 Testing the deep learning-based control system on gesture 4 33

List of Tables

1.1 Project Timeline with Milestones 6

3.1 Sample CNN Layer Structure 15

4.1 CNN Layer Structure for Baseline Gesture Recognition Algorithm . . . 20
4.2 Training Options for the Baseline Gesture Recognition Algorithm . . . 20
4.3 CNN Layer Structure for Final Gesture Recognition Algorithm 22
4.4 Training Options for Final Gesture Recognition Algorithm 23
4.5 Performance Metrics for Baseline and Final Gesture Recognition Models 25
4.6 CNN Layer Structure for Baseline Speech Recognition Algorithm . . . 26
4.7 Training Options for the Baseline Speech Recognition Algorithm 26
4.8 CNN Layer Structure for Final Speech Recognition Algorithm 28
4.9 Training Options for Final Speech Recognition Algorithm 29
4.10 Performance Metrics for Baseline and Final Speech Recognition Models 31
4.11 Pseudo Code for Servo Control 32

6.1 MATLAB Training Options and Specific Solver Settings 41

Chapter 1

Introduction

1.1 Background and Motivation

Industries have been through several transformative phases. Each phase is marked by significant technological advancements. These advancements revolutionised the way industries operate. The first industrial revolution, or Industry 1.0, is where mechanical devices were deployed in production. The second industrial revolution, or Industry 2.0, introduced assembly lines and many advancements in machinery. The Third Industrial Revolution, or Industry 3.0, used Programmable Logic Controllers (PLCs) to enable limited automation in production stages. Each stage brought in a noticeable increase in efficiency and productivity. The latest industrial revolution is Industry 4.0, which uses digital technologies such as artificial intelligence (AI), machine learning (ML), and the Internet of Things (IoT). These are combined to create interconnected systems that are intelligent.

Industries move beyond traditional automation using PLC and integrate AI and ML into robotics. This paves the way for more adaptive autonomous systems. More advancements in the fields of deep learning techniques like convolutional neural networks (CNNs) mean robotics is not confined to only performing repetitive tasks and is now capable of learning from data, making real-time decisions, and continuously improving their performance. This is further boosted by IoT, which supports seamless connection, communication, and data exchange between the devices, creating a more responsive and efficient industrial environment.

In this rapidly evolving era, this project explores the use of deep learning (DL) to control Youbionic Handy Lite. This is a robotic hand designed to perform a range of tasks with high precision.

Through this project, I want to contribute my findings to the application of AI-driven robotics. This falls under the Industry 4.0 standards. The impact of such developments extends beyond the industries, into healthcare, manufacturing, and much more. This dissertation is structured to give an informed introduction of the technology, followed by detailed discussion on methodology and conclusion with a discussion of the implications and future works of AI-integrated robotic systems.

1.2 Problem Definition

As the use of automation in industries evolves over time, the challenge evolves from automating repetitive tasks to creating intelligent, adaptive systems that are capable of performing complex operations on their own. The use of deep learning in robotics gives a promising solution to this challenge, but there are a few hurdles to be addressed to make this a reality.

Traditional methods of control can be effective for simpler tasks. Controlling the robotic arm, Youbionic HandyLite, requires responsive and precise servo control. The system should interpret and execute all the commands in real-time. We need a flexible controller that can adapt to modern industries. This project aims to develop a control system for the robotic arm using deep learning.

This project uses MATLAB to use deep learning techniques. This platform offers robust toolboxes for implementing and testing deep learning models. The use of MATLAB not only helps in developing deep learning algorithms but also helps in seamless integration with servos to enable real-time control of the robot.

This project uses classification outputs generated from a trained neural network to con-

trol the robotic arm. The system will classify input data and generate appropriate output to control the servos. It will facilitate controlling the robotic hand to perform the desired task. This approach can result in adaptive learning, which means that the robot can improve its performance based on the feedback.

In summary, this project addresses the following key problems:

- Developing a servo control system for the Youbionic HandyLite
- Implementing deep learning algorithms for gesture and speech recognition
- Control the robotic hand using classification output

This dissertation focusses on using the current state of AI to improve robotics by tackling these challenges. It also demonstrates the practical application of deep learning in controlling robotic systems.

1.3 Aims and Objectives

The primary aim of this project is to control the robotic hand using servos with deep learning. So create a control system that is capable of using gesture and speech recognition algorithms.

Objectives are as follows:

- Develop a simple gesture recognition algorithm using deep learning.
- Enhance the algorithm based on performance metrics
- Control the robotic hand using classification output.

Additional objectives are as follows:

- Develop a simple speech recognition algorithm using deep learning.
- Enhance the algorithm based on performance metrics
- Control the robotic hand using classification output.

1.4 Overview of the Report

This dissertation is organised into six chapters, along with references and appendices. Each chapter builds on the previous one to give a comprehensive explanation of the project. Find the overview of the structure of this report below:

- **Chapter 1: Introduction**

  This chapter contains the background and motivation for the project. It introduces the key concepts of Industry 4.0, deep learning, and robotics. It outlines the aims and objectives of the project.

- **Chapter 2: Literature Review**

  This chapter reviews existing research and literature relevant to the project. It also identifies gaps in the current research.

- **Chapter 3: Methodology**

  This chapter gives detailed information on the methodology used to develop the control system for the Youbionic Handy Lite. It includes the design and implementation of deep learning algorithms in MATLAB. It also contains the process of integrating these algorithms with servo control.

- **Chapter 4: System Design and Implementation**

  This chapter describes the system design and implementation of the control system. It explains how the deep learning models were trained, validated, and deployed to control the robotic hand. The chapter also covers the hardware and software components used in the system.

- **Chapter 5: Results and Discussion**

  This chapter contains the results obtained from the training and testing process. It includes performance metrics of the deep learning algorithms and the overall performance of the control system. These results are discussed in line with objectives.

- **Chapter 6: Conclusion and Future Work**

  This chapter summarises the findings of the project and suggests potential areas for future research and improvements to the system.

- **References**

  This section lists all the sources cited throughout the report.

- **Appendices**

  The appendices contain code listings.

1.5 Project Management

1.5.1 Project Planning

Project planning involves all the key activities, milestones, and timelines required to achieve the objectives. This explains all the required steps to create a deep-learning-based control system to control the robotic hand. This project was divided into the following phases to ensure progress:

- **Phase 1: Research and Literature Review**

  Conducted a comprehensive review of existing literature on controlling a robotic hand using servos. This phase involved identifying research gaps to form methodology.

- **Phase 2: Design and Development**

  Developed deep learning algorithms for gesture and speech recognition using MATLAB. This phase involved creating models, selecting datasets, and determining performance metrics.

- **Phase 3: System Integration**

  Integrated the developed algorithms with the servo control system. This phase focused on control of the robotic hand based on the output from deep learning models.

- **Phase 4: Testing and Evaluation**

  Conducted testing to validate the performance of the deep learning-based control system. This phase involved iterative testing, performance evaluation, and optimisation.

- **Phase 5: Oral Presentation and Final Report Submission**

  Prepared the dissertation report, which included a detailed explanation of the

methodology, results, and conclusions. This phase also included the preparation of presentation materials.

1.5.2 Gantt Chart

Figure 1.1 illustrates the Gantt chart developed to outline the timeline of the project phases. The chart shows the revised timeline based on progress and unforeseen delays.

Task	June Week 1	June Week 2	June Week 3	June Week 4	July Week 1	July Week 2	July Week 3	July Week 4	August Week 1	August Week 2	August Week 3	August Week 4
Review exisiting materials and resources provided, initial planning	5 hours											
Literature review on control methods for robotic arm using servos		6 hours	10 hours									
Implement gesture recognition using Machine learning in MATLAB			6 hours	20 hours								
Implement gesture recognition using Deep Learning in MATLAB				10 hours	25 hours		10 hours		10 hours			
Arduino Coding for controlling robotic arm using servos					4 hours	20 hours						
Implement speech recognition using Deep learning in MATLAB							6 hours	15 hours				
Implement servo control with gesture recognition								12 hours	20 hours			
Implement servo control with speech recognition										5 hours	10 hours	
Documentation and Final Testing										15 hours	10 hours	
Writing Report					5 hours	5 hours	5 hours	5 hours	5 hours	5 hours	15 hours	25 hours
Completed	Ongoing				More work hours needed				Pending further action			

Figure 1.1: Gantt Chart for Project Timeline

Table 1.1 below gives a brief insight into the phase-wise completion of tasks and summarises the gantt chart.

Table 1.1: Project Timeline with Milestones

Milestone	**Task**	**Deadline**
Phase 1	Research and Literature Review	Week 3, June
Phase 2	Design and Development	Week 1, August
Phase 3	System Integration	Week 3, August
Phase 4	Testing and Evaluation	Week 3, August
Phase 5	Oral Presentation and Final Report Submission	Week 4, August

1.5.3 Progress Overview and risks

Regular reviews were conducted throughout the project to track progress. Time adjustments were made to the timeline wherever required, like in the case of training models for gesture and speech recognition and the control system for robotic arms. This was a potential risk, as training and tuning the deep learning algorithms often takes hypothesis testing to be carried out. Training models after hyperparameter tuning often takes 1-2

hours for the current dataset, and getting the optimal set of hyperparameters is often the challenge. The Gantt chart was updated to reflect all these changes. The core objectives of developing a robust deep-learning-based control system for the robotic hand were achieved.

Chapter 2

Literature Review

2.1 Introduction

Over the last few decades, the development of robotic arms has made substantial progress, owing to advances in control mechanisms, design techniques, and application fields. This chapter examines the existing literature on robotic arm design, gesture-based control systems, Arduino implementations, and advanced control approaches. The review's goal is to identify major contributions, analyse trends, and indicate areas that require additional research.

2.2 Robotic Arm Design and Control

Research has focused on designing robotic arms that are precise, flexible, and cost-effective. Bora and Nandi [1] developed a low-cost articulated robotic arm with shadow function-based control, making it suitable for budget-sensitive applications. Pol et al. [3] developed a four-degree-of-freedom (DoF) robotic arm using LabVIEW, emphasising versatility and ease of integration.

In the context of assistive technology, Schrock et al. [15] developed a modular robotic arm for wheelchair users, allowing them to do daily activities independently. Eldridge et al. [14] investigated the use of a remote centre of motion robotic arm in computer-assisted surgery, proving its precision and control in medical settings.

2.3 Gesture-Controlled Robotic Arms

Gesture recognition as a control interface has grown in popularity due to its intuitive nature. Pal and Kakade [2] used a Kinect sensor to recognise dynamic hand gestures and accurately operate robotic arms. Mardiyanto et al. [5] created a system that uses accelerometers and gyroscopes to recognise hand movements. The system was successfully used to drive an underwater robotic arm, demonstrating its robustness in various settings.

Sihombing et al. [10] developed a system that uses finger and hand gestures to control a robotic arm, enhancing gesture recognition precision. Bakri et al. (2019) improved gesture control by using a wireless NRF24L01 transmitter, increasing system range and usability.

2.4 Arduino-Based Robotic Arms

Arduino platforms are extensively utilised due to their low cost, ease of use, and extensive community support. Bhargava and Kumar [4] showed an Arduino-controlled robotic arm, providing a versatile solution for educational and hobbyist applications. Agrawal et al. [9] and Ali et al. [12] added IoT capabilities to Arduino-based robotic arms, allowing for remote monitoring and control and expanding their use in smart settings.

