San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

Spring 2022

Gesture Recognition using Neural Networks

Ashwini Kurady
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

 Part of the Artificial Intelligence and Robotics Commons

Recommended Citation

Kurady, Ashwini, "Gesture Recognition using Neural Networks" (2022). *Master's Projects*. 1081.
DOI: https://doi.org/10.31979/etd.jmhw-h8eh
https://scholarworks.sjsu.edu/etd_projects/1081

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Writing Project

Gesture Recognition using Neural Networks

Final Report

Author

Ashwini Kurady

CS 298

05/16/2022

Advisor

Dr. Chris Tseng

A Writing Project Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment of the Requirements for the Degree: Master of Science

© 2022

Ashwini Kurady

ALL RIGHTS RESERVED

The Designated Committee Approves the Master's Project Titled

Gesture Recognition using Neural Network

By

Ashwini Kurady

Approved for the Department of Computer Science

San José State University

May 2022

Dr. Chris Tseng Department of Computer Science	Date
Dr. Ching-Seh Wu Department of Computer Science	Date
Mr. Nikhil Hiremath Software Developer, Adobe, CA	Date

Acknowledgements

I am heartfully grateful to my project advisor, Dr. Chris Tseng, for his expertise, guidance, encouragement, and time. I would like to thank my committee members, Dr. Ching-Seh Wu and Mr. Nikhil Hiremath, for their inputs and suggestions. Lastly, I would like to thank my family and friends for their kind support.

Abstract

The advances in technology have brought in a lot of changes in the way humans go about their lives. This has enhanced the significance of Artificial Neural Networks and Computer Vision-based interactions with the world. Gesture Recognition is one of the major focus areas in Computer Vision. This involves Human Computer Interfaces (HCI) that would capture and understand human actions. In this project, we will explore how Neural Network concepts can be applied in this challenging field of Computer Vision. By leveraging the latest research for Gesture Recognition, we researched on how to capture the movement across different frames of the gestures in videos. We experimented on preprocessed 2D and 3D data by applying various Neural Network models such as Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) using Time Series Classification Technique to recognize the Gesture.

Index Terms – **Neural Networks, Computer Vision, Gesture Recognition, Time-Series Classification.**

TABLE OF CONTENTS

1 Problem Statement

Computer Vision is defined as the capacity of the said system to arrive at a decision or responsive action based on analysis of information derived from visual data that is perceived from the environment. This provides the computer and/or computer-controlled devices like automobiles the intelligence akin to human beings by being able to perceive and perform actions based on the perceived data. This part of the Artificial Intelligence helps to identify and classify objects. Earlier approaches were efficient in this process in general but failed to identify when new data is given as input. This is because these approaches were based on deterministic logic and algorithm which fail due to uncertainty of prediction when faced with uncertainty in data coming in from new surroundings and situations [1]. With internet explosion, there was a big change in the amount of data available and data driven methods were preferred over logic-based methods with the drastic increase in raw data (images and videos) due to heavy usage of internet [2]. Statistical Modeling Techniques were introduced to provide a highly approximate results of the real-world datasets instead of traditional deterministic techniques [3]. Yann LeCun [4] proposed a new approach to categorize each pixel in an image to the object it can belong to by using Convolutional Network Feature Extractor. Though this approach used Deep Learning Unsupervised Methods, it was not a great approach to learn low level features of an image. A Neural Network Model called AlexNet provided impeccable results with the advent of GPUs a decade later. This significantly overcame the top 5 errors in ImageNet ILSVRC challenge in 2012 and ever since, every ImageNet challenge has been won by model based on Convolutional Neural Nets (CNNs) [5].

A wide range of applications can be found under Computer Vision such as video surveillance, tracking, Human-Computer Interaction. Basic foundation of Computer Vision falls under

image/video analysis and pattern recognition. However, accurate vision recognition system continues to be a challenging area of research in Artificial Intelligence. Various techniques have been explored to leverage the power of Neural Network and with coupling with digital sensors to improve the accuracy of the system. In this project, we will explore various techniques and related problems in the field of Computer Vision. We will focus on different Neural Network methods for Gesture Recognition. With this, we address the questions: *How accurate does Neural Network-based Computer Vision models predict identify Gesture Pattern? How Time Series Classification using Neural Network helps to recognize the Gestures? which Neura; Network models will help to enhance the Gesture Recognition? what are the limitations to for Neural Network to perform better in identifying Gesture?*

We will explore methodologies to identify Objects and Gestures in an image with the goal of imitating the Human Visual System and also, we will look at how Time Series Classification can be applied using Neural Network to improve the accuracy of the said system.

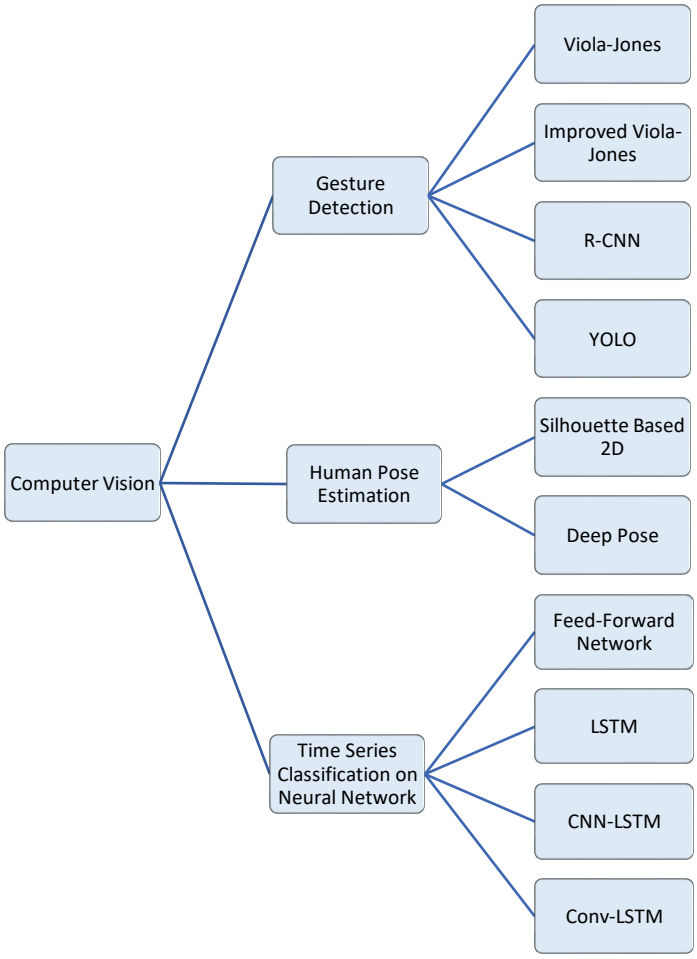


Fig. 1 Overview of Computer Vision Techniques

2 Prior Work

In this section, we will look at some of the important aspects of Gesture Recognition namely, Image Segmentation, Image Classification and Object Detection. Before we dive into classifying and detecting the image, we divide the image into segments so that we process the object information instead of processing the whole image. Image Classification refers to classifying the image to the class it belongs to.

2.1 Object Detection

For Object Detection, Viola-Jones[6] proposed Rapid Object Detection using Critical Visual features which is a fast yet efficient machine learning approach for object detection. This process has three steps including integral image representation of a frame, selecting the best visual features using AdaBoost learning algorithm and cascading complex classifiers so that the object is given more attention than the background and the noise. While this approach works well for facial recognition, it is difficult to detect other objects/gestures with this technique.

Qian Li[7] proposed an approach to improve Viola-Jones Object Detector by training the classifier with multiple feature images, changing the predefined threshold for classifier training and improving the algorithm by use of Support Vector Machine.

Region Convolution Neural Network (R-CNN) is a simple Deep Learning Object Detection scheme which makes use of selective search and CNN-based classification and scoring to detect a particular object among multiple objects [8][9]. A bounding box is created for Image Classification followed by feature vector computation using convolutional and connected layers. Scoring is done by using Support Vector Machines (SVMs) and the model is tuned until satisfactorily good prediction is obtained.

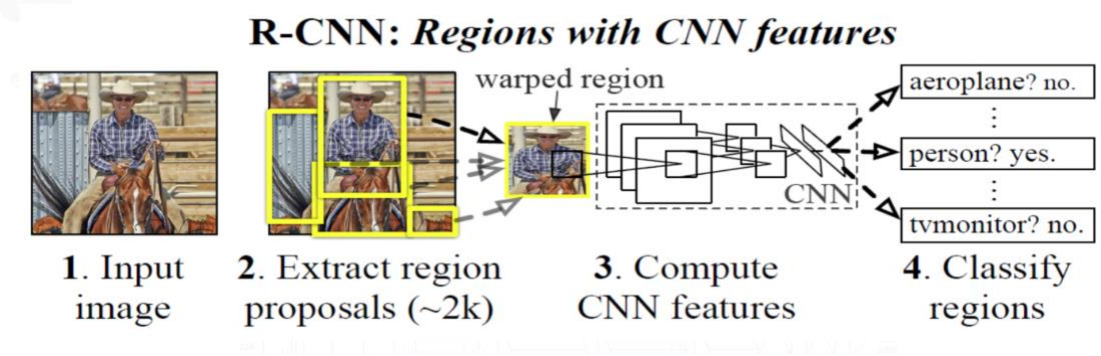


Fig. 2 RCNN with CNN Features [8]

An alternate approach which is not dependent on selecting segments of image for processing is YOLO (You Only Look Once) [10]. Since this is based on regression, this is better suited for Real Time Object Detection. This is also called Single Shot Object Detection. But due to the regression, this can be computational intensive. Zihan ni [11] proposed a Light YOLO model for hand Gesture Recognition. They strengthen the existing base YOLOv2 model by spatial refinement (They down- sampled the convolutional layers from Conv6 to Conv4). These changes increased the accuracy of the model from 96.8% to 98.6%.

2.2 Human Pose Estimation

Extending the Object Detection approach to detect the gesture of the human body, given an input data such as images and videos, Human Pose Estimation approaches help to build the human body representation from the input data. Over the last decade, this approach has had an increased focus from the research community and has a wide range of applications such as Human-Computer Interaction, motion analysis and many more. The critical parameter here is to detect only those human poses which involve movement in the body. To detect motion, images should be framed in order with moments apart. Hence, Human Pose Estimation can be done by comparing object position with respect to sequence of images. Below are a few of the great approaches to detect 2D and 3D Human Pose Estimation.

2.2.1 Silhouette-Based 2D Human Pose Estimation

All traditional approaches for the Human Pose Estimation were based on predefined framework for a pose or a standard template which is not dependent on the image data. These approaches limited the expressiveness of the estimation. Hence, Meng Li[12] proposed a new silhouette

based model to estimate 2D Human Pose. This base model is composed of 15 joints and 14 segments. Gaussian Mixture Background Model (GMM) was used to extract the silhouette from the image and they used techniques such as Distance Transform (DT) and Principle Component Analysis (PCA) to decide base joint for the model to estimate the human pose. They made joint A to be the base point for whole model and the length of each segment is also considered constant. It is feasible to apply this approach in real time as the number of iterations for each frame stands 1.8 on an average. This shows that the model can perform faster computational complexity but as this approach takes one point to be the base point for the whole model, the approach may not be feasible for scenarios where there is a change in the position of the object.

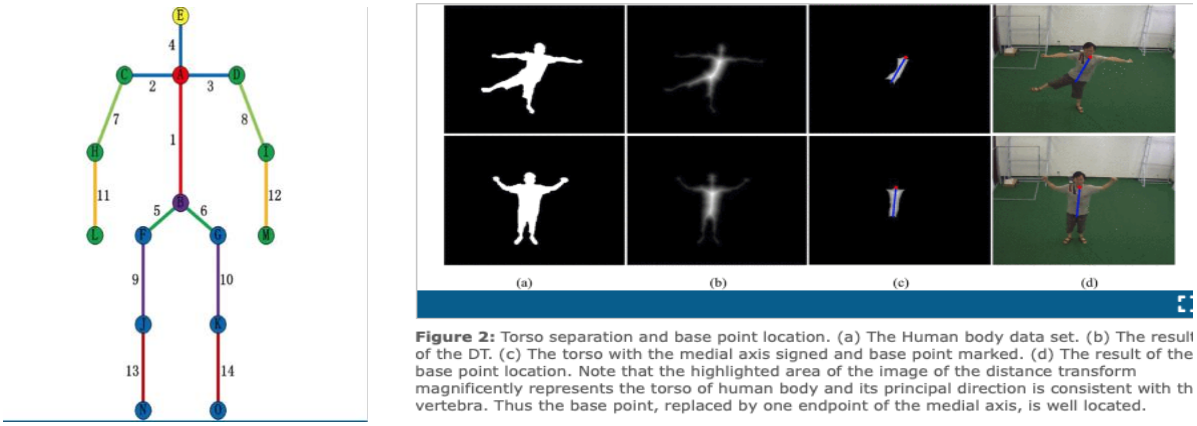


Fig. 3 Estimating 2D Human Pose using Silhouette Structure [12]

2.2.2 Deep Pose: Human Pose Estimation via Deep Neural Networks

The techniques used for Human Pose Estimation changed drastically by using Convolutional Neural Network (CNN)[13]. Here, they consider pose estimation to be a joint regression problem where the location of each body joint is regressed and then passed through 7 CNN layers-each

being a linear transformation followed by a non-linear transformation. This process is repeated to refine the predictions from previous stage. The last layer outputs 2k joint coordinates of each body joint. The main advantages of this approach are that it captures all the details about each joint and it is a much simpler method when compared to graphical models such as silhouette or usage of any heatmaps. This model doesn't require any detectors explicitly. This approach has proven that CNN can be used for both classification and localization tasks.

