SKELETAL ANALYSIS USING COMPUTER VISION AND DEEP LEARNING TECHNIQUES TO DETECT STROKE

A PROJECT REPORT

Submitted by

CB.EN.U4CCE19002	**ADARSH SINGH**
CB.EN.U4 CCE19004	**ADITHYA AS**
CB.EN.U4 CCE19011	**ASHWIN G**
CB.EN.U4 CCE19038	**NIKHIL A**
CB.EN.U4 CCE19045	**ROHAN C**

Under the guidance of

Dr. SANTOSH KUMAR C

in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER AND COMMUNICATION ENGINEERING

AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE 641112

May 2023

AMRITA SCHOOL OF ENGINEERING
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE 641112

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**SKELETAL ANALYSIS USING COMPUTER VISION AND DEEP LEARNING TECHNIQUES TO DETECT STROKE",**

submitted by

CB.EN.U4CCE19002	**ADARSH SINGH**
CB.EN.U4CCE19004	**ADITHYA AS**
CB.EN.U4CCE19011	**ASHWIN G**
CB.EN.U4CCE19038	**NIKHIL A**
CB.EN.U4CCE19045	**ROHAN C**

In partial fulfillment of the requirements for the award of the **Degree of Bachelo**r **of Technology** in **COMPUTER AND COMMUNICATION ENGINEERING,** is a bonafide record of the work carried out under my guidance and supervision at the Amrita School of Engineering, Coimbatore.

Project Advisor
Name: Dr Santosh kumar C
Designation: Professor

Project Coordinator
Name:Sudheesh P
Designation: Assistant Professor

Chairman
Dept. of Elec. & Commn. Engineering

The project was evaluated by us on:

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

This project has been completed and successful owing to the persistent support and guidance from our project supervisor, Dr. C Santosh Kumar and Research Scholar, Mr. Sreekumar KT during the course of our Project. We would also like to extend our thanks to the Chairperson of ECE Department, Dr. Madhu Mohan N, for providing us with the opportunity to dedicate ourselves for this project.

We owe a big thanks to our Project review panel members, Mr. Sudheesh P and Dr. Rolant Gini J to have been patient all the way and also to have provided us with the best ideas to improve the project through continuous feedbacks.

Lastly, we would like to thank our parents and friends to have been kind and supportive in this journey.

ABSTRACT

The recent surge in demand and advancements in artificial intelligence has enabled computers to solve many real-life problems, such as face recognition, speech recognition, etc. The potential for artificial intelligence in health care is vast, as AI increases the ability of healthcare professionals to better understand day-to-day patterns and provide better feedback. This principle can be utilized to detect diseases that need the expertise of an expert clinician to detect motor impairment diseases such as stroke and cerebral palsy. Early diagnosis and treatment of cerebral palsy help significantly improve the quality of the life of the child to near normal.

Caregivers can record their child utilizing our mobile application to get feedback, if the model finds any symptoms of motor-impaired diseases, the video can be then sent to the neonatal specialists for further analysis. Before the emergence of AI, this scenario could only be identified by trained neurologists who could make informed decisions on the motor-impaired child. This model aims to address these motor-impaired children who do not have access to skilled neurologists.

We extracted the skeletal structure from the video data that contains the semantic pose points of different joints in the body. This information is used to track the movement of the joints. Time domain analysis and frequency domain analysis were performed to identify distinguishing characteristics between normal and motor-impaired persons. The data collected was used to train the model using machine learning and deep learning models like SVM, LSTM, and CNN, the performance of these algorithms was compared and evaluated to find the most efficient algorithm to detect patients affected by motor-impaired diseases

TABLE OF CONTENTS

LIST OF SYMBOLS

SYMBOL	DESCRIPTION	PAGE NO
%	percentage	
σ	standard deviation	22
Σ	Summation	22
μ	mean (average)	22
X_i	i'th Coordinate value	22
Y(x)	Input class	18
b	Bias term	18
h(x,y)	Filter	24
f(i,j)	Input data	24
g(i,j)	Output of convolution layer	24
k	Size of pooling window	25

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION	PAGE NO
2D	2 Dimension	35
3D	3 Dimension	6
AI	Artificial Intelligence	i
AR	Augmented Reality	7
BPTT	backpropagation through time	30
CNN	Convolutional Neural Networks	3
COCO	Common Objects in Context	6
COVID 19	Corona Virus Disease of 2019	16
CSV	Comma Separated Values	49
FFT	Fast Fourier Transform	39
HCI	Human-Computer Interaction	7
IOS	iPhone Operating System	13
LSTM	Long Short-Term Memory	I
MPII	Multi-Person Pose Estimation	6
MRI	Magnetic resonance imaging	56
PTSD	post-traumatic stress disorder	56
R-CNN	regions with convolutional neural networks	12
RBF	Radial Basis Function	20
RLU	Rectifies Linear Unit	27
RMS	root mean square	44
RNN	recurrent neural network	27
SVM	Support Vector Machine	37

TS	Tensor flow	15
TSN	Temporal Segment Network	9
VR	Virtual Reality	7
WHOP	World Health Organization	2
YOLO	You only look once	12

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1. INTRODUCTION

Stroke is a neurological disorder caused by the intervention of blood supply to the brain. It affects people regardless of their age, making it a leading cause of disability and death worldwide. However, detecting stroke in infants and young kids is a challenge due to the lack of communication skills and observation skills in them. In this project, we aim to explore the use of Computer Vision and Deep Learning techniques for stroke detection in infants and young children.