Ahmed et al. [11] designed and implemented an IoT-controlled robotic arm using Arduino ESP32, resulting in improved connection and real-time control. The trend towards IoT integration emphasises the increasing necessity of connectivity in current robotic systems.

2.5 Computer Vision and Advanced Control Techniques

The use of computer vision and powerful control algorithms has greatly increased the capabilities of robotic arms. Sadeghi et al. [7] tackled the difficulty of viewpoint-invariant visual servoing using recurrent control, underscoring the necessity for robust algorithms that improve the flexibility of robotic arms in dynamic settings. Bolc et al. [13] explored computer vision techniques for robotic control, especially in circum-

stances that need great precision and adaptability.

Verma [18] suggested a system that employs computer vision to identify hand gestures and then use them to operate a robotic arm, highlighting the possibility of visual input as an effective control method. Kadir et al. [16] and Cela et al. [17] highlighted the importance of sophisticated algorithms and connectivity in improving the robustness and flexibility of robotic systems, especially for real-time applications.

2.6 Applications and Future Directions

Robotic arms are used in a variety of applications, including medical, industrial, and household settings. Eldridge et al. (1996) demonstrated the promise of robotic arms in precision-critical tasks like computer-assisted surgery. Similarly, Mardiyanto et al. [5] investigated the usage of robotic arms in underwater environments and shown their adaptation to harsh circumstances.

Despite these advancements, significant difficulties remain. The integration of advanced AI approaches for gesture detection and control, as mentioned by Verma [18] and Sadeghi et al. [7], remains a potential field for further research. Furthermore, building more intuitive and user-friendly interfaces, as well as improving the robustness of robotic systems in uncertain contexts, are important areas for future research.

2.7 Conclusion

The literature demonstrates significant development in the design and control of robotic arms, particularly in the areas of cost-effectiveness, gesture-based control, and Arduino-based systems. While great progress has been achieved in combining IoT and computer vision technology, issues such as flexibility, sophisticated AI integration, and real-time control in dynamic contexts remain. Addressing these problems through ongoing research and innovation is critical to realising the full potential of robotic arms in a variety of applications.

Chapter 3

Methodology

3.1 Research Design

The design for my project is based on both experimental and computational works. Controlling the robotic hand requires experimental work, which requires fine-tuned controlling of the servos and establishing a bridge between MATLAB and the laptop. Developing and enhancing the deep learning models require computational work, as lots of images need training and are then classified under respective labels.

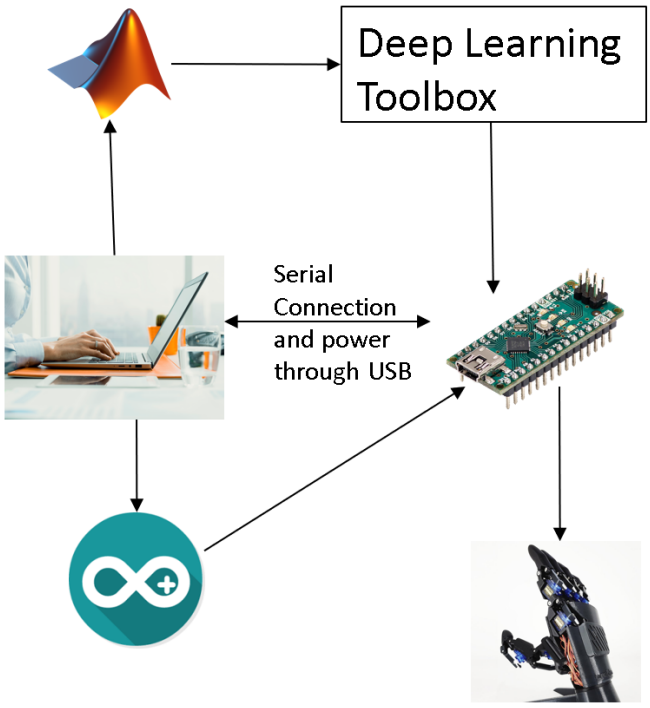

Figure 3.1: System Architecture Flowchart

3.1.1 Experimental Setup

Youbionic Handy Lite, the robotic hand used in this project, has 11 servo motors, SG90, which weigh 90g each. These motors are connected to the HCPCA9685 16-bit, 12-channel servo motor driver board, which in turn is connected to the Arduino Nano. Arduino Nano is used to control and power the servo motors and, in turn, is connected to the laptop for serial communication and power supply.

3.1.2 Algorithm Development

For both the gesture and speech recognition algorithms, the process of designing and iterating the deep learning network is similar. We start with data acquisition, where we use the webcam and microphone to capture data. It is then preprocessed to make sure it is suitable to feed into the algorithm. First, a random seed is setup to generate the same output every time the code is run. Using the Deep Learning Toolbox in MATLAB, a basic layer is created that contains the image input layer, followed by blocks of the convolutional layer, batch normalisation layer, ReLU layer, and a Max Pooling layer, followed by a dropout layer, and finally ending with the block containing the fully connected layer, softmax function layer, and output layer. There are parameters for controlling the filter's kernel window size, number of filters used, Max Pooling kernel size, and stride, among others, that can be fine-tuned. This is how the deep learning algorithms are structured and refined.

3.1.3 Integration and Testing

As shown in Figure 3.1, the laptop acts as a central hub, which uses MATLAB and Arduino IDE for software interfacing, a wired connection to Arduino Nano through the USB for hardware interfacing, and the connection to all the servos of the arm using Arduino. Arduino IDE is used to upload the code for defining servo movements. MATLAB is used to train deep learning networks, and their classification output is sent to Arduino Nano, through which the servo control is managed.

3.2 Data Collection Methods

For deep learning models, the data being trained would greatly benefit if the capture was done using the same source. This will help ease the classification as the model knows the expected image clarity. Data is collected through the laptop's internal webcam and converted to greyscale to maintain uniformity with the existing dataset. For the speech recognition model, we use the microphone of a laptop to record an audio dataset and use the same sampling rate as the existing dataset. The publicly available dataset provided by the supervisor had 1500 samples per class for gesture and 1700 samples per class for speech. An additional 300 samples per class for gesture and 250 samples per class for audio were added through data acquisition.

3.3 Tools and Techniques

The specific tools, techniques, and software used in the project are discussed in this section. Figure 3.2 gives a clear explanation of the workings of the entire process while outlining the tools and techniques used.

3.3.1 Software Tools

MATLAB is the primary software tool used for developing and testing deep learning algorithms. It provided solid support for mathematical computations and training. MATLAB is used to train the gesture and speech recognition algorithms for data acquisition, image and speech data preprocessing, and testing the algorithms for classification outputs. It helped in incorporating the algorithms with the robotic hand's control system. Arduino IDE is used once to upload the code for servo control onto the Arduino Nano, through which the robot hand knows the gesture to be performed.

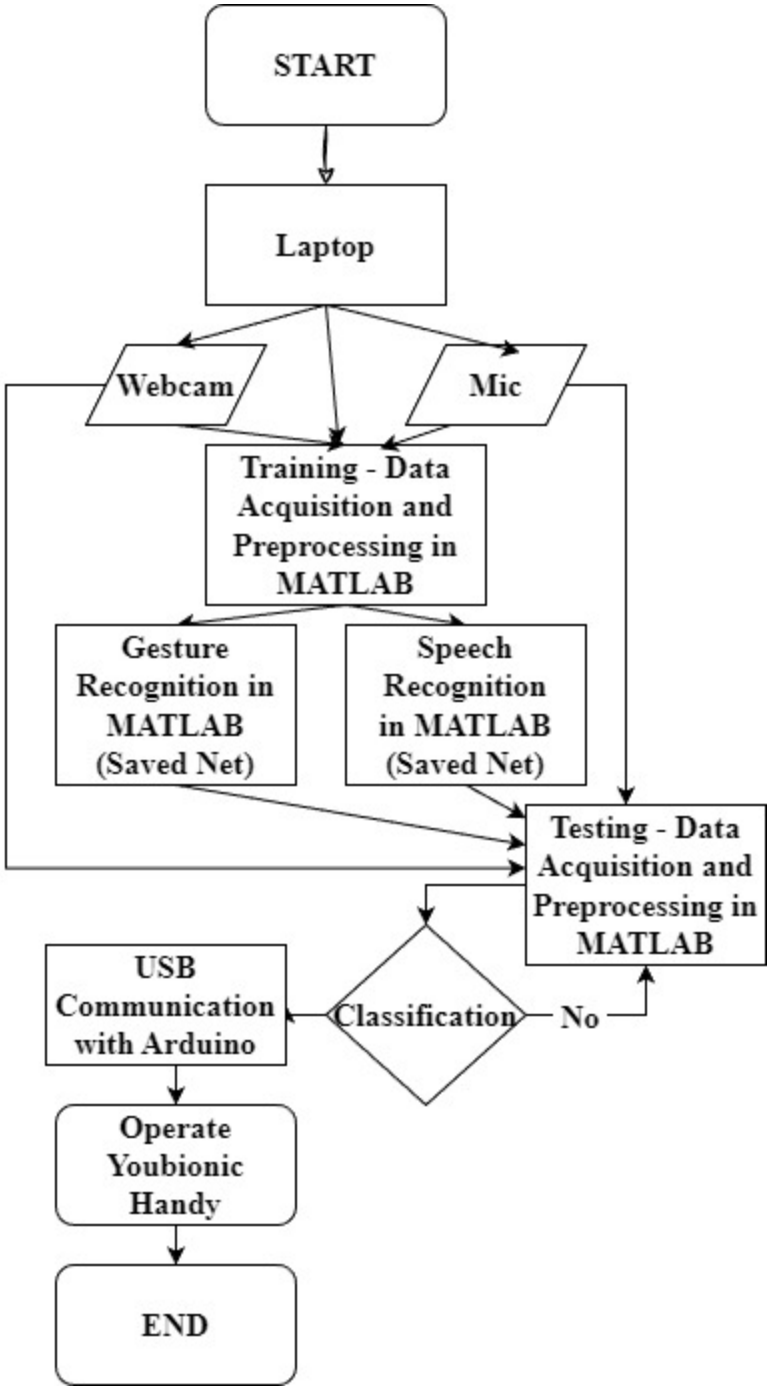

Figure 3.2: Process Flowchart

3.3.2 Hardware Tools

Youbionic Handy Lite uses an Arduino Nano for communication as well as power. It has a PCA9685 servo driver board connected, to which all the 11 servo motors are wired, which helps in equal distribution of current and voltage to these servos. After the classification output is generated, it is sent to Arduino Nano, which then executes

the required action on the robotic hand.

3.3.3 Techniques

Table 3.1 shows a sample CNN layer structure. It starts by taking an input in the form of an image. The image size is output size. Next Convolution 2D layer is used where Rectified Layer unit is used as the activation function. Convolutional neural networks use a filter to generate a feature map that summarises the presence of observed features in the input. The rectified linear unit (ReLU) or rectifier activation function brings nonlinearity into a deep learning network and resolves the vanishing gradients issue by interpreting the positive part of its argument. Batch normalisation can help reduce overfitting and improve the model's generalisability by normalising a layer's activations. Batch normalisation can lower the model's sensitivity to the initial weights, making it easier to train. Max pooling minimises the spatial dimensions of features by selecting the highest value in each small window or region. The fully connected layer will execute high-level reasoning and decision-making using the features retrieved by the prior layers. It achieves this by learning sophisticated nonlinear mappings between the input and output data. The softmax activation function converts the neural network's raw outputs into a vector of probabilities, which is effectively a probability distribution over the input classes. Consider a multiclass classification task involving N classes.