Figure 2. Left: schematic view of the DNN-based pose regression. We visualize the network layers with their corresponding dimensions, where convolutional layers are in blue, while fully connected ones are in green. We do not show the parameter free layers. Right: at stage s, a refining regressor is applied on a sub image to refine a prediction from the previous stage.

Fig. 4 Human Pose Estimation using CNN [13]

3 Time Series Classification on Neural Networks

3.1 Artificial Neural Network (ANN)

There has been an explosion in the amount of data being collected in the modern world. The aim of this is to make a sense of the data and the understanding of the domain and help in arriving at decisions. The collected data is subjected to data exploration for data analysis followed by identification and summarizing the main features of the data. Many times, these datapoints are needed to be visualized in real or spatial dimensions in order to make sense of the data. This is followed by feature engineering and feature selection which are used to derive better representations of the features with respect to the underlying problem. Since this process

involves a significant amount of time and effort, there is a need to develop a more efficient means to aid with the process of feature engineering. Artificial Neural Networks (ANNs) provide the means to do most of the feature engineering and feature selection by self-learning the model with respect to the target outcome.

Artificial Neural Networks derive their inspiration from the brain and the nervous system and comprise of computing nodes called neurons and network of interconnections between neurons with weights assigned to them. Convolutional Neural Network are a special kind in this which excel in Pattern Recognition tasks in images, video frames etc.

A typical Artificial Neural Network consists of an input layer, any number of hidden layers and an output layer. Input layer is where a single vector input is loaded and gets propagated to the hidden layer with a weight. The hidden layer consists of a set of neurons with each neuron connected to every other neuron (node) in the previous layer. The neurons are independent without sharing connection to any other connections within the layer. Neurons consist of a computational unit which receives input from previous layer and performs dot products of inputs with weights and adds the bias to generate the output Y [14]. A Neural Network is said to be fully connected if every node in the layer is connected to every other node in the previous layer. Depth of the network is the number of hidden layer along with the output layer. The output is passed to an activation function to add non-linearity to the model. Removing a connection in the network amounts to setting the weight of a particular connection to zero.

$$Y = \sum (\text{input} \times \text{weight}) + \text{bias}$$

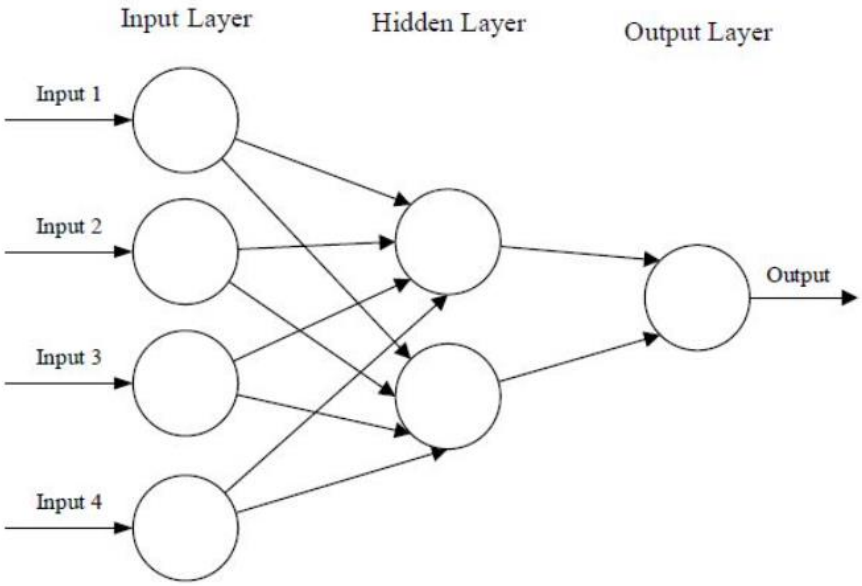


Fig. 5 Simple Neural Network [14]

One important point to note is that the Artificial Neural Networks can be quite compute hungry due to the sheer volume of data that they need to handle. Since the weights scale with each hidden layer, the number of weights can easily scale up to thousands and millions.

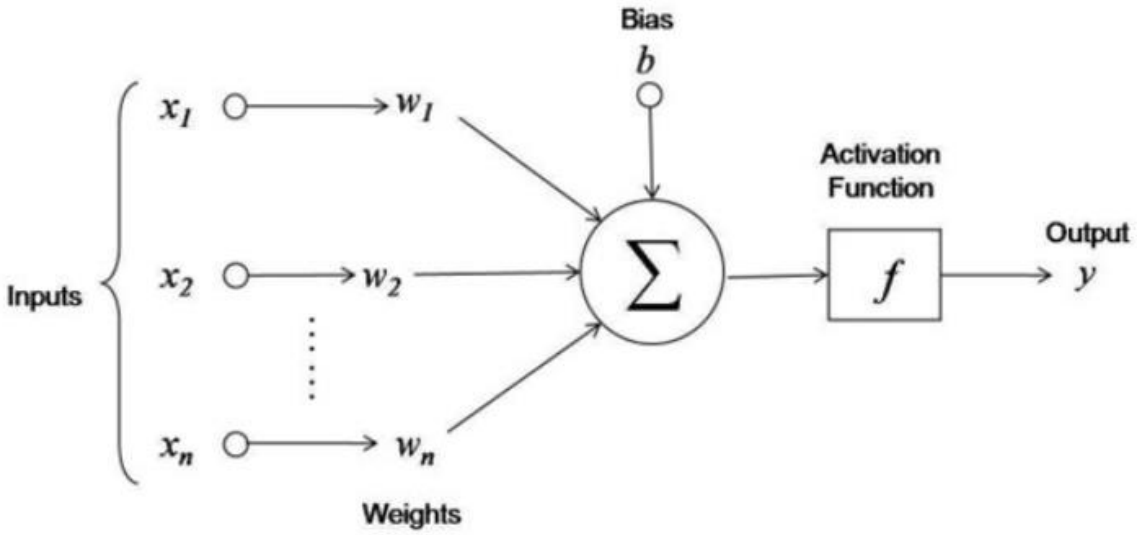


Fig. 6 A Neuron Node in an Artificial Neural Network [14]

There are 3 main types of Activation Functions which can be used to add non-linearity to the model [15]. These Activation Functions decide when the neurons show fire or not fire.

3.1.1 Sigmoid Function

A Sigmoid Function can be thought of as having the basic shape of a step function except that it has a smooth transition resulting in various gradient at various points in the function. This leads to the non-linear nature of the function. The Sigmoid Function can take inputs between $(-\infty, +\infty)$ but leads to a finite output that ranges between [0,1]. The Sigmoid Function does take more time to train the model to converge since at the bookends, the output doesn't change as quickly as the input. In spite of this, the bounded output of the Sigmoid Function fits perfectly well in giving a binary classification and makes it well suited to model the probability.

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

3.1.2 ReLU Function

The Rectified Linear Unit Function addresses the issue of large convergence training time and is quite simple and fast. It also does not have the issue of vanishing gradients. However, it is prone to the issue of knockout called dying ReLU problem when many neurons only output the value of 0. This can happen due to learning large negative bias term for weights. This issue can however be solved by employing Leaky ReLUs with a small positive gradient for negative inputs.

$$f(x) = \max(0, x)$$

3.1.3 Tanh Function

Tanh Function is quite similar to the sigmoid function with the difference being that the output can range from [-1,1] and is quite frequently used in conjunction with the Sigmoid Function. In terms of the gradients, it is important to note that the tanh function has stronger gradient than sigmoid function. Both Tanh and Sigmoid Functions are better at recovering from similar issue as dying ReLU due to the gradients.

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

3.2 Neural Networks and Time Series Classification

Time series data is the information collected successfully in time. Since processes are often measured with respect to time, temporal data has increased significantly and has many real-world applications such as detecting stock anomalies, medical tests. Time Series Classification is being

one of the challenging problems in the field of Computer Vision. Given a set of time series with class labels, we check if we can train a model to correctly predict the class of new time series [16]. As Gesture Recognition is a problem of classifying sequences of movements, using Deep Learning methods for Time Series Classification provides state-of-the-art results with little or no data feature engineering.

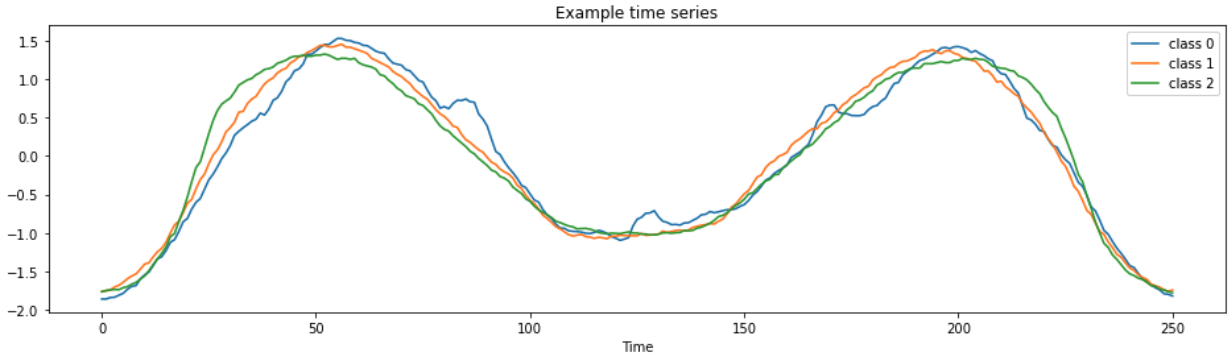


Fig. 7 Predicting Class Labels in Time Series [16]

Below are the different Neural Network models for the Time Series Classification:

3.2.1 Feed Forward Network

In Feed Forward Network, the input is fed to the network where they map the data to different categories/labels. This will eventually help to recognize the pattern in the data to identify the correct label. With supervised training, a Feed Forward Network is trained until the error is minimized. Convolutional Neural Network is a Feed Forward Network. Using CNN for Time Series Classification [17] has major advantages such as highly noise resistant, feature engineering without any manual intervention but Feed Forward Networks usually do not have time order. It can provide the prediction results just based on the current pattern it has been exposed to and doesn't remember the recent past. A trained CNN model can be exposed to any

random classification tasks, but decisions taken to classify the first image doesn't alter on how the model classifies the second image, hence CNN model is best used for spatial data more than Time Series data.

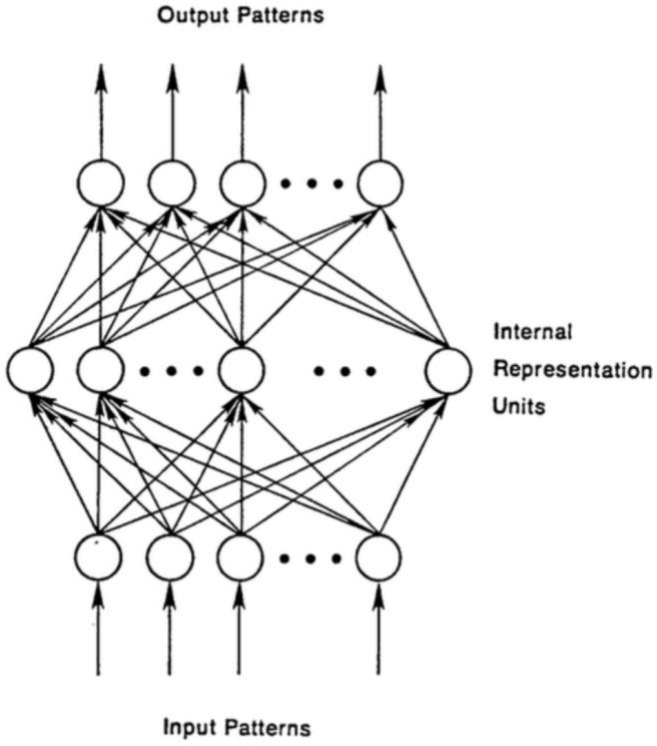


Fig. 8 Internal Representation of CNN [17]

3.2.2 Long Short-Term Memory Recurrent Neural Network

This is a type of Artificial Neural Network designed to recognize the patterns in the sequential input. RNN have memory to store temporal information available in time series data. Below is a simple example of a Recurrent Neural Network proposed by Elman [18]. Here, the box of letters labelled as "BTSXVPE" represent the current input data and the box labeled as "Context Units" shows the previous output. Both are deciding factors for the model. Hence, decision made by the network at time step (t-1) has an impact on the decision made at time step (t).

