Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Undergraduate Dissertation

Human Gesture Recognition
Skeletal Data based Gesture Recognition Analysis

Author:
Pulkit Sharma

Supervisor:
Dr Gerard Lacey

A dissertation submitted in partial fulfillment of the requirements for

the degree:

Bachelor's in Computer Engineering

Submitted to the University of Dublin, Trinity College, May 2021

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

Name _____ **Date**_____

Acknowledgment

I would like to express my thanks and gratitude to my supervisor Dr. Gerard Lacey for his valuable contribution and guidance throughout my dissertation. He has been very patient with me and has given me valuable feedback after every meeting we had regarding the project. Without his continuous support and advice, this project would not have been possible.

I would also like to thank my family and my friends for their constant support and patience throughout this project.

Abstract

Human Gestures can be defined as a non-verbal symbol of physical behavior or emotional expression that contains some information. Human gesture recognition has a wide range of applications in computer vision, especially in human-computer interaction applications. Training Gesture recognition models have a variety of approaches such as the 3D model approach, skeletal approach, etc. We are interested in implementing a skeletal model-based approach in this project.

BlazePose is a lightweight architecture for human pose estimation based on CNN which divides the body into 33 key skeletal points which can then be used for fitness tracking, sign language recognition, etc. We use this architecture to convert our dataset's images into a time series of skeletal data points which will then serve as an input to our deep learning model. Advanced deep learning techniques such as CNN have been successful in both spatial and temporal domains due to their powerful hierarchical features. Hence, we use a parallel convolution model and train it on these skeletal coordinates and evaluate the performance of the model.

We are interested in finding out How does a high-level representation of data i.e. skeletal coordinates of key landmark points of the body affect the accuracy and training time of a deep learning model? We will investigate our Blazepose + CNN model on the Jester 20BN dataset to accurately track human gestures. We are also interested in finding out how does a model that is trained on these skeletal points generalize i.e. what happens to the train and test accuracies of the model when it is trained on one type of dataset and tested on other.

Table of Contents

List of Tables

List of Figures

List of Equations

Chapter 1

Introduction:

In this chapter, we go into the background of Gesture recognition and the motivation for choosing this project. We also briefly discuss the project objectives in section 1.3 and the flow of this dissertation in section 1.4

1.1 Background:

Human Gestures can be defined as a non-verbal symbol of physical behavior or emotional expression that contains some information. According to Edward T.Hall, up to 60% of all our communications are non-verbal[1]. Vision-based technology of Gesture recognition determines the user intent by recognizing gestures or movements of body parts. It has excellent applications in research areas such as Human-Computer Interaction(HCI), Human-Robot Interaction(HRI), Sign language recognition, and augmented reality (virtual reality)[2], [3]. Human Gesture recognition from video sequences has many practical applications in computer vision, such as surveillance systems, motion analysis in sports, monitoring of patients, and human behavior analysis. Researchers see gesture recognition as a way for machines to understand human body language better, bridging the gap between machines and humans and allowing them to communicate without any mechanical devices like a mouse and keyboard.

1.2 Motivation:

Gesture recognition has recently become one of the most attractive research fields in pattern recognition due to its use cases like HCI. While many studies and approaches have been based on training models directly on the dataset's images, there are not many approaches that investigate training a model on sparse representation of data(Skeletal coordinates). This was due to the lack of relevant architectures that could extract the skeletal data from RGB images in real-time and give consistent results. But currently, due to advancements and research in the computer vision field, many architectures like LeapMotion, OpenPose, BlazePose have been developed, which allows users to calculate skeletal data in real-time and give consistent results. We will be experimenting with BlazePose architecture in this Project. We are particularly interested in this approach because training a model on images/videos requires state-of-the-art hardware. It still takes a lot of training time because processing an RGB image is GPU intensive task. Processing a 3D skeletal image is relatively less GPU intensive[4]. If this approach yields good results, it will shorten the time for training Gesture Recognition models. Hence, we are interested in implementing this idea of the BlazePose + CNN model and evaluating the results.

If this alternate approach performs well, it will also enable people who do not have access to state-of-the-art hardware to train their deep learning models. We are also interested in finding out how this model generalizes, i.e. how the model performs when tested on a different dataset than the one it has been trained on. We want to understand the data sensitivity of this type of model. We want to see if this approach is scalable and if a model trained on skeletal data be used for transfer learning.

1.3 Project Objectives:

The desired outcome for this project would be to successfully train a convolution neural network model on real-time skeletal coordinates.

- Implement a vision-based approach such as Blazepose to extract skeletal coordinates from the gestures sequences images in the Jester 20bn dataset. Save the extracted coordinates to a pckl or JSON file; we want to feed this time sequence of gesture coordinates as the training input for the Gesture Recognition model and gain insights into how this model performs.

- Find out how a model trained on skeletal data performs when trained on this skeletal data instead of raw RGB data.

- Find out model accuracy, loss, confusion matrix, and classification report of the model.

- Train the model on different datasets to see how well the model generalizes.

1.4 Dissertation Overview:

This thesis aims to give a detailed description and documentation of Training a Human Gesture Recognition neural network model trained on skeletal coordinates of input gestures.

Chapter 2 provides an overview of related work in the Gesture Recognition field, and different implementations produced in recent years relevant to this project. Chapter 2 also explores the core background concepts and theory that will be used in this project.

Chapter 3 focuses on the design and implementation of this proposed model.

Chapter 4 gives the model's performance and critical findings on different experiments.

Chapter 5 discusses the results, limitations of the model, scalability, and future work that this project can expand upon.

CHAPTER 2

Literature Review:

2.1 Gesture Recognition

There are many different approaches for interpreting a gesture while processing the data for training a deep learning model. According to [5], there are two main approaches: 3D model-based and appearance-based. The 3D model-based uses 3D information of key landmarks in the body parts to obtain key parameters like the position of fingers, left hand, right hand, etc. The appearance-based systems make use of images and videos for direct interpretation. Following are the different approaches are used for Gesture Recognition.

1) 3D Model-Based Approach: This approach uses skeletal or volumetric models or sometimes even a combination of both. Volumetric Approach is widely used in computer vision and the models are created from 3D surfaces like polygon meshes.

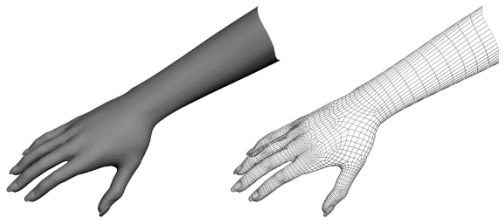

Figure 2.1: Real Hand Interpreted as 3D textured volumetric model from [5].

This approach is computationally intensive and still not very well developed for real time analysis which serves as its major drawback.

2) Skeletal Based Approach: This approach makes use of a simplified version of joint angle parameters with segment lengths instead of processing 3D models like the previous approach. This makes it less intensive on systems because there are comparably fewer parameters to deal with. This approach makes a virtual skeleton of the body where key landmark features are computed and mapped to different segments. Different positions and orientations are then analyzed to train a model.

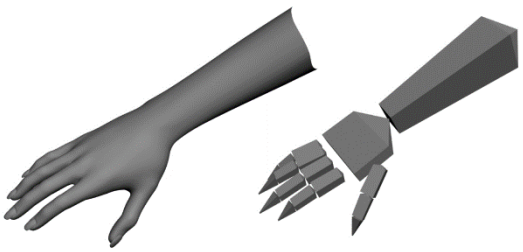

Figure 2.2: Real Hand Interpreted as 3D skeletal Model from [5]

This approach is faster than the rest because only the key landmark points are analyzed. We can extract the key points from different datasets and test them against

a common template for a gesture. This narrows down the detection program to just focus on the key landmarks of the body.

3) Appearance-based approach: This approach uses a template database to derive the parameters directly from images and videos instead of using the spatial model. This is based on deformable 2D templates of hands or poses.

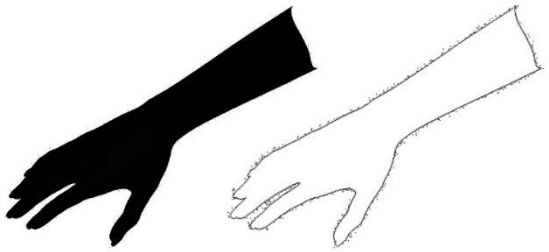

Figure 2.3: Real Hand Interpreted as silhouette and contour from [5]

These template-based models are used mainly for hand-tacking but can also be used for gesture recognition.

There are many different challenges and trade-offs for using different approaches. The limitations of using an image-based gesture recognition model are the equipment and image noise. A model trained on one camera may not work for another camera or the images with varying lighting conditions, and different subjects may give out incorrect results. To overcome these limitations, we try to implement the skeletal-based approach which extracts the key landmark coordinates of the body. It does not work on the raw RGB image which is subject to change. Therefore, this approach should perform better and on a broader dataset by working on the skeletal data and not RGB images.

2.2 Related Work

According to [7] Deep learning can be defined as a subclass of machine learning where algorithms f use a cascade of computational layers fi for feature extraction and learning. Human Gesture recognition is a complex system made up of Gesture Modelling, Gesture analysis and machine learning. CNN are widely used for image based learning as they do not require users to develop a complex features extraction algorithms for images but through the use of convolution and sub sampling layers, the invariant features are allowed with little dislocation[8]. [9] talks about mixing deep learning algorithms and Gesture Recognition using Convolution Neural Networks on RGB-D images. [8] directly trains the CNN on RGB images of sequences to classify.[6] investigates a CNN architecture for RGB-D images where the classifier is made of high resolution and low-resolution networks. However, robust classification of gestures under widely varying lighting conditions, and from different subjects is still a challenging problem. One of the solutions is to train the model on skeletal data instead of raw RGB images which is the motivation behind this project.

Recently, classification with deep convolution neural networks has been very successful in recognition research in various fields [9]. A multi-column CNN employs multiple parallel networks which improve the recognition rates of image classification by 30-80% [10]. [6] develops an effective method for dynamic hand gesture recognition with a 3D convolution neural network by using a classifier that fuses the motion volume of normalized depth and image gradient values. These research papers show great potential in using CNN for gesture recognition which inspired us to use the Convolution neural network model in the project.

SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset[11] presents and evaluates a 3D dynamic hand gesture dataset(SHREC 2017) which provides the sequence of hand skeletal data in addition to depth images. They feed the gesture coordinate sequences into a convolution neural network and obtain an accuracy of 88.24% on the 14 gesture classes. In [12], the author provides an alternate approach where the author color-coded the skeletal joints of a 3D skeleton model across time. The colored trajectories are projected on 2D planes to obtain images that serve as input of CNNs which then classifies gesture class probabilities.

Recurrent Neural Networks [13] or Long Short Term Memory(LSTM) have been widely used for achieving the state of the art results for gesture recognition where the data is in a time-like sequence. [13] combines the use of LSTM with the Spatio-temporal attention mechanism and achieves great results. Still, we chose CNNs instead of RNNs because, in recent studies, it was found that they perform better than RNNs. This is because RNNs present significant issues such as being sensitive to the first examples; they also sometimes cause chaotic behavior while training due to their complex dynamics[14].

Research on skeletal data done in [15] provides a state-of-the-art model with the highest performance on the DHG dataset. This research also uses a parallel model with 3 branches, a high-resolution branch, low-resolution branches, and a branch that serves as an identity function to reduce overfitting and achieves 91% accuracy for the 14 gesture classes. This research has achieved great results on the dataset with skeletal data and depth images, but it does not experiment with the data sensitivity of the model. The model is trained and tested on only one dataset. In our project, we take this parallel convolution model [15] and train/test it on different datasets to see how the model generalizes. The main aim of using the skeletal approach is that the model should not be sensitive to light and subject conditions. We aim to test how the model performs on different datasets and make conclusions based on that.

2.3 Neural Networks

Neural Networks aim to mimic how the human brain works with the help of a series of algorithms recognizing underlying relationships in data. It adapts to the diverse input given in Machine learning models and tries to generate the most accurate prediction without re-adjusting the design criteria of the output. Convolution Neural Networks are a type of Neural network which is currently state of the art in designing deep learning prediction models. Fig 2.4 shows the implementation of an Artificial Neural Network.

Every neural network has three layers: 1) Input Layer 2) Hidden Layer 3) Output layer

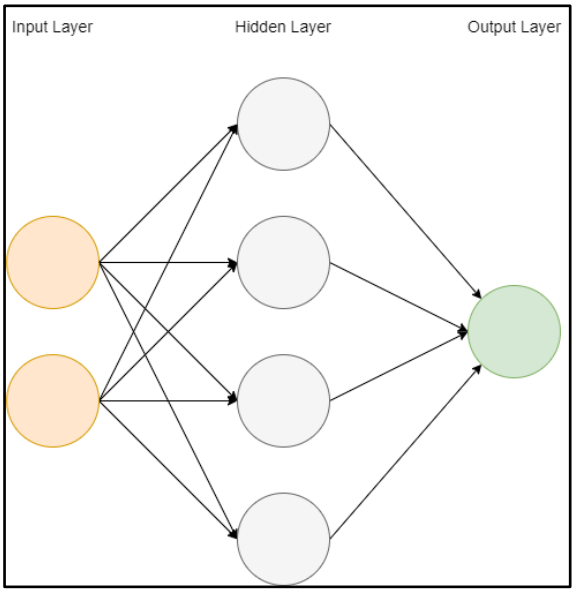

Figure 2.4: Basic implementation of an Artificial Neural Network

1) Input Layer: It is the initial input data that is fed into the network. The Yellow circles in Fig 2.4 represent the input layer. They are generally denoted as Vector X

2) Hidden Layer: It can contain multiple layers where all the computation is done. The Grey circle in Fig 2.4 represents the hidden layer. They are generally denoted as W or θ.

3) Output Layer: It produces the result for the given input data. The Green circle in Fig 2.4 represents the output layer.

Each Node is connected to every node from the previous layer and every connection has a certain weight attached to it. Weight is the impact that the previous node will have on the next node. These nodes have a predefined activation function that defines which node will be activated for different values. Unit Step function, Linear Function, and sigmoid function are all different types of activation functions. In neural networks, the Sigmoid function is the most widely used activation function because it is a nonlinear function that allows the node to take values ranging from (0, 1). The equation of the sigmoid function is given below:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^{x}}{e^{x}+1}.$$

Equation 2.1: Sigmoid Activation Function for Neural Network

When there are multiple possible outcomes, it gives us the probabilities of “activation” for each output class. The answer is the outcome with the highest probability value. Neural Networks have a node called bias or weights. Bias allows the activation function to shift left or right depending on what is required to make the better fit for the data. It is critical for a successful deep learning model. Neural network’s ability to distinguish subtle nonlinear interdependencies and patterns is the reason they are so widely used.

2.3.1 Convolution Neural Network

Convolution Neural Network is a multilayered neural network with special architecture to detect complex features in data. They are commonly used for analyzing visual images, self-driving vehicles, etc. CNN's use relatively less pre-processing compared to other algorithms. This is because CNNs learn to optimize the filters(kernels) through automated learning. CNNs are regularized version of multilayer perceptrons. Multilayer perceptron means fully connected networks where each neuron in one layer is connected to all the neurons in the next layer. This prevents overfitting of data. CNNs can be broadly divided into two parts, as shown in Fig 2.5: Feature Extraction and Classification. We go into the detail of each layer to better understand their use case.