CNN Layer Structure (Total Learnables: 480, Layers: 5)			
Layer Type	**Parameters**	**Output Size**	**Activation Function**
Image Input	Image Size: [HxWxD]	[HxWxD]	-
Convolution 2D	Filter Size: MxN, Number of Filters: F	[HxWxF]	ReLU
Batch Normalisation	-	[HxWxF]	-
Max Pooling 2D	Pool Size: PxQ, Stride: S	[H/P x W/Q x F]	-
Fully Connected	Number of Classes: C	[1x1xC]	-
Softmax	-	[1x1xC]	Softmax
Classification	-	[1x1xC]	-

Table 3.1: Sample CNN Layer Structure

3.3.4 Training Options

SGD, Adam, and RMSprop are some of the most common optimisers. SGD is simple but effective, although Adam and RMSprop are more computationally costly but frequently converge faster. They are all gradient descent-based methods, which means that the model weights are iteratively updated in the direction that minimises loss. Adam is most popular because the learning rate can be set to adaptive. Other options like L1 or L2 regularisation can be applied. Early stopping can be implemented using ValidationPatience. Output Fcn can be used to run custom functions while training data, and MiniBatchSize is used to declare the size of mini-batches. The mini-batch is a set number of training instances that is smaller than the total dataset. So, in each cycle, we train the network on a different group of samples until the entire dataset is consumed. We can set ExecutionEnvironment to make use of CPU, GPU, or if we have multiple GPUs, multi-gpu and parallel processing. Refer to table **??** in the Appendix section for detailed training options.

3.4 Data Analysis

The data collected during experiments and algorithm testing has been evaluated for performance using validation accuracy, precision, recall and F1 scores.The standard metrics like accuracy, precision, recall, and F1 score were used because they provide a comprehensive evaluation of the algorithm.

Accuracy measures overall correctness of predictions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.1}$$

Precision is the accuracy of positive predictions. Positive predictions means sum of all true positive and false positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

Recall, also known as sensitivity, is the ability of the model to identify all correct positive instances. Positive instances is the sum of all true positives and false negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

F1 score is used when a single metric is required to assess the model. It is the harmonic mean of Precision and Recall.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.4}$$

For training the data, iterative methods to validate the hypothesis was used till acceptable results were achieved. For testing the data, a confidence threshold was set to ensure the data classified is correct. While training, hypothetical cases of background and unknown classes were setup and that classification is clearly seen while testing. A confusion matrix has been generated to look at class-wise distribution of data across all the classes. Throughout the process various graphs and figures were generated to visualise with clarity and made sure it was correctly expressed.

3.5 Rationale

In this section, we justify the choices we have taken and the way these choices affected the project.

3.5.1 Deep Learning

The choice of using deep learning for enhancing gesture and speech recognition instead of machine learning models like random forest classifiers or support vector machines is justified because deep learning employs neural networks that mimic the neurons in our brains. Neural networks use networks of neurons, as the name suggests, all interconnected to each other over layers in which they try to correlate the features split across all neurons to classify the data. Deep learning for machine vision and image classification uses convolutional neural networks (CNNs). CNNs are known for feature mapping. It essentially enables the neurons to pick the best features for an image, correlate them, and use them to classify. So CNNs with deep learning are used for this project.

3.5.2 MATLAB

MATLAB is chosen as the software executing deep learning algorithms as it has an inbuilt toolbox for deep learning, image processing, computer vision, audio, and hardware support packages for USB webcams and Arduino. MATLAB is used by scientists

and engineers and is taught during education as well. It was designed for mathematical operations, especially matrix manipulations, plotting functions and data, implementing programs, and interfacing with other applications. It was also a part of modules and coursework for MSc Robotics at the University of Sheffield.

3.5.3 Youbionic Handy Lite

Youbionic Handy Lite was provided by the supervisor due to its advanced features, adaptability, and functionality. As mentioned earlier, it has 11 servos, 2 servos for each finger but 3 for the thumb. It is capable of mimicking the human hand to a certain extent. Since gesture recognition is being used, making the same gesture as the user is easier for this robotic hand. It makes the project that much simpler in terms of execution. The hand is 3D printed and houses all the servos along with an Arduino Nano and a HCPCA9685 servo controller board.

3.5.4 Performance Metrics

The metrics chosen, accuracy, precision, recall, and F1 score, are standard in the field as they provide a good understanding of the algorithm and each offer different insights about the model's performance.

Chapter 4

Results and Discussion

In this chapter, the results obtained from the experiments and analyses are presented. It is followed by a discussion of their significance and interpretation of the results.

4.1 Gesture Recognition Results and Discussion

In this section, the algorithm structure, training options, training results, confusion matrices, and performance evaluation of both the baseline and final models are presented and discussed.

4.1.1 Baseline Model Performance Analysis

The structure of the baseline model can be seen in 4.1. It is a fairly simple structure, with 480 learnable parameters and 5 layers. It contains a simple block with 1 Convolution2D layer, 1 batch normalisation, 1 ReLU, and 1 MaxPooling layer. This was chosen as the starting model to establish a performance benchmark for gesture recognition.

CNN Layer Structure (Total Learnables: 480, Layers: 5)			
Layer Type	**Parameters**	**Output Size**	**Activation Function**
Image Input	[98x50x3]	[98x50x3]	-
Convolution 2D	Filter Size: 3x3, Number of filters: 16	[98x50x16]	ReLU
Batch Normalisation	-	[98x50x16]	-
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[49x25x16]	-
Fully Connected	Number of Classes: 11	[1x1x11]	-
Softmax	-	[1x1x11]	Softmax
Classification	-	[1x1x11]	-

Table 4.1: CNN Layer Structure for Baseline Gesture Recognition Algorithm

The table 4.2 shows the training options used for this simple structure. It uses an Adam solver as opposed to a gradient-based solver due to its advantage of having adaptive learning rates. This helps in faster convergence and improved quality of the solution. We have a 0.001 learning rate, with 128 mini-batch size in 10 epochs of training, and early stopping was implemented based on validation performance.

Option	**Value**
Solver	Adam
MiniBatchSize	128
MaxEpochs	10
Shuffle	Every Epoch
ValidationData	dsVal
ValidationFrequency	20
Verbose	True
Plots	Training-progress
ExecutionEnvironment	GPU
InitialLearnRate	0.001
ValidationPatience	5

Table 4.2: Training Options for the Baseline Gesture Recognition Algorithm

As can be seen in Figure 4.1, the baseline model achieved a training accuracy of 68.11%, which indicates limited generalisation ability. Given the simple structure and

relatively simple training options, the accuracy is quite good for the baseline model.

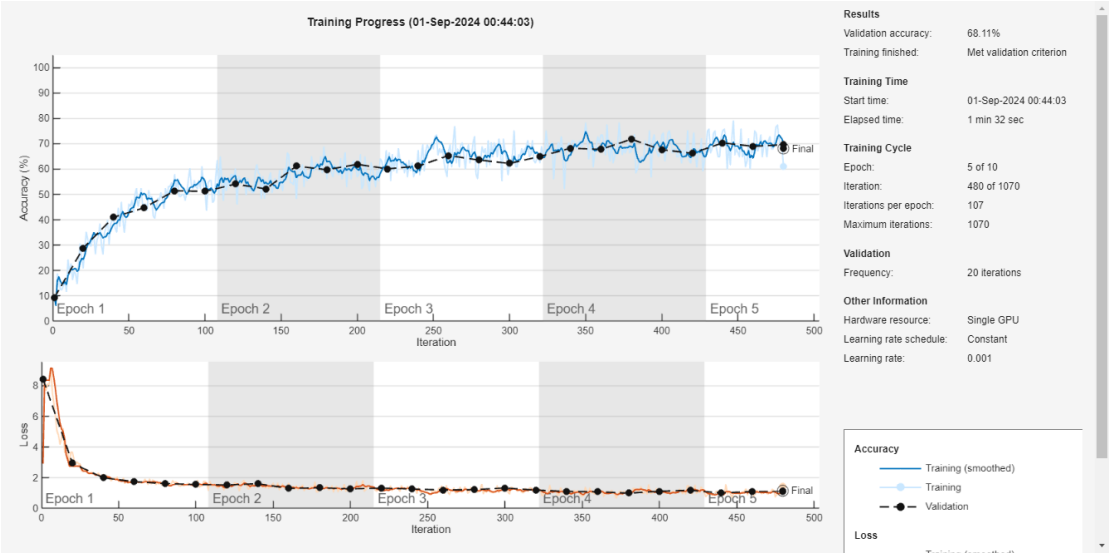

Figure 4.1: Training result for baseline gesture recognition algorithm

The figure 4.2 shows the confusion matrix for the training. We can see the distribution has lots of misclassifications, especially among similar gestures.

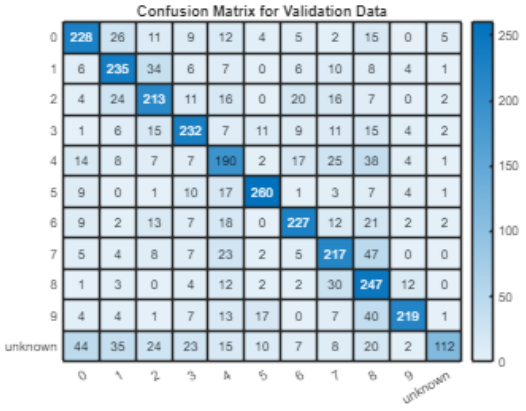

Figure 4.2: Confusion Matrix for baseline gesture recognition algorithm

The model's overall performance can be seen in table 4.5. If we take an F1 score of 0.68, we can say that the network is too basic and simple for effective gesture recognition.