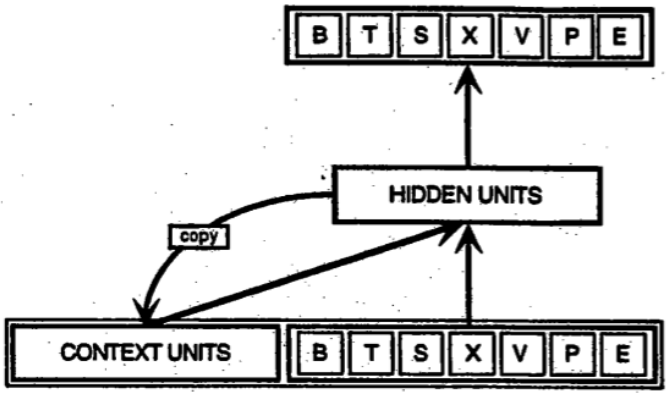


Fig.9 Concept of Long Short-Term Memory [18]

Recurrent Neural Network stand different when compared to Feed Forward Network due to this feedback loop. But due to this loop, whenever a cost function is calculated to improve the network, they need to be propagated all the way back through time to the neurons. Due to the temporal loops, same weight gets multiplied multiple times resulting to either an exploding gradient or the Vanishing Gradient [19]. Due to this problem, whole network will be not trained properly.

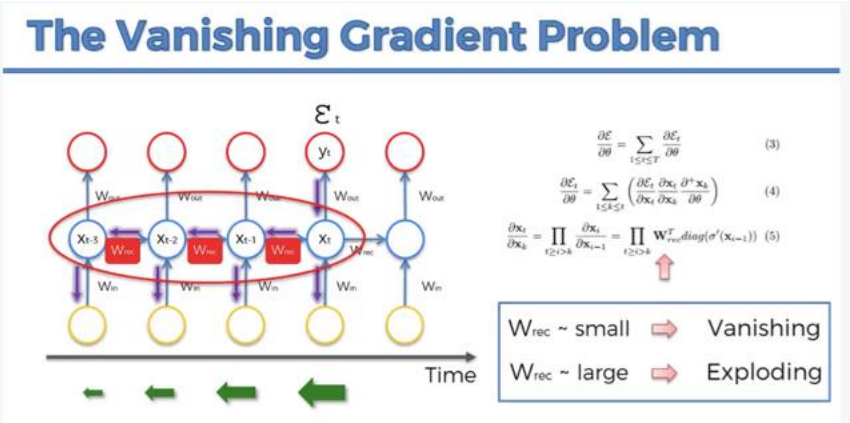


Fig.10 Vanishing Gradient Problem [19]

To overcome the problem of Vanishing Gradient, in mid-90's, LSTM (Long-Short Term Memory) Recurrent Neural Network was introduced by German researchers Sepp Hochreiter and Juergen Schmidhuber [20]. LSTMs maintain a constant error that can be back propagated through times and layers. This technique allows the network to continue to learn over many time steps without getting affected by the temporal loops.

3.2.3 Convolutional Neural Network Long Short-Term Memory Network (CNN-LSTM) Model

This approach makes use of the Convolutional Neural Network layers for feature extraction on input data and the Long-Short Term Memory to support sequence prediction [21]. Since these are both spatially and temporally deep, they can be used to sequence a variety of vision task related inputs and outputs. Especially, they can be used to address the following problems:

1. Activity Recognition – Process a sequence of images to generate textual description of the activity in the image sequence (video).
2. Image Recognition – This involves processing a single image and developing a description of the image input.
3. Video Recognition – This involves processing a sequence of images to generate a description of the video.

Also called the Long-Term Recurrent Convolutional Network (LRCN), CNN-LSTMs use a CNN pretrained for image classification problem and repurpose it for caption generating problem through feature extraction.

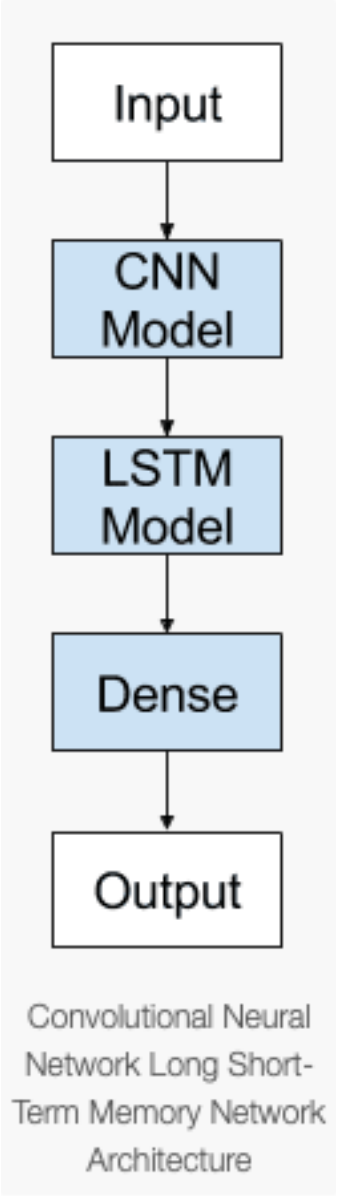


Fig. 11 Structure of CNN-LSTM Model [21]

The CNN-LSTM model is well suited for applications which involve 2D data structure or pixels in their input like images or 1D structure of words. It is also well suited for inputs that have temporal structure such as order of images in a video or words in a text. Consequently, CNN-LSTMS also find applications in speech recognition and natural language processing where CNNs are used for feature extraction for LSTMs on audio and textual input data.

3.2.4 Convolutional Long Short-Term Memory Recurrent (Conv-LSTM) Network Model

The Conv LSTM is an extension of the CNN-LSTM model and is used for spatio-temporal data. Unlike the CNN-LSTM which uses output of the CNN models for interpretation, Conv-LSTM uses convolutions directly as part of reading inputs into LSTM inputs themselves. The Conv-LSTM is a Recurrent Layer just like the LSTM. However, in Conv-LSTM, the internal matrix multiplications are replaced with convolution operations on the inputs. Hence, the data that flows through a Conv-LSTM cell keeps the input dimension (3D/2D) instead of being just a 1D vector with features. The future state of a cell in the grid is determined by inputs and past states of local neighbor cells by a process of convolution in the state-to-state and input-to-state transitions [22]

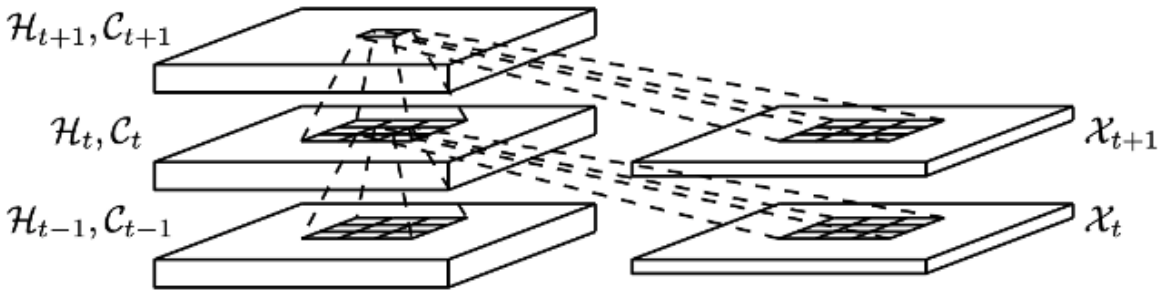


Fig. 12 Internal Working of Conv-LSTM Model [22]

A Conv-LSTM with a larger kernel should be able to capture faster motions while one with a smaller kernel would be able to capture slower motions in the spatio-temporal data. Since this needs the number of rows and columns to be the same in states and inputs, padding might be necessary before performing the convolution operation.

4 Dataset

After going through all the approaches mentioned above, we experiment on implementing a Deep Learning model combining visual features and temporal features in the data.

To train the model, we used a Gesture Commands for Robot Interaction Dataset (GRIT) Dataset created by Tsironi *et al* [23]. This is one of the popular datasets for the Gesture Recognition. This corpus contains nine Human-Robot Interaction (HRI) command gestures. The nine activities are as follows: Abort, Circle, Hello, No, Stop, Turn Right, Turn Left, and Warn. Six different subjects participated to gesture. Each of them performed the same gesture at least 10 times. A total of 543 sequences were recorded. With each of the gesture sequence being segmented and labeled with one of the nine activities.

Gesture Description is as follows:

	Gesture Name	Gesture description
1.	Abort – 57 Samples	This gesture requires the motion of the hand in front of the throat with palm facing downwards.
2.	Circle – 60 Samples	This gesture is a cyclic movement starting from the shoulder with the arm and the index finger being stretched out. The movement is taking place in front of the body of the user while he/she is facing the direction of the capturing sensor.
3.	Hello - 59 Samples	As its name denotes, it is the typical greeting gesture, with the hand waving while facing the direction of the robot. The waving starts from the elbow rotation.

4.	No - 62 Samples	To perform this gesture the arm and the index finger need to be stretched out towards the direction of the robot and then the repeating wrist rotation alternately to the right and the left.
5.	Turn Left - 62 Samples	As the name of gesture denotes, the arm is pointing to the left.
6.	Turn Right - 60 Samples	In the same sense as Turn Left, this gesture consists of the arm pointing to the right.
7.	Stop - 60 Samples	The gesture includes raising the arm in front of the body with the palm facing the robot.
8.	Turn - 63 Samples	This gesture is a cyclic movement starting with a rotation from the elbow including the simultaneous rotation of the wrist with the index finger stretched pointing downwards. The signature of this movement seems like a circle.
9.	Warn - 60 Samples	This gesture firstly requires raising the hand in front of the body having an angle between the upper and the lower arm greater than 90 degrees and then rotating the elbow while the palm is being stretched out.

Fig. 13 Gestures in GRIT Dataset [23]

5 Gesture Recognition with 2D Data

5.1 Data Pre-Processing

The structure of the dataset is that there are 9 gestures, and each gesture has approximately 60 samples and each sample has approximately 24 timesteps [60X24X9].

5.1.1 Understanding 2D (2-dimensional) points on the image

First part of the project was to check on how to represent an image as a input data to the model. We used a technique proposed by Zhe Cao[24] where we can estimate how a subject is showing gestures by identifying 17 joints of the body parts referred as “key points” as shown in the below image.

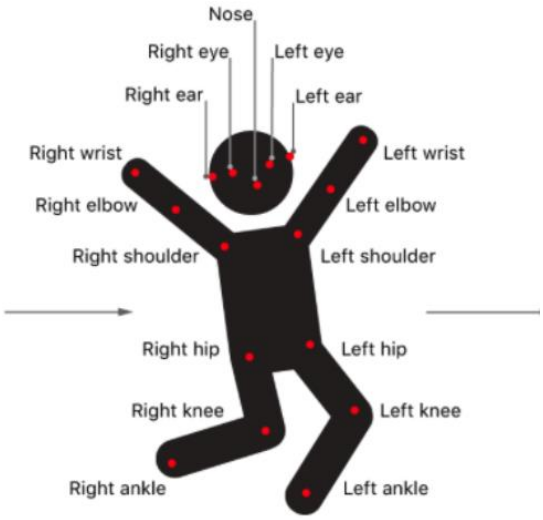


Fig. 14 Key-points of each Frame in Gesture Data [25]

The dataset has array representation of each frame. From these, we were able to extract the coordinate points and draw the ellipse points on the image as shown below:

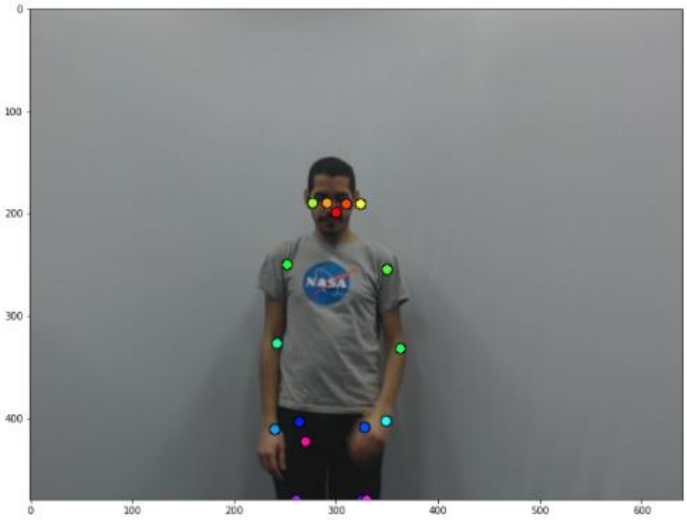


Fig. 15 Representing Key-points on a Sample Gesture Image [23]

0 [295, 194] - Nose
1 [305, 186] - left eye
2 [286, 185] - Right eye
3 [319, 186] - left ear
4 [272, 185] - Right ear
5 [345, 250] - left shoulder
6 [247, 245] - Right shoulder
7 [358, 327] - left elbow
8 [237, 322] - Right elbow
9 [344, 398] - left wrist
10 [235, 406] - Right wrist
11 [323, 404] - left hip
12 [259, 399] - Right hip
13 [320, 475] - left knee
14 [256, 475] - Right knee
15 [325, 475] - left Ankle
16 [265, 418] - Right Ankle

Fig. 16 Labels for each of the Key-points from the above sample gesture image

5.1.2 Finding the Origin for all the Key-points in a Gesture

Initially, we aimed to find the origin for all the key-points in a gesture because origin can act as a relative measure between different timesteps. In order to find the origin, it was important to understand how each ellipse point traverses in time. We performed an experiment by picking the

left elbow point and looped through different time frames. Below is the graph which shows that all the data points are referred relative to the top left. From this, we inferred that the origin is set to the top left corner.