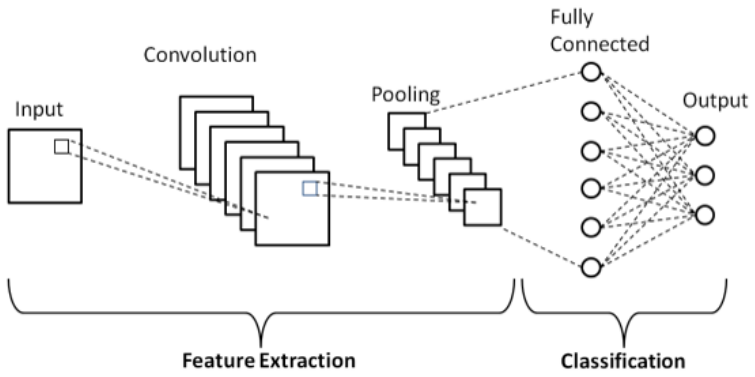

Figure 2.5: Basic implementation of a Convolution Neural Network[16]

1) Input Data: Images are made up of pixels. At a lower level, every pixel is a matrix with numerical values between 0 and 255. This is called the digital representation of the image, which is how computers interpret images, as shown in Fig 2.6.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Figure 2.6: Lower Level Representation of an image in form of a matrix

2) Convolution Layer: This layer consists of feature detectors or kernels, which are matrices usually of size 3x3, as shown in Fig 2.7. The matrix representation of the image is multiplied elementwise with the feature detector to make a feature map, as shown in Fig 2.8. This reduces the size of the image so that it is faster to process computationally. The kernel is chosen to retain the unique features in the image.

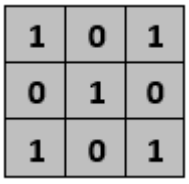

Figure 2.7: Sample Convolution Kernel

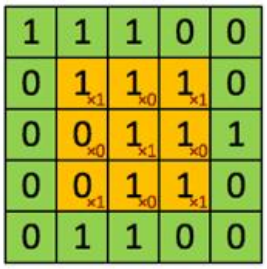

Figure 2.8: Convolution Operation on the image matrix in Fig 2.6

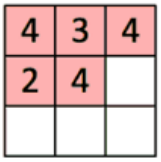

Figure 2.9: Feature Map Obtained

3) Activation Function: This layer is added to help add nonlinearity to the network to learn complex patterns in data. It decides what is part of features must be activated or "fired up" in the next layer. We use ReLU(rectified linear activation function) function in this project because it is easier to compute and does not cause the vanishing gradient problem. This allows models to learn faster and perform better. The graph of the ReLU function is shown in Fig 2.10

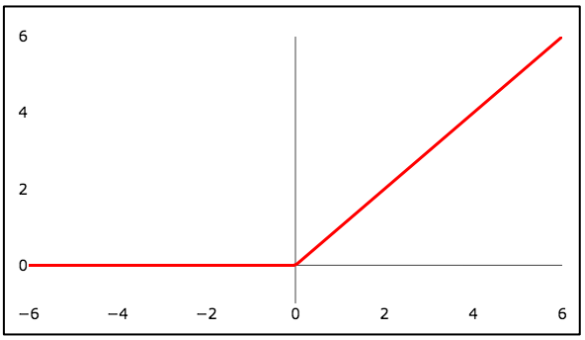

Figure 2.10: ReLU activation Function

Source: TowardsDataScience: Activation Functions in Neural Networks

4) Pooling Layer: This layer reduces the dimensions of data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. This allows the network to detect features in various images irrespective of the difference in lighting or angles in the pictures. The most used pooling techniques are Max Pooling and average Pooling. Fig 2.11 shows max-pooling, which works by placing a 2x2 matrix on the feature map and picking up the largest value.

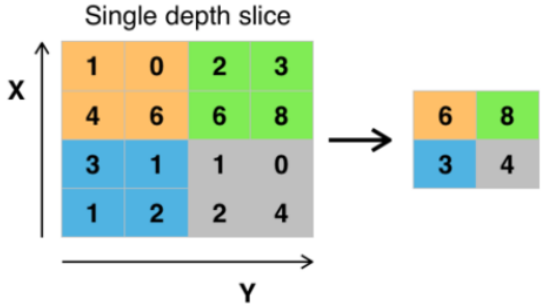

Figure 2.11: Maximum Pooling operation done on a feature map

5) Fully Connected Layer: This is the final layer in a CNN that is connected to every neuron. This layer classifies the data into different output classes using the softmax activation function and gives us the prediction of the output classes as a number between 0 and 1.

We use a parallel convolution model in this project, a modified version of CNN where different branches are working in parallel to reduce the computation time. We go into the detail of the model in 3.4.2.

2.4 Pose Estimation

Pose estimation is an important area of research in Computer Vision. Pose estimation plays a critical role in projecting overlay of digital content over the physical world in Augmented Reality, Quantifying physical exercises, sign language recognition. It forms the basis for fitness, yoga, weightlifting applications. The standard approach for human pose estimation is to produce heatmaps for different joints along with refining offsets for each coordinate. A lot of progress has been made in the pose estimation field[17]. While this approach shows promising results, it makes the model for a single person considerably larger than what is suitable for mobile phones. Currently, the standard for pose estimation is COCO topology[18]. It makes 17 landmark points for the body, localizing ankle and wrist points. It lacks scale and orientation information for hands which is pivotal for sign language, gesture recognition, or any fitness tracking. BlazePose introduces a different approach which significantly speeds up this model with minimal to no quality degradation[19]. Hence we selected Blazepose for this project to experiment with this new approach and see how accurately it predicts the landmarks of the body.

2.5 Technologies Used

2.5.1 Google Colab

Google Collaboratory is free to use the product from Google Research. It is well suited to train deep learning models and requires no setup. It can be used to write and execute arbitrary python code through the browser and gives free access to computing resources, including GPUs such as Nvidia K80s, T4s, P4s, and P100s. Colab is based on Jupyter, which allows it to use and share Jupyter notebooks without having to download or install anything. The code is executed in a virtual machine that is assigned to the user at the runtime and is private to the user's account. This is the only downside of using Google Colab that sometimes the resources assigned to a user are not guaranteed or unlimited. It fluctuates according to the demand and free resources at the time. Colab maintains the flexibility to adjust usage limits to accommodate fluctuations in demand to accommodate every user using the service.

Since this project requires working with feature extraction and training a deep learning model, using Google Colab was an effective choice so that the model utilizes the free-to-use hardware and not the PC's resources.

2.5.2 Keras and TensorFlow

TensorFlow is a free and open-source software library for machine learning, data flow, and differential programming developed by Google. Its particular focus is on training and inference of deep neural networks. Keras is also an open-source neural network and machine learning library written in Python. It is integrated inside of TensorFlow and focuses on implementing deep neural networks. It is a very user-friendly and modular library.

This project uses both Keras and TensorFlow for training a CNN model for performing Gesture recognition using skeletal data.

2.5.3 MatPlotLib

MatPlotLib is an extension of NumPy used for creating static, animated, and interactive visualizations in Python. It provides a MATLAB-like interface with the ability to code in Python. This is useful for the project to plot Validation and Testing accuracy and loss for the Gesture Recognition model.

2.5.4 BlazePose

Blazepose was presented by Google at the CV4ARVR workshop at CVPR 2020 as a lightweight real-time solution for pose estimation on mobile devices. Blazepose provides a new approach for pose detection by inferring 33 2D landmarks on a body in a single frame of a video that achieves real-time performance on mobile phones compared to state-of-the-art approaches which reply on intensive CPU/GPU hardware as shown in Fig 2.12. Blazepose enables us to find these 33 key landmark points in terms of coordinates which we will use to run ML models for face or hand-tracking.

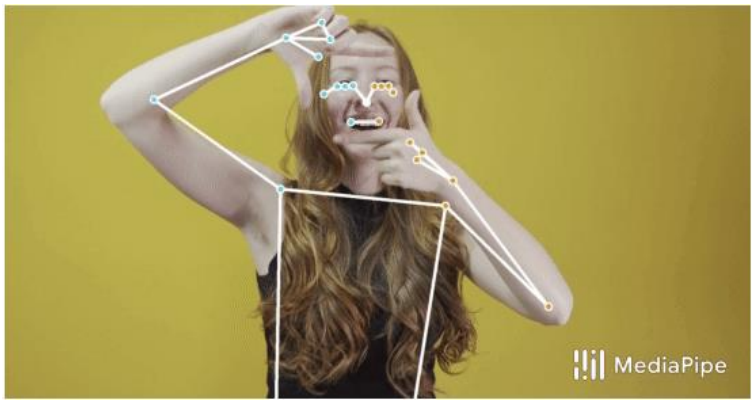

Figure 2.12: BlazePose skeletal on a sample image[19]

Blazepose uses a two-step detector-tracker ML pipeline[19], as shown in Fig 2.13. It first locates the pose region of interest and then estimates the 33 landmarks from this ROI. Each of these 33 key points has 3 degrees of freedom(x, ylocation, and visibility). The critical thing to notice here is that the detector is only run once, i.e. for the first frame. For the rest of the frames, it is derived from the previous frame's pose key points as shown in Fig 2.13. Instead of using heatmap prediction, BlazePose uses regression supervised by heatmap/ offset prediction of all key points.

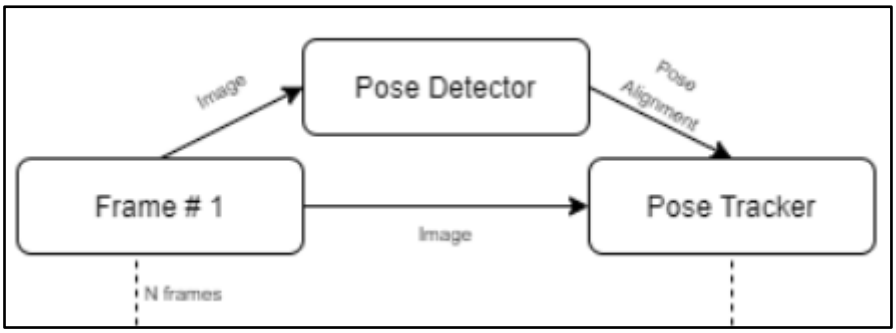

Figure 2.13: BlazePose 2 step detector ML Pipeline[19]

The 33 key landmark points are depicted in Fig 2.14 below.

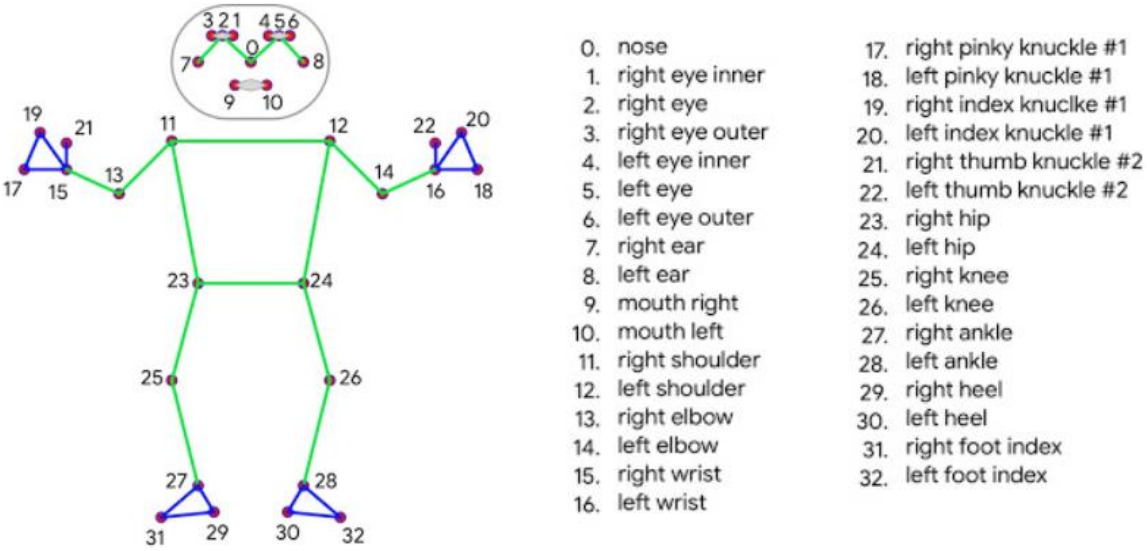

Figure 2.14: Key landmark points detected by BlazePose[19]

Since this project focuses on the Human/Hand Gestures where the lower body is either not useful or not visible in the datasets, BlazePose gives an option to select the 25 upper body landmarks from Nose to left Hip. This project will be using this UPPER_BODY_ONLY function, which only connects the first 25 points when set to True.

2.6 Datasets Used

2.6.1 Jester 20BN V1

This is a large collection of labeled video clips randomly recorded in front of a laptop camera or a webcam showing humans performing pre-defined hand gestures[20]. This dataset allows training robust machine learning models and is available free of charge for academic research, making it a viable option for this project's approach. There are a total of 148092 videos corresponding to 27 defined gestures shown in Fig 2.16. Each gesture video has a folder containing jpg images with a height of 100px and variable width. The jpg images were extracted from the videos at 12 frames per second, and the number of JPGs varies with different gestures.

Figure 2.15: Sample Gesture images from Jester Dataset [20]

Classes	
Doing other things	12,416
Thumb Down	5,460
Thumb Up	5,457
Drumming Fingers	5,444
Pushing Hand Away	5,434
Stop Sign	5,413
Sliding Two Fingers Down	5,410
Pulling Hand In	5,379
Zooming Out With Two Fingers	5,379
Pushing Two Fingers Away	5,358
Zooming In With Two Fingers	5,355
Sliding Two Fingers Left	5,345
No gesture	5,344
Zooming Out With Full Hand	5,330
Pulling Two Fingers In	5,315
Shaking Hand	5,314
Zooming In With Full Hand	5,307
Swiping Down	5,303
Sliding Two Fingers Up	5,262
Sliding Two Fingers Right	5,244
Swiping Up	5,240
Rolling Hand Forward	5,165
Swiping Left	5,160
Swiping Right	5,066

Figure 2.16: Gesture Classes in Jester Dataset

This is the largest dataset of video clips showing human gestures. It is ideal for this project because it provides a space for cross-testing and training to see how the model generalizes as well.

2.6.2 SHREC 2017

This is a 3D dynamic hand Gesture dataset that provides sequences of hand skeletal data and depth images[21]. This dataset was created to encourage researchers to challenge their recognition algorithms for hand gestures using depth images or skeletal data points. This dataset directly contains the hand gestures in the form of skeletal sequences instead of regular RGB images. This dataset contains 14 hand gestures; each gesture is performed between 1 and 10 times by 28 participants. There are coordinates of 22 joints of the hand, as shown in Fig 2.17, that forms a full hand skeletal. The dataset is collected on Intel RealSense short-range depth camera, as shown in Fig 2.17. The depth images were captured at 30 frames per second with a resolution of 640x480.