4.1.2 Final Model Performance Analysis

CNN Layer Structure for Gesture Recognition Algorithm (Total Learnables: 2.8M, Layers: 30)			
Layer Type	**Parameters**	**Output Size**	**Activation Function**
Image Input	-	[98x50x3]	-
Convolution 2D	Filter Size: 5x5, Filters: 128, Padding: Same	[98x50x128]	ReLU
Batch Normalisation	-	[98x50x128]	-
ReLU	-	[98x50x128]	ReLU
Convolution 2D	Filter Size: 5x5, Filters: 128, Padding: Same	[98x50x128]	ReLU
Batch Normalisation	-	[98x50x128]	-
ReLU	-	[98x50x128]	ReLU
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[49x25x128]	-
Convolution 2D	Filter Size: 5x5, Filters: 128, Padding: Same	[49x25x128]	ReLU
Batch Normalisation	-	[49x25x128]	-
ReLU	-	[49x25x128]	ReLU
Convolution 2D	Filter Size: 5x5, Filters: 128, Padding: Same	[49x25x128]	ReLU
Batch Normalisation	-	[49x25x128]	-
ReLU	-	[49x25x128]	ReLU
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[25x13x128]	-
Convolution 2D	Filter Size: 5x5, Filters: 128, Padding: Same	[25x13x128]	ReLU
Batch Normalisation	-	[25x13x128]	-
ReLU	-	[25x13x128]	ReLU
Convolution 2D	Filter Size: 5x5, Filters: 128, Padding: Same	[25x13x128]	ReLU
Batch Normalisation	-	[25x13x128]	-
ReLU	-	[25x13x128]	ReLU
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[13x7x128]	-
Dropout	Dropout Rate: dropout_rate	[13x7x128]	-
Fully Connected	Number of Classes: 11	[11]	-
Softmax	-	[11]	Softmax
Classification	-	[11]	-

Table 4.3: CNN Layer Structure for Final Gesture Recognition Algorithm

Table 4.3 shows the final model structure with 2.8 million learnable parameters and 30 layers. This shows that a significantly deeper architecture is designed, which can capture finer details in images. Layer blocks are designed to have 2 sets of Convolution2D, Batch Normalisation, and ReLU before having a MaxPooling layer. 4 blocks as such are designed before having a dropout layer followed by the fully connected layers, softmax function, and classification output. The filter size increased to 5x5 from 3x3, and

the number of filters increased to 128. This helps in having more data in increased numbers of feature maps, which greatly enhances the recognition performance. A dropout layer with 0.2 was also added to ensure random connections between neural networks are cut-off such that the model can prevent overfitting and memorising the data rather than learning it. This helps in better model generalisation.

Option	**Value**
Solver	Adam
MiniBatchSize	64
LearnRateSchedule	Piecewise
LearnRateDropFactor	0.1
LearnRateDropPeriod	3
MaxEpochs	10
Shuffle	Every Epoch
L2Regularization	0.0001
ValidationData	dsVal
ValidationFrequency	20
Verbose	True
Plots	Training-progress
ExecutionEnvironment	GPU
InitialLearnRate	0.0001
ValidationPatience	5

Table 4.4: Training Options for Final Gesture Recognition Algorithm

The table 4.4 shows the training options used. The full potential of the Adam solver is used here, where the piecewise learning rate is scheduled with a drop factor of 0.1 for every 3 epochs. The lower initial learning rate of 0.0001 as well as the reduced mini-batch size to 64 mean faster convergence and more batches per epoch. The data was also shuffled every epoch to have new data splits under training and validation while also having L2 regularisation for reducing the impact of bias due to large weights. This also helps prevent overfitting.

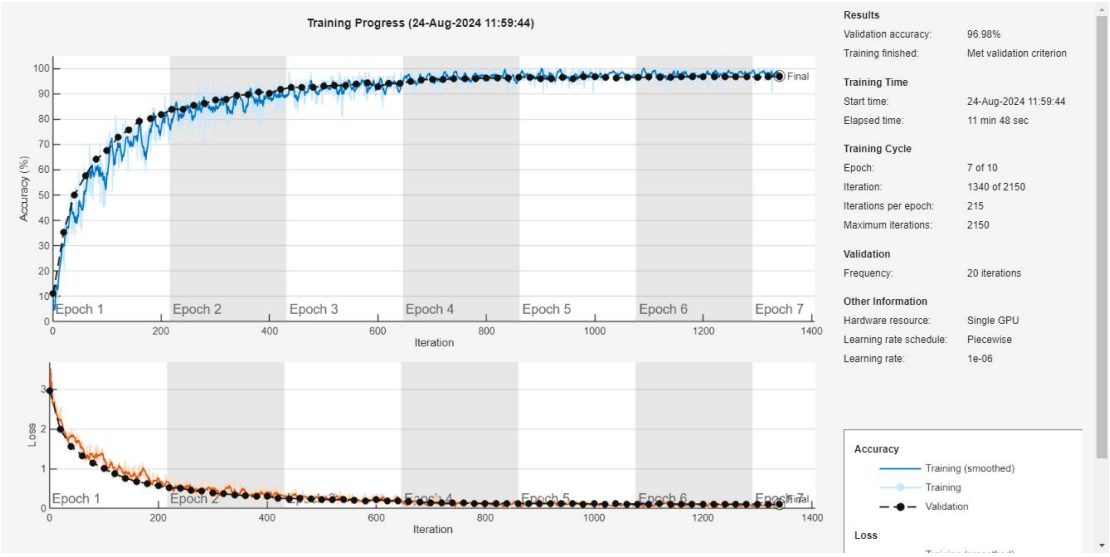

Figure 4.3: Training result for final gesture recognition algorithm

As can be seen in Figure 4.3, the training reaches up to 80% accuracy within the first epoch itself. This is due to enhanced training options that promote faster convergence. As we see the graph, the convergence flattens and hovers around a percentage range.So, as seen, the training stops early due to reaching validation criteria. The final model hence reaches 96.98

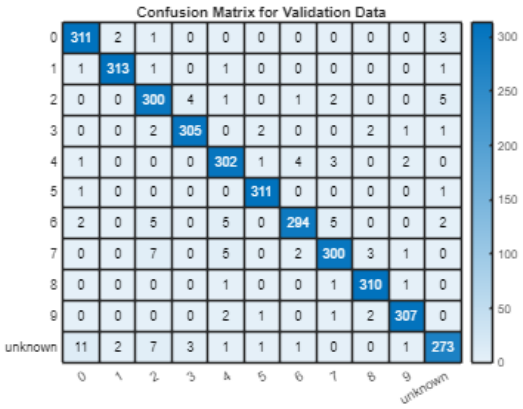

Figure 4.4: Confusion matrix for final gesture recognition algorithm

Figure 4.4 shows the confusion matrix for the final model. It shows that the distribution of training data has substantially improved with significant reductions in misclassifications between similar gestures.

Performance Metric	Baseline Model	Final Model
Accuracy	68.11%	96.98%
Precision	0.68	0.97
Recall	0.67	0.97
F1 Score	0.68	0.97

Table 4.5: Performance Metrics for Baseline and Final Gesture Recognition Models

As can be seen in Table 4.5, the model has 96.98% training accuracy with an F1 score of 0.97 that shows robust performance across all gesture classes. So the performance has improved from barely usable to highly usable between baseline and final models.

4.2 Speech Recognition Results and Discussion

In this section, the algorithm structure, training options, training results, confusion matrices, and performance evaluation of both the baseline and final models are presented and discussed.

4.2.1 Baseline Model Performance Analysis

The structure of the baseline model can be seen in 4.1. It is a fairly simple structure, with 192 learnable parameters and 5 layers. It contains a simple block with 1 Convolution2D layer, 1 batchbatch normalisation, 1 ReLU, and 1 MaxPooling layer. This was chosen as the starting model to establish a performance benchmark for gesture recognition.

CNN Layer Structure (Total Learnables: 192, Layers: 5)			
Layer Type	**Parameters**	**Output Size**	**Activation Function**
Image Input	-	[99x50x3]	-
Convolution 2D	Filter Size: 3x3, Number of Filters: 16	[99x50x16]	ReLU
Batch Normalization	-	[99x50x16]	-
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[49x25x16]	-
Fully Connected	Number of Classes: 11	11	-
Softmax	-	11	Softmax
Classification	-	11	-

Table 4.6: CNN Layer Structure for Baseline Speech Recognition Algorithm

Option	**Value**
Solver	Adam
MiniBatchSize	128
MaxEpochs	10
Shuffle	Every-epoch
ValidationData	dsVal
ValidationFrequency	20
Verbose	True
Plots	Training-progress
ExecutionEnvironment	GPU
InitialLearnRate	0.001
ValidationPatience	5

Table 4.7: Training Options for the Baseline Speech Recognition Algorithm

Table 4.7 shows the training options used for this simple structure. It uses Adam solver due to adaptive learning rates, which helps in faster convergence. We have a 0.001 learning rate, with 128 mini-batch size in 10 epochs of training, and early stopping was implemented based on validation performance.

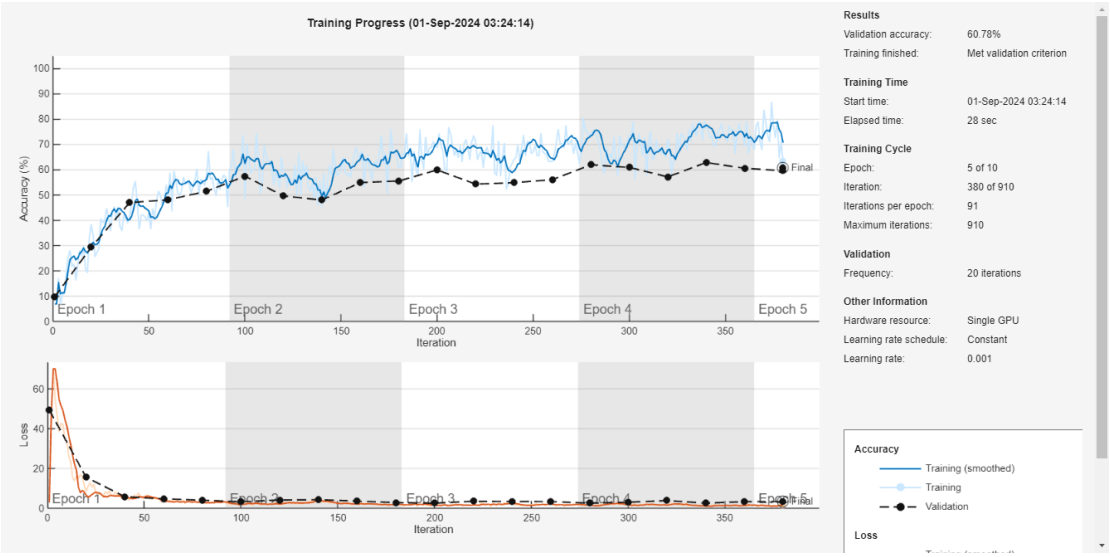

Figure 4.5: Training result for baseline speech-speech recognition algorithm

As can be seen in Figure 4.5, the baseline model achieved a training accuracy of 60.76%, which indicates limited generalisation ability. Given the simple structure and relatively simple training options, the accuracy sets performance metrics for the baseline model.

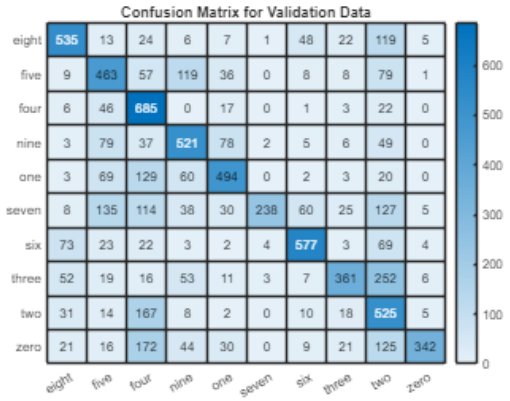

Figure 4.6: Confusion Matrix for baseline Speech Recognition algorithm

The overall performance of the model can be seen in table 4.10. If we take an F1 score of 0.61, we can say that the network is too basic and not suited for effective speech recognition.