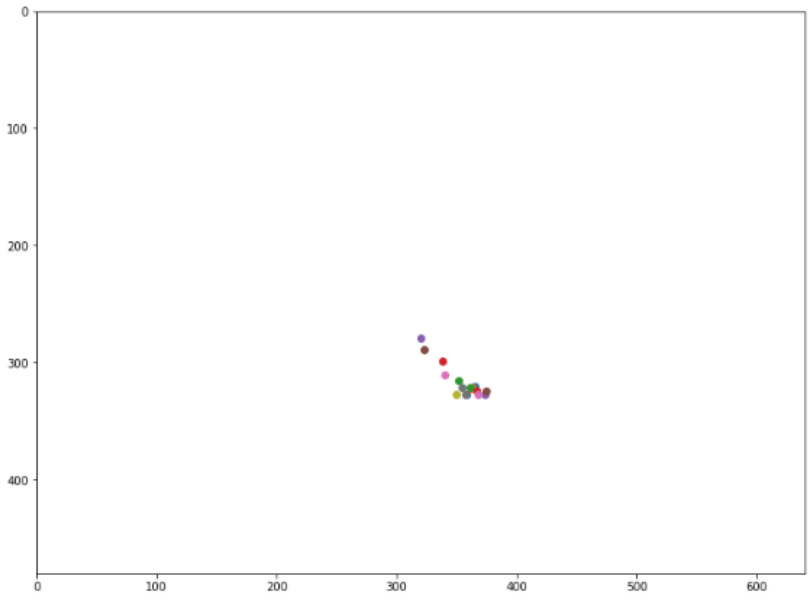


Fig. 17 Movement of Left Elbow Key-points across Different Timeframe

5.1.3 Storing Coordinates relative to a Joint position

The origin was shifted to '0' key point – Nose as origin and all other joint positions are calculated based on this origin. By making this change, though the relative measure remains the same, calculating with respect to the object position can be easy to refer back while applying complex methods.

Here, we take origin as Nose point as it remained approximately constant for all gestures and then stored the coordinates of 17 joint positions of each image frame in an array representative form.

5.1.4 Using Temporal Difference Method

Once we have the coordinates of the joint positions ready, it is important to know how to identify motion between the sequence of frames. Temporal difference [26] is a frame differencing based method where we extract the difference between two successive frames using any mathematical measures to get a relation between successive frames.

There are other methods such as optical flow which are better for dynamic environment, but they have high computation complexity with more noise data. Considering our method to work fast for dynamic and real time environment, we decided to go with temporal difference method.

5.1.5 Calculating Polar Coordinates and Angular Velocity

The movement data can be determined by velocity. Next step in the project was to check how we can relate different time steps of each gesture with accurate prediction.

As we know that, In a rectangular coordinate system, we plot points based on an ordered pair of (x,y) but in Polar Coordinate System, the ordered pair will now be (r,θ) where each point in the plane is determined by a radical distance from the reference point and polar angle(θ) from the reference direction.

Below is the graph illustrating the relationship between Polar Coordinates and Cartesian Coordinates [27]

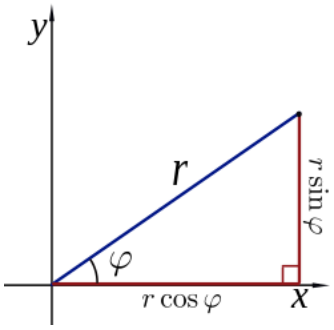


Fig. 18 Representation of Polar Coordinates

To convert from Cartesian to Polar Coordinates, we use:

Radical distance: $r = \sqrt{x^2+y^2}$

Polar Angle: $\theta = \tan^{-1}(y/x)$

Angular Velocity is the rate of change of Polar Angle of each point with respect to time. It is represented as below:

$$\text{Angular Velocity: } \omega = \frac{\delta\theta}{\delta t}$$

Below is plot for 'frame_id' (number of frames in each gesture) with respect to 'Angular Velocity'(rate of change in polar angle between successive frames) of Right shoulder, Right elbow, and Right Wrist joint positions for a sample of Abort Gesture.

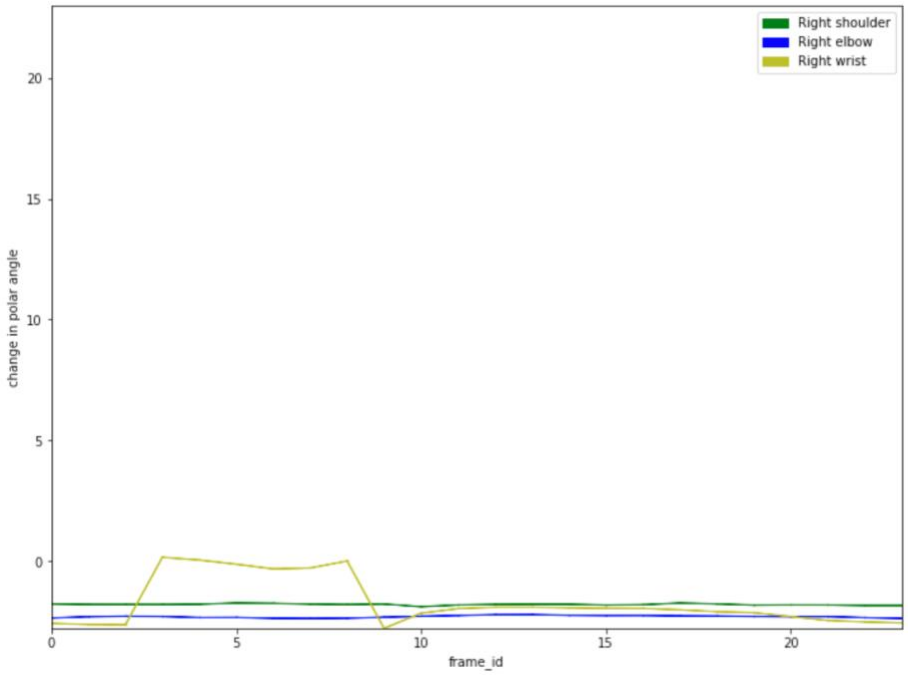


Fig. 19 Change in Joint Positions across Different Gesture Frames using Angular Velocity

We can see from the graph that the right shoulder and right elbow's positions remains almost the same, but the right wrist movement stands different from the other two. This example shows that calculating Polar Angle and Angular Velocity help to predict the gesture correctly.

Next step is to calculate and replace just the (x,y) pair of coordinates to (x,y, θ, ω) – x,y, Polar Angle and Angular Velocity for joint positions of all sequence of image from each gesture.

For our experiments, we modify values of 6 joint positions:

'left-shoulder', 'left-elbow', 'left-wrist'

'right-shoulder', 'right-elbow', 'right-wrist'

As we calculate Polar Angle and Angular Velocity for each joint positions, we get 12 features for 6 joint positions:

'left-shoulder polar angle', 'left-shoulder velocity'

'left-elbow polar angle', 'left-elbow velocity'

'left-wrist polar angle', 'left-wrist velocity'

'right-shoulder polar angle', 'right-shoulder velocity'

'right-elbow polar angle', 'right-elbow velocity'

'right-wrist polar angle', 'right-wrist velocity'

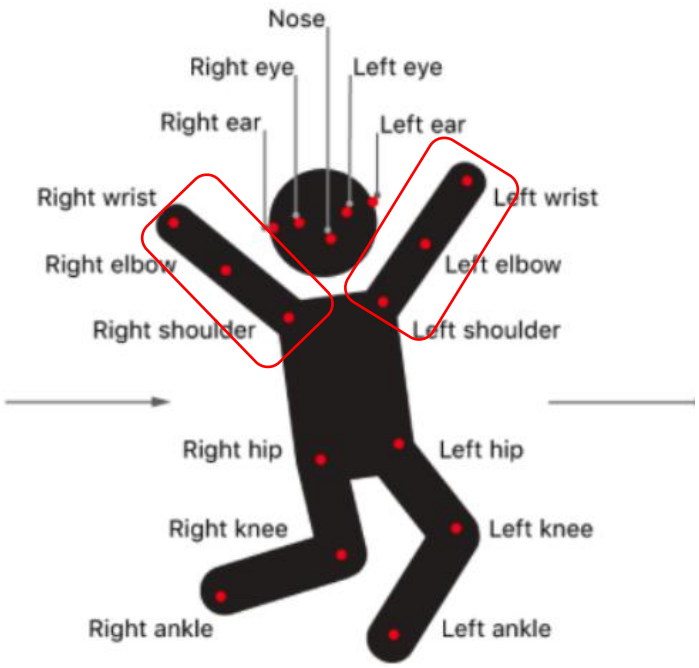


Fig. 20 Selected Joint Position (Key-points) for Experiment [25]

Frame #	Polar Angle	Change in Polar Angle (Angular Velocity)
0	1	-
1	1.5	((1.5-1) / 1) = 0.5
2	2.2	((2.2-1.5) / 1) = 0.7

3	1.8	((1.8-2.2)/1) = -0.4
4	2.9	((2.9-1.8)/1) = 1.1
5	3.3	((3.3-2.9)/1) = 0.4

5.1.6 Calculating Angular Velocity and Acceleration

Though we can approximately predict the movement of the data using Polar angle and Angular velocity, it is proven that the movement can be accurately captured if we can predict using acceleration data [28]. Hence, this next experiment is about calculating Acceleration using Angular Velocity.

'Acceleration' is the rate of change of Angular Velocity of each point with respect to time. It is represented as below:

$$a = \frac{\delta\omega}{\delta t}$$

Now, we store data as (x,y, θ , ω) – [x-coordinate, y-coordinate, Polar Angle, Angular Velocity] to (x,y, ω, a) – [x-coordinate, y-coordinate, Angular Velocity, Angular Acceleration] for joint positions of all sequence of image from each gesture.

For our experiments, we modify values of 6 joint positions:

'left-shoulder', 'left-elbow', 'left-wrist'

'right-shoulder', 'right-elbow', 'right-wrist'

As we calculate Polar Angle and Angular Velocity for each joint positions, we get 12 features for 6 joint positions:

‘left-shoulder velocity’, ‘left-shoulder acceleration’

‘left-elbow velocity’, ‘left-elbow acceleration’

‘left-wrist velocity’, ‘left-wrist acceleration’

‘right-shoulder velocity’, ‘right-shoulder acceleration’

‘right-elbow velocity’, ‘right-elbow acceleration’

‘right-wrist velocity’, ‘right-wrist acceleration’

Frame #	Polar Angle	Change in Polar Angle (Angular Velocity - ω)	Change in Angular Velocity (Acceleration - a)
0	1	-	-
1	1.5	((1.5-1) / 1) = 0.5	-
2	2.2	((2.2-1.5) / 1) = 0.7	((0.7-0.5)/1) = 0.2
3	1.8	((1.8-2.2)/1) = -0.4	((-0.4-0.7)/1) = -1.1
4	2.9	((2.9-1.8)/1) = 1.1	((1.1-(-0.4))/1) = 1.5
5	3.3	((3.3-2.9)/1) = 0.4	((0.4-1.1)/1) = -0.7

5.2 Experiments

1. Using an interactive environment called a Google Colaboratory Notebook [29]
2. Language used: Python 3.6
3. Following packages are used: sklearn, pandas, NumPy, matplotlib, Keras, Tenserflow.

5.2.1 Experiment 1 – LSTM-Only Model on 2 gestures with Polar Angle and Angular Velocity

We initially focus on using LSTM for Gesture Recognition. The main advantage of using LSTM [30] is that they can learn from raw time series data directly without any domain expertise to feature engineering.