1.1 BACKGROUND AND MOTIVATION

The World Health Organization (WHO) says in its reports that stroke is the second major cause of death worldwide, responsible for 11.8% of all demises.[18] Stroke is a medical emergency that requires immediate care as it can lead to permanent brain damage and disability. The symptoms of stroke include abrupt onset of physical behavior in a person such as unilateral weakness or numbness, confusion, slurred speech, and severe headaches.

While it is frequently associated with the elderly, it can also affect infants and young children, with causes such as cardiovascular disease, sickle cell anemia, and blood clotting disorders differing from those which affects the adults. Detecting stroke in children can be especially difficult due to the lack of evident symptoms and communication between the patient and the doctor.[20] The symptoms of stroke in children can be subtle and difficult to detect, which often leads to delayed diagnosis and treatment.

Even with advancements in technologies which help in diagnosis of neurological disorders, these technologies are found in rarity in areas which have not had the privilege of any such development. Parents and family members find difficulty in approaching any kind of medical consulting, given the distance they have to travel and the longer waiting times to meet with the professionals.

Delay in diagnosis and treatment of stroke in children can lead to severe consequences, including permanent brain damage, disability, and death. Stroke also can affect the mental health of the family members, especially the parents. Parents of children who have suffered a stroke may experience anxiety, depression, and post-traumatic stress disorder (PTSD).

Moreover, they may face challenges in caring for their child, including financial strain, emotional distress, and social isolation.

1.2 PROBLEM STATEMENT

The difficulty in detecting neurological disorders in infants and young children is a critical problem and finding a solution for it is the need of the hour. Currently, no specific diagnostic tool or method has been developed to detect stroke in children, which often leads to delayed diagnosis and treatment. The lack of visible symptoms and the inability of children to communicate their symptoms make it challenging for doctors to diagnose stroke in children. Moreover, the diagnostic process involves invasive procedures such as MRI, which may not be suggested for infants and young children.

Late diagnosis and treatment of stroke in infants and kids can have severe consequences, including permanent brain damage, disability, and losing lives. Early diagnosis and treatment of stroke helps in significantly improving the prognosis and reducing the risk of long-term ailments. Therefore, there is a need for an accurate and non-invasive diagnostic tool that can detect stroke in infants and young children.

In this project, we aim to explore the use of the various available libraries in computer vision and deep learning domains for detecting stroke in infants and young children. Mediapipe is an open-source library developed by Google that detects the skeletal pose points of a human being with the help of a simple camera feed.[19] CNN is a deep learning architecture which used for image recognition tasks. By combining the mediapipe and CNN, we aim to develop a tool for stroke detection in infants and kids.

In conclusion, detecting stroke in infants and young children is a crucial problem that needs to be resolved. Delay in diagnosis and treatment of stroke in children can have critical consequences, including permanent brain damage, disability, and also loss of life. The lack of visible symptoms and the communication skills make this a challenging task for neurological professionals to diagnose stroke in infants and kids. Therefore, there is a need for an accurate and non-invasive diagnostic tool that can detect stroke in children. In this project, we aim to explore the use of the mediapipe library and CNN for stroke detection in children, which can help in improving the diagnosis and treatment of stroke, providing relief to the patients as well as their family. This project can also contribute to the development of new technologies and methods for diagnosing neurological disorders in children. Ultimately, the development of a diagnostic tool for stroke detection in children can have a great impact on public health, improving the lives of the kids and their families.

CHAPTER 2
LITERATURE REVIEW

2. LITERATURE REVIEW

Pose estimation is the process of estimating the 3D pose of an object or person from a 2D image or video. It has become a critical component of many applications in various fields due to its ability to provide accurate and real-time estimation of an object's position and orientation. In this literature survey, we provide an overview of the application of pose estimation in various fields.

2.1 Overview of pose estimation and its various models

Pose estimation involves detecting key points, also known as landmarks or joints, in the human body or objects, and estimating the relative positions and orientations of these points. Different key points can be opted from the skeletal structure based on the purpose of the application and the type of pose estimation model used. For example, the pose points can be the joints of a human being and their facial parts if the model has to detect a person, whereas it can be edges and corners in cases of object detection.

There are many pose estimation models which differ in specifications such as speed, complexity and accuracy. OpenPose is one of the popular pose estimation models founded by Carnegie Mellon University in 2017. OpenPose uses CNN to locate the key points and estimate the pose of a person, operating with a real-time, multi-modal and multi-person system. OpenPose has achieved state-of-the-art performance on several benchmarks, such as the COCO dataset.

Alpha Pose is another popular pose estimation model that was introduced in 2018 by the Chinese Academy of Sciences. Alpha Pose is also a real-time, multi-person, and multi-modal pose estimation system that uses a CNN to detect key points and estimate the pose of humans. Alpha Pose has achieved high accuracy and efficiency on several benchmarks, such as the MPII and COCO datasets.

Mediapipe is a framework developed by Google that provides various computer vision and machine learning components, including pose estimation. Mediapipe' s pose estimation module uses a CNN to detect key points and estimate the pose of humans or animals. Mediapipe also provides several additional features, such as face detection and tracking, hand tracking, and object detection. [21]

2.2 Application of Pose Estimation in Various Models

Robotics:

Robots might have to understand the environment in which they are present and interact with it, making pose estimation a necessary task for them. Object recognition, manipulation, grasping and navigation are some of the applications in which pose estimation is required in robotics. Pose estimation helps robots to perfectly locate and grasp objects, maneuver in complex environments, and perform manipulation tasks.