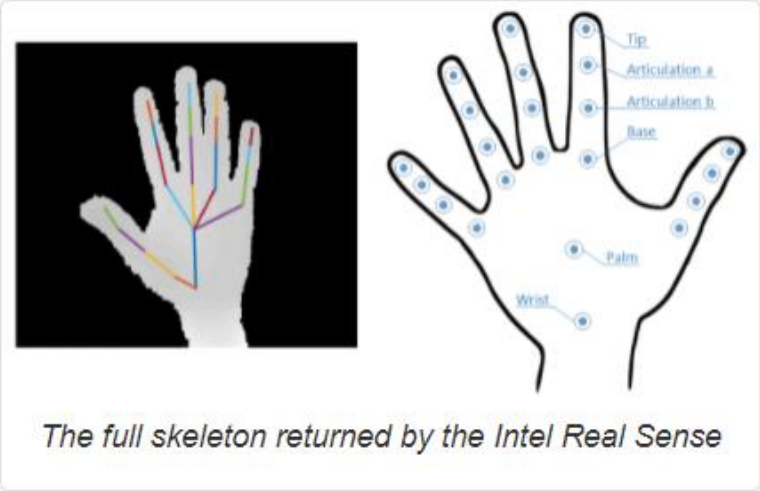

Figure 2.17: Depth Images and Skeletal Points in SHREC Dataset[21]

Gesture	Label	Tag name
Grab	Fine	G
Expand	Fine	E
Pinch	Fine	P
Rotation CW	Fine	R-CW
Rotation CCW	Fine	R-CCW
Tap	Coarse	T
Swipe Right	Coarse	S-R
Swipe Left	Coarse	S-L
Swipe Up	Coarse	S-U
Swipe Down	Coarse	S-D
Swipe X	Coarse	S-X
Swipe V	Coarse	S-V
Swipe +	Coarse	S-+
Shake	Coarse	Sh

Figure 2.18: Gesture Classes in SHREC Dataset[21]

This dataset is ideal for an alternate approach if the initial approach of extracting coordinate points using Blazepose gives us inconsistent results. This dataset serves the purpose of giving the skeletal points of gestures directly in the dataset, which can be extracted quickly and can be used to train the model.

2.6.1 DHG 14/28

This is an almost identical dataset to SHREC 2017 with the difference in the number of participants and the times a video was recorded. Each Gesture in this Dataset is performed five times by 20 participants. Data is recorded and stored in the same way as the SHREC dataset. This also contains 14 Gesture classes stated in the SHREC dataset. This Dataset will be used to cross-train/test the model to evaluate the data sensitivity of the model.

CHAPTER 3

Design and Implementation:

This chapter discusses different approaches taken to achieve our target of training a gesture recognition model on skeletal coordinates. Section 3.1 provides an overview of the whole project design, while subsequent sections provide in-depth descriptions of the steps taken during the development of each stage of the design.

3.1 Initial Project Design

This project contains two major software components- Data Processing and Training the model. We use two separate Google Colab notebooks to implement the components to make the code easy to implement and debug. Fig 3.1 shows the detailed overview of how the initially planned approach for extracting the skeletal coordinates and then training the model looks like.

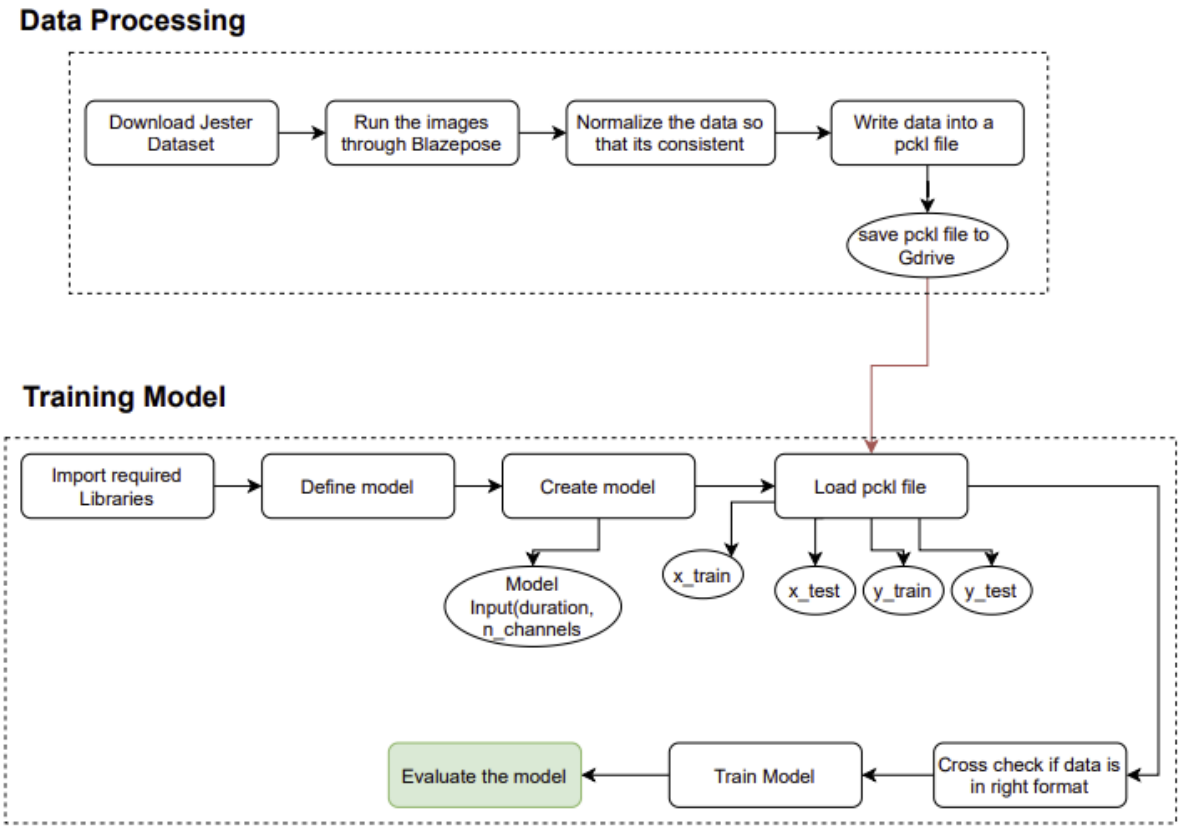

Figure 3.1: Overview of Initial Project Design

3.1.1 Data Pre-Processing

This is the process of preparing raw data and make it suitable for feeding it as an input to the deep learning model. Real-world data contains noise, inconsistencies, etc. which cannot be directly used for deep learning because the models are very sensitive towards this. The main problem with extracting coordinates from images is the different scales each gesture is recorded in. For example, if a person is sitting close to the camera and the other is sitting far away from the camera, both are recording the same gesture but on a different scale. The issue here is that the model may give more attention to one feature based on its value. Therefore, we need to make this data consistent by normalization before training the model on this input data.

3.1.2 Normalization

Normalization is a scaling technique that shifts and rescales the data converts it into a value between 0 and 1. This is also called Min-Max scaling; the equation is given below:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Equation 3.1: Min-Max Normalization Equation

X_{min} and X_{max} are the minimum and maximum values of the features, respectively. The minimum value of the feature is transformed to 0, the maximum value to 1, and every other value is transformed into a decimal between 0 and 1 with the help of the equation given above. We plan to use this normalization after extracting the skeletal data from the Jester dataset images to make it consistent.

3.2 Extracting Skeletal Coordinates

BlazePose offers two frameworks: the full-body framework and the upper body framework. As discussed in section 2.3.4, we will be using the upper body framework to extract the upper body landmark features. We were able to implement the live version of BlazePose, and as shown in Fig 3.2, BlazePose is running successfully on the upper body. We then extracted the raw coordinate points for the first 25 key points from the raw RGB image, as shown in Fig 3.2 and Fig 3.3. This was proof of the concept that BlazePose was successfully running on the system and giving out correct coordinates. The next step is to download and extract the Jester 20bn dataset and run the Gesture images through the same framework and check if we are getting consistent results.

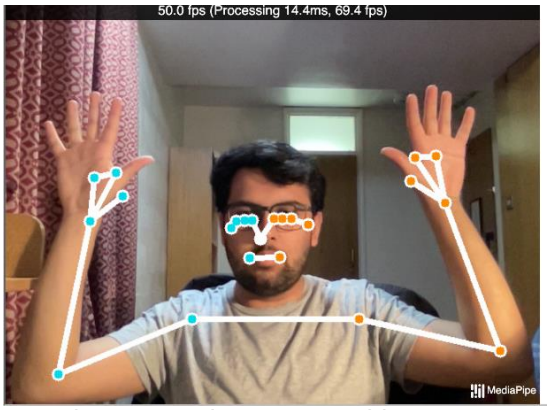

Figure 3.2: BlazePose Architecture on Author's image

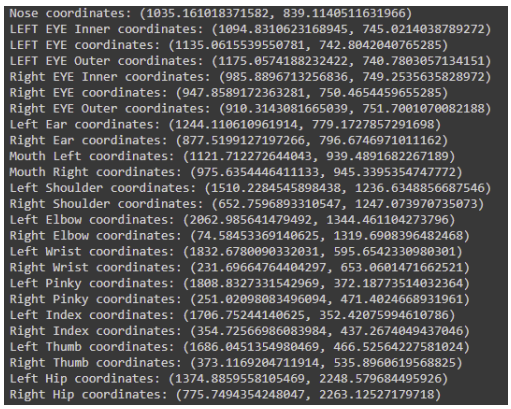

Figure 3.3: Coordinates of 25 key landmark points from Fig 3.2

3.3 Implementation Issues

After successfully implementing BlazePose and extracting the coordinates from the RGB images in the previous section, we tried to recreate the results on Jester Dataset images. But the issue we ran into was that the Jester Dataset is recorded very randomly in front of a webcam or laptop. It has not been recorded, especially for extracting key point coordinates from the images. This was leading to major inconsistencies in the result. An example of such inconsistency is shown below in Fig 3.4 and Fig 3.5:

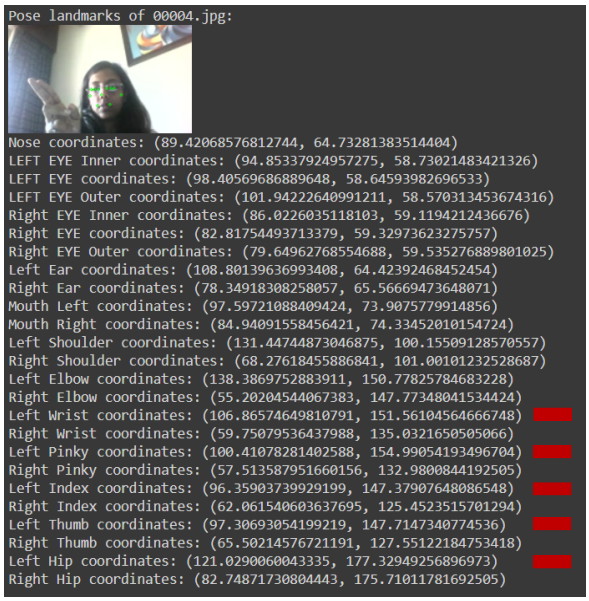

Figure 3.4: BlazePose Implement Issue 1: Picking up coordinates of Points that aren't in the image

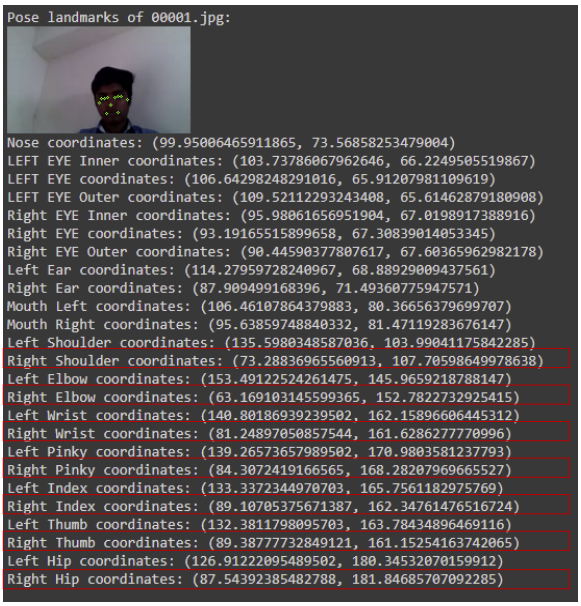

Figure 3.5: BlazePose Implement Issue 2: Picking up coordinates of Points that aren't in the image

As we can see from the implementation, BlazePose is picking up coordinates that are not there. In Fig 3.4, we can see that there is no left hand; still, BlazePose is picking up the coordinates for the same. In fig 3.5, we can see that there are no hands in the image still BlazePose is picking up coordinates for the same. These are just two examples; we noticed that more than 60% of videos in BlazePose are recorded in the same way where some key features are missing from the images. Training the model on these wrong coordinates would lead to inconsistent results in the latter part of the project, so we decided to change our approach for the project altogether because training the model on wrong coordinates will give us no significant results.

3.4 Alternate Design

The inconsistency in the Jester dataset leads us to find an alternate approach for the project. We were able to find the datasets, SHREC 2017 and DHG 14/18, as described in section 2.6, which already contains the data in the form of skeletal coordinate points and images. Given the limited time left for the project, we decided to proceed with this dataset where our motive for the project stays the same, but instead of trying to extract the coordinates in real-time, we work with the datasets which already have the skeletal data. The scalability of this approach is discussed in section 5.2. Our approach stays the same; we will have two Colab notebooks, one for downloading the SHREC dataset and processing the input data and the second one to create a CNN model which will take this skeletal data as input and train the gesture recognition model. The alternate design for the project is given in Fig 3.6 below:

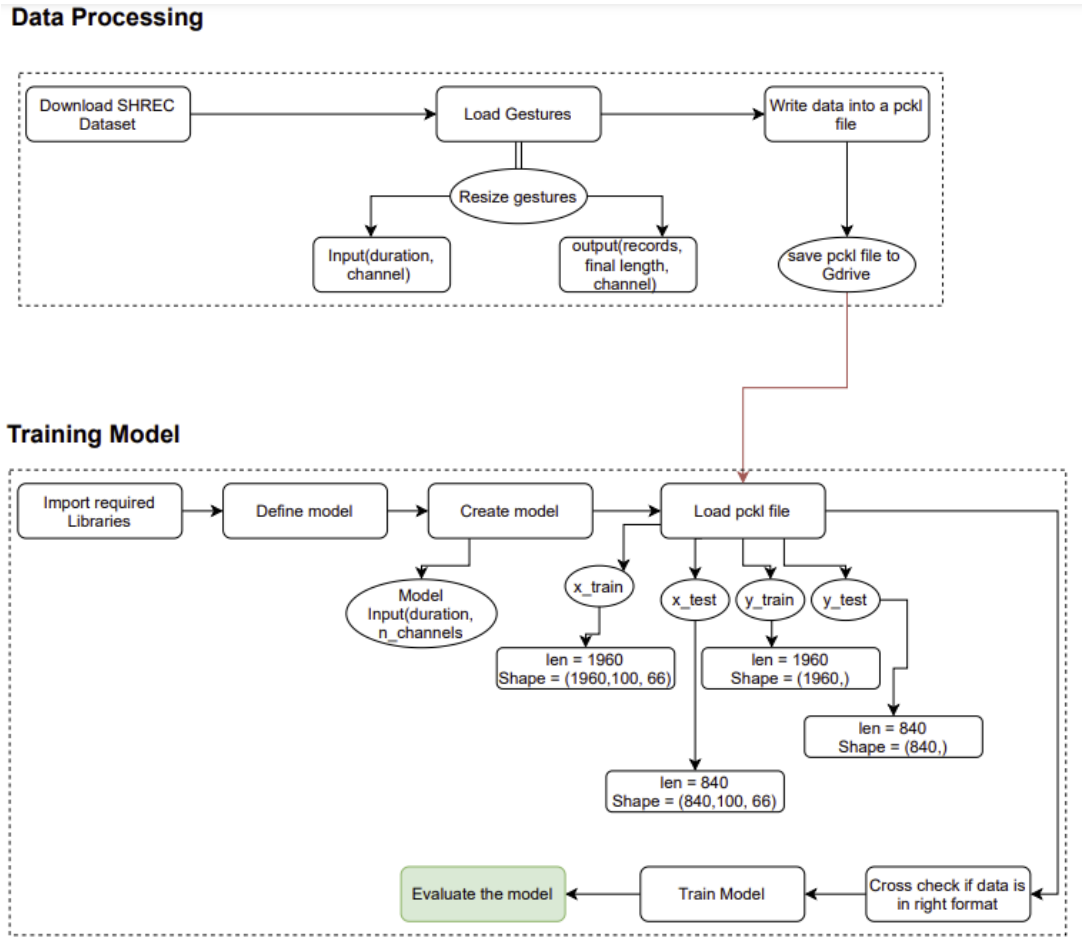

Figure 3.6: Overview of Alternate Project Design

3.4.1 Data Preparation

Since the data in the SHREC and DHG datasets are already in the form of skeletal coordinates, we only have to process and define it in a consistent form so we can feed it as the input for the model. Fig 3.7 shows the hand skeleton of the dataset returned by the Intel RealSense camera containing 22 joints in the Hand. Each frame of the sequence contains a depth image and the coordinates of 22 joints both in 2D depth image and 3D world space, forming a full hand skeleton as shown in Fig 3.8. The dataset is divided into two gesture classes, 14 and 28; for this project, we focus on only 14 gesture classes given in Fig 2.18.