5.2.1.1 Architectural Setup:

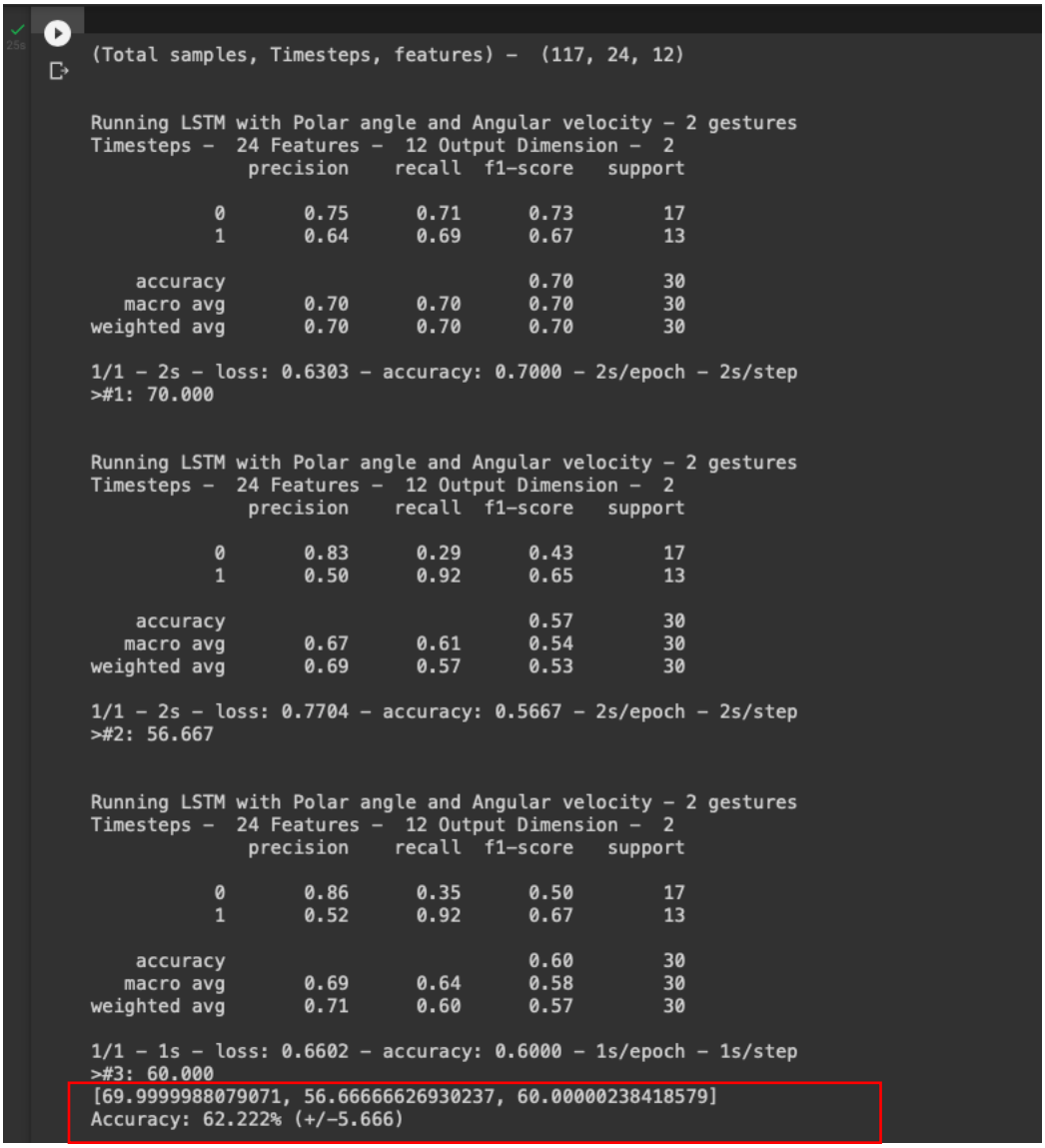
Architecture setup of the LSTM model using Kera's Deep Learning library are as follows:

Experiment on 2 Gestures 'abort' and 'stop'.

1. Input Layer: A single 3D array with input as [samples, timesteps, features (Polar Angle, Angular Velocity)]
2. Hidden Layer1: Single LSTM hidden layer.
3. Hidden Layer2: Dropout Layer with a dropout rate 0.5
4. Hidden Layer3: Dense Fully Connected Layer with an output size 100, a 'ReLU' Activation Function.
5. Output Layer: Dense Fully connected layer with two-element vector containing the probability of a given window to each of the 2 gesture types, a Softmax Activation Function.
6. Categorical_crossentropy and 'adam' stochastic gradient descent parameters to optimize the network.
7. The model is fit for 20 epochs per iteration and total sample (117) as a batch size. Batch size was not divided as input data range is small.

5.2.1.2 Results:

1. Current results are only for 'Abort' and 'Stop' gestures and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps. Abort =>57 samples and stop => 60 samples. Here, we keep the frame length to 24 as average number of frames in each sample is ~ 24.
2. Auto train/test split of the 2 gestures data with 75% for training and 25% for testing using train/test split technique.
3. Training was done with repeats – 3.
4. We got 62% average test accuracy as shown below with classification report.



```
(Total samples, Timesteps, features) -  (117, 24, 12)


Running LSTM with Polar angle and Angular velocity - 2 gestures
Timesteps -  24 Features -  12 Output Dimension -  2
              precision    recall  f1-score   support

           0       0.75      0.71      0.73        17
           1       0.64      0.69      0.67        13

    accuracy                           0.70        30
   macro avg       0.70      0.70      0.70        30
weighted avg       0.70      0.70      0.70        30

1/1 - 2s - loss: 0.6303 - accuracy: 0.7000 - 2s/epoch - 2s/step
>#1: 70.000


Running LSTM with Polar angle and Angular velocity - 2 gestures
Timesteps -  24 Features -  12 Output Dimension -  2
              precision    recall  f1-score   support

           0       0.83      0.29      0.43        17
           1       0.50      0.92      0.65        13

    accuracy                           0.57        30
   macro avg       0.67      0.61      0.54        30
weighted avg       0.69      0.57      0.53        30

1/1 - 2s - loss: 0.7704 - accuracy: 0.5667 - 2s/epoch - 2s/step
>#2: 56.667


Running LSTM with Polar angle and Angular velocity - 2 gestures
Timesteps -  24 Features -  12 Output Dimension -  2
              precision    recall  f1-score   support

           0       0.86      0.35      0.50        17
           1       0.52      0.92      0.67        13

    accuracy                           0.60        30
   macro avg       0.69      0.64      0.58        30
weighted avg       0.71      0.60      0.57        30

1/1 - 1s - loss: 0.6602 - accuracy: 0.6000 - 1s/epoch - 1s/step
>#3: 60.000
[69.9999988079071, 56.66666626930237, 60.00000238418579]
Accuracy: 62.222% (+/-5.666)
```

5.2.1.3 Observations:

1. The obtained accuracy looks good, but we cannot guarantee the correctness as the input data range is very small.
2. We can also see that the range of difference in accuracy is also high (+/- 5.66). This is due to high frequency of biased input. As the test data is less, they can be more correct predictions for one gesture than the other which gives biased results.

5.2.2 Experiment 2 – LSTM-Only Model on 2 gestures with Angular Velocity and Acceleration

5.2.2.1 Architectural Setup

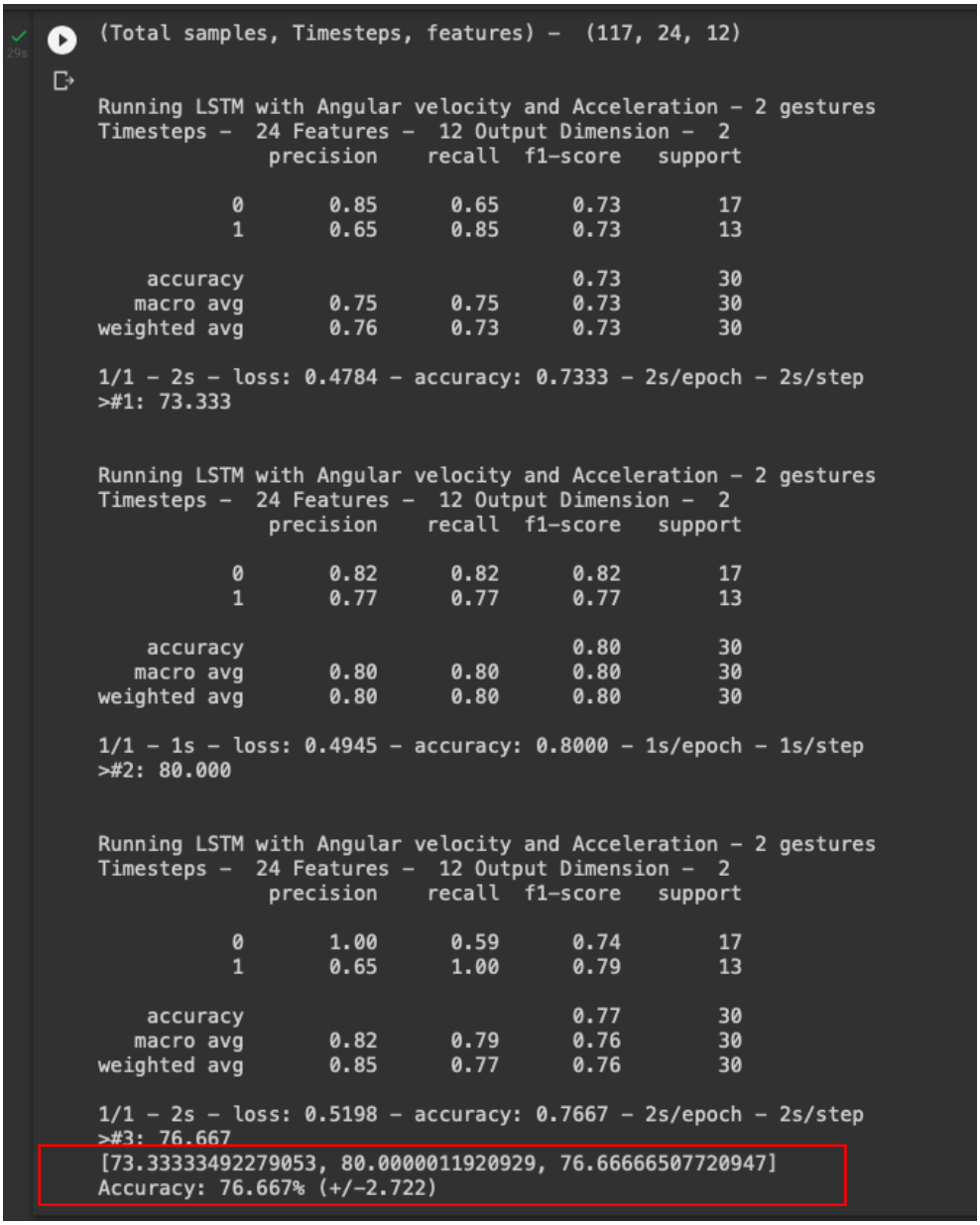
Experiment on 2 gestures ‘abort’ and ‘stop’:

1. Input Layer: A single 3D array with input as [samples, timesteps, features (Angular Velocity, Acceleration)]
2. Hidden Layer1: Single LSTM hidden layer.
3. Hidden Layer2: Dropout Layer with a dropout rate 0.5
4. Hidden Layer3: Dense Fully Connected Layer with an output size 100, a ‘ReLU’ Activation Function.
5. Output Layer: Dense Fully Connected Layer with two-element vector containing the probability of a given window to each of the 2 gesture types, a Softmax Activation Function.
6. Categorical_crossentropy and ‘adam’ stochastic gradient descent parameters to optimize the network.
7. The model is fit for 20 epochs per iteration and total sample (117) as a batch size.

5.2.2.2 Results

1. Current results are only for 'Abort' and 'Stop' gestures and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps. Abort =>57 samples and stop => 60 samples. Here, we keep the frame length to 24 as average number of frames in each sample is ~ 24.
2. Auto train/test split of the 2 gestures data with 75% for training and 25% for testing using train/test split technique.

3. Training was done with repeats – 3.
4. We got 76% average test accuracy as shown below with classification report.



```
(Total samples, Timesteps, features) -  (117, 24, 12)


Running LSTM with Angular velocity and Acceleration - 2 gestures
Timesteps -  24 Features -  12 Output Dimension -  2
              precision    recall  f1-score   support

           0       0.85      0.65      0.73        17
           1       0.65      0.85      0.73        13

    accuracy                           0.73        30
   macro avg       0.75      0.75      0.73        30
weighted avg       0.76      0.73      0.73        30

1/1 - 2s - loss: 0.4784 - accuracy: 0.7333 - 2s/epoch - 2s/step
>#1: 73.333


Running LSTM with Angular velocity and Acceleration - 2 gestures
Timesteps -  24 Features -  12 Output Dimension -  2
              precision    recall  f1-score   support

           0       0.82      0.82      0.82        17
           1       0.77      0.77      0.77        13

    accuracy                           0.80        30
   macro avg       0.80      0.80      0.80        30
weighted avg       0.80      0.80      0.80        30

1/1 - 1s - loss: 0.4945 - accuracy: 0.8000 - 1s/epoch - 1s/step
>#2: 80.000


Running LSTM with Angular velocity and Acceleration - 2 gestures
Timesteps -  24 Features -  12 Output Dimension -  2
              precision    recall  f1-score   support

           0       1.00      0.59      0.74        17
           1       0.65      1.00      0.79        13

    accuracy                           0.77        30
   macro avg       0.82      0.79      0.76        30
weighted avg       0.85      0.77      0.76        30

1/1 - 2s - loss: 0.5198 - accuracy: 0.7667 - 2s/epoch - 2s/step
>#3: 76.667
[73.33333492279053, 80.0000011920929, 76.66666507720947]
Accuracy: 76.667% (+/-2.722)
```

5.2.2.3 Observations

1. We experimented by taking the same parameters as in Experiment 1 but considered features as Angular Velocity and Acceleration instead of Polar Angle and Angular Velocity.