Augmented Reality (AR):

Digital information is overlaid onto the real word using Augmented Reality. Pose estimation is important for AR, as the 3D pose of the camera is estimated to overlay digital content onto the real world. Gaming, education, training and marketing are some of the applications of AR in which pose estimation is put to use. [22]

Virtual Reality (VR):

Another emerging technology is virtual reality which creates an immersive virtual environment. The user's hand and head movement are tracked using pose estimation in VR, ensuring close to reality interaction with the virtual environment. Pose estimation in VR is used in various applications, such as gaming, simulation, training, and therapy.[23]

Human-Computer Interaction (HCI):

The learning process of interaction between human and computers is called Human-Computer Interaction. To enable natural and intuitive interactions with the computer, pose estimation is used in HCI. Various applications in HCI such as sign language recognition, facial expression recognition and gesture recognition use pose estimation.

Sports:

Performance of athletes of various sports can be analyzed using pose estimation. It can track body movements such as running, throwing, jumping and many more, and provide feedback on the same. Pose estimation is used in various sports, such as athletics, gymnastics, and dance.

Medical Imaging:

Movement of internal organs and bones can be analyzed using pose estimation in the medical field. Pose estimation can be used to track the movement of organs during surgery or to monitor the progress of bone healing. Pose estimation is used in various medical applications, such as surgical planning, physical therapy, and rehabilitation.

2.3 Real Time Human Action Recognition

Real-time human action recognition is an important and challenging task in computer vision. It involves recognizing human actions from video streams in real-time, which has a wide range of applications, including surveillance systems, human-robot interaction, and gaming, among others. Traditional computer vision methods have been limited in their ability to recognize complex human actions in real-time due to the high variability in human actions and the need for real-time performance. However, latest advancements in deep learning techniques such as Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM), along with pose estimation techniques, have shown appreciable results in real-time human action recognition.[24]

Image and video recognition tasks, which also tracks the human action employ CNNs. This approach typically uses a 3D CNN architecture, which takes in a series of frames as input and extracts features such as facial and temporal ones. The classifier takes the input of the extracted features to recognize human action. A two-stream CNN is an example of such an approach which uses both spatial and temporal features for capturing appearance and motion information. The spatial stream processes each frame independently, while the temporal stream processes the motion between frames.

LSTM networks are another popular deep learning technique for human action recognition. LSTM networks are designed to handle sequential data and can capture long-term dependencies between frames in a video sequence. Spatiotemporal feature representation of video or image data is used in LSTM-based approaches to classify different human actions. Temporal Segment Network (TSN) is one such example. It divides the video sequence into several segments and extracts features from each segment. The extracted features are then fed into the LSTM network to classify the human action.

3D pose-based action recognition is another example of a pose-based approach. It estimates the 3D poses of human body parts from the video sequence. Human action is recognized by feeding the estimated 3D poses into a classifier.[25] 3D pose-based approaches can capture temporal and spatial information from video sequences.

More sophisticated models for real-time human action recognition have been developed with the help of recent advancements in deep learning techniques. Convolutional LSTM is one such model which combines the spatiotemporal information in a video sequence. This network consists of a series of convolutional and LSTM layers that are connected to form a spatiotemporal network. This network can learn spatial and temporal dependencies simultaneously and can handle variable-length input sequences.

Despite the promising results that deep learning and pose estimation techniques provide in real-time human action recognition, several challenges are yet to be addressed. One challenge is the lack of labelled training data for complex human actions, which makes it difficult to train deep learning models. Another challenge is the variability in human actions, such as variations in clothing, lighting, and camera angles, which can affect the performance of the models. Furthermore, the computational complexity of deep learning models can make real-time human action recognition challenging on resource-limited devices.

In conclusion, real-time human action recognition is an important and challenging task in computer vision with numerous applications in various fields. Recent advances in deep learning techniques, such as CNNs, LSTM networks, and pose estimation, have shown promising results in real-time human action recognition. However, there are still several challenges that need to be addressed to improve the performance of these models in real-world scenarios. Future research in this area should focus on developing more robust and efficient models that can handle complex human actions in real-time with limited computational resources.

CHAPTER 3
METHODOLGY

3. METHODOLOGY

This chapter discusses the proposed approach for pose estimation as well as the various algorithms utilized in creating and improving the performance of a model which distinguishes between a person affected by stroke and a healthy individual.

3.1 Pose Estimation Approach

The increasing prevalence of technology has amplified the demand for artificial intelligence. One of the prominent areas where AI stands out is the potential AI has on healthcare, hence by leveraging certain technologies and frameworks like mediapipe, yolo4, and LSTM, we can assist the healthcare professionals by providing a way to identify whether a patient has a stroke or not with 3D skeletonization. This paper proposes the integration of mediapipe framework which uses an algorithm similar to R-CNN architecture with long short-term memory (LSTM) and recurrent neural network for processing the video. We have also explored various other frameworks like yolo4 and chose mediapipe for integration due to its versatility. The extracted landmarks from skeletal data and other features are directed into the LSTM module to generate materialistic features (training).