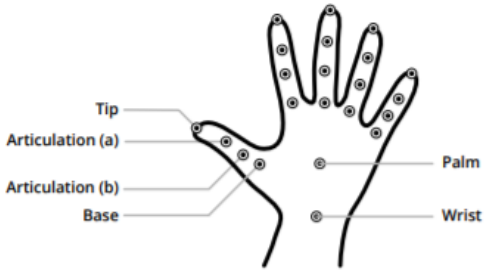

Figure 3.7: 22 Key points in Hand Skeletal Model [21]

Figure 3.8: Depth Image of Hand in SHREC Dataset [21]

We define a 3D skeletal data sequence *s* as a vector whose components p_i are multivariate time sequences where each component $p_i = (p_i(t))t \in N$ represents a multivariate sequence with three univariate sequence components.

$$s = (p_1 \cdots p_n)^T \qquad p_i = (x^{(i)}, y^{(i)}, z^{(i)})$$

These altogether represent a time sequence of the positions $p_i(t)$ of the i^{th} skeletal joint j_i. There are a total of 22 skeletal joints, which represents a distinct part of the Hand in the skeletal model given in Fig 3.7. The data is already in normalized form. At each time step t, the 3D hand skeleton consists of an ordered list of 22 joints with their positions $p_i = (x_i, y_i, z_i) \in R^3$, $\forall$ i $\in$ 1; 22 in the 3D space. These 22 joints have three channels each, so the total channels come out to be 66, as shown in Fig 3.9.

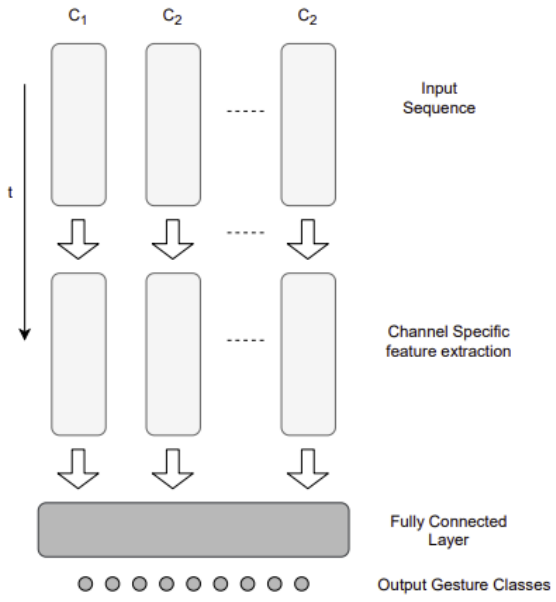

Figure 3.9: Overview of form of input data fed into the model

As shown in the code snippet in Fig 3.10, We divide the SHREC dataset into test train split where x_train and x_test contain the vector s, i.e. the multivariate time sequences and y_train, and y_test consist of their respective gesture class. We perform a test train split of 30/70 of the total 2800 gesture sequences, which comes out to be 1960 train sequences and 840 test sequences. The Gestures can have variable durations; hence we resize the time series by interpolating them to the same length. The output tensor should be of the shape:

(len(input_gesture_sequence), duration, channels)

```
gestures, labels_14, labels_28 = load_gestures(dataset='shrec',
                                               root='/tmp/dataset_shrec2017/',
                                               version_x='3D',
                                               version_y='both',
                                               resize_gesture_to_length=100)

# Test Train Split for Dataset 70/30
x_train, x_test, y_train_14,  y_test_14, y_train_28, y_test_28 = train_test_split(gestures, labels_14, labels_28, test_size=0.30)

# Save the dataset
data = {
    'x_train': x_train,
    'x_test': x_test,
    'y_train_14': y_train_14,
    'y_train_28': y_train_28,
    'y_test_14': y_test_14,
    'y_test_28': y_test_28
}
write_data(data, filepath='shrec_data.pckl')
```

Figure 3.10: Code Snippet showing splitting the processed dataset and saving it into pckl file

After defining the training and test parameters, we print the shape of NumPy.ndarray tensor and gesture classes list to cross-check if the parameters are loaded from the dataset correctly. As we can see from Fig 3.12, the parameters are loaded correctly. The shape of NumPy.ndarray tensor x_train is coming out to be (1960, 100, 66), which corresponds to 1960 multivariate gesture sequences 100-time duration and 66 channels, precisely what we wanted. This approach for data processing is inspired by [15], where the same process has been performed for the DHG dataset.

```
#-----------------------------------
# Printing parameters to cross check
#-----------------------------------
print(x_train.shape)
print(x_test.shape)
print(len(y_train_14))
print(len(y_test_14))
```

Figure 3.11: Code Snippet to check the form of data

```
(1960, 100, 66)
(840, 100, 66)
1960
840
```

Figure 3.12: Output of the code snipped in Fig 3.11

Now we have successfully processed our data and saved the parameters into a pckl file. We now define the model and feed these parameters as the input for training the model.

3.4.2 Parallel Convolution Model

The parallel convolution model used in this project is inspired by [15], which focuses on the DHG dataset. We take the model and train it on the SHREC dataset without changing any parameters to see how it performs on a different dataset. In this section, we go into detail about this parallel convolution model used for gesture recognition.

This is a multi-channel convolution neural network with two feature extraction branches and a residual branch per channel. The model architecture is shown in Fig 3.13. The architecture is inspired by the high and low-resolution network [6]. The use of residual branches makes it easier to optimize the model because of better gradient backpropagation in the training phase. It ultimately increases the accuracy of the deep learning model. This approach is inspired by [22], which talks about using residual branches in image recognition in depth.

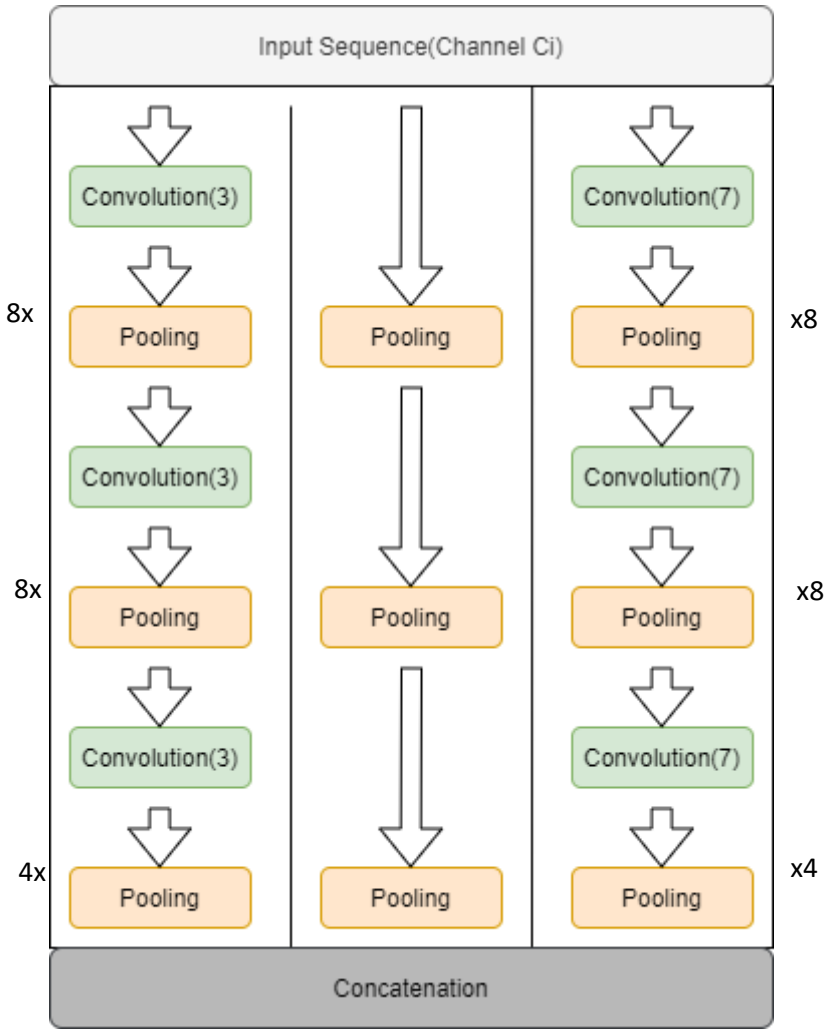

Figure 3.13: Detailed structure of Parallel Convolution Model from [15]

The input for this model consists of fixed length, 1D sequences of vectors (s_1, s_2,…., s_c) where c is the number of channels. These sequences are directly used as an input to all the three parallel branches in the model.

- Low Res Branch: This branch consists of 3 convolution and pooling layers. In this branch, we use three kernels for each convolution layer. The difference between the three convolution layers resides in the number of feature maps used. For the first two layers, we use eight feature maps, whereas, for the last layer, we use four feature maps.

- Residual Branch: This branch serves as an identity function. We do not put any convolution filters in this branch. We perform a pooling operation on the input so that when this branch is finally concatenated with the outputs of other branches, it does not create any risk of overfitting.

- High Res Branch: This branch is almost identical to the low-res branch. It consists of 3 convolution and pooling layers. But in this branch, we use seven kernels for each convolution layer instead of 3. The difference between the three convolution layers resides in the number of feature maps used. For the first two layers, we use eight feature maps, whereas, for the last layer, we use four feature maps

All the neurons in the model use ReLU activation function: σ(x) = ReLU(x) = max(0,x) other than the output neuron which uses SoftMax activation function. All the pooling layers use average pooling with a pool size of 2. We implement this model in Google Colab using Keras as a high-level library above TensorFlow. Fig 3.14 shows the actual code snippet from Google Colab used for creating the model.

```
def create_model(n_classes, duration, n_channels, dropout_probability=0.2):
    # Define model, using functional API
    model_input = Input(shape=(duration, n_channels))

    # slice into channels
    channel_inputs = Lambda(lambda x: tensorflow.split(x, num_or_size_splits=n_channels, axis=-1))(model_input)

    features = []
    for channel in range(n_channels):
        channel_input = channel_inputs[channel]
        # high branch
        high = Conv1D(filters=8, kernel_size=7, padding='same', activation='relu', input_shape=(100, 1))(channel_input)
        high = AveragePooling1D(pool_size=2)(high)
        high = Conv1D(filters=4, kernel_size=7, padding='same', activation='relu')(high)
        high = AveragePooling1D(pool_size=2)(high)
        high = Conv1D(filters=4, kernel_size=7, padding='same', activation='relu')(high)
        high = Dropout(dropout_probability)(high)
        high = AveragePooling1D(pool_size=2)(high)
        # low branch
        low = Conv1D(filters=8, kernel_size=3, padding='same', activation='relu', input_shape=(100, 1))(channel_input)
        low = AveragePooling1D(pool_size=2)(low)
        low = Conv1D(filters=4, kernel_size=3, padding='same', activation='relu')(low)
        low = AveragePooling1D(pool_size=2)(low)
        low = Conv1D(filters=4, kernel_size=3, padding='same', activation='relu')(low)
        low = Dropout(dropout_probability)(low)
        low = AveragePooling1D(pool_size=2)(low)
        # pooling branch
        ap_residual = AveragePooling1D(pool_size=2, input_shape=(100, 1))(channel_input)
        ap_residual = AveragePooling1D(pool_size=2)(ap_residual)
        ap_residual = AveragePooling1D(pool_size=2)(ap_residual)
        # channel output
        channel_output = concatenate([high, low, ap_residual])
        features.append(channel_output)

    features = concatenate(features)
    features = Flatten()(features)
    features = Dense(units=1936, activation='relu')(features)

    model_output = Dense(units=n_classes, activation='softmax')(features)

    model = Model(inputs=[model_input], outputs=[model_output])
    return model
```

Figure 3.14: Code Snippet showing the code for creating the model from [4]

3.5 Training

In this section, we discuss in detail the hyperparameters defined for training the model. We also discuss different possibilities of cross-training and testing between DHG and SHREC datasets to check the data sensitivity of the model.

1. Input Data: After processing the data as discussed in Section 3.4.1, we have unidimensional sequences of skeletal points where each batch contains a set of 32 skeletal gesture sequences of length 100. The datapoints outside the boundaries of the input are filled with a reflection using linear interpolation of time series. After this, every sequence is of length 100-time steps.

2. Weights: We set initial weights for the model using Xavier initialization[23]. This approach is also called Glorot uniform initializer, which draws a sample from a uniform distribution within a (-limit, limit) where the limit is sqrt(6 / (n_{in} + n_{out})). n_{in} or number in is the number of input units in weight tensor, and n_{out} or number out is the number of outputs in weight tensor. This method works better than uniform random initialization and eliminates the need to guess good values for training the model.

3. Hardware: We trained the model using Google Colab, which allows the GPU which are available at the time of connecting to Colab's Virtual Machine. In most cases, while training and testing this model, we were assigned either Nvidia T4 or K4.

4. Loss Function: We use negative log-likelihood as the cost function for this model. This is used to calculate the model error and minimizes it. We chose the negative log-likelihood function because it works well with the SoftMax activation function used in output neurons.

5. Optimizer: We use the Adam optimization algorithm[24] for this model. We chose Adam optimizer for stochastic gradient descent for training deep learning models. It is relatively easier to configure and handles sparse gradients on noisy problems. It calculates an exponential moving average of the gradient and squared gradient. We set the decay rates β1 = 0.9, β2 = 0.98, α = 10−3 for the learning rate, and epsilon ε = 10−8 to prevent any division by zero in the implementation.

6. Regularization: We use the dropout as a regularizer to reduce overfitting in the model[25]. The key idea is to randomly drop units from the neural network during training to prevent units from co-adapting too much. This significantly reduces the overfitting in neural networks. We set the dropout rate to be 0.2 after expedients conducted in [15] show that higher dropout rates reduce the training accuracy but do not affect the validation accuracy.

We first try to recreate the results in [15] on the DHG dataset. We want to use the same Hyperparameters and model on SHREC 2017 datasets and evaluate the results obtained. We are also interested in finding out the data sensitivity of the model, i.e. we will run multiple combinations to see how a change in test and train dataset affects the model's accuracy. We will train the model on the SHREC dataset and test it on the DHG dataset and vice-versa. We will also try to train the model on both datasets and then test it on either of the one datasets to see how it performs. We go into these details in the next chapter.