2. The obtained accuracy might not guarantee the correctness due to small dataset range, but the biased result is reduced by 3% when compared to our first experiment because, Angular Velocity and Acceleration are more related to each other when compared to Polar Angle and Angular Velocity.

5.2.3 Experiment 3 – LSTM-only Model on 9 gestures with Angular Velocity and Acceleration

5.2.3.1 Architectural Setup

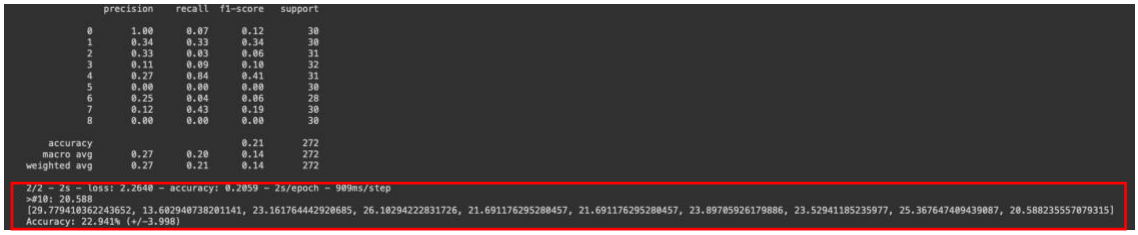
Experiment on all the 9 gestures:

1. Input Layer: A single 3D array with input as [samples, timesteps, features (Angular Velocity, Acceleration)].
2. Hidden Layer1**:** Single LSTM hidden layer (543 samples)
3. Hidden Layer2: Dropout Layer with a dropout rate 0.5
4. Hidden Layer3: Dense Fully Connected Layer with an output size 100, a ‘ReLU’ Activation Function.
5. Output Layer: Dense Fully Connected Layer with nine-element vector containing the probability of a given window to each of the 9 gesture types, a Softmax Activation Function.
6. Categorical_crossentropy and ‘adam’ stochastic gradient descent parameters to optimize the network.
7. The model is fit for 50 epochs per iteration and batch size (271). Total sample size (543) is divided into 2 batches before getting updated in network.

5.2.3.2 Results

1. Results are for 9 gestures [‘Stop’, ‘Abort’, ‘Warn’, ‘Circle’, ‘hello’, ‘turn’, ‘turn-right’, ‘turn-left’, ‘no’] and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps. Here, we keep the frame length to 24 as average number of frames in each sample is ~ 24.
2. Stratified shuffle train/test split of the 9 gestures data with 75% for training and 25% for testing.
3. Run experiments with repeats – 10
4. We got 26% average test accuracy as shown below with classification report.

```
Running LSTM with Angular velocity and Acceleration - 9 gestures
Timesteps -  24 Features -  12 Output Dimension -  9
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        30
           1       0.22      0.67      0.34        30
           2       0.00      0.00      0.00        31
           3       0.12      0.09      0.11        32
           4       0.53      0.84      0.65        31
           5       0.00      0.00      0.00        30
           6       0.56      0.18      0.27        28
           7       0.10      0.30      0.16        30
           8       0.50      0.03      0.06        30

    accuracy                           0.24       272
   macro avg       0.23      0.23      0.18       272
weighted avg       0.22      0.24      0.18       272

2/2 - 2s - loss: 2.2606 - accuracy: 0.2353 - 2s/epoch - 925ms/step
>#8: 23.529


Running LSTM with Angular velocity and Acceleration - 9 gestures
Timesteps -  24 Features -  12 Output Dimension -  9
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        30
           1       0.57      0.40      0.47        30
           2       0.14      0.03      0.05        31
           3       0.16      0.22      0.19        32
           4       0.27      0.90      0.41        31
           5       0.00      0.00      0.00        30
           6       0.75      0.43      0.55        28
           7       0.12      0.27      0.16        30
           8       0.33      0.03      0.06        30

    accuracy                           0.25       272
   macro avg       0.26      0.25      0.21       272
weighted avg       0.26      0.25      0.21       272

2/2 - 2s - loss: 2.5451 - accuracy: 0.2537 - 2s/epoch - 930ms/step
>#9: 25.368
```



5.2.3.3 Observations

LSTM is applied for all 9 gestures. We can see that the accuracy dropped drastically. This drop is accuracy is due incapability for the model to adapt for large dataset and also, we need a Convolutional Model to capture spatial features along with temporal features for each timeframe. We can also see that there are a few labels which are not predicted. This is due to large number of gestures without adequate size of dataset for LSTM only model.

5.2.4 Experiment 4 – CNN LSTM Model on 9 gestures with Angular Velocity and Acceleration

5.2.4.1 Architectural Setup

Experiment on all the 9 gestures:

1. A single 3D array with input as [samples, timesteps, features (Angular Velocity, Acceleration)].
2. Hidden Layer1**:** Time Distributed Layer to read sequence of timesteps.
3. Hidden Layer2: Flattening Layer before sending output from CNN to LSTM model
4. Hidden Layer3: Dropout Layer with a dropout rate 0.5
5. Hidden Layer4: Dense Fully Connected Layer with an output size 100, a ‘ReLU’ Activation Function.

6. Hidden Layer 5**:** One CNN layer with a max pooling layer.
7. Output Layer: Dense Fully Connected Layer with nine-element vector containing the probability of a given window to each of the 9 gesture types, a Softmax Activation Function.

5.2.4.2 Results

1. Results are for 9 gestures and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps and keeping frame length to be 24.
2. train/test split of the 9 gestures data with 75% for training and 25% for testing.
3. Run experiments with repeats – 3
4. We got 69% average test accuracy as shown below with classification report.

```
(Total samples, Timesteps, features) -  (543, 24, 12)

Running CNN-LSTM with Angular velocity and Acceleration-2D - 9 gestures
Timesteps -  24 Features -  12 Output Dimension -  9
              precision    recall  f1-score   support

           0       0.80      0.29      0.42        14
           1       0.70      0.70      0.70        20
           2       0.43      0.46      0.44        13
           3       0.65      0.87      0.74        15
           4       0.85      0.85      0.85        13
           5       0.47      0.54      0.50        13
           6       0.71      0.80      0.75        15
           7       0.57      0.86      0.69        14
           8       0.91      0.53      0.67        19

    accuracy                           0.65       136
   macro avg       0.68      0.65      0.64       136
weighted avg       0.69      0.65      0.65       136

>#1: 65.441


Running CNN-LSTM with Angular velocity and Acceleration-2D - 9 gestures
Timesteps -  24 Features -  12 Output Dimension -  9
              precision    recall  f1-score   support

           0       0.58      0.50      0.54        14
           1       0.78      0.70      0.74        20
           2       0.36      0.31      0.33        13
           3       0.79      0.73      0.76        15
           4       0.85      0.85      0.85        13
           5       0.48      0.85      0.61        13
           6       0.75      0.80      0.77        15
           7       0.74      1.00      0.85        14
           8       1.00      0.53      0.69        19

    accuracy                           0.69       136
   macro avg       0.70      0.70      0.68       136
weighted avg       0.72      0.69      0.69       136

>#2: 69.118
```

```
Running CNN-LSTM with Angular velocity and Acceleration-2D - 9 gestures
Timesteps -  24 Features -  12 Output Dimension -  9
              precision    recall  f1-score   support

           0       1.00      0.36      0.53        14
           1       0.88      0.70      0.78        20
           2       0.44      0.92      0.60        13
           3       0.85      0.73      0.79        15
           4       0.92      0.85      0.88        13
           5       0.64      0.69      0.67        13
           6       0.68      0.87      0.76        15
           7       0.81      0.93      0.87        14
           8       0.93      0.68      0.79        19

    accuracy                           0.74       136
   macro avg       0.79      0.75      0.74       136
weighted avg       0.81      0.74      0.74       136

>#3: 74.265
[65.4411792755127, 69.11764740943909, 74.26470518112183]
Accuracy: 69.608% (+/-3.619)
```

5.2.4.3 Observations

We see that the model achieved a performance of 69% with a standard deviation of about 3%. Here, CNN layer helped to extract spatial features for sequence of timesteps and then LSTM layer interprets the feature extracted. This approach gave good results but instead of running CNN over frames and send the feature sequence to LSTM, we can make the LSTM to perform convolutional operations within itself using ConvLSTM method. This method can help to correctly extract the features and reduce errors due to any numerical precisions internally.

5.2.5 Experiment 5 – ConvLSTM Model on 9 gestures with Angular Velocity and Acceleration

5.2.5.1 Architectural Setup

Experiment on all the 9 gestures:

1. Input Layer: 5D tensor with shape [samples, time, rows, cols, channels]
    i. Samples: number of samples in the dataset (543)
    ii. Time: split of the timesteps (1)
    iii. Rows: shape of each sequence (1)
    iv. Columns: 24 timesteps for input sequence (24)
    v. Channels: input features (12)
2. Hidden Layer1**:** ConvLSTM 2D Layer with two dimensional kernel size (1 row and 1 column of the timesteps)
3. Hidden Layer2: Dropout Layer with a dropout rate 0.5
4. Hidden Layer3: Flattening Layer before passing to Dense Layer

5. Hidden Layer4: Dense Fully connected Layer with an output size 100, a ‘ReLU’ Activation Function.
6. Hidden Layer 5**:** Compile Layer with loss function as categorical cross entropy and adam optimizer.
7. Output Layer: Dense Fully Connected Layer with nine-element vector containing the probability of a given window to each of the 9 gesture types and a Softmax Activation Function.

5.2.5.2 Results

1. Results are for 9 gestures and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps and keeping frame length to be 24.
2. Train/test split of the 9 gestures data with 75% for training and 25% for testing.
3. Run experiments with repeats – 3
4. We got 70% average test accuracy as shown below with classification report.

```
run_experiment()

(Total samples, Timesteps, features) -  (543, 24, 12)


Running ConvLSTM with Angular velocity and Acceleration-2D - 9 gestures
              precision    recall  f1-score   support

           0       0.50      0.71      0.59        14
           1       0.78      0.70      0.74        20
           2       0.38      0.46      0.41        13
           3       0.71      0.67      0.69        15
           4       0.92      0.85      0.88        13
           5       0.53      0.77      0.62        13
           6       1.00      0.80      0.89        15
           7       0.79      0.79      0.79        14
           8       0.82      0.47      0.60        19

    accuracy                           0.68       136
   macro avg       0.71      0.69      0.69       136
weighted avg       0.72      0.68      0.69       136

>#1: 68.382


Running ConvLSTM with Angular velocity and Acceleration-2D - 9 gestures
              precision    recall  f1-score   support

           0       0.60      0.21      0.32        14
           1       0.83      0.75      0.79        20
           2       0.50      0.62      0.55        13
           3       0.75      0.80      0.77        15
           4       0.93      1.00      0.96        13
           5       0.45      0.77      0.57        13
           6       0.92      0.80      0.86        15
           7       0.72      0.93      0.81        14
           8       0.86      0.63      0.73        19

    accuracy                           0.72       136
   macro avg       0.73      0.72      0.71       136
weighted avg       0.74      0.72      0.71       136

>#2: 72.059
```

```
Running ConvLSTM with Angular velocity and Acceleration-2D - 9 gestures
              precision    recall  f1-score   support

           0       0.57      0.57      0.57        14
           1       0.93      0.65      0.76        20
           2       0.44      0.62      0.52        13
           3       0.68      0.87      0.76        15
           4       0.79      0.85      0.81        13
           5       0.71      0.77      0.74        13
           6       0.92      0.80      0.86        15
           7       0.67      0.71      0.69        14
           8       0.87      0.68      0.76        19

    accuracy                           0.72       136
   macro avg       0.73      0.72      0.72       136
weighted avg       0.75      0.72      0.73       136

>#3: 72.059
[68.38235259056091, 72.0588207244873, 72.0588207244873]
Accuracy: 70.833% (+/-1.733)
```

5.2.5.3 Observations

In ConvLSTM model, convolutional operations take place instead of matrix multiplication within the LSTM. We can see that the accuracy improved by 1% with reduce in standard deviation by 2%.