4.2.2 Final Model Performance Analysis

CNN Layer Structure for Speech Recognition Algorithm (Total Learnables: 1M, Layers: 30)			
Layer Type	**Parameters**	**Output Size**	**Activation Function**
Image Input	-	[99x50x1]	-
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[99x50x128]	ReLU
Batch Normalisation	-	[99x50x128]	-
ReLU	-	[99x50x128]	ReLU
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[99x50x128]	ReLU
Batch Normalisation	-	[99x50x128]	-
ReLU	-	[99x50x128]	ReLU
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[50x25x128]	-
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[50x25x128]	ReLU
Batch Normalisation	-	[50x25x128]	-
ReLU	-	[50x25x128]	ReLU
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[50x25x128]	ReLU
Batch Normalisation	-	[50x25x128]	-
ReLU	-	[50x25x128]	ReLU
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[25x13x128]	-
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[25x13x128]	ReLU
Batch Normalisation	-	[25x13x128]	-
ReLU	-	[25x13x128]	ReLU
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[25x13x128]	ReLU
Batch Normalisation	-	[25x13x128]	-
ReLU	-	[25x13x128]	ReLU
Max Pooling 2D	Pool Size: 2x2, Stride: 2	[13x7x128]	-
Convolution 2D	Filter Size: 3x3, Number of Filters: 128, Padding: Same	[13x7x128]	ReLU
Batch Normalisation	-	[13x7x128]	-
ReLU	-	[13x7x128]	ReLU
Max Pooling 2D	Pool Size: 2x1, Stride: 1	[1x7x128]	-
Dropout	Dropout Rate: 0.2	[1x7x128]	-
Fully Connected	Number of Classes: 11	[11]	-
Softmax	-	[11]	Softmax
Classification	-	[11]	-

Table 4.8: CNN Layer Structure for Final Speech Recognition Algorithm

Table 4.8 shows the final model structure with 1 million learnable parameters and 30 layers. This shows that a significantly deeper architecture is designed, which can capture finer details in images. Layer blocks are designed to have 2 sets of Convolution2D, Batch Normalisation, and ReLU before having a MaxPooling layer. 4 blocks as such are designed before having a dropout layer followed by the fully connected layers, softmax

function, and classification output. The filter size is still 3x3, and the number of filters increased to 128. This helps in having more data in increased numbers of feature maps, which greatly enhances the recognition performance. A dropout layer 18 with 0.2 was also added to ensure random drop between connections of the neural network such that the model can prevent overfitting or memorising the data. This helps in learning the patterns and better model generalisation.

Option	**Value**
Solver	Adam
MiniBatchSize	64
LearnRateSchedule	Piecewise
LearnRateDropFactor	0.1
LearnRateDropPeriod	5
MaxEpochs	20
Shuffle	Every Epoch
L2Regularization	0.0001
ValidationData	dsVal
ValidationFrequency	20
Verbose	True
Plots	Training-progress
ExecutionEnvironment	GPU
InitialLearnRate	0.0001

Table 4.9: Training Options for Final Speech Recognition Algorithm

The table 4.9 shows the training options used. The full potential of the Adam solver is used here, where a piecewise learning rate is scheduled with a drop factor of 0.1 for every 5 epochs. The lower initial learning rate of 0.0001 as well as the reduced mini-batch size to 64 mean faster convergence and more batches per epoch. The data was shuffled every epoch to have new samples under training and validation while also having an L2 regularisation value of 0.0001 to reduce the impact of bias due to large weights and prevent overfitting.

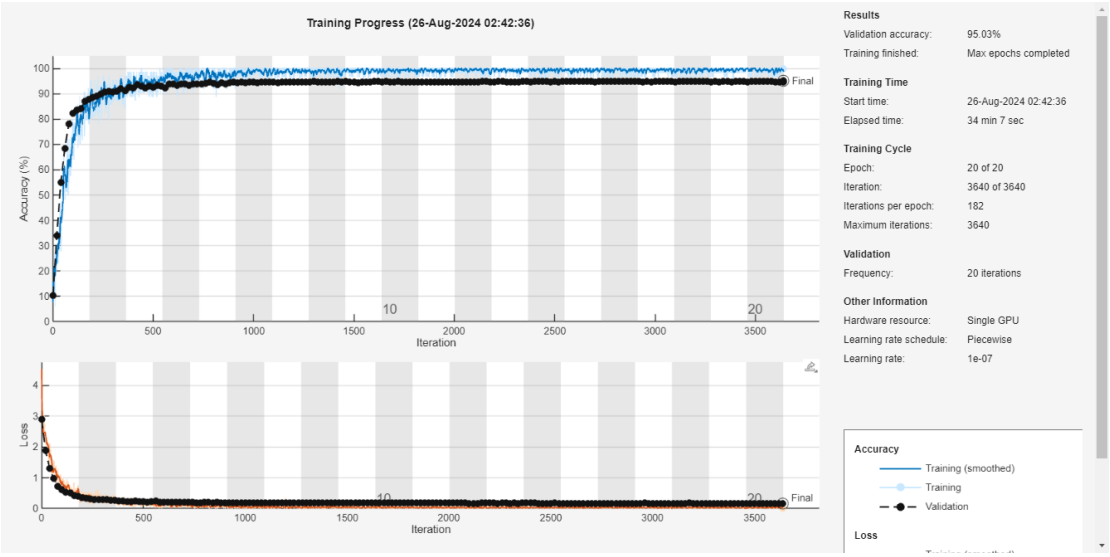

Figure 4.7: Training result for final Speech Recognition algorithm

As can be seen in Figure 4.7, the training reaches up to 80% accuracy within the first epoch itself. This is due to enhanced training options that promote faster convergence. Early stopping is not implemented here. The final model hence reaches 95.03% training accuracy, which shows a great improvement over the baseline.

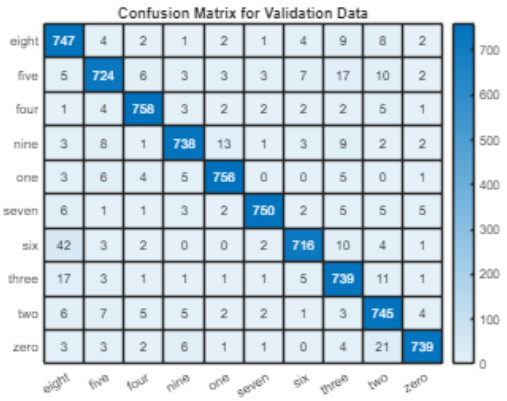

Figure 4.8: Confusion matrix for final speech recognition algorithm

Figure 4.8 shows the confusion matrix for the final model. It shows that the distribution of training data has substantially improved with significant reductions in misclassifications between similar commands.

Performance Metric	Baseline Model	Final Model
Accuracy	60.76%	95.03%
Precision	0.61	0.95
Recall	0.69	0.95
F1 Score	0.61	0.95

Table 4.10: Performance Metrics for Baseline and Final Speech Recognition Models

As can be seen in Table 4.10, the model has 95.03% training accuracy with an F1 score of 0.95 that shows robust performance across all command classes. So the performance has improved from barely usable to highly usable between baseline and final models.

4.3 Integrated System Performance and Discussion

4.3.1 Control of the Robotic Arm

Table 4.11: Pseudo Code for Servo Control

Initialize Libraries and Variables:

Include `Wire.h` and `Adafruit_PWMServoDriver.h`.

Initialize `Adafruit_PWMServoDriver pwm`.

Define `numServos = 11`.

Define `servoChannels[numServos] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`.

Define `positions[10][numServos]` with servo positions for gestures 0-9.

Setup Function:

Begin serial communication at 57600 baud.

Initialize the PWM driver.

Set PWM frequency to 60 Hz.

Print "Ready to receive gestures..."

Main Loop:

While Serial input is available:

Parse `gestureLabel` as an integer.

Clear the Serial buffer.

If `gestureLabel` is valid (0-9):

Print the received gesture.

Call `performGesture(gestureLabel)`.

Else:

Print "Invalid gesture received!"

Function: performGesture(number):

For each servo `i` from 0 to `numServos`:

Map the position to a PWM pulse width (150-600).

Set PWM for the servo channel `servoChannels[i]`.

The table 4.11 shows the pseudo code used in controlling the robotic arm. This code is uploaded into the memory of the Arduino Nano inside the Youbionic Handy Lite. As can be seen, the control for each digit is mapped onto the corresponding gesture to enact the digit. The classification output is sent over the com port in the form of a number,

which helps to enact the corresponding mapped gesture.

4.3.2 Overall System Results

Figures 4.9 and 4.10 demonstrate the live testing using gesture recognition. The robotic hand successfully executes the recognised gestures. The system's responsiveness and classification highlight its potential for real-world applications.