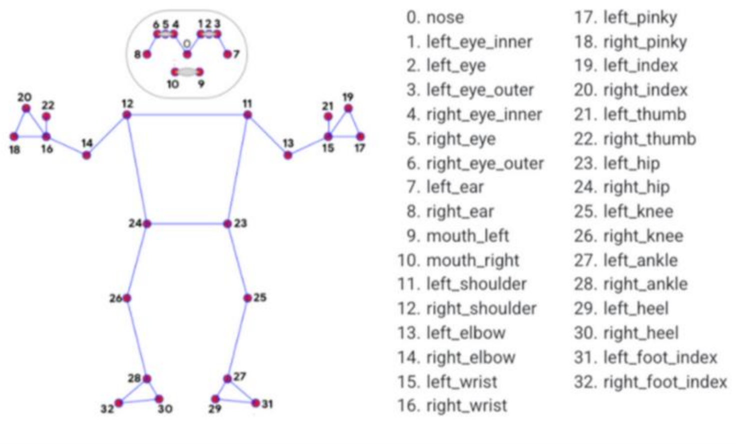


Fig 3.1 Semantic key points detected by mediapipe

Extremely valuable features are captured in each frame of video, we used 30 frames/second. Finally, the movements are detected from the information feature with the aid of mediapipe and other software modules. Mediapipe permits us to access pre-trained

Deep Learning models for Audio, Image, and Text processing. These pre-trained models provided by Mediapipe are trained on large datasets using state-of-the-art techniques. This model considers the gait of human to detect stroke and no stroke. Mediapipe is an extremely significant pose estimation framework that poses as a platform pipeline framework [15]. The working of mediapipe is quite straightforward, in our case mediapipe is used to track the semantic key points of a patient. This is known as holistic detection, if there are any inconsistencies with the distance between certain or subsequent nodes (semantic key points), we can conclude that the respective patient has stroke. If there are no inconsistencies present we can safely conclude that the patient is healthy. Mediapipe constructs custom machine-learning solutions for streaming as well as live media. Mediapipe works by tracing semantic key points in objects or humans in order to perform pose estimation [15]. It is also worth noting that mediapipe can be used for various other purposes such as object detection and tracking, face detection, hair segmentation, face mesh and hand tracking as all of these methods require tracing semantic key points from the concerned individual or object.

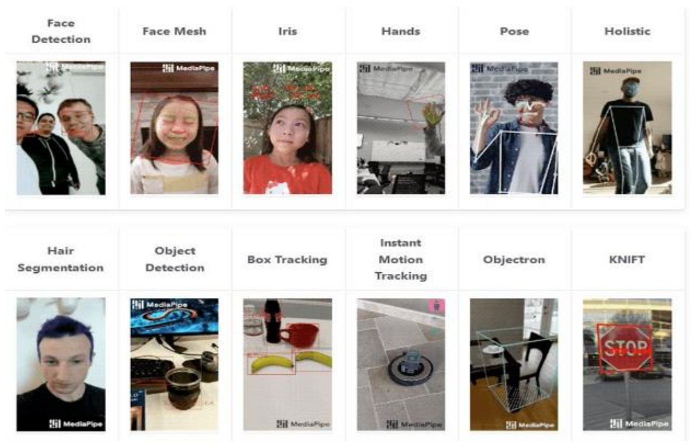


Fig 3.2 mediapipe used for various applications

Mediapipe is commonly used as it manages resources like CPU and GPU efficiently to attain low latency performance, mediapipe also takes care of the synchronization of time-stamped data (these include video frames and audio). Mediapipe is also highly suitable for various platforms (IOS, Android, IOT), hence users can deploy anywhere as interoperability and unity is present. Mediapipe is also open source and free as the framework of mediapipe is licensed under Apache 2.0 which is extremely customizable. The main components of mediapipe include the packet, graph, nodes as well streams. The basic data flow unit is regarded as a packet, the packet is composed of a numeric timestamp as well as a shared pointer to an immutable payload [15]. The subsequent processing takes place inside graphs, graphs designate the flow paths of packets between nodes. A graph can possess any number of inputs or outputs and split or merge data.

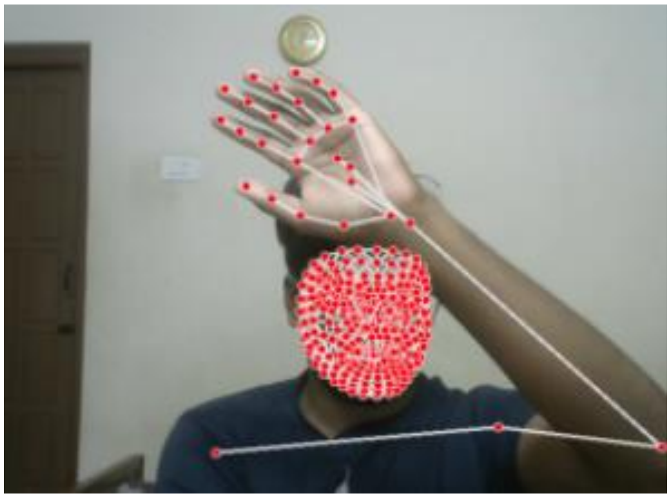


Fig 3.3 Graph of hand detection (collection of nodes and streams are present)

Nodes are referred to the semantic key points; in the above example the points are considered as the nodes. Nodes consume and produce packets, each node or semantic key point defines a certain number of input as well as output ports. Streams are regarded as the connections between subsequent nodes, they carry the respective packets or sequence of packets with increasing timestamps. In the mediapipe infrastructure, the pipeline can be refined drastically by inserting or removing nodes anywhere in the graph.

3.1.1 Reason for Choosing Mediapipe

Choices available: YOLOv7 Pose

About Mediapipe:

In various applications, including understanding sign language, measuring physical activity and even controlling full-body gestures, human position estimation from video is quintessential. It serves as the foundation for exercises like fitness, yoga, and even dancing. Also, it permits augmented reality to layer digital information and material on top of the real environment. Numerous software-based approaches for pose detection have been introduced for this application. These include but are not limited to: YOLOv7 Pose, Alpha Pose, Apple Vision and, TF.js.

For our application i.e., stroke detection using skeletal analysis, we have used mediapipe over other frameworks due following reasons.