5.2.6 Experiment 6 – LeakyReLU + ConvLSTM Model

For all the models above, we used ReLU activation function. This function is linear for all positive values and zero for all negative values. If the slope of the ReLU is negative, it ends up being zero which makes it a dead neuron. These situations occur when there are big jumps in the slope due to high learning rate. To overcome these errors, we can use LeakyRelu which sets small slopes in negative range. This makes the training process balanced and much faster.

5.2.6.1 Architectural Setup

Experiment on all the 9 gestures:

5. Input Layer: 5D tensor with shape [samples, time, rows, cols, channels]
    i. Samples: number of samples in the dataset (543)
    ii. Time: split of the timesteps (1)
    iii. Rows: shape of each sequence (1)
    iv. Columns: 24 timesteps for input sequence (24)
    v. Channels: input features (12)
6. Hidden Layer1**:** ConvLSTM 2D Layer with two-dimensional kernel size (1 row and 1 column of the timesteps)
7. Hidden Layer2: Dropout Layer with a dropout rate 0.5
8. Hidden Layer3: Flattening Layer before passing to Dense Layer

3. Hidden Layer4: Dense Fully Connected Layer with an output size 100, a 'LeakyReLU' Activation Function.
4. Hidden Layer 5**:** Compile Layer with loss function as categorical cross entropy and adam optimizer.
5. Output Layer: Dense Fully Connected Layer with nine-element vector containing the probability of a given window to each of the 9 gesture types and a Softmax Activation Function.

5.2.6.2 Results

1. Results are for 9 gestures and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps and keeping frame length to be 44.
2. Train/test split of the 9 gestures data with 75% for training and 25% for testing.
3. Run experiments with repeats – 3

```
(Total samples, Timesteps, features) -  (543, 44, 12)


Running ConvLSTM with Angular velocity and Acceleration-2D - 9 gestures
              precision    recall  f1-score   support

           0       0.62      0.36      0.45        14
           1       0.75      0.75      0.75        20
           2       0.40      0.62      0.48        13
           3       0.73      0.73      0.73        15
           4       0.93      1.00      0.96        13
           5       0.44      0.54      0.48        13
           6       0.71      0.67      0.69        15
           7       0.82      1.00      0.90        14
           8       0.92      0.58      0.71        19

    accuracy                           0.69       136
   macro avg       0.70      0.69      0.69       136
weighted avg       0.72      0.69      0.69       136

>#1: 69.118
```

```
Running ConvLSTM with Angular velocity and Acceleration-2D - 9 gestures
              precision    recall  f1-score   support

           0       0.67      0.57      0.62        14
           1       0.78      0.70      0.74        20
           2       0.50      0.62      0.55        13
           3       0.67      0.80      0.73        15
           4       0.85      0.85      0.85        13
           5       0.47      0.62      0.53        13
           6       0.85      0.73      0.79        15
           7       0.81      0.93      0.87        14
           8       0.77      0.53      0.62        19

    accuracy                           0.70       136
   macro avg       0.71      0.70      0.70       136
weighted avg       0.71      0.70      0.70       136

>#2: 69.853
```

```
Running ConvLSTM with Angular velocity and Acceleration-2D - 9 gestures
              precision    recall  f1-score   support

           0       0.55      0.43      0.48        14
           1       0.76      0.80      0.78        20
           2       0.41      0.54      0.47        13
           3       0.81      0.87      0.84        15
           4       1.00      0.92      0.96        13
           5       0.58      0.54      0.56        13
           6       0.85      0.73      0.79        15
           7       0.82      1.00      0.90        14
           8       0.82      0.74      0.78        19

    accuracy                           0.74       136
   macro avg       0.73      0.73      0.73       136
weighted avg       0.74      0.74      0.73       136

>#3: 73.529
[69.11764740943909, 69.85294222831726, 73.52941036224365]
Accuracy: 70.833% (+/-1.930)
```

5.2.6.3 Observations

Testing Accuracy of 70%. Though we do not see any major change in accuracy, this experiment was performed on 44 timesteps than the previous experiment where we used 24 timesteps. Increase in timesteps can raise noisy data as most of the timesteps will be padded with zero at the

end. Hence, even with the noise data, the model was able to produce approximately same testing accuracy.

We performed many experiments on 2D data and found that, by using CNN-LSTM and ConvLSTM models improved the accuracy. Hence, we now apply these models on 3D data to achieve real time Gesture Recognition.

6 Gesture Recognition on 3D Data

6.1 Data Pre-Processing

6.1.1 Understanding 3D points on the image

As in 2D data, we can estimate gestures by identifying 17 joints of the body parts referred as “Key-points”. Here, the position of each joint is represented by (x,y,z) coordinates. We store the coordinates of each frame relative to one of its joint positions (Nose) as in 2D data.

6.1.2 Calculating Angular velocity and Acceleration

To represent 3 coordinates(x,y,z) to calculate the movement, we use Spherical Coordinate system[31].

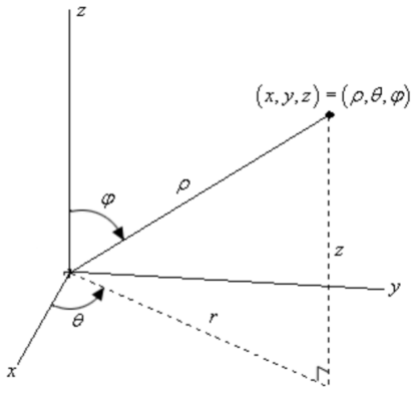


Fig. 21 Representation of Spherical Coordinates [31]

In the above figure, Point P (r, ø, θ) has distance(r) from the origin(O) to the point(p), Theta (θ) – angle measured on the XY plane and phi(ø) is an angle to rotate around the x-axis to get to the point(p). It is measured on the Z-axis. [32]

We apply the relation between Spherical Coordinates and Cartesian Coordinates as follows:

$$r = \sqrt{(x^2 + y^2 + z^2)}$$

$$\theta = \begin{cases} \tan^{-1}\left(\frac{y}{x}\right), x \neq 0 \\ \frac{\pi}{2}, \quad x = 0 \end{cases}$$

$$\phi = \begin{cases} \tan^{-1}\left(\frac{\sqrt{x^2 + y^2}}{z}\right), z \neq 0 \\ \frac{\pi}{2}, \quad z = 0 \end{cases}$$

There are many different conventions for Spherical Coordinates. The convention used here is common in mathematics. To calculate the movement in gestures, we store (r, ϕ, θ) for each joint instead of (x,y,z) coordinates.

To capture the dynamics, we consider two velocities in 3D data:

Angular Velocity(θ) is the rate of change of Polar Angle of each point with respect to time. This is same as in Polar Coordinates and is represented as below:

$$\omega = \frac{\delta\theta}{\delta t}$$

Phi Velocity (ϕ) is the rate of change of Phi Velocity of each point with respect to time.

$$\beta = \frac{\delta\phi}{\delta t}$$

Acceleration is the rate of change of Phi Velocity of each point with respect to time. It is represented as below:

$$a = \frac{\delta\beta}{\delta t}$$

For our experiments, we modify values of 6 joint positions:

‘left-shoulder’, ‘left-elbow’, ‘left-wrist’

‘right-shoulder’, ‘right-elbow’, ‘right-wrist’

As we calculate Angular Velocity and Acceleration for each joint positions, we get 18 features for 6 joint positions:

'left-shoulder velocity, 'left-shoulder phi-velocity', ‘left-shoulder acceleration’

'left-elbow polar angle’, 'left-elbow velocity', 'left-elbow acceleration'

'left-wrist polar angle', 'left-wrist velocity', 'left-wrist acceleration'

'right-shoulder polar angle’, 'right-shoulder velocity', 'right-shoulder acceleration'

'right-elbow polar angle’, 'right-elbow velocity', 'right-elbow acceleration'

'right-wrist polar angle', 'right-wrist velocity', 'right-wrist acceleration'

6.2 Experiments

6.2.1 Experiment 1 – ConvLSTM Model on 2 gestures

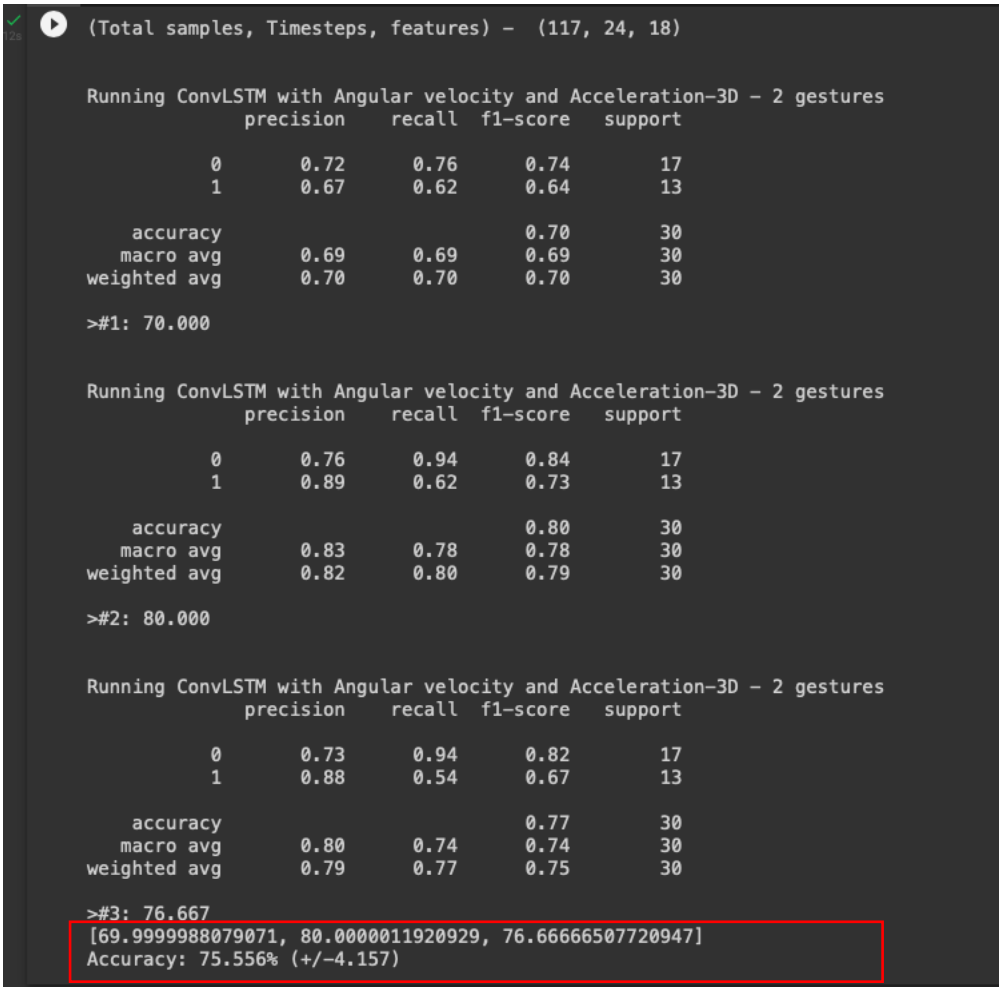
6.2.1.1 Architectural Setup

Experiment on 2 gestures:

1. Input Layer: 5D tensor with shape [samples, time, rows, cols, channels]
    i. Samples: number of samples in the dataset (543)
    ii. Time: split of the timesteps (1)
    iii. Rows: shape of each sequence (1)
    iv. Columns: 24 timesteps for input sequence (24)
    v. Channels: input features (18)
2. Hidden Layer1**:** ConvLSTM Layer with 1 filter of 11 X 11 dimension.
3. Hidden Layer3: Dropout Layer with a dropout rate 0.5
4. Hidden Layer4: Dense Fully Connected Layer with an output size 100, a 'ReLU' Activation Function.
5. Hidden Layer 5**:** One CNN Layer with a max pooling layer.
6. Output Layer: Dense Fully Connected Layer with nine-element vector containing the probability of a given window to each of the 9 gesture types, a Softmax Activation Function.
7. The model is fit for 60 epochs per iteration and total sample (60) as a batch size.

6.2.1.2 Results

1. Results are for 2 gestures for 3D data and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps and keeping frame length to be 24.
2. train/test split of the 2 gestures data with 75% for training and 25% for testing.
3. Run experiments with repeats – 3
4. We got 75% average test accuracy as shown below with classification report.



```
(Total samples, Timesteps, features) -  (117, 24, 18)


Running ConvLSTM with Angular velocity and Acceleration-3D - 2 gestures
              precision    recall  f1-score   support

           0       0.72      0.76      0.74        17
           1       0.67      0.62      0.64        13

    accuracy                           0.70        30
   macro avg       0.69      0.69      0.69        30
weighted avg       0.70      0.70      0.70        30

>#1: 70.000


Running ConvLSTM with Angular velocity and Acceleration-3D - 2 gestures
              precision    recall  f1-score   support

           0       0.76      0.94      0.84        17
           1       0.89      0.62      0.73        13

    accuracy                           0.80        30
   macro avg       0.83      0.78      0.78        30
weighted avg       0.82      0.80      0.79        30

>#2: 80.000


Running ConvLSTM with Angular velocity and Acceleration-3D - 2 gestures
              precision    recall  f1-score   support

           0       0.73      0.94      0.82        17
           1       0.88      0.54      0.67        13

    accuracy                           0.77        30
   macro avg       0.80      0.74      0.74        30
weighted avg       0.79      0.77      0.75        30

>#3: 76.667
[69.9999988079071, 80.0000011920929, 76.66666507720947]
Accuracy: 75.556% (+/-4.157)
```

6.2.1.3 Observations

Testing Accuracy is 75%. The increase in the testing accuracy is due to the addition of the 3D information in the form of Angular Velocity for both Theta and Phi Coordinates. We used the same ConvLSTM model applied for 2D data with few changes in numeric precisions and hyper parameters such as batch size, epoch, number of filters, number of steps and length of the input shape. This model is better suited to handle the spatial-temporal data and is able to make predictions better.