Chapter 4

Results and Evaluations

In this section, we will be covering the results obtained while testing the model on different datasets. We will be coving model accuracy, loss, confusion matrix, and classification report for both the datasets. Section 4.1 will cover the basics of evaluation parameters used to evaluate a deep learning model and subsequent sections covering the results obtained.

4.1 Evaluation Parameters

In this section, we briefly discuss the parameters we use for evaluating a deep learning model.

1) Accuracy: This is the fraction of predictions the model predicts correctly divided by the total number of predictions.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Equation 4.1: Accuracy for a model

In the case of binary classification accuracy can be calculated in terms of positives and negatives as well.

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

Equation 4.2: Accuracy for a binary classification model

Where TP= True Positive, TN = True Negative, FP= False Positive, FN = False Negative

2) Loss: This is defined as a penalty for bad prediction. Loss is a number indicating the uncertainty of a prediction based on how much the prediction varies from the true value. The more the model's prediction is from the actual result, the greater is the loss.

3) Confusion Matrix: This summarizes the performance of a classification algorithm. We use this for this project because classification accuracy alone is misleading when there are multiple classes and an unequal number of observations in each class in the dataset.

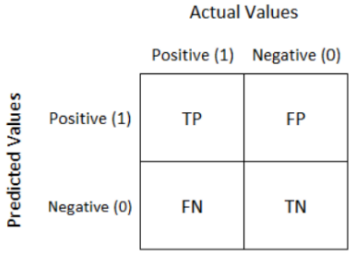

Figure 4.1: Confusion Matrix for a Binary Classification model

The number of correct/incorrect predictions are summarized and broken down by each class which shows where the model is confused while making predictions.

4) Classification Report: This reports the key metrics in a classification problem. It consists of Precision, Recall, and F1-Score:

- Precision: This is defined as the ratio between True Positives and Total Positives(TP+FP). This calculates the proportion of positive identifications that were actually correct.

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

Equation 4.3: Precision of a model

- Recall: This is the measure of the model's ability to correctly identify True Positive. This calculates the proportion of actual positives identified correctly.

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

Equation 4.4: Recall of a model

- F1-Score: This is the weighted average of precision and recall. It reaches its best value at 1 and worst value at 0. The relative contribution of precision and recall to the F1-score is equal.

5) Overfitting: This happens when a deep learning model learns the details and noise in the training data very closely which negatively affects the performance of the model. This results in poor test accuracy and poor generalization of the model as it picks up noise and random fluctuations in training data and learns them as features. These features may not apply to the new data that is being tested which then results in poor performance of the model. We can see overfitting trends in Fig 4.2 and Fig 4.3.

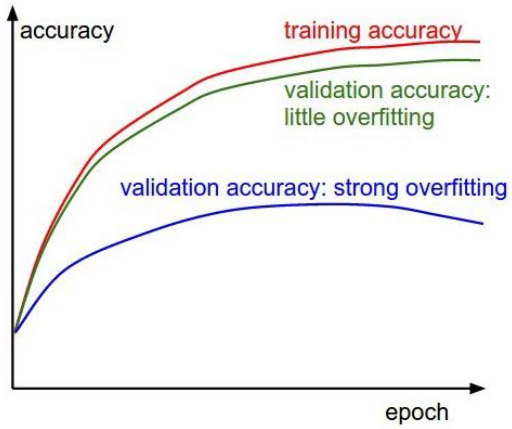

Figure 4.2: Different overfitting curves observed in a Accuracy Graph for a deep (Source: CS231n Convolutional Neural Networks for Visual Recognition)

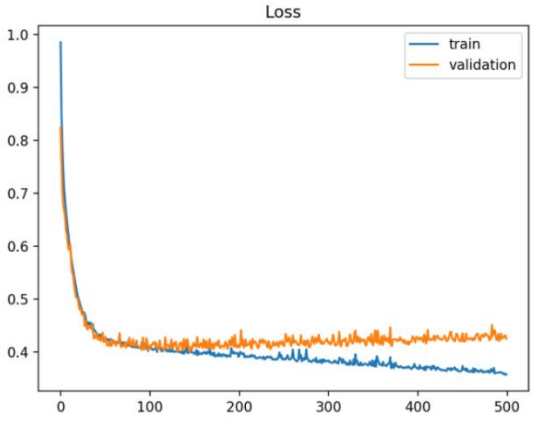

Figure 4.3: Overfitting curves observed in a Loss Graph for a deep learning model.(Source: Learning Curves to Diagnose Machine Learning Model Performance)

6) Underfitting: This happens when a deep learning model can neither learn the features in the training data nor generalize to the new data. This means that the algorithm cannot capture the feathers in the training data itself. This results in poor test and train accuracies. This is harder to detect. We need to look at the learning curve of training loss to identify underfitting. Underfitting is shown in Fig 4.4

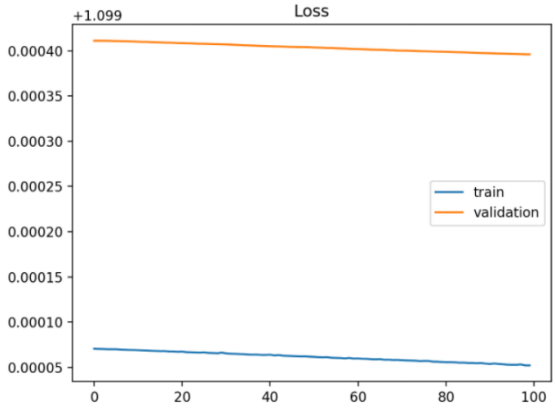

Figure 4.4: Underfitting curves observed in a Loss Graph for a deep learning model.(Source: Learning Curves to Diagnose Machine Learning Model Performance)

We use the hyperparameters explained in section 3.5 to train and test the model. The results obtained are on 14 gesture classes. Each gesture falls into one of 14 categories :

Grab (G), Tap (T), Expand (E), Pinch (P), Rotation clockwise (RC), Rotation counter-clockwise (RCC), Swipe right (SR), Swipe left (SL), Swipe up (SU), Swipe down (SD), Swipe x (SX), Swipe + (S+), Swipe v (SV), Shake (Sh).

We opted to keep all the hyperparameters the same throughout the testing on different datasets to ensure that the comparison between different combinations is consistent. The subsequent sections show the results obtained during training and testing on single and mixed datasets and we discuss these results in the next Chapter.

4.2 SHREC 2017

In this section, we primarily focus on SHREC 2017 dataset presented in section 2.3. The model achieves a classification accuracy of 91.79% on 14 Gesture classes. The classification report consisting of precision, recall, and F1-Score is given in Table 4.1.

Gesture Class	Precision	Recall	F1-Score
G	0.88	0.89	0.89
T	0.75	0.87	0.81
E	0.96	0.86	0.91
P	0.77	0.88	0.82
RC	0.89	0.8	0.84
RCC	0.97	0.91	0.94
SR	0.93	0.95	0.94
SL	0.95	0.95	0.95
SU	0.94	0.94	0.94
SD	0.92	0.91	0.91
SX	0.97	0.97	0.97
S+	0.98	0.96	0.97
SV	0.97	0.92	0.94
Sh	0.91	0.94	0.92
Accuracy	91.79%		

Table 4.1: Classification report for Model trained and tested on SHREC Dataset

This model's accuracy and loss graphs are shown in Fig 4.5 and Fig 4.6:

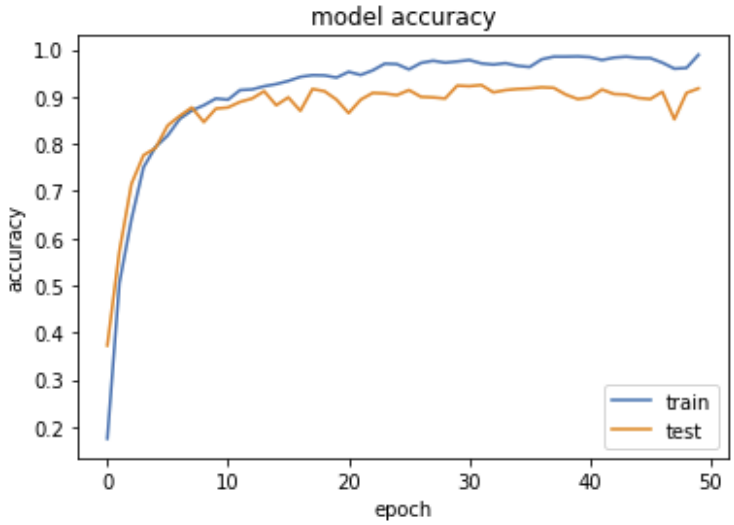

Figure 4.5: Accuracy curve for Model trained and tested on SHREC Dataset

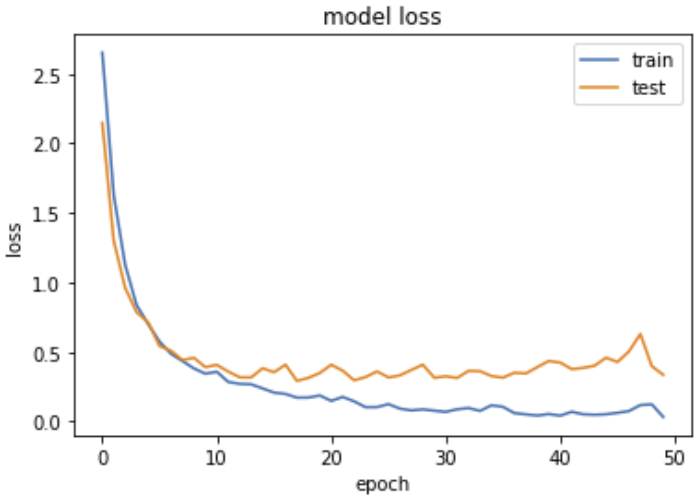

Figure 4.6: Loss curve for Model trained and tested on SHREC Dataset

Table 4.2 shows the confusion matrix obtained for different gesture classes:

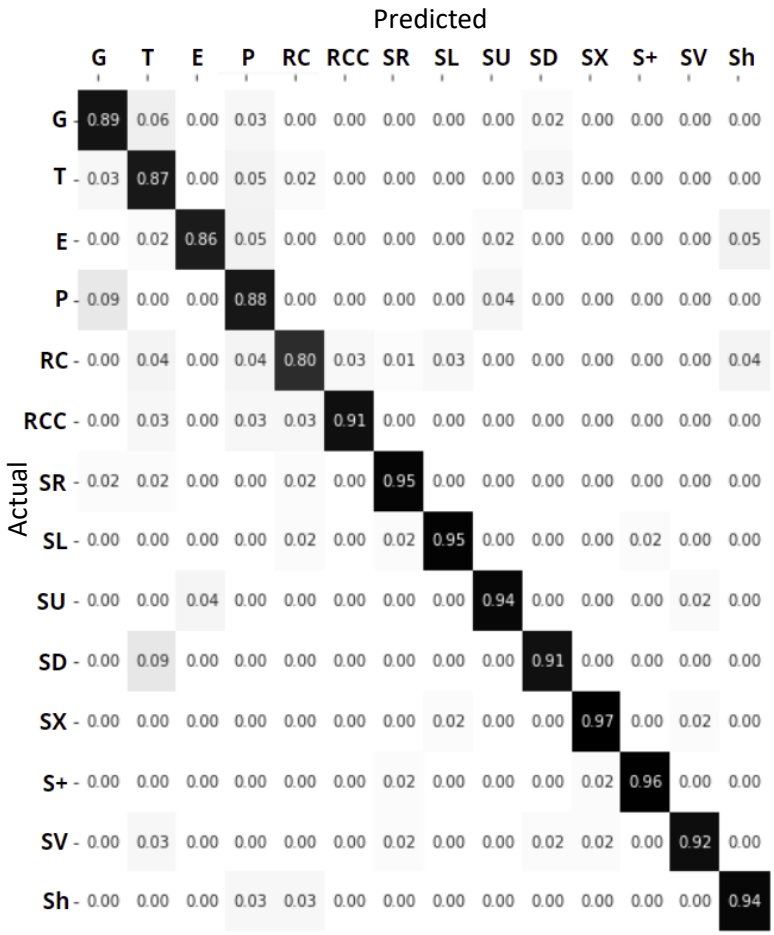

Table 4.2: Confusion Matrix for Model trained and tested on SHREC Dataset

4.3 DHG 14/28

In this section, we try to recreate the results of [15] on DHG 14/28 dataset using the same model and we obtained 85.60% accuracy on 14 gesture classes. The classification report consisting of precision, recall, and F1-Score is given in Table 4.3.

Gesture Class	Precision	Recall	F1-Score
G	0.76	0.58	0.65
T	0.86	0.88	0.87
E	0.93	0.84	0.88
P	0.63	0.91	0.74
RC	0.85	0.83	0.84
RCC	0.92	0.93	0.93
SR	0.92	0.81	0.86
SL	0.84	0.97	0.9
SU	0.85	0.94	0.89
SD	0.92	0.71	0.8
SX	0.97	0.93	0.95
S+	1	0.93	0.96
SV	0.9	0.93	0.92
Sh	0.84	0.89	0.87
Accuracy	85.60%		

Table 4.3: Classification report for Model trained and tested on DHG Dataset

This model's accuracy and loss graphs are shown in Fig 4.7 and Fig 4.8:

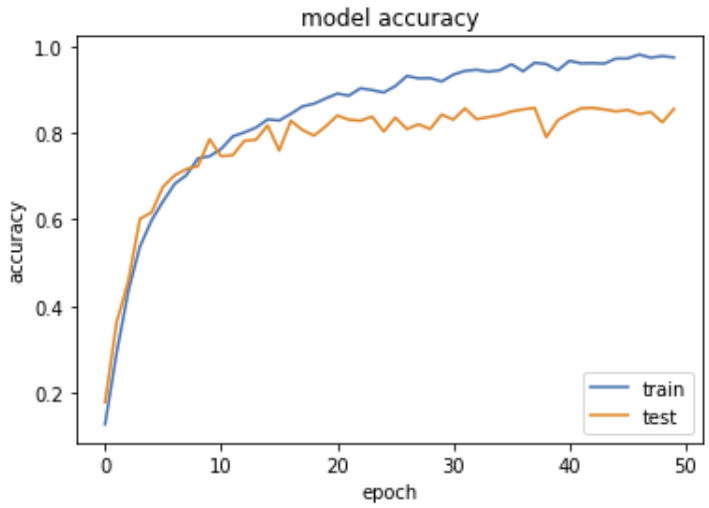

Figure 4.7: Accuracy curve for Model trained and tested on DHG Dataset

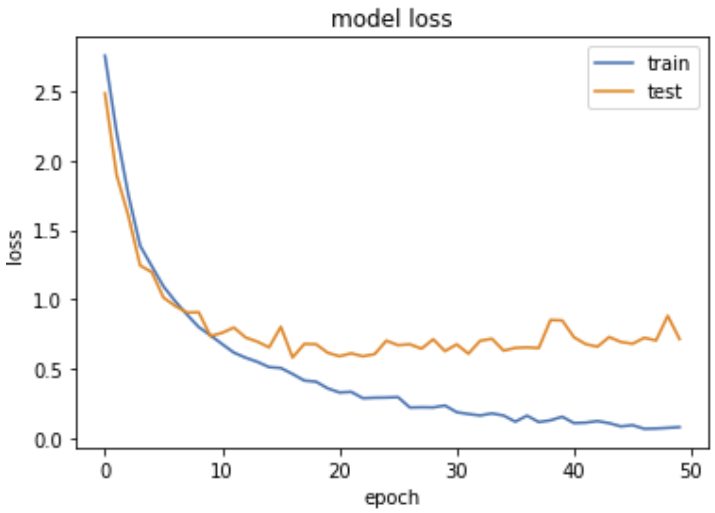

Figure 4.8: Loss curve for Model trained and tested on DHG Dataset

Table 4.4 shows the confusion matrix obtained for different gesture classes:

Predicted

Actual	G	T	E	P	RC	RCC	SR	SL	SU	SD	SX	S+	SV	Sh
G	0.58	0.09	0.03	0.27	0.02	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
T	0.04	0.88	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.01
E	0.02	0.00	0.84	0.00	0.02	0.00	0.00	0.00	0.11	0.00	0.00	0.00	0.00	0.02
P	0.05	0.02	0.00	0.91	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
RC	0.02	0.02	0.00	0.02	0.83	0.00	0.02	0.06	0.00	0.02	0.00	0.00	0.00	0.02
RCC	0.00	0.02	0.00	0.02	0.00	0.93	0.00	0.03	0.00	0.00	0.00	0.00	0.00	0.00
SR	0.00	0.00	0.04	0.00	0.02	0.00	0.81	0.00	0.02	0.00	0.00	0.00	0.07	0.04
SL	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.97	0.00	0.00	0.00	0.00	0.00	0.01
SU	0.04	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.00	0.00	0.00	0.02	0.00
SD	0.03	0.01	0.00	0.12	0.03	0.03	0.00	0.03	0.00	0.71	0.01	0.00	0.00	0.03
SX	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.00	0.93	0.00	0.02	0.02
S+	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.02	0.02	0.93	0.00	0.00
SV	0.00	0.00	0.00	0.02	0.00	0.00	0.05	0.00	0.00	0.00	0.00	0.00	0.93	0.00
Sh	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.04	0.00	0.02	0.00	0.00	0.00	0.89

Table 4.4: Confusion Matrix for Model trained and tested on DHG Dataset

4.3 Mixed Dataset

In this section, we try different combinations of Dataset to test the sensitivity of the model. We train the model on the DHG dataset and test it on SHREC and vice-versa. We also train the model on both DHG and SHREC and test it on both datasets separately.

Trained on DHG, Tested on SHREC

We train the model on the DHG dataset and test it on the SHREC dataset. The model achieves a classification accuracy of 34.64% on 14 Gesture classes. The classification report consisting of precision, recall, and F1-Score is given in Table 4.5.

Gesture Class	Precision	Recall	F1-Score
G	0.24	0.37	0.29
T	0.3	0.05	0.08
E	0.4	0.78	0.53
P	0.43	0.28	0.34
RC	0.67	0.11	0.2
RCC	0.55	0.35	0.43
SR	0.22	0.3	0.25
SL	0.45	0.28	0.35
SU	0.11	0.43	0.17
SD	0.5	0.23	0.31
SX	0.48	0.18	0.26
S+	0.52	0.57	0.54
SV	0.38	0.3	0.33
Sh	0.84	0.72	0.77
Accuracy	34.64%		

Table 4.5: Classification report for Model trained on DHG Dataset and tested on SHREC Dataset

This model's accuracy and loss graphs are shown in Fig 4.9 and Fig 4.10:

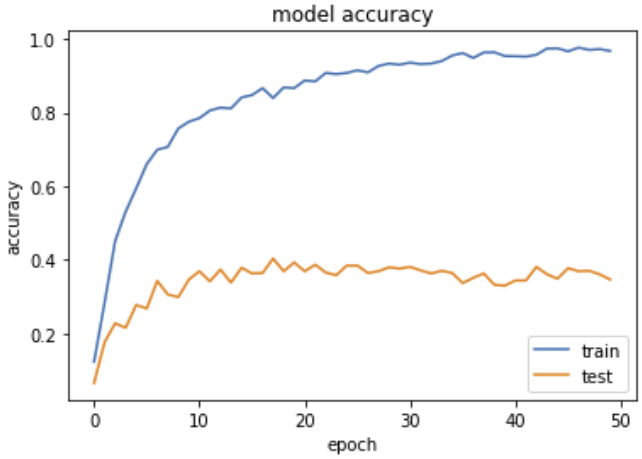

Figure 4.9: Accuracy curve for Model trained on DHG Dataset and tested on SHREC Dataset

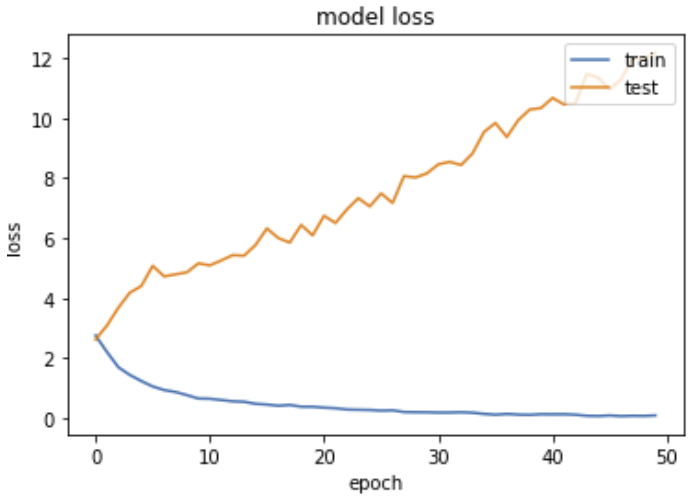

Figure 4.10: Loss curve for Model trained on DHG Dataset and tested on SHREC Dataset

Table 4.6 shows the confusion matrix obtained for different gesture classes:

Actual \ Predicted	G	T	E	P	RC	RCC	SR	SL	SU	SD	SX	S+	SV	Sh
G	0.37	0.02	0.05	0.17	0.00	0.05	0.02	0.00	0.32	0.00	0.00	0.00	0.02	0.00
T	0.21	0.05	0.13	0.03	0.02	0.00	0.02	0.00	0.56	0.00	0.00	0.00	0.00	0.00
E	0.02	0.02	0.78	0.00	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.02
P	0.56	0.04	0.11	0.28	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00
RC	0.01	0.01	0.11	0.03	0.11	0.10	0.07	0.14	0.31	0.03	0.00	0.00	0.01	0.04
RCC	0.08	0.00	0.25	0.00	0.03	0.35	0.06	0.00	0.14	0.00	0.00	0.00	0.05	0.05
SR	0.00	0.02	0.11	0.00	0.02	0.00	0.30	0.18	0.11	0.00	0.11	0.00	0.14	0.02
SL	0.03	0.00	0.02	0.00	0.00	0.12	0.27	0.28	0.11	0.00	0.00	0.06	0.11	0.00
SU	0.20	0.00	0.06	0.08	0.00	0.00	0.00	0.00	0.43	0.20	0.00	0.00	0.00	0.02
SD	0.02	0.00	0.04	0.02	0.00	0.02	0.00	0.00	0.55	0.23	0.00	0.11	0.02	0.00
SX	0.10	0.00	0.00	0.00	0.00	0.00	0.15	0.00	0.11	0.00	0.18	0.31	0.15	0.00
S+	0.00	0.00	0.00	0.00	0.00	0.00	0.41	0.00	0.02	0.00	0.00	0.57	0.00	0.00
SV	0.07	0.00	0.13	0.00	0.00	0.00	0.00	0.02	0.43	0.00	0.07	0.00	0.30	0.00
Sh	0.00	0.02	0.09	0.02	0.00	0.00	0.05	0.02	0.06	0.00	0.03	0.00	0.00	0.72

Table 4.6: Confusion Matrix for Model trained on DHG Dataset and tested on SHREC Dataset

Trained on SHREC, Tested on DHG

We train the model on the SHREC dataset and test it on the DHG dataset. The model achieves a classification accuracy of 27.02% on 14 Gesture classes. The classification report consisting of precision, recall, and F1-Score is given in Table 4.7.

Gesture Class	Precision	Recall	F1-Score
G	0.5	0.14	0.22
T	0.51	0.38	0.43
E	0.5	0.08	0.14
P	0.18	0.82	0.29
RC	0.71	0.09	0.17
RCC	0.4	0.23	0.29
SR	0.47	0.13	0.2
SL	0.24	0.24	0.24
SU	0.14	0.06	0.08
SD	0.23	0.16	0.19
SX	0.36	0.07	0.11
S+	0.86	0.42	0.56
SV	0.42	0.25	0.31
Sh	0.18	0.76	0.29
Accuracy	27.02%		

Table 4.7: Classification report for Model trained on SHREC Dataset and tested on DHG Dataset

This model's accuracy and loss graphs are shown in Fig 4.11 and Fig 4.12:

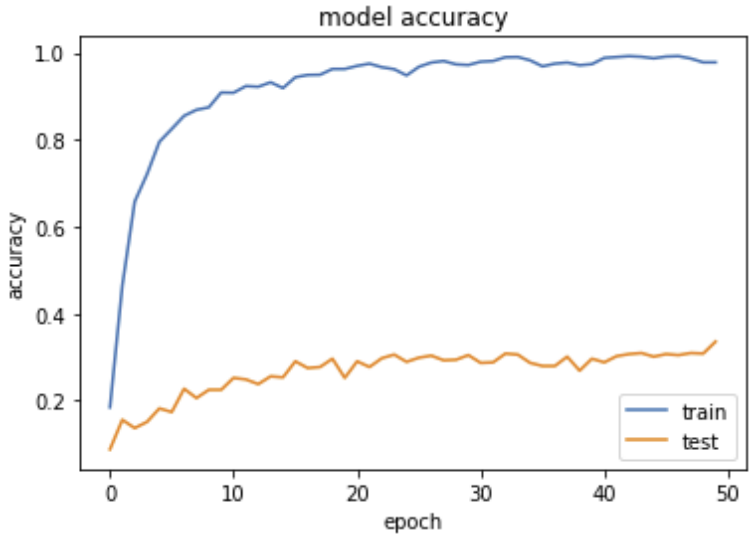

Figure 4.11: Accuracy curve for Model trained on SHREC Dataset and tested on DHG Dataset

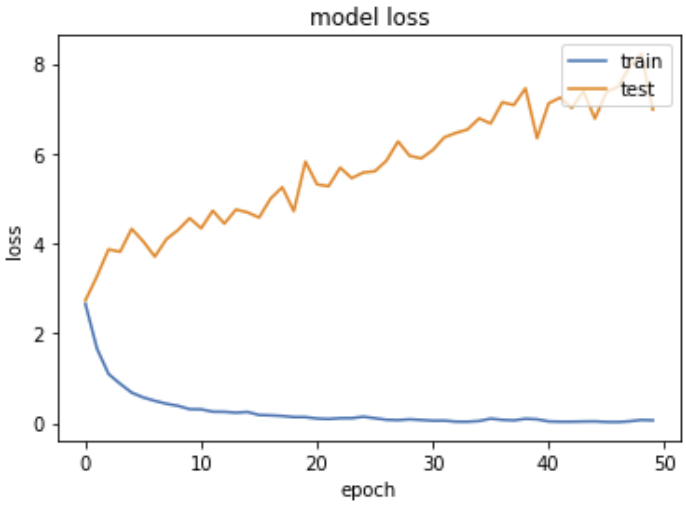

Figure 4.12: Loss curve for Model trained on SHREC Dataset and tested on DHG Dataset

Table 4.8 shows the confusion matrix obtained for different gesture classes:

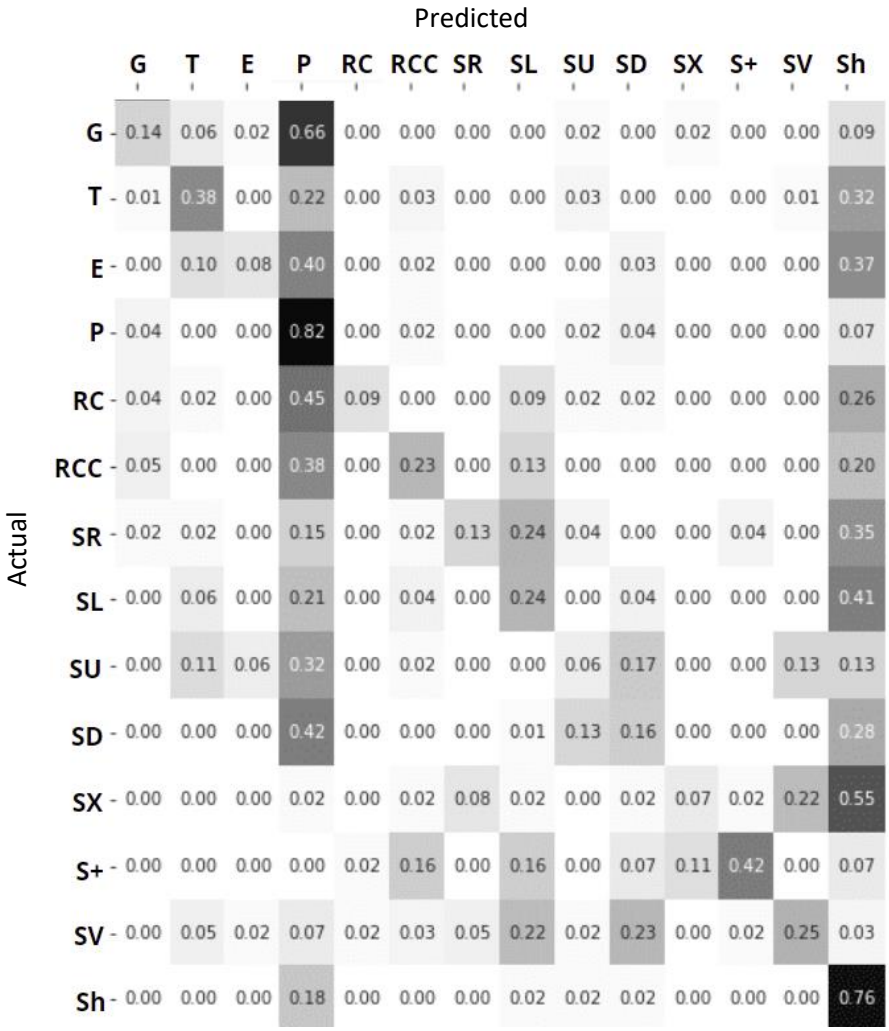

Table 4.8: Confusion Matrix for Model trained on SHREC Dataset and tested on DHG Dataset

Trained on both SHREC and DHG, Tested on SHREC

We train the model on both datasets and test it on the SHREC dataset. The model achieves a classification accuracy of 49.34% on 14 Gesture classes. The classification report consisting of precision, recall, and F1-Score is given in Table 4.9.