Mediapipe vs YOLOv7:

YOLO v7 Pose has 17 key points [16] COCO. Whereas Mediapipe Pose has 33 key points COCO. In addition to that mediapipe also has palm and face detection. YOLO can detect multiple persons and mediapipe can detect only one at a time, this brings in a plus point to our system by using mediapipe. Since our system deals with detection of stroke for a single user, there’s no need for detection for multiple users in a single frame. In turn if detection for multiple users is done in a single frame, this would not even increase the complexity of the system substantially, but also decrease the accuracy of our system to a great extent.

There are many reasons why we choose mediapipe over YOLO in our project. Few are as follows:

- **a. FPS**: On comparison on default settings we can see that Mediapipe has more frames per second than YOLOv7 [16].

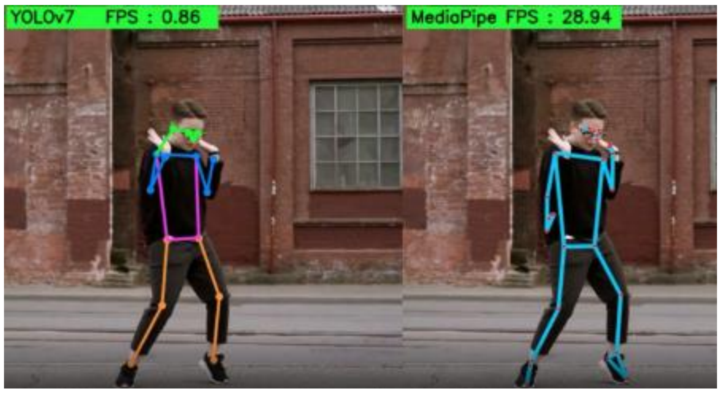


Fig 3.4 Difference in FPS- YOLOv7 vs MediaPipe

b. **Low light conditions:** The following comparison shows YOLOv7 and mediapipe comparison in low light conditions. Clearly mediapipe outperforms YOLOv7 in in all aspects, be it fps or pose recognition [16].

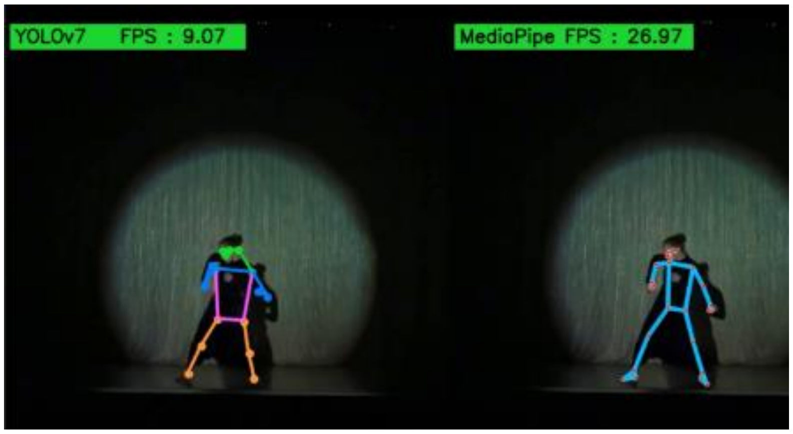


Fig 3.5 Low light comparison of YOLOv7 vs MediaPipe

c. **Far away detection:** In conditions, where our subject is far away from camera mediapipe performs decently well than compared to YOLOv7. While YOLOv7 is unable to detect the person [16], Mediapipe is not only able to detect the person but also with decent accuracy. This can be a great feature in our model in times like COVID-19 where the subject is expected to maintain social distancing.

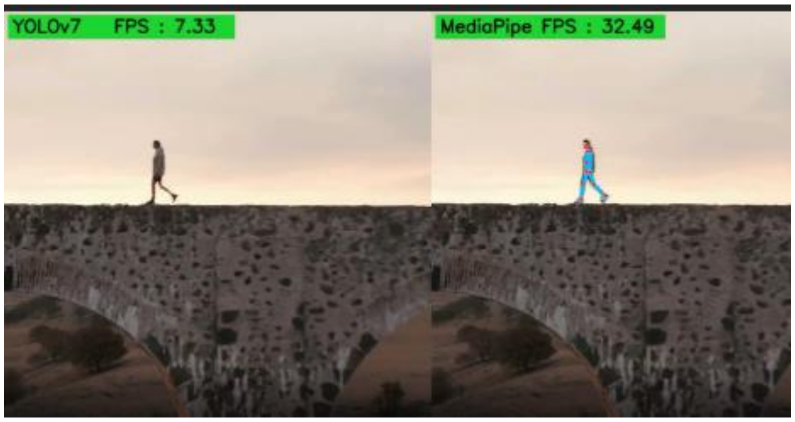


Fig 3.6 Faraway away detection – YOLOv7 vs MediaPipe

d. **Yoga Posture detection:** Mediapipe performs substantially well when it comes to Yoga pose detection. On the other hand, YOLOv7 struggles in detecting the pose when it comes to yoga poses. Since our model detection is no different than yoga pose estimation and mediapipe performs well in this case, this is another reason why we choose mediapipe over any other framework.

3.2 SUPPORT VECTOR MACHINES

A Support Vector Machine (SVM) is a strong and generally involved regulated learning algorithm for grouping, relapse, and exception identification. A discriminative classifier learns a boundary (hyperplane) between classes that expands the edge between them. Along these lines, SVM is a boundary-based strategy, meaning it centers around the boundary of the classes instead of on the circulation of the data focuses.