6.2.2 Experiment 2 – ConvLSTM Model on 9 gestures

6.2.2.1 Architectural Setup

Experiment on data of 9 gestures:

1. Input Layer: 5D tensor with shape [samples, time, rows, cols, channels]
    i. Samples: number of samples in the dataset (543)
    ii. Time: split of the timesteps (1)
    iii. Rows: shape of each sequence (1)
    iv. Columns: 24 timesteps for input sequence (30)
    v. Channels: input features (18)
2. Hidden Layer1**:** ConvLSTM Layer with 1 filter of 8 X 8 dimension.
3. Hidden Layer2: Dropout Layer with a dropout rate 0.5
4. Hidden Layer3: Flattening Layer before passing to Dense Layer
5. Hidden Layer4: Dense Fully Connected Layer with an output size 1000, a 'LeakyReLU' Activation Function.
6. Hidden Layer 5**:** Compile Layer with loss function as categorical cross entropy and adam optimizer.
7. Output Layer: Dense Fully Connected Layer with nine-element vector containing the probability of a given window to each of the 9 gesture types, a Softmax Activation Function.
6. The model is fit for 50 epochs per iteration and total sample (60) as a batch size.

6.2.2.2 Results

5. Results are for 9 gestures for 3D data and for 6 joint positions of each of the gestures as mentioned in data pre-processing steps and keeping frame length to be 30.

6. train/test split of the 9 gestures data with 75% for training and 25% for testing.
7. Run experiments with repeats – 3
8. We got ~49% average test accuracy as shown below with classification report.

6.2.2.3 Observation

Testing accuracy is ~50%. We can see that there is a drop in accuracy. This can be considered as a normal behavior because of increase in the input size for the model. We tuned the model by changing its hyperparameters like changing to 'Tanh' Activation Function and reducing the filter size. This behavior may also be due to a smaller number of datapoints for the predicting labels. Being able to reduce the noise in the data and experimenting the model with different parameters can still outperform 2D model.

```
(Total samples, Timesteps, features) -  (543, 30, 18)
30 18 9


Running ConvLSTM with Angular velocity and Acceleration-3D - 9 gestures
              precision    recall  f1-score   support

           0       0.47      0.57      0.52        14
           1       0.65      0.75      0.70        20
           2       0.26      0.46      0.33        13
           3       0.44      0.53      0.48        15
           4       0.73      0.85      0.79        13
           5       0.38      0.23      0.29        13
           6       0.64      0.47      0.54        15
           7       0.64      0.64      0.64        14
           8       0.43      0.16      0.23        19

    accuracy                           0.51       136
   macro avg       0.52      0.52      0.50       136
weighted avg       0.52      0.51      0.50       136

>#1: 51.471


Running ConvLSTM with Angular velocity and Acceleration-3D - 9 gestures
              precision    recall  f1-score   support

           0       0.54      0.50      0.52        14
           1       0.68      0.75      0.71        20
           2       0.25      0.31      0.28        13
           3       0.43      0.60      0.50        15
           4       0.69      0.69      0.69        13
           5       0.19      0.23      0.21        13
           6       0.27      0.20      0.23        15
           7       0.50      0.57      0.53        14
           8       0.50      0.21      0.30        19

    accuracy                           0.46       136
   macro avg       0.45      0.45      0.44       136
weighted avg       0.46      0.46      0.45       136

>#2: 45.588
```

```
Running ConvLSTM with Angular velocity and Acceleration-3D - 9 gestures
              precision    recall  f1-score   support

           0       0.62      0.57      0.59        14
           1       0.68      0.65      0.67        20
           2       0.29      0.31      0.30        13
           3       0.42      0.67      0.51        15
           4       0.75      0.69      0.72        13
           5       0.25      0.23      0.24        13
           6       0.50      0.47      0.48        15
           7       0.44      0.57      0.50        14
           8       0.60      0.32      0.41        19

    accuracy                           0.50       136
   macro avg       0.51      0.50      0.49       136
weighted avg       0.52      0.50      0.50       136

>#3: 50.000
[51.47058963775635, 45.588234066963196, 50.0]
Accuracy: 49.020% (+/-2.500)
```

7 ANALYSIS

For 2D data, we started with LSTM only model using Polar Angle and Angular Velocity as features in Experiment-A and we were able to achieve 62% testing accuracy with high range of deviation. In the next experiment-Experiment B, we replaced the features from Polar Angle , Angular Velocity to Angular Velocity , Acceleration as acceleration helps to capture movement data in a better way. we were able to achieve a better testing accuracy of 76%. we extended our next experiment by adding more data. Instead of 2 gestures, we calculated features for 9 gestures. In this experiment, the accuracy dropped by 50%. This is due to applying same model on the large dataset. By applying Only LSTM model in these three experiments, we came to know that it captures temporal changes in sequential data, but it is incapable of handling spatial features. To overcome this problem, in our next experiment, we combined CNN and LSTM model i.e., running Conv1D layer on 24 timesteps to capture spatial features and then sending the output to the LSTM improved testing accuracy to 69% for all the 9 gestures. By applying CNN LSTM model on the data gave good accuracy but the internal computation for CNN LSTM is complex. To overcome this problem, we experimented our data on ConvLSTM model which replaces matrix multiplication with convolutional operations without keeping CNN as a separate layer. This experiment resulted in 70% accuracy. Though the computation was easy, we couldn’t find any major changes in the results. To improve the validation accuracy, we performed next experiment to change activation function from "ReLU" to "LeakyReLU". This function helps to reduce generating high slopes for learning, thereby balancing the biased weights in the network. For this experiment, we increased noise and the data by increasing timesteps to its maximum (from 24 to 44). Even with added disturbance, the model was able to achieve 70% testing accuracy. From all these experiments on 2D data, we came to know that running ConvLSTM

model on the data yield better results. For the next experiment, we applied ConvLSTM on 2 gestures 'Abort' and 'stop' of 3D data which has 18 features instead of just 12 features. we found that the testing accuracy raised to 75%. Though the accuracy dropped with large dataset, by experimenting with calculated velocity and acceleration of 3D data, even with variable frame length and padded zeros, we were able to extract and predict the features efficiently.

8 CONCLUSION

Computer Vision [33] is a scientific discipline in the Artificial Intelligence field which trains computers to gain high level understanding from digital images or videos and has a wide range of application in various field such as Gesture Recognition, Automatic Inspections, Visual Surveillance, Object Detection, Medical Image Analysis and many more. This project explores the various existing techniques along with their limitations to arrive at the best suited technique for Gesture Recognition.

Long-Short Term Memory along with its enhancements in Convolutional Neural Network-LSTM (CNN-LSTM) and Convolutional LSTM have exhibited best accuracy for handling spatio-temporal data and hence, these techniques were further explored upon. Hyper parameter tuning such as modifying activation functions, varying filter size and appropriate data preparation was employed in order to help improve the model prediction. Experiments on both 2D & 3D data conform with our hypothesis that Conv-LSTM is better suited to handle complex data due to the convolution operations. However, it is to be noted that varying data length can be somewhat of a limitation since convolution needs the size of states to be the same as the size of the input data stream.

8.1 Future Work

Below is the scope of improvements that can be made to the model developed here to enhance its prediction accuracy.

1. Update the model to handle Angular Velocity and Acceleration for both theta and phi Polar Coordinates.
2. Reduce the noise introduced by zero padding of the input sequences due to variable frame length.
3. Improve the model to accept variable frame length for correct accuracy.
4. Enable the model to handle all joint positions instead of the 6 joints used presently.
5. Perform more experiments on the 3D joint data and better correlate the results obtained to the parameters.

References

[1] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, no. 0001-0782, pp. 84--90, 2017.

[2] M. Tomasz, "Tombone's Computer Vision Blog," 08 April 2015. [Online]. Available: http://www.computervisionblog.com/2015/04/deep-learning-vs-probabilistic.html. [Accessed 17 May 2022]

[3] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein and W. Hong, "Model-driven Data Acquisition in Sensor Networks," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, Toronto, Canada, 2004.

[4] C. Farabet, C. Couprie, L. Najman and Y. LeCun, "Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers," *CoRR*, vol. abs/1202.2160, 2012.

[5] F. F. Li, "CS231n Convolutional Neural Networks for Visual Recognition," [Online]. Available: http://cs231n.github.io/convolutional-networks/#case. [Accessed 17 May 2022]

[6] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in - *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, . DOI: 10.1109/CVPR.2001.990517.

[7] Q. Li, U. Niaz and B. Merialdo, "An improved algorithm on viola-jones object detector," in - *2012 10th International Workshop on Content-Based Multimedia Indexing (CBMI),* 2012, . DOI: 10.1109/CBMI.2012.6269796.

[8] R. B. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition,* vol. abs/1311.2524, no. CVPR '14, pp. 580--587, 2014.

[9] https://medium.com/coinmonks/review-r-cnn-object-detection-b476aba290d1 [Accessed 17 May 2022]

[10] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, 2016.

[11] Ni, Chen, J., Sang, N., Gao, C., & Liu, L. (2018). Light YOLO for High-Speed Gesture Recognition. *2018 25th IEEE International Conference on Image Processing (ICIP)*, 3099–3103. https://doi.org/10.1109/ICIP.2018.8451766

[12] Meng Li, Tao Yang, Runping Xi, & Zenggang Lin. (2009). Silhouette-Based 2D Human Pose Estimation. *2009 Fifth International Conference on Image and Graphics*, 143–148. https://doi.org/10.1109/ICIG.2009.91

[13] A. Toshev and C. Szegedy, "DeepPose: Human pose estimation via deep neural networks," in - *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, . DOI: 10.1109/CVPR.2014.214.

[14] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *CoRR*, vol. abs/1511.08458, 2015

[15] A. Sharma, "Understanding Activation Functions in Neural Networks," 30 March 2017. [Online] Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 [Accessed 17 May 2022]

[16] https://wiki.pathmind.com/lstm [Accessed 17 May 2022]

[17] https://towardsdatascience.com/how-to-use-convolutional-neural-networks-for-time-series-classification-56b1b0a07a57 [Accessed 17 May 2022]

[18] https://web.stanford.edu/group/pdplab/pdphandbook/handbookch8.html [Accessed 17 May 2022]

[19] https://www.superdatascience.com/blogs/recurrent-neural-networks-rnn-the-vanishing-gradient-problem [Accessed 17 May 2022]

[20] https://www.researchgate.net/publication/13853244_Long_Short-term_Memory [Accessed 17 May 2022]

[21] https://machinelearningmastery.com/cnn-long-short-term-memory-networks/ [Accessed 17 May 2022]

[22] https://paperswithcode.com/method/convlstm [Accessed 17 May 2022]

[23] E. Tsironi, P. Barros and S. Wermter, "Knowledge Technology Corpora," 2016. [Online]. Available: https://www2.informatik.uni-hamburg.de/wtm/datasets/ [Accessed 17 May 2022]

[24] Z. Cao *et al*, "OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 43, *(1),* pp. 172-186, 2021. . DOI: 10.1109/TPAMI.2019.2929257.

[25] https://developer.apple.com/documentation/coreml/model_integration_samples/detecting_human_body_poses_in_an_image [Accessed 17 May 2022]

[26] S. Wei, Z. Chen, M. Li and L. Zhuo, "An Improved Method of Motion Detection Based on Temporal Difference," in *2009 International Workshop on Intelligent Systems and Applications*, 2009

[27] https://en.wikipedia.org/wiki/Polar_coordinate_system [Accessed 17 May 2022]

[28] https://www.researchgate.net/publication/221601229_Gesture_Recognition_with_a_3-D_Accelerometer [Accessed 17 May 2022]

[29] https://colab.research.google.com/ [Accessed 17 May 2022]

[30] https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/ [Accessed 17 May 2022]

[31] https://mathinsight.org/spherical_coordinates [Accessed 17 May 2022]

[32] https://tutorial.math.lamar.edu/classes/calciii/SphericalCoords.aspx [Accessed 17 May 2022]

[33] https://en.wikipedia.org/wiki/Computer_vision#Applications [Accessed 17 May 2022]