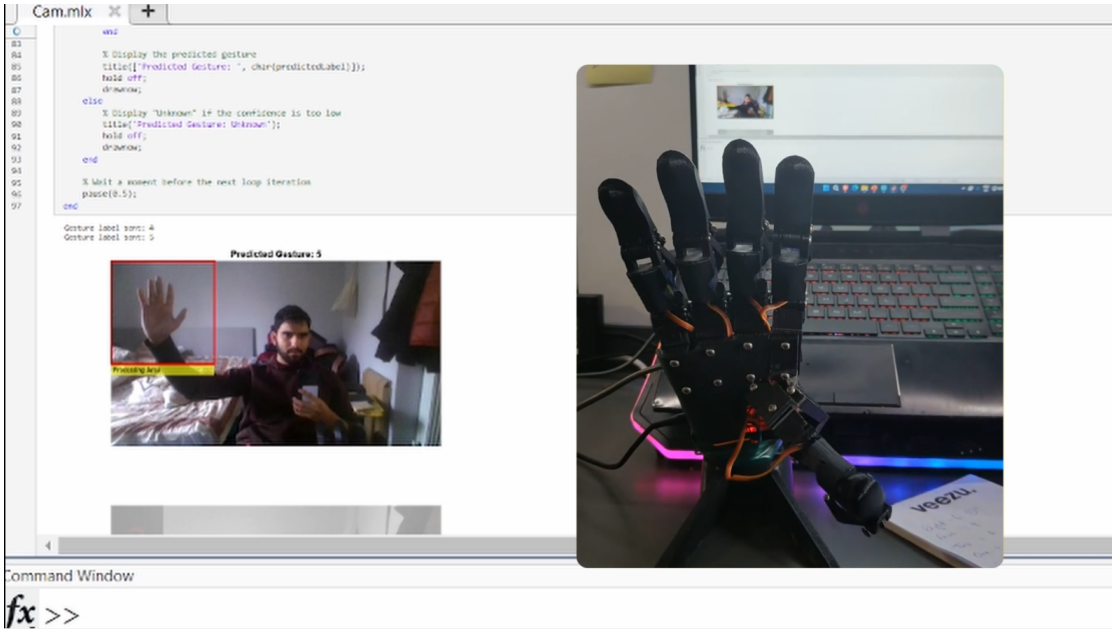

Figure 4.9: Testing the deep learning-based control system on gesture 5

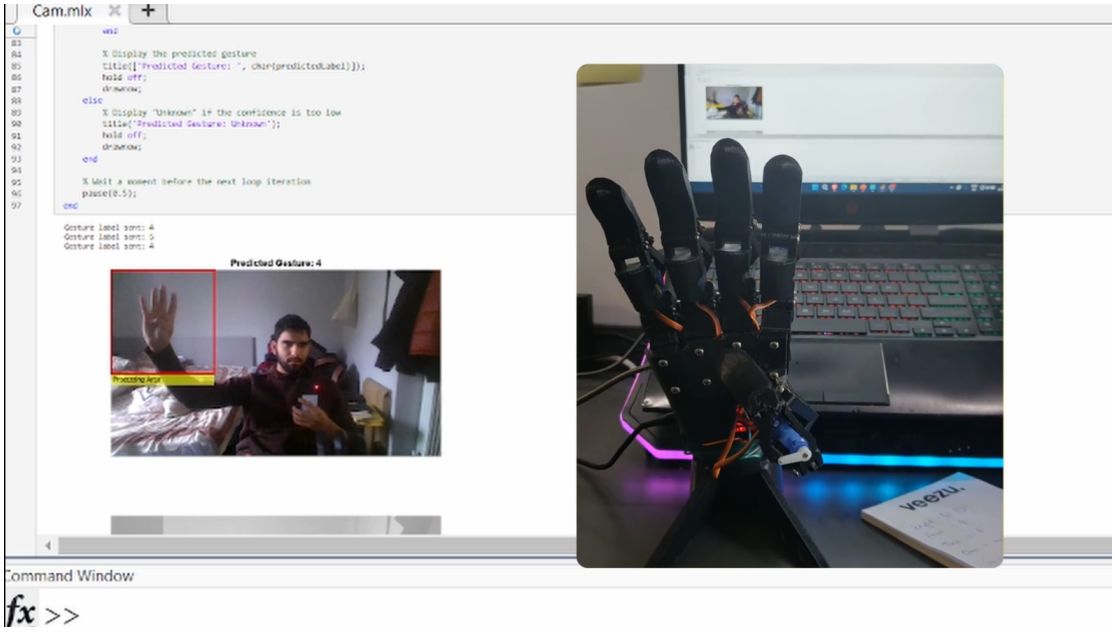

Figure 4.10: Testing the deep learning-based control system on gesture 4

4.4 Summary of Findings

The final models for both gesture and speech recognition significantly outperformed their baseline models with accuracies of 96.98% and 95.03%, respectively. Integrating the gesture model into a robotic arm control system showed the practical applicability of deep learning in assistive technologies.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this project, the aim was to control the robotic hand using a servo with the help of deep learning. Gesture and speech recognition algorithms were designed and enhanced using the Deep Learning Toolbox in MATLAB. Their final performance peaked, and when compared with the baseline models, they improved from unusable to robust performing models for classifying gestures and commands. Robotic hands were controlled using gesture recognition classification. All the main objectives are achieved, with a part of additional objectives achieved.

5.2 Limitations

Due to time constraints, controlling the robotic hand using speech recognition classification could not be achieved. The lack of huge computational overhead meant training the algorithms on massive datasets is not achievable. The scope of the study could be increased to using transfer learning from other popular algorithms to achieve better accuracy.

5.3 Future Work

Future work involves deploying speech recognition for testing. Employing Mel-Frequency Cepstral Coefficients (MFCCs)-based Recurrent Neural Networks (RNNs) for speech

recognition. Creating a finger and hand mapping algorithm in MATLAB if possible. Python has better support packages for the same, and using Python to recreate this project will give better approachability and inclusiveness to the project.

REFERENCES

[1] P. Bora and V. Nandi, "Low cost shadow function based articulated robotic arm," in *Proc. 2015 Int. Conf. Energy, Power and Environment: Towards Sustainable Growth (ICEPE)*, Shillong, India, Jun. 2015, pp. 1–4, doi: 10.1109/EPETSG.2015.7510079.

[2] D. H. Pal and S. M. Kakade, "Dynamic hand gesture recognition using kinect sensor," in *Proc. 2016 Int. Conf. Global Trends Signal Process. Inf. Comput. Commun. (ICGTSPICC)*, Jalgaon, India, Dec. 2016, pp. 448–453, doi: 10.1109/ICGTSPICC.2016.7955343.

[3] R. S. Pol, S. Giri, A. Ravishankar, and V. Ghode, "LabVIEW based four DoF robotic ARM," in *Proc. 2016 Int. Conf. Adv. Comput. Commun. Inform. (ICACCI)*, Jaipur, India, Sep. 2016, pp. 1791–1798, doi: 10.1109/ICACCI.2016.7732308.

[4] A. Bhargava and A. Kumar, "Arduino controlled robotic arm," in *Proc. 2017 Int. Conf. Electron. Commun. Aerosp. Technol. (ICECA)*, Coimbatore, India, Apr. 2017, pp. 376–380, doi: 10.1109/ICECA.2017.8212837.

[5] R. Mardiyanto, M. F. R. Utomo, D. Purwanto, and H. Suryoatmojo, "Development of hand gesture recognition sensor based on accelerometer and gyroscope for controlling arm of underwater remotely operated robot," in *Proc. 2017 Int. Seminar Intell. Technol. Appl. (ISITIA)*, Surabaya, Indonesia, Aug. 2017, pp. 329–333, doi: 10.1109/ISITIA.2017.8124104.

[6] R. K. Megalingam, S. Boddupalli, and K. G. S. Apuroop, "Robotic arm control through mimicking of miniature robotic arm," in *Proc. 2017 4th Int. Conf. Adv.*

Comput. Commun. Syst. (ICACCS), Coimbatore, India, Jan. 2017, pp. 1–7, doi: 10.1109/ICACCS.2017.8014622.

[7] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2Real Viewpoint Invariant Visual Servoing by Recurrent Control," in *Proc. 2018 IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, Jun. 2018, pp. 4691–4699, doi: 10.1109/CVPR.2018.00493.

[8] A. B. Bakri, R. Adnan, and F. A. Ruslan, "Wireless Hand Gesture Controlled Robotic Arm Via NRF24L01 Transceiver," in *Proc. 2019 IEEE 9th Symp. Comput. Appl. Ind. Electron. (ISCAIE)*, Malaysia, Apr. 2019, pp. 16–22, doi: 10.1109/ISCAIE.2019.8743772.

[9] N. K. Agrawal, V. K. Singh, V. S. Parmar, V. K. Sharma, D. Singh, and M. Agrawal, "Design and Development of IoT based Robotic Arm by using Arduino," in *Proc. 2020 4th Int. Conf. Comput. Methodol. Commun. (ICCMC)*, Erode, India, Mar. 2020, pp. 776–780, doi: 10.1109/ICCMC48092.2020.ICCMC-000144.

[10] P. Sihombing, R. B. Muhammad, H. Herriyance, and E. Elviwani, "Robotic Arm Controlling Based on Fingers and Hand Gesture," in *Proc. 2020 3rd Int. Conf. Mech. Electron. Comput. Ind. Technol. (MECnIT)*, Medan, Indonesia, Jun. 2020, pp. 40–45, doi: 10.1109/MECnIT48290.2020.9166592.

[11] A. S. Ahmed, H. A. Marzog, and L. A. Abdul-Rahaim, "Design and implement of robotic arm and control of moving via IoT with Arduino ESP32," *Int. J. Electr. Comput. Eng.*, vol. 11, no. 5, p. 3924, Oct. 2021, doi: 10.11591/ijece.v11i5.pp3924-3933.

[12] H. M. Ali, Y. Hashim, and G. A. Al-Sakkal, "Design and implementation of Arduino based robotic arm," *Int. J. Electr. Comput. Eng.*, vol. 12, no. 2, p. 1411, Apr. 2022, doi: 10.11591/ijece.v12i2.pp1411-1418.

[13] L. Bolc, R. Tadeusiewicz, L. J. Chmielewski, and K. Wojciechowski, Eds., *Computer Vision and Graphics: Int. Conf., ICCVG 2012, Warsaw, Poland, Sep. 24-26, 2012. Proc.*, ser. Lecture Notes in Comput. Sci., vol. 7594, Berlin, Heidelberg: Springer, 2012, doi: 10.1007/978-3-642-33564-8.

[14] B. Eldridge, K. Gruben, D. LaRose, J. Funda, S. Gomory, J. Karidis, G. McVicker, R. Taylor, and J. Anderson, “A remote center of motion robotic arm for computer assisted surgery,” *Robotica*, vol. 14, no. 1, pp. 103–109, Jan. 1996, doi: 10.1017/S0263574700018981.

[15] P. Schrock, F. Farelo, R. Alqasemi, and R. Dubey, “Design, simulation and testing of a new modular wheelchair mounted robotic arm to perform activities of daily living,” in *Proc. 2009 IEEE Int. Conf. Rehabil. Robot. (ICORR)*, Kyoto, Japan, Jun. 2009, pp. 518–523, doi: 10.1109/ICORR.2009.5209469.

[16] W. M. H. W. Kadir, R. E. Samin, and B. S. K. Ibrahim, “Internet Controlled Robotic Arm,” *Procedia Eng.*, vol. 41, pp. 1065–1071, 2012, doi: 10.1016/j.proeng.2012.07.284.

[17] A. Cela, J. Yebes, R. Arroyo, L. Bergasa, R. Barea, and E. López, “Complete Low-Cost Implementation of a Teleoperated Control System for a Humanoid Robot,” *Sensors*, vol. 13, no. 2, pp. 1385–1401, Jan. 2013, doi: 10.3390/s130201385.

[18] S. Verma, “Hand Gestures Remote Controlled Robotic Arm,” *Adv. Electron. Electr. Eng.*, vol. 3, no. 5, pp. 601–606, 2013.

Chapter 6

Appendix

6.1 Video Links

Please find the videos for training and testing int he following Google Drive:
https://drive.google.com/drive/folders/1KP0POeDZWSfnxr6uRZd1hl3x-wURtOya?usp=sharing

6.2 Training options

MATLAB Training Options and Specific Solver Settings	
Training Option	**Possible Values**
`Solver`	`'adam', 'sgdm', 'rmsprop', 'sgd'`
`InitialLearnRate`	Any positive scalar (e.g., 0.001, 0.01)
`Momentum` (for `sgdm`, `sgd`)	Any scalar between 0 and 1 (e.g., 0.9)
`Beta1, Beta2` (for `adam`)	Any positive scalar (e.g., 0.9, 0.999)
`Rho` (for `rmsprop`)	Any positive scalar (e.g., 0.9)
`LearnRateSchedule` (for `adam`, `sgdm`)	`'none', 'piecewise', 'cosine'`
`LearnRateDropFactor` (for `adam`, `sgdm`)	Any positive scalar (e.g., 0.1, 0.5)
`LearnRateDropPeriod` (for `adam`, `sgdm`)	Any positive integer (e.g., 10, 20)
`Epsilon` (for `adam`, `rmsprop`)	Any positive scalar (e.g., 1e-8, 1e-6)
`GradientDecayFactor` (for `adam`)	Any positive scalar (e.g., 0.9)
`SquaredGradientDecayFactor` (for `rmsprop`)	Any positive scalar (e.g., 0.99)
`MaxEpochs`	Any positive integer (e.g., 10, 50, 100)
`MiniBatchSize`	Any positive integer (e.g., 16, 32, 64)
`Shuffle`	`'never', 'once', 'every-epoch'`
`Verbose`	`true, false`
`VerboseFrequency`	Any positive integer (e.g., 50, 100)
`ValidationData`	`Validation dataset (e.g., XVal, YVal)`
`ValidationFrequency`	Any positive integer (e.g., 50, 100)
`ValidationPatience`	Any positive integer (e.g., 5, 10)
`Plots`	`'none', 'training-progress', 'training-error'`
`GradientThreshold`	Any positive scalar (e.g., 1, 10)
`GradientThresholdMethod`	`'l2norm', 'global-l2norm', 'absolute-value'`
`L2Regularization`	Any positive scalar (e.g., 0.0001, 0.01)
`ExecutionEnvironment`	`'auto', 'cpu', 'gpu', 'multi-gpu', 'parallel'`
`CheckpointPath`	`File path to save checkpoints`
`OutputFcn`	`Handle to custom function`