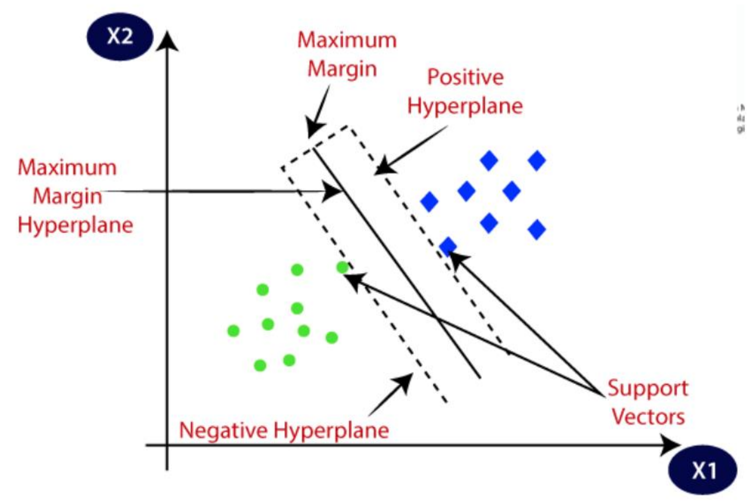


Fig 3.7 SVM classifier

In binary classification, SVM aims to find a hyperplane in the n-dimensional feature space that separates the data into two classes 0 or 1. The hyperplane is defined by the below equation:

$$Y(x) = w^T x + b \quad \textbf{(3.1)}$$

Where w is a vector perpendicular to the hyperplane, x is the input vector, b is a bias term, and y(x) is the output (class) of the SVM for input x. The SVM finds the optimal best hyperplane by maximizing the size of margin, which is the distance between the hyperplane and the closest data points of each class [17].

3.2.1 Types of SVM

- Linear
- Non Linear

Linear SVM:

Linear SVM is a kind of SVM that utilizes a linear kernel to track down the ideal choice boundary between two classes. A linear kernel is a straightforward, linear capability that computes the dot product between two element vectors. The choice boundary in a linear SVM is a hyperplane that isolates the two classes in the element space.

Linear SVM is a paired classifier that works by boosting the margin between the classes. The margin is the distance between the choice boundary and the nearest data points of each class, and expanding the margin assists with diminishing overfitting and further develop speculation [17]. The ideal choice boundary is found by tackling a constrained improvement issue that expands the margin while guaranteeing that all data focuses are accurately ordered.

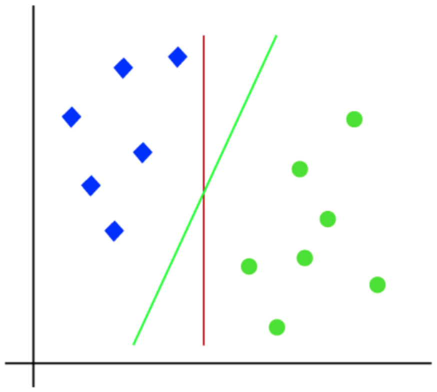


Fig 3.8 Linear SVM classifier

One of the benefits of linear SVM is that it is computationally productive and can deal with enormous datasets with high-layered include spaces. It is likewise less inclined to overfitting contrasted with different kinds of SVM that utilization more intricate kernels.

Linear SVM has been broadly utilized in numerous applications like text arrangement, picture characterization, and bioinformatics [17]. It has displayed to accomplish high precision and vigor in different errands, particularly when the component space is linearly distinguishable. Be that as it may, in situations where the data isn't linearly detachable, utilizing a linear kernel may not be ideal and different kinds of SVM with non-linear kernels might be more proper.

Non-Linear SVM:

Be that as it may, generally speaking, the data isn't linearly detachable, and in this way, a direct hyperplane can't be found. To resolve this issue, SVM utilizes a strategy called kernel stunt that maps the input data into a higher-layered space, where a linear boundary can be found. This mapping is finished by utilizing a kernel capability that processes the dot product between two vectors in the higher-layered space, without really registering the mapping.

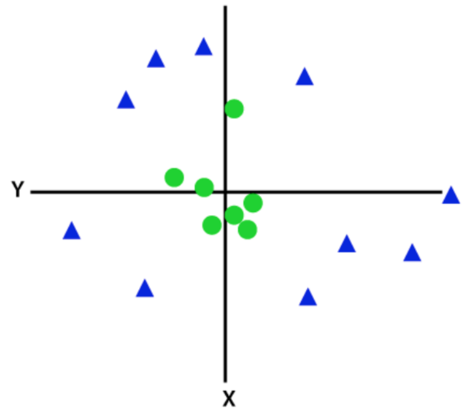


Fig 3.9 Non Linear SVM classifier

The most commonly used kernel functions are:

- Linear kernel
- Polynomial kernel
- Radial basis function (RBF) kernel

Where gamma, coef0, and degree are hyper parameters that need to be tuned to achieve the best performance.

To find the optimal hyperplane, SVM solves a convex optimization problem that minimizes the error of classification and maximizes the margin.

The problem can be formulated as:

Minimize (1/2) ||w||^2 + C sum_i=1^m xi_**i** **(3.2)**

Subject to y_i (w^T x_i + b) >= 1 - xi_i for i = 1, m

Where C is a regularization boundary that controls the compromise between margin boost and grouping blunder, xi_i are slack factors that take into account the misclassification, and m is the quantity of data focuses. The answer for this issue is a bunch of support vectors, which are the data focuses nearest to the choice boundary, and their related loads.

In outline, SVM is a strong and flexible algorithm that can deal with both linearly and non-linearly detachable data by utilizing a kernel capability to plan the data into a higher-layered space. It is additionally strong to overfitting and can deal with high-layered data effectively.