Gesture Class	Precision	Recall	F1-Score
G	0.44	0.34	0.38
T	0.19	0.06	0.06
E	0.44	0.79	0.57
P	0.52	0.56	0.54
RC	0.78	0.44	0.56
RCC	0.9	0.42	0.57
SR	0.4	0.52	0.45
SL	0.76	0.55	0.64
SU	0.21	0.57	0.31
SD	0.46	0.3	0.36
SX	0.57	0.39	0.47
S+	0.81	0.78	0.79
SV	0.26	0.34	0.29
Sh	0.76	0.86	0.81
Accuracy	49.34%		

Table 4.9: Classification report for Model trained on both SHREC and DHG Dataset and tested on SHREC Dataset

This model's accuracy and loss graphs are shown in Fig 4.13 and Fig 4.14:

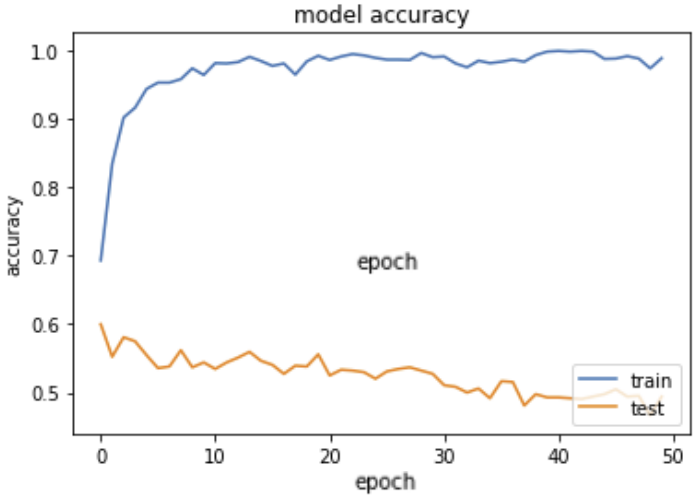

Figure 4.13: Accuracy curve for Model trained on both SHREC and DHG Dataset and tested on SHREC Dataset

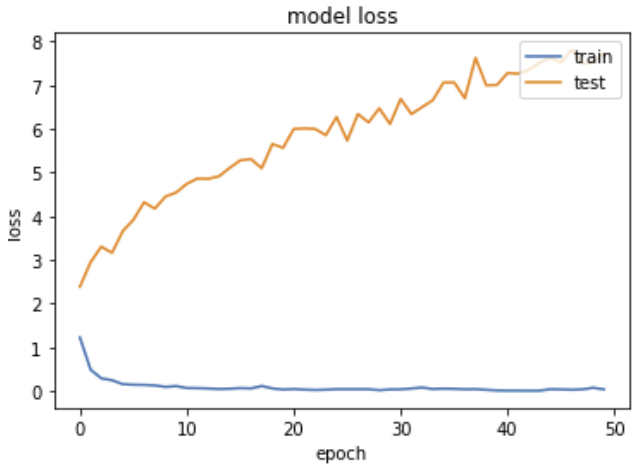

Figure 4.14: Accuracy curve for Model trained on both SHREC and DHG Dataset and tested on SHREC Dataset

Table 4.10 shows the confusion matrix obtained for different gesture classes:

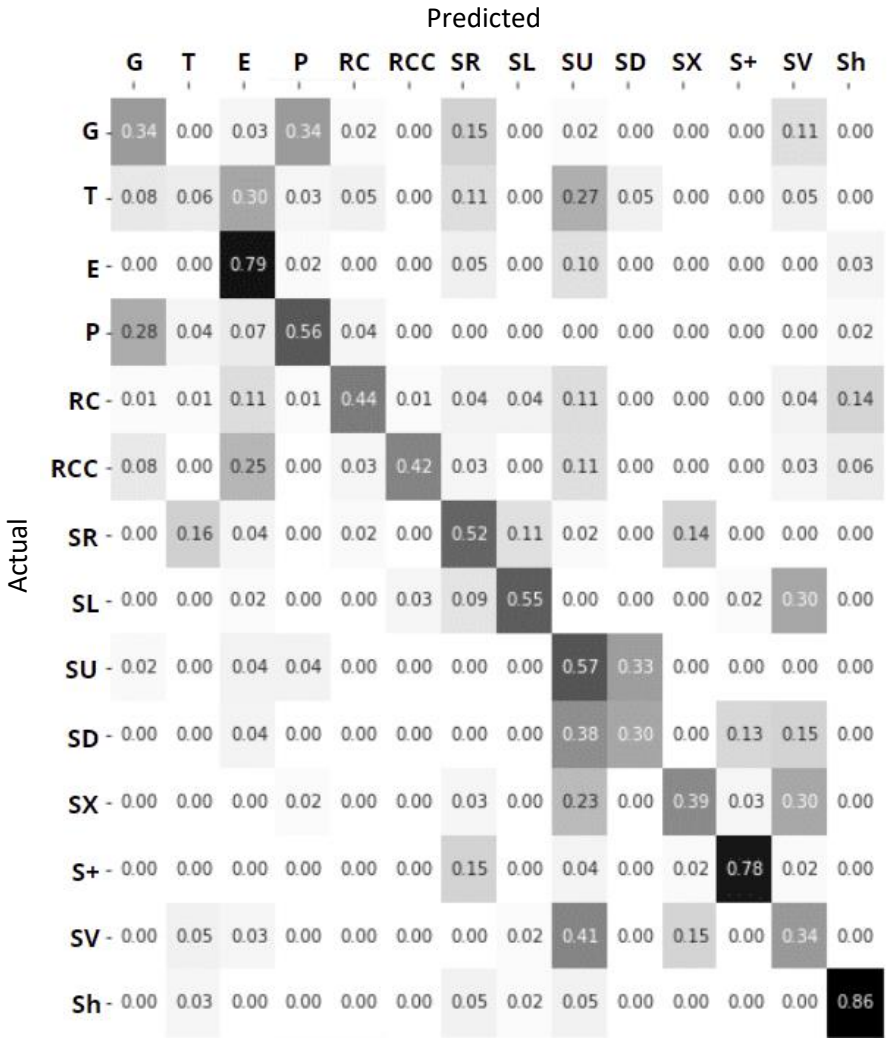

Table 4.10: Confusion Matrix for Model trained on both SHREC and DHG Dataset and tested on SHREC Dataset

Trained on both DHG and SHREC, Tested on DHG

We also tried training the model on both datasets and test it on the SHREC dataset. The model achieves a classification accuracy of 49.34% on 14 Gesture classes. The classification report consisting of precision, recall, and F1-Score is given in Table 4.11.

Gesture Class	Precision	Recall	F1-Score
G	0.52	0.17	0.26
T	0.56	0.64	0.59
E	0.81	0.21	0.33
P	0.24	0.75	0.36
RC	0.6	0.4	0.48
RCC	0.43	0.53	0.47
SR	0.67	0.11	0.19
SL	0.38	0.51	0.43
SU	0.12	0.04	0.06
SD	0.22	0.23	0.22
SX	0.78	0.52	0.62
S+	0.63	0.58	0.61
SV	0.5	0.18	0.27
Sh	0.37	0.87	0.52
Accuracy	41.60%		

Table 4.11: Classification report for Model trained on both DHG and SHREC Dataset and tested on DHG Dataset

This model's accuracy and loss graphs are shown in Fig 4.15 and Fig 4.16:

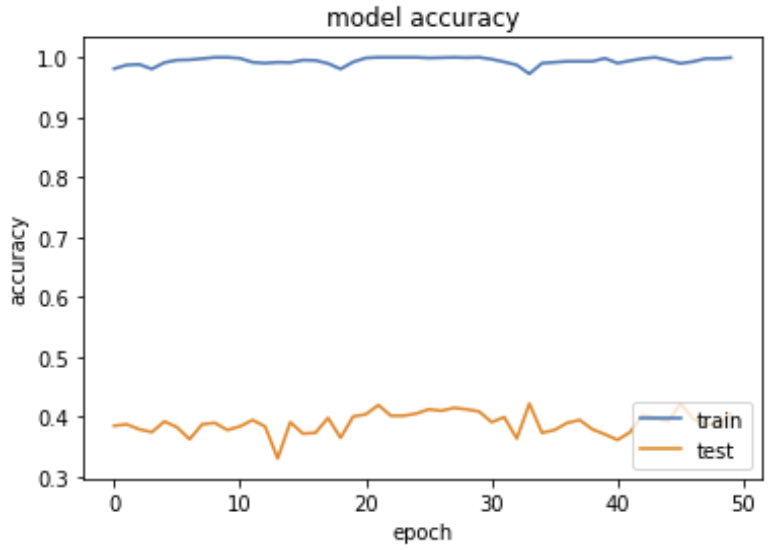

Figure 4.15: Accuracy curve for Model trained both DHG and SHREC Dataset and tested on DHG Dataset

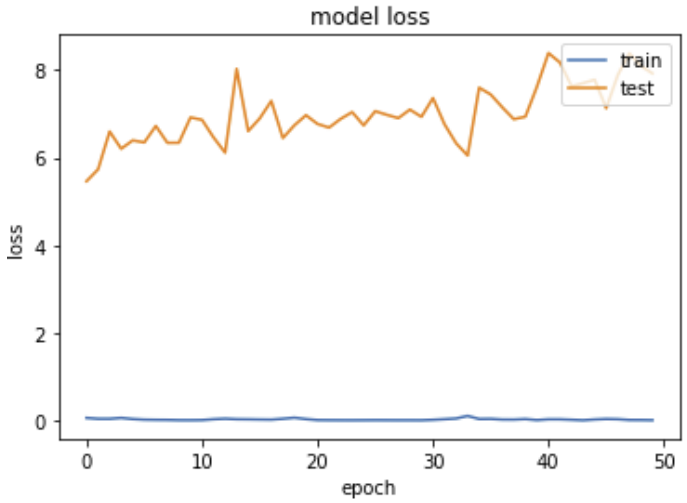

Figure 4.16: Accuracy curve for Model trained both DHG and SHREC Dataset and tested on DHG Dataset

Table 4.12 shows the confusion matrix obtained for different gesture classes:

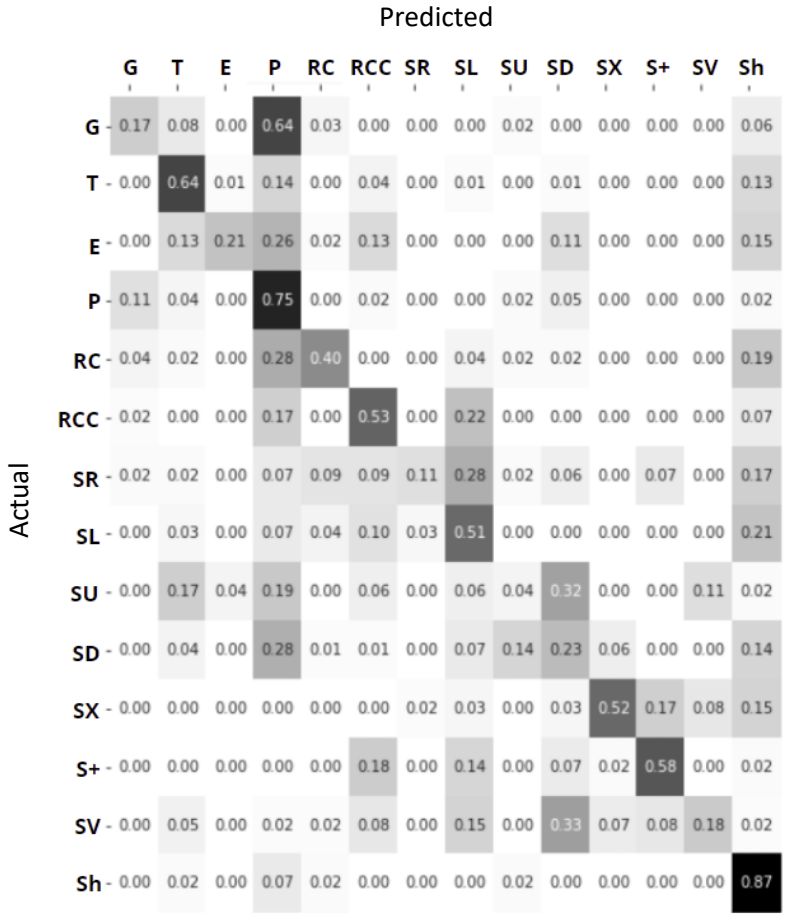

Table 4.12: Confusion Matrix for Model trained on both DHG and SHREC Dataset and tested on DHG Dataset

Chapter 5

Conclusion

In this chapter, the results obtained in the previous chapter are discussed thoroughly to draw conclusions. Section 5.1 discusses the different results obtained while subsequent chapters discuss the scalability and future work that can be done on the project.

5.1 Discussion

5.1.1 SHREC 2017

We obtain an accuracy of 91.79 on the SHREC 2017 dataset, which is slightly higher than the accuracy obtained in [15]. We also get high precision and recall values, as we can see from the classification report in section 4.2 table 4.1. High precision relates to a low false-positive rate which means that the model is correctly learning features and giving correct results when tested without labels.

Looking at the accuracy graph in Fig 4.5, we can infer that there is minimal overfitting in the model. It does not affect the accuracy of the model that much as the train and test accuracies are close to each other. The Loss graph in Fig 4.6 corresponds to zero, which is also a good sign showing that there is very little underfitting occurring in the model.

We look at the confusion matrix in Table 4.2 and see how the actual gesture class is classifying for every other predicted gesture class. We can infer from the high values of matrix diagonal that all the gestures classes are performing well and giving out correct classifications while predicting gestures.

Overall, the model trained and tested on SHREC 2017 dataset is performing really well and giving us great results.

5.1.2 DHG 14/18

We obtain an accuracy of 85.60 on the DHG 14/28 dataset, which is lower than the accuracy obtained on the SHREC dataset and [15]. We get high precision and recall values, as we can see from the classification report in section 4.3 Table 4.3. This means that this model correctly learns the features and gives correct results when tested on data without labels.

Looking at the accuracy graph in Fig 4.7, we can infer that there is some overfitting in the model. The Loss graph in Fig 4.8 corresponds to zero, but there is noticeable overfitting occurring in the model than the model trained on the SHREC dataset. This could be due to a smaller number of epochs. We can try to increase epochs until the validation curve stops improving. We can also add an early stopping call back to identify the number of epochs required.

Looking at the confusion matrix in Table 4.4, we can see that it performs well for all gesture classes other than G(Grab) class, where it is only giving 58% accuracy. We can see from the confusion matrix that it is wrongly classifying Grab as Pinch with 27% accuracy. This could be because pinch and grab are very close gestures sequences in terms of hand movement. They can easily be confused as if a person tries to grab something with only index finger and thumb that can be confused as a pinch. This could be one of the reasons for this confusion.

5.1.3 Mixed Datasets

Trained on DHG, Tested on SHREC

We obtain an accuracy of 34.64% when the model is trained on the DHG dataset and tested on the SHREC dataset. There is a significant drop in the accuracy of the model. There a lot of false positives in the model, as we can see from the confusion matrix. This is leading to poor precision and recall values in the classification report of the model.

Looking at the accuracy graph in Fig 4.9, we can infer that there is a strong case of overfitting in the model. The model is too closely aligned to a limited set of data points which greatly affects the learning capacity of the model. The Loss graph in Fig 4.10 clearly shows a strong case of an overfit model with too large of a capacity and learning rate. We can conclude this because the training loss continues to decrease with number of epochs, whereas the testing loss decreases to a point and then suddenly starts increasing with a number of epochs. We can try to fix this by trying different combinations of regularization, learning rate, and number of epochs while training the model.

Looking at the confusion matrix in Table 4.6, we can see that it performs well for only two gesture classes, E(Expand) and Sh(Shake). It is classifying almost every other gesture with very low accuracy(less than 50%). We can look at the confusion matrix and try to if there is a pattern in confusion between different classes:

1. Grab and Tap is confused heavily with Swipe up. This could be due to similarity in the sequence of hand movements while performing these gestures.
2. Pinch is confused with Grab. This is the opposite of the confusion occurring in the DHG dataset, but the reason could be similar.
3. Rotation Clockwise and Rotation counterclockwise classes are confused with Expand class. This could again be due to the similarity in the movement of the hand while performing these gestures.
4. Swipe left(SL), and Swipe right(SR) are confused with each other.
5. Swipe up(SU) and Swipe down(SD) are confused with each other.
6. Swipe + is confused with a Swipe right. This could be confused when a person tries to make + with a horizontal line first.
7. Swipe V is confused with Swipe Up.