Table 6.1: MATLAB Training Options and Specific Solver Settings

6.3 MATLAB Code Listings for Gesture Recognition

6.3.1 Data Acquisition using webcam

```
clc;
clear all;
close all;
warning off;

c = webcam;
```

```
7
8  x = 0;
9  y = 0;
10 height = 200;
11 width = 200;
12 bboxes = [x y height width];
13
14 num_classes = 10;
15 num_images = 100;
16
17 class_names = {'zero', 'one', 'two', 'three', 'four', 'five', 'six',
       'seven', 'eight', 'nine'};
18
19 for class_idx = 1:num_classes
20     output_folder = class_names{class_idx};
21     if ~exist(output_folder, 'dir')
22         mkdir(output_folder);
23     end
24
25     disp(['Capturing images for class: ', output_folder]);
26
27     temp = 0;
28     tic;
29     while temp < num_images
30         e = c.snapshot;
31
32         IFaces = insertObjectAnnotation(e, 'rectangle', bboxes, '
               Processing Area');
33
34         imshow(IFaces);
35
36         es = imcrop(e, bboxes);
37
38         es = imresize(es, [98 50]);
39
40         es_gray = rgb2gray(es);
41
42         filename = fullfile(output_folder, strcat(num2str(temp), '.
               bmp'));
```

```
43
44          imwrite(es_gray, filename);
45
46          temp = temp + 1;
47
48          drawnow;
49      end
50      elapsedTime = toc;
51      disp(['Time taken to capture 100 images for class ',
            output_folder, ': ', num2str(elapsedTime), ' seconds']);
52  end
53
54  clear c;
55
56  disp('Image capturing complete.');
```

Listing 6.1: Code for performing data acquisition using webcam

6.3.2 Gesture Recognition Algorithm training

```
1   clc
2   clear all
3   close all
4   rng(1,'twister');
5
6   imds = imageDatastore('NumbersInSignLanguage_16500', '
        IncludeSubfolders', true, 'LabelSource', 'foldernames');
7
8   labelCountBeforeSplit = countEachLabel(imds);
9   disp('Number of images per label before splitting:');
10  disp(labelCountBeforeSplit);
11
12  [imdsTrain, imdsVal] = splitEachLabel(imds, 0.8, 'randomized');
13
14  labelCountTrain = countEachLabel(imdsTrain);
15  labelCountVal = countEachLabel(imdsVal);
16
17  disp('Number of images per label in training set:');
18  disp(labelCountTrain);
```

```
19
20 disp('Number of images per label in validation set:');
21 disp(labelCountVal);
22
23 imageAugmenter = imageDataAugmenter('RandXScale',[0.85 1.35],'
       RandYScale',[0.85 1.35]);
24
25 image_size = [98 50 3];
26
27 dsTrain = augmentedImageDatastore(image_size, imdsTrain, '
       ColorPreprocessing', 'gray2rgb', 'DataAugmentation',
       imageAugmenter);
28 dsVal = augmentedImageDatastore(image_size, imdsVal, '
       ColorPreprocessing', 'gray2rgb', 'DataAugmentation',
       imageAugmenter, 'OutputSizeMode', 'resize');
29
30 numTrainImages = numel(dsTrain.Files);
31 numValImages = numel(dsVal.Files);
32
33 disp(['Number of training images after augmentation: ', num2str(
       numTrainImages)]);
34 disp(['Number of validation images after augmentation: ', num2str(
       numValImages)]);
35
36 YValidation = imdsVal.Labels;
37 num_classes = numel(categories(imdsTrain.Labels));
38 num_filters = 128;
39 dropout_rate = 0.2;
40 learning_rate = 0.001;
41
42 layers = [
43     imageInputLayer([image_size])
44
45     convolution2dLayer([5 5],num_filters,'Padding','same')
46     batchNormalizationLayer
47     reluLayer
48
49     convolution2dLayer([5 5],num_filters,'Padding','same')
50     batchNormalizationLayer
```

```
    reluLayer

    maxPooling2dLayer(2,"Stride",2)

    convolution2dLayer([5 5],num_filters,'Padding','same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer([5 5],num_filters,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,"Stride",2)

    convolution2dLayer([5 5],num_filters,'Padding','same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer([5 5],num_filters,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,"Stride",2)

    convolution2dLayer([5 5],num_filters,'Padding','same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer([5 5],num_filters,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,"Stride",2)

    dropoutLayer(dropout_rate)
    ];

analyzeNetwork(layers);
```

```
layers = [
    layers

    fullyConnectedLayer(num_classes)
    softmaxLayer
    classificationLayer
    ];

validationPatience = 5;

options = trainingOptions('adam', ...
"MiniBatchSize",64, ...
'LearnRateSchedule', 'piecewise', ...
'LearnRateDropFactor',0.1, ...
'LearnRateDropPeriod',3, ...
'MaxEpochs',10, ...
'Shuffle','every-epoch', ...
'L2Regularization',0.0001, ...
'ValidationData',dsVal, ...
'ValidationFrequency',20, ...
'Verbose',true, ...
'Plots','training-progress', ...
'ExecutionEnvironment','gpu', ...
'InitialLearnRate', 0.0001, ...
'ValidationPatience', validationPatience);

net = trainNetwork(dsTrain, layers, options);

save('trainedGestureModel.mat', 'net');

YPred = classify(net,dsVal);

YPred_onehot = zeros(numel(YPred), num_classes);
for j = 1:numel(YPred)
    YPred_onehot(j, YPred(j)) = 1;
end

confMat = confusionmat(YValidation, YPred);
figure
```

```
129
130 heatmap(categories(YValidation), categories(YValidation), confMat);
131 title('Confusion Matrix for Validation Data')
132
133 TP = diag(confMat);
134 FP = sum(confMat, 2) - TP;
135 FN = sum(confMat, 1)' - TP;
136 precision = TP ./ (TP + FP);
137 recall = TP ./ (TP + FN);
138 F1 = 2 * (precision .* recall) ./ (precision + recall);
139 accuracy = sum(YPred == YValidation)/numel(YValidation);
140
141 fprintf('Validation Accuracy: %.2f%%\n', accuracy*100);
142 fprintf('Precision: %.2f\n', mean(precision));
143 fprintf('Recall: %.2f\n', mean(recall));
144 fprintf('F1 Score: %.2f\n', mean(F1));
```

Listing 6.2: Code to train the Gesture Recognition Algorithm

6.3.3 Testing Gesture Recognition Algorithm using Webcam

```
1  clc;
2  close all;
3  clear all;
4
5  c = webcam;
6  load('trainedGestureModel.mat');
7
8  roiX = 1;
9  roiY = 1;
10 roiWidth = 200;
11 roiHeight = 200;
12
13 bboxes = [roiX roiY roiWidth roiHeight];
14
15 arduinoSerial = serialport('COM4', 57600);
16
17 previousGesture = "";
18
```

```
while true
    for countdown = 3:-1:1
        img = snapshot(c);
        annotatedImg = insertText(img, [roiX, roiY], num2str(
            countdown), ...
                                  'FontSize', 80, 'TextColor', 'red',
                                      ...
                                  'BoxColor', 'black', 'BoxOpacity',
                                     0.6);
        imshow(annotatedImg);
        pause(1);
    end

    img = snapshot(c);
    annotatedImg = insertObjectAnnotation(img, 'rectangle', bboxes, '
        Processing␣Area');
    roi = img(roiY:roiY+roiHeight-1, roiX:roiX+roiWidth-1, :);
    roiGray = rgb2gray(roi);
    roiRGB = cat(3, roiGray, roiGray, roiGray);
    roiResized = imresize(roiRGB, [98 50]);

    [predictedIdx, scores] = classify(net, roiResized);
    predictedLabel = string(predictedIdx);

    confidenceThreshold = 0.3;

    imshow(annotatedImg);
    hold on;
    rectangle('Position', [roiX, roiY, roiWidth, roiHeight], ...
              'EdgeColor', 'r', 'LineWidth', 2);

    if max(scores) >= confidenceThreshold && predictedLabel ~= "
        unknown"
        if predictedLabel ~= previousGesture {
            writeline(arduinoSerial, num2str(double(predictedIdx) -
                1));
            disp(['Gesture␣label␣sent:␣', num2str(double(predictedIdx
                ) - 1)]);
            previousGesture = predictedLabel;
```

```
        end

        title(['Predicted Gesture: ', char(predictedLabel)]);
        hold off;
        drawnow;
    else
        title('Predicted Gesture: Unknown');
        hold off;
        drawnow;
    end

    pause(0.5);
end

clear c;
clear arduinoSerial;
```

Listing 6.3: Code for using webcam to test the trained data

6.4 MATLAB Code Listings for Speech Recognition

6.4.1 Data Acquisition using microphone

```
clc;
clear all;
close all;

fs = 16000;
recordingDuration = 3;
numRecordings = 40;
pauseDuration = 1;
outputBaseDir = 'AudioDataset';
targetDuration = 1;

classLabels = {'zero', 'one', 'two', 'three', 'four', 'five', 'six',
    'seven', 'eight', 'nine'};
for i = 1:length(classLabels)
    classDir = fullfile(outputBaseDir, classLabels{i});
```

```
    if ~exist(classDir, 'dir')
        mkdir(classDir);
    end
end

startIndex = 1711;
recorder = audiorecorder(fs, 16, 1);

for digitIdx = 1:length(classLabels)
    digitLabel = classLabels{digitIdx};
    fprintf('Now recording for digit: %s\n', digitLabel);

    for recNum = 1:numRecordings
        fprintf('Recording %d of %d for digit %s\n', recNum,
            numRecordings, digitLabel);

        for countdown = 3:-1:1
            fprintf('%d...\n', countdown);
            pause(1);
        end

        fprintf('Start speaking now!\n');
        recordblocking(recorder, recordingDuration);
        audioData = getaudiodata(recorder);
        fprintf('Recording completed for this instance.\n');

        trimmedAudio = trimAudioToSpeech(audioData, fs,
            targetDuration);

        filename = fullfile(outputBaseDir, digitLabel, sprintf('%s_%d
            .wav', digitLabel, startIndex + recNum - 1));
        audiowrite(filename, trimmedAudio, fs);

        fprintf('Please wait...\n');
        pause(pauseDuration);
    end

    fprintf('Finished recording for digit: %s\n\n', digitLabel);
end
```

```
51
52 fprintf('All recordings completed successfully!\n');
53 clear recorder;
54
55 function trimmedAudio = trimAudioToSpeech(audioData, fs,
       targetDuration)
56     energy = audioData.^2;
57     threshold = 0.01 * max(energy);
58     speechIndices = find(energy > threshold);
59
60     if isempty(speechIndices)
61         trimmedAudio = audioData(1:min(targetDuration*fs, length(
               audioData)));
62     else
63         startIndex = max(speechIndices(1) - round(0.1 * fs), 1);
64         endIndex = min(startIndex + targetDuration * fs - 1, length(
               audioData));
65         trimmedAudio = audioData(startIndex:endIndex);
66     end
67
68     if length(trimmedAudio) < targetDuration * fs
69         trimmedAudio = [trimmedAudio; zeros(targetDuration * fs -
               length(trimmedAudio), 1)];
70     end
71 end
```