3.2.2 SVM for Human Action Detection

Human action detection is a well-known utilization of SVM in computer vision. In this errand, the objective is to characterize human actions in a video grouping, like strolling, running, bouncing, or waving. [26]

To involve SVM for human action detection, we really want to initially separate features from the video approaches that catch the important data about the action being performed. There are a few sorts of features that can be utilized, like variety histograms, optical stream, and spatiotemporal features. Spatiotemporal features are ordinarily utilized for action acknowledgment, and they portray the movement of the human body over the long run [17]. These features can be separated utilizing techniques, for example, 3D Filter or thick directions.

When the features are extricated, we can utilize SVM to characterize the actions. SVM works by tracking down the choice boundary that isolates the various classes with the biggest margin. On account of human action detection, the classes are the various kinds of actions, and the choice boundary is a hyperplane in the component space that isolates the features comparing to various actions.

The SVM algorithm can be trained utilizing a named dataset of video successions, where every video grouping is marked with the comparing action [17]. The SVM algorithm learns the ideal choice boundary by limiting the grouping mistake subject to a requirement that guarantees the margin is boosted.

To group another video succession, we first concentrate the spatiotemporal features from the video casings, and afterward apply the trained SVM model to order the action. The

SVM model outputs the anticipated action mark, which can be utilized for additional examination or applications.

Standard deviation can be utilized as an element for SVM-based human action detection. Standard deviation is a factual measure that shows how much the data focuses in a succession shift from the mean worth. With regards to human action detection, standard deviation can catch the variety of the movement examples of various actions. Standard deviation is determined by the equation,

$$\sigma = \sqrt{\frac{\sum(X-\mu)^2}{n}} \quad . \qquad \textbf{(3.3)}$$

Where,

- σ = standard deviation
- Σ = the sum of…
- *X* = Each data point in dataset (coordinates)
- μ = mean (average)
- *n* = Number of data points /samples

In outline, SVM can be utilized for human action detection by extricating spatiotemporal features from video casings and utilizing SVM to order the actions. This approach has been displayed to accomplish high precision in real life acknowledgment errands and has been utilized in various applications, like observation, sports examination, and diversion.

3.3 Overview of Convolutional Neural Network

Convolutional Neural Networks (CNNs) are deep learning models, which over the past few years have proven to be extremely efficacious in tasks related to image processing, video processing, speech recognition, natural language processing, and many others. CNNs have been used in various applications such as object/face recognition and automated cars. This chapter aims to provide a comprehensive overview of the foundation of CNNs, including the architecture, training process, and how it has been implemented in our model.

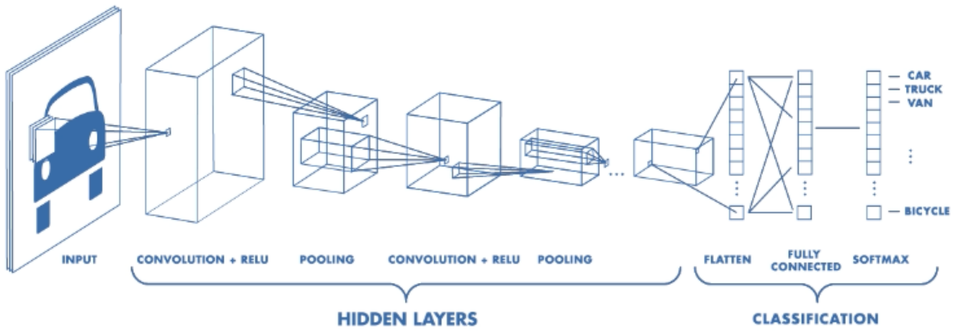


Fig 3.10 Typical CNN Architecture

3.3.1 Foundation of CNN

CNN is a feedforward neural network that uses three main layers to extract data from the input and provide an accurate result – Convolutional Layer, Pooling Layer and Fully Connected Layer. Filters in convolutional layers are used to extract features from the input data. The features are then forwarded to a fully connected layer that performs classification or regression tasks.

Convolutional Layer

The convolutional layer is the building block of CNNs. This layer is designed to extract spatial features from the previous input layer data by performing convolution operations [15]. A 3D tensor is fed forward to the convolutional layer which represents the image or video data. The tensor consists of three dimensions: width, height, and depth, where depth represents the number of channels in the image.

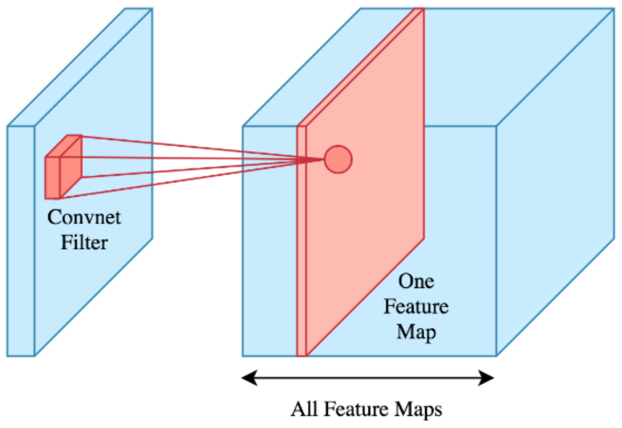


Fig 3.11 Convolution process using ConvNet filter

The output of the convolutional layer is also a 3D tensor, which represents the features extracted from the input data.