Trained on SHREC, Tested on DHG

We obtain an accuracy of 27.02% when the model is trained on the SHREC dataset and tested on the DHG dataset. This was expected as we concluded in the previous experiment that the model is overfitting and not learning the features of Gesture. We can confirm this by looking at the accuracy graph in Fig 4.11 and the loss graph in Fig 4.12, where the same problem discussed in the previous experiment arises in this model as well. Both models are performing poorly when tested on some other dataset.

Looking at the confusion matrix in Table 4.8, we can see that it performs well for only two gesture classes Sh(Shake) and P(Pinch). It is classifying almost every other Gesture with very low accuracy(less than 50%). We can look at the confusion matrix and try to if there is a pattern in confusion between different classes:

1. Grab and Expand are heavily confused with Pinch.
2. Rotation Clockwise, Rotation Counter-Clockwise, Swipe up, Swipe down are confused with Pinch.
3. Swipe left, and Swipe right are confused with Shake.

We can notice in this confusion matrix that most of the classes are getting confused with Pinch and Shake Gesture Classes. It means that the model might be learning these classes closely and the rest of the classes. This could be due to overfitting. But we can see some patterns emerging that majority of classes are getting confused with Pinch and Shake gesture classes.

DHG and SHREC datasets are recorded in the exact same way with the same number of gesture classes, hence our initial thought was that the results should be consistent on both. But after conducting an experiment on cross-train and test we find that the accuracy significantly drops. We can notice certain patterns here. SL, SR, and SU, SD are opposite to each other, and the model is getting confused between these. This could be due to the way data pre-processing is performed. The confused gesture classes might have data points outside the boundaries of input which means that they were filled with a reflection using interpolation as mentioned in Section 3.5. The way swipe left and swipe right gestures are performed should be exactly opposite to each other but if we fill the input data with a reflection of swipe left it becomes swipe right which is one of the reasons why the model is performing poorly on a different dataset than the one it has been trained on. We discuss the next results to see if this hypothesis is correct or not.

Trained on both SHREC and DHG, Tested on SHREC

When we train the model on both datasets, first on the SHREC dataset and then on the DHG dataset, and then test accuracy on SHREC increases to 49.34%. This was expected as we could see the model overfitting in previous experiments. We can confirm this by looking at the accuracy graph in Fig 4.13 and the loss graph in Fig 4.14, where the same problem discussed in the previous experiments arises in this model as well. The models are performing better than previous experiments, but the problem of overfitting still exists.

Looking at the confusion matrix in Table 4.10, we can see that it performs well for only three gesture classes E(Expand), S+(Swipe+), and Sh(Shake). We can look at the confusion matrix and try to if there is a pattern in confusion between different classes:

1. Grab is confused as Pinch and vice versa.
2. Tap is confused with Swipe up and Expand.
3. Swipe up and Swipe down are confused with each other.
4. Swipe V is confused with SU.

This experiment shows better results than the previous two as we can see that 8 out of 14 gesture classes show more than 50% classification accuracy. We observe the same confusion patterns as in the previous cross-train/test experiment but with little improvement. The model is trained on both datasets, so it performs a little better when cross-tested. The accuracy obtained is still not desirable, but we are definitely on the correct path to improving the model's performance on cross-testing.

Trained on both DHG and SHREC, Tested on DHG

When we train the model on both datasets, the test accuracy on DHG increases to 41.60%. This was expected as we could see the model overfitting in previous experiments. We can confirm this by looking at the accuracy graph in Fig 4.15 and the loss graph in Fig 4.16, where the same problem discussed in the previous experiments arises in this model as well. The models are performing better than previous experiments, but the problem of overfitting still exists.

Looking at the confusion matrix in table 4.12, we can see that it performs well for only three gesture classes T(Tap), P(Pinch), and Sh(Shake). We can look at the confusion matrix and try to if similar patterns are emerging as the previous experiments:

1. Grab, Expand, and Rotation Clockwise are confused as Pinch.
2. Swipe up and Swipe down are confused with each other.
3. Swipe V is confused with Swipe down.
4. Mostly every gesture class is confused with Pinch and Shake.

This experiment shows better results than the previous cross-train experiment as we can see that 7 out of 14 gesture classes show more than 50% classification accuracy

We can find similar patterns emerging after discussing the confusion matrices from all our different experiments. Some of the key observations are:

1. When cross-testing on the SHREC dataset, most of the Gesture Classes are confused with Swipe up and Expand.
2. When cross-testing on the DHG dataset, most of the Gesture Classes are confused with Pinch and Shake
3. Swipe Up/ Swipe Down and Swipe Left/ Swipe Right is getting confused with each other in almost every cross-train experiment.

We can see the performance of the model improving when we are increasing the training data. But it is still way low compared to the accuracy we were obtaining on individual datasets. This could be due to the interpolation of input data points as discussed previously. When we perform error analysis and look at the datasets carefully, we understand the problem might be in the way both of the datasets are recorded. SHREC is This dataset contains 14 hand gestures; each gesture is performed between 1 and 10 times by 28 participants whereas the DHG dataset is performed five times by 20 participants. This is creating a discrepancy in the number of data points in test and training sets which might be the reason for poor performance on the cross-testing.

5.2 Performance

Overall, we conclude that the model is performing well when trained and tested on the same dataset but after looking at the results in different experiments results we see that this parallel CNN model does not generalize very well, with accuracy dropping by around 50% when we cross-train/test. We also see that when we increase the size of the training data to double the number of gestures, i.e. train the model on both SHREC and DHG datasets the accuracy increases to some extent. But this is still leading to a lot of confusion between the gesture classes. We need to make the model robust enough to differentiate between the similar gestures classes such as swipe up and swipe down.

5.3 Scalability

The approach taken in this project is limited to only the datasets which already have the gesture sequence in skeletal coordinates form. There are many architectures other than BlazePose, which extract the skeletal coordinates of RGB images in real-time. Due to a lack of time and resources, we could not implement these architectures in this project, but we will briefly discuss them in this section.

1. LeapMotion: This optical hand tracking module captures the moment than in real-time accurately, as shown in Fig 5.1. We can use this module to create our own dataset of Gesture Sequences which will be stored in coordinate or skeletal form instead of RGB data. We can then use the recorded dataset to test the model discussed in this project.

Figure 5.1: LeapMotion Module Implementation

2. MediaPipe Hands: This is a high-fidelity hand and finger tracking architecture[26]. This tacks 21 3D landmarks points of a hand in one frame, as shown in Fig 5.2. This architecture can be used on consistent datasets to try and extract the coordinates of the landmark features and see how the model performs when train and tested on extracted data.

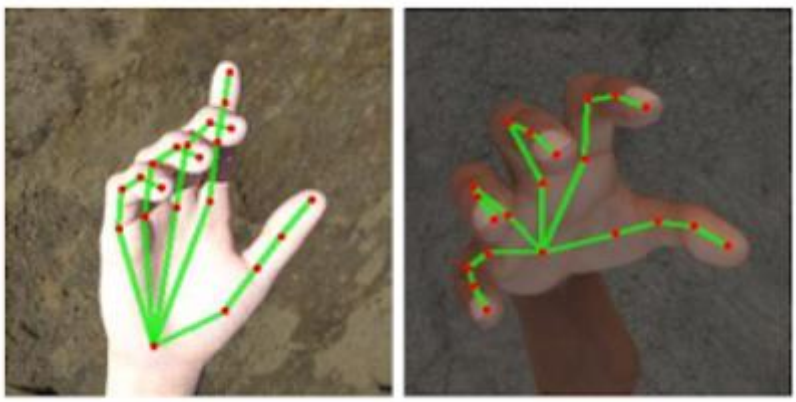

Figure 5.2: Mediapipe Hands Implementation [21]

5.4 Future Work

In this section, we will be talking about different approaches we can take in the future to try and improve the results. We will also be talking about one of the practical applications of this type of skeletal-based model.

5.4.1 Improvement

The approach taken in this project was successful for the individual datasets but performed poorly when we tried to cross-train/test the datasets. We concluded that the model is overfitting when tested on mixed datasets. There are many areas that this project can improve upon in the future for better performance:

1. One of the reasons for poor performance on the cross-train/test could be due to the same Hyperparameters used throughout. In the future, we would like to try and test if changing these parameters can lead to better performances in the mixed datasets; for instance, we could change the regularization or increase the dropout to reduce overfitting. Changing hyperparameters would improve our results to some level.

2. We can also try to implement any one of the approaches discussed in section 5.2 and extract the skeletal coordinates in real-time to see how that affects the performance of the model.

3. We need to fix the way data is pre-processed and make the model robust enough to work on the variable number of time steps. This gesture model currently only work on completed sequences. We need to think of a way to make the model work on half or incomplete gesture sequences because, in real-time, consistency in data is not necessary, which could lead to poor performance of this approach.

4. We also wanted to try to combine the datasets so that the training data is a single NumPy array rather than training on both datasets separately to see how that performs on different datasets. We were not able to follow this approach in this project due to a lack of time.

5.4.2 Applications

This project can have various applications in computer vision where it's required that the identity of the person remains protected. We will discuss one of the applications in this section.

According to various studies[27], injuries in or near bathtub or shower accounts for more than 66% of all emergency room visits. 31% of these injuries are head or neck related, leading to death if not addressed immediately. This project can be installed in bathrooms as a potential IoT solution that alerts family members or emergency services that there has been some accident inside the bathroom, requiring attention. This project does not require raw RGB images as its input data but the 3D skeletal data to detect the pose of a person, which makes it perfect for protecting the privacy of users inside the bathrooms because we do not want to invade the privacy of the person inside the bathroom. The model can be trained to recognize accidents which can then be connected to an app that sends out emergency response for help that can save a person's life.

References

[1] G. Imai, “Gestures: Body language and nonverbal communication,” *Retrieved Oct*, 2005.

[2] R. Yang, S. Sarkar, and B. Loeding, “Handling movement epenthesis and hand segmentation ambiguities in continuous sign language recognition using nested dynamic programming,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 3, pp. 462–477, 2009.

[3] D. Wickeroth, P. Benolken, and U. Lang, “Markerless gesture based interaction for design review scenarios,” in *2009 Second International Conference on the Applications of Digital Information and Web Technologies*, 2009, pp. 682–687.

[4] S. Laraba, M. Brahimi, J. Tilmanne, and T. Dutoit, “3D Skeleton-Based Action Recognition by Representing Motion Capture Sequences as 2D-RGB Images,” *Comput. Animat. Virtual Worlds*, vol. 28, May 2017, doi: 10.1002/cav.1782.

[5] V. I. Pavlovic, R. Sharma, and T. S. Huang, “Visual interpretation of hand gestures for human-computer interaction: a review,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, no. 7, pp. 677–695, Jul. 1997, doi: 10.1109/34.598226.

[6] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, “Hand gesture recognition with 3D convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 1–7.

[7] L. Deng and D. Yu, “Deep Learning: Methods and Applications,” *Found. Trends Signal Process.*, vol. 7, no. 3–4, pp. 197–387, Jun. 2014, doi: 10.1561/2000000039.

[8] H.-I. Lin, M.-H. Hsu, and W.-K. Chen, “Human hand gesture recognition using a convolution neural network,” in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, Aug. 2014, pp. 1038–1043, doi: 10.1109/CoASE.2014.6899454.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[10] D. Ciresan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, Jun. 2012, pp. 3642–3649, doi: 10.1109/CVPR.2012.6248110.

[11] Q. D. Smedt, H. Wannous, J.-P. Vandeborre, J. Guerry, B. L. Saux, and D. Filliat, “3D Hand Gesture Recognition Using a Depth and Skeletal Dataset,” *Eurographics Workshop 3D Object Retr.*, p. 6 pages, 2017, doi: 10.2312/3DOR.20171049.

[12] P. Wang, Z. Li, Y. Hou, and W. Li, “Action Recognition Based on Joint Trajectory Maps Using Convolutional Neural Networks,” *ArXiv161102447 Cs*, Nov. 2016, Accessed: May 02, 2021. [Online]. Available: http://arxiv.org/abs/1611.02447.

[13] S. Song, C. Lan, J. Xing, W. Zeng, and J. Liu, “An End-to-End Spatio-Temporal Attention Model for Human Action Recognition from Skeleton Data,” *ArXiv161106067 Cs*, Nov. 2016, Accessed: May 02, 2021. [Online]. Available: http://arxiv.org/abs/1611.06067.

[14] T. Laurent and J. von Brecht, “A recurrent neural network without chaos,” *ArXiv161206212 Cs*, Dec. 2016, Accessed: Apr. 23, 2021. [Online]. Available: http://arxiv.org/abs/1612.06212.

[15] G. Devineau, F. Moutarde, W. Xi, and J. Yang, “Deep learning for hand gesture recognition on skeletal data,” in *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, 2018, pp. 106–113.

[16] Phung and Rhee, “A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets,” *Appl. Sci.*, vol. 9, p. 4500, Oct. 2019, doi: 10.3390/app9214500.

[17] K. Sun, B. Xiao, D. Liu, and J. Wang, “Deep High-Resolution Representation Learning for Human Pose Estimation,” *ArXiv190209212 Cs*, Feb. 2019, Accessed: Apr. 04, 2021. [Online]. Available: http://arxiv.org/abs/1902.09212.

[18] “COCO - Common Objects in Context.” https://cocodataset.org/#keypoints-2020 (accessed Apr. 05, 2021).

[19] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, “BlazePose: On-device Real-time Body Pose tracking,” *ArXiv200610204 Cs*, Jun. 2020, Accessed: Apr. 04, 2021. [Online]. Available: http://arxiv.org/abs/2006.10204.

[20] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, “The Jester Dataset: A Large-Scale Video Dataset of Human Gestures,” in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Seoul, Korea (South), Oct. 2019, pp. 2874–2882, doi: 10.1109/ICCVW.2019.00349.

[21] Q. De Smedt, H. Wannous, and J.-P. Vandeborre, “Skeleton-Based Dynamic Hand Gesture Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Las Vegas, NV, USA, Jun. 2016, pp. 1206–1214, doi: 10.1109/CVPRW.2016.153.

[22] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *ArXiv151203385 Cs*, Dec. 2015, Accessed: Apr. 22, 2021. [Online]. Available: http://arxiv.org/abs/1512.03385.

[23] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” p. 8.

[24] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Dec. 2014, Accessed: Apr. 23, 2021. [Online]. Available: https://arxiv.org/abs/1412.6980v9.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 56, pp. 1929–1958, 2014.

[26] F. Zhang *et al.*, “MediaPipe Hands: On-device Real-time Hand Tracking,” Jun. 2020, Accessed: Apr. 27, 2021. [Online]. Available: https://arxiv.org/abs/2006.10214v1.

[27] S. J. Mao, L. B. McKenzie, H. Xiang, and G. A. Smith, “Injuries associated with bathtubs and showers among children in the United States,” *Pediatrics*, vol. 124, no. 2, pp. 541–547, Aug. 2009, doi: 10.1542/peds.2008-2489.

Code for the project can be found at https://github.com/psharma228/FYP-Human-Gesture-Recognition