Listing 6.4: Code for performing data acquisition using microphone

6.4.2 Preprocessing audio files to generate spectrograms

```
1 clear all;
2 rng(1,'twister');
3
4 datasetFolder = 'Speech_Numbers_17000';
5 ads = audioDatastore(datasetFolder, 'IncludeSubfolders', true, '
      FileExtensions', '.wav', 'LabelSource', 'foldernames');
6
7 commands = categorical(["one","two","three","four","five","six","
      seven","eight","nine","zero","background"]);
```

```
isCommand = ismember(ads.Labels, commands);
isUnknown = ~ismember(ads.Labels, [commands, "_background_noise_"]);

includeFraction = 1;
mask = rand(numel(ads.Labels), 1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

adsSubset = subset(ads, isCommand | isUnknown);
countEachLabel(adsSubset)

p1 = 0.6;
p2 = 0.4;
[adsTrain, adsValidation] = splitEachLabel(adsSubset, p1);
numUniqueLabels = numel(unique(adsTrain.Labels));

numTrainFiles = numel(adsTrain.Files);
numValidationFiles = numel(adsValidation.Files);

disp(['Number of training files: ', num2str(numTrainFiles)]);
disp(['Number of validation files: ', num2str(numValidationFiles)]);

fs = 16e3;
segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;
segmentSamples = round(segmentDuration * fs);
frameSamples = round(frameDuration * fs);
hopSamples = round(hopDuration * fs);
overlapSamples = frameSamples - hopSamples;
FFTLength = 512;
numBands = 50;

afe = audioFeatureExtractor('SampleRate', fs, 'FFTLength', FFTLength,
    'Window', hann(frameSamples, 'periodic'), 'OverlapLength',
   overlapSamples, 'barkSpectrum', true);
setExtractorParameters(afe, 'barkSpectrum', 'NumBands', numBands);

numHops = floor((segmentSamples - overlapSamples) / hopSamples) + 1;
```

```
45
46 XTrain = zeros(numHops, numBands, 1, numel(adsTrain.Files));
47 XValidation = zeros(numHops, numBands, 1, numel(adsValidation.Files))
       ;
48
49 subds = partition(adsTrain, 1, 1);
50 for idx = 1:numel(subds.Files)
51     x = read(subds);
52     xPadded = [zeros(floor((segmentSamples - numel(x)) / 2), 1); x;
           zeros(ceil((segmentSamples - numel(x)) / 2), 1)];
53     features = extract(afe, xPadded);
54     if size(features, 1) < numHops
55         paddedFeatures = zeros(numHops, numBands);
56         paddedFeatures(1:size(features, 1), :) = features;
57         XTrain(:,:,:,idx) = paddedFeatures;
58     else
59         XTrain(:,:,:,idx) = features(1:numHops, :);
60     end
61 end
62
63 subds = partition(adsValidation, 1, 1);
64 for idx = 1:numel(subds.Files)
65     x = read(subds);
66     xPadded = [zeros(floor((segmentSamples - numel(x)) / 2), 1); x;
           zeros(ceil((segmentSamples - numel(x)) / 2), 1)];
67     features = extract(afe, xPadded);
68     if size(features, 1) < numHops
69         paddedFeatures = zeros(numHops, numBands);
70         paddedFeatures(1:size(features, 1), :) = features;
71         XValidation(:,:,:,idx) = paddedFeatures;
72     else
73         XValidation(:,:,:,idx) = features(1:numHops, :);
74     end
75 end
76
77 unNorm = 2 / (sum(afe.Window)^2);
78 epsil = 1e-6;
79
80 XTrain = XTrain / unNorm;
```

```
XTrain = log10(XTrain + epsil);

XValidation = XValidation / unNorm;
XValidation = log10(XValidation + epsil);

YTrain = removecats(adsTrain.Labels);
YValidation = removecats(adsValidation.Labels);

specMin = min(XTrain, [], 'all');
specMax = max(XTrain, [], 'all');
idx = randperm(numel(adsTrain.Files), 3);
figure('Units', 'normalized', 'Position', [0.2 0.2 0.6 0.6]);
for i = 1:3
    [x, fs] = audioread(adsTrain.Files{idx(i)});
    subplot(2, 3, i)
    plot(x)
    axis tight
    title(string(adsTrain.Labels(idx(i))))

    subplot(2, 3, i + 3)
    spect = (XTrain(:,:,1,idx(i))');
    pcolor(spect)
    caxis([specMin specMax])
    shading flat

    sound(x, fs)
    pause(2)
end

imageDir = 'SpeechImageData';
if ~exist(imageDir, 'dir')
    mkdir(imageDir);
end

for i = 1:size(XTrain, 4)
    label = YTrain(i);
    labelDir = fullfile(imageDir, 'TrainingData', char(label));
    if ~exist(labelDir, 'dir')
        mkdir(labelDir);
```

```
    end
    tmp_image = XTrain(:,:,1,i);
    tmp_image = mat2gray(tmp_image);
    fileName = fullfile(labelDir, ['image' num2str(i) '.png']);
    imwrite(tmp_image, fileName);
end

for i = 1:size(XValidation, 4)
    label = YValidation(i);
    labelDir = fullfile(imageDir, 'ValidationData', char(label));
    if ~exist(labelDir, 'dir')
        mkdir(labelDir);
    end
    tmp_image = XValidation(:,:,1,i);
    tmp_image = mat2gray(tmp_image);
    fileName = fullfile(labelDir, ['image' num2str(i) '.png']);
    imwrite(tmp_image, fileName);
end

numProcessedTrainFiles = numel(adsTrain.Files);
numProcessedValidationFiles = numel(adsValidation.Files);

fprintf('Preprocessing is complete.\n');
fprintf('Number of training files processed: %d\n',
    numProcessedTrainFiles);
fprintf('Number of validation files processed: %d\n',
    numProcessedValidationFiles);
```

Listing 6.5: Code to convert audio files into spectrograms

6.4.3 Speech Recognition Algorithm training

```
clc;
clear all;
rng(1, 'twister');

dsTrain = imageDatastore('TrainingData', "IncludeSubfolders", true, "
    LabelSource", "foldernames");
```

```
dsVal = imageDatastore('ValidationData', "IncludeSubfolders", true, "
    LabelSource", "foldernames");

image_size = [99 50 1];

timepoolSize = 12;
YValidation = dsVal.Labels;
num_classes = numel(categories(dsTrain.Labels));
num_filters = 128;
filter_sizes = [3 3];
dropout_rate = 0.2;
learning_rate = 0.001;

layers = [
    imageInputLayer(image_size)

    convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, "Stride", 2)

    convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2, "Stride", 2)

    convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
```

```
44
45      convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
46      batchNormalizationLayer
47      reluLayer
48
49      maxPooling2dLayer(2, "Stride", 2)
50
51      convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
52      batchNormalizationLayer
53      reluLayer
54
55      convolution2dLayer(filter_sizes, num_filters, 'Padding', 'same')
56      batchNormalizationLayer
57      reluLayer
58
59      maxPooling2dLayer([timepoolSize, 1], "Stride", 1)
60
61      dropoutLayer(dropout_rate)
62      ];
63
64  analyzeNetwork(layers);
65
66  layers = [
67      layers
68
69      fullyConnectedLayer(num_classes)
70      softmaxLayer
71      classificationLayer
72      ];
73
74  validationPatience = 5;
75  options = trainingOptions('adam', ...
76      "MiniBatchSize", 64, ...
77      'LearnRateSchedule', 'piecewise', ...
78      'LearnRateDropFactor', 0.1, ...
79      'LearnRateDropPeriod', 5, ...
80      'MaxEpochs', 20, ...
81      'Shuffle', 'every-epoch', ...
82      'L2Regularization', 0.0001, ...
```

```
    'ValidationData', dsVal, ...
    'ValidationFrequency', 20, ...
    'Verbose', true, ...
    'Plots', 'training-progress', ...
    'ExecutionEnvironment', 'gpu', ...
    'InitialLearnRate', 0.0001);

net = trainNetwork(dsTrain, layers, options);

save('trainedSpeechModel.mat', 'net');

YPred = classify(net, dsVal);
confMat = confusionmat(YValidation, YPred);
figure;
heatmap(categories(YValidation), categories(YValidation), confMat);
title('Confusion Matrix for Validation Data');

TP = diag(confMat);
FP = sum(confMat, 2) - TP;
FN = sum(confMat, 1)' - TP;
precision = TP ./ (TP + FP);
recall = TP ./ (TP + FN);
F1 = 2 * (precision .* recall) ./ (precision + recall);
accuracy = sum(YPred == YValidation) / numel(YValidation);

fprintf('Validation Accuracy: %.2f%%\n', accuracy * 100);
fprintf('Precision: %.2f\n', mean(precision));
fprintf('Recall: %.2f\n', mean(recall));
fprintf('F1 Score: %.2f\n', mean(F1));
```

Listing 6.6: Code to train the Speech Recognition Algorithm

6.5 Arduino Code for Servo Control

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
```

```
5
6  const int numServos = 11;
7  int servoChannels[numServos] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
8
9  int positions[10][numServos] = {
10   {30, 30, 0, 0, 0, 0, 0, 0, 60, 90, 180},
11   {170, 180, 0, 0, 0, 0, 0, 0, 60, 120, 180},
12   {170, 180, 170, 180, 0, 0, 0, 0, 30, 0, 180},
13   {180, 180, 180, 180, 0, 0, 0, 0, 180, 90, 180},
14   {180, 180, 180, 180, 180, 180, 180, 180, 0, 90, 180},
15   {180, 180, 180, 180, 180, 180, 180, 180, 180, 90, 180},
16   {180, 180, 180, 180, 180, 180, 0, 0, 40, 120, 180}, //
17   {180, 180, 180, 180, 0, 0, 180, 180, 0, 90, 180}, //
18   {180, 180, 0, 0, 180, 180, 180, 180, 60, 120, 180}, //
19   {90, 0, 180, 180, 180, 180, 180, 180, 60, 150, 180}  //
20 };
21
22 void setup() {
23   Serial.begin(57600);
24   pwm.begin();
25   pwm.setPWMFreq(60);
26
27   Serial.println("Ready␣to␣receive␣gestures...");
28 }
29
30 void loop() {
31   if (Serial.available()) {
32     int gestureLabel = Serial.parseInt();
33
34     while (Serial.available() > 0) {
35       Serial.read();
36     }
37
38     if (gestureLabel >= 0 && gestureLabel < 10) {
39       Serial.print("Performing␣gesture␣for:␣");
40       Serial.println(gestureLabel);
41       performGesture(gestureLabel);
42     } else {
43       Serial.println("Invalid␣gesture␣received!");
```

```
    }
  }
}

void performGesture(int number) {
  for (int i = 0; i < numServos; i++) {
    int pulse = map(positions[number][i], 0, 180, 150, 600);
    pwm.setPWM(servoChannels[i], 0, pulse);
  }
}
```

Listing 6.7: Arduino Code for Servo Control of Youbionic Handy Lite