The convolution operation can be defined as follows:

Where

- Filter = h(x,y)
- Input data = f(i,j)
- Output of convolutional layer = g(i,j)

The filter h(x,y) is a weighted matrix that keeps learning during the training process. The convolutional filter slides over the input data in a stride, which is a hyperparameter, defined by the user, and performs the convolution operation at each position in the matrix. The output g (i,j) is the feature map, which represents the extracted features.

Three hyperparameters are present in a Convolutional layer which determine the size of the output volume – Stride, Depth and Zero Padding. Stride refers to the number of pixels the filter moves after an iteration. Depth simply means the number of filters which have been used in layer. Zero padding is used to retain the dimensions of the input volume in the output layer.

Pooling Layer

The pooling layer is used to down-sample the spatial size of the feature maps but in the meantime retain the important features. The pooling operation can be described as follows:

- Input data = f(i,j)
- Output of convolutional layer = g(i,j)
- Size of pooling window = k

The pooling operation calculates the maximum value (max-pooling) or average value (average-pooling) of the input data within the defined window [17]. This layer is used to reduce the size of the feature maps and the sensitivity of the network to small variations in the input data.

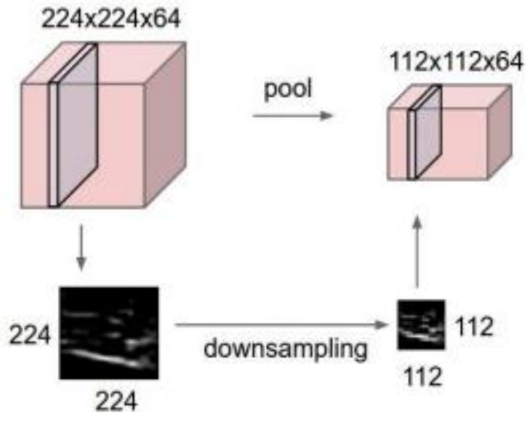


Fig 3.12 Illustration of pooling process

Fully Connected Layer

Fully connected layer is used for performing classification or regression task. The extracted features from the convolutional layer is fed to the fully connected layer to produce the final output. The fully connected layer mimics the traditional neural network layer, where each neuron is connected to all the neurons in the previous layer. The output of the fully connected layer is a vector, in which each element of the output layer provides the probability of the input data belonging to a particular class [17]. Two types of activation functions can be used in the fully connected layer – softmax for output of multiple classes, sigmoid for output of a single class.

3.3.2 Training Process

The training process of CNNs involves the following steps:

1. Initialization: Random initialization of network weights.
2. Forward Propagation: Data propagation from input layer to the output layer; various tasks take places in the middle layers.
3. Calculation: Difference between the predicted output and actual output is calculated.
4. Back Propagation: Error difference sent back to the network to modify its weights.
5. Update Weights: The weights are updated corresponding to the calculated error value.
6. Repeat: Steps 1-5 are repeated for specified number of iterations or until the network converges.

The objective of the training process is to reduce the error between the calculated output and predicted output. The error is calculated using a loss function. Some examples of loss functions are cross-entropy loss and mean squared error. The backpropagation algorithm is used to calculate the gradient of the loss function with respect to the network weights. Then, the gradient descent algorithm is used to update the weights of the network.

In the training phase of the model, the network learns to extract features from the input layer that provide relevance to the classification or regression task. The first few layers of the network extract low-level features such as edges and corners, while the deeper layers extract high-level features such as objects and scenes.

In this project we have implemented a sequential Convolutional Neural architecture with the following layers in order:

i. Conv 1D layer
ii. Max Pooling layer
iii. Flatten layer
iv. Dense / Fully connected layer
v. Dropout layer
vi. Final dense layer

This architecture, apart from the different layers present, needs to be specified of some characteristics for it to function as desired.

- The Conv1D layer has a kernel size = 3 and ReLU activation.
- Max Pooling layer has a pool size = 2
- The first Dense layer has a ReLU activation function while the last Dense layer has sigmoid activation function
- Dropout layer has 40% dropout capacity, propagating only 60% of data from the previous layer to the next layer

3.4 Long Short-Term Memory (LSTM)

LSTM, which represents Long Short-Term Memory, is a kind of recurrent neural network (RNN) engineering intended to catch long-term conditions in successive information. It was first presented by Hoch Reiter and Schmidhuber in 1997. [1]

At its center, a LSTM cell comprises of a memory cell, an input gate, an output gate, and a forget gate. The memory cell is liable for putting away and refreshing data after some time, while the input gate controls the progression of new data into the memory cell. The forget gate, then again, determines which data to dispose of from the memory cell. At last, the output gate controls the progression of data from the memory cell to the remainder of the network.

The critical advancement of LSTM is that it utilizes these gates to control the progression of data through the network specifically. This permits it to learn long-term conditions without being impacted by the evaporating slope issue, which happens when the angles in backpropagation become excessively little to actually refresh the network boundaries [18].

To prepare a LSTM network, the loads of the different gates and the memory cell are refreshed utilizing backpropagation through time (BPTT), a variation of the standard backpropagation calculation that is intended for recurrent networks. During preparing, the network is taken care of a grouping of inputs and the comparing outputs, and the loads are acclimated to limit the blunder between the anticipated and real outputs.

LSTM networks have been broadly utilized in various applications, including regular language handling, discourse acknowledgment, and time series forecast. They have shown to be especially compelling in errands that require catching long-term conditions in successive information, like language interpretation and text generation.

3.4.1 LSTM architecture

At a significant level, a LSTM network comprises of a grouping of LSTM cells that are associated with one another in a chain-like construction [18]. Each LSTM cell has a memory cell, an input gate, an output gate, and a forget gate. These gates permit the network to specifically control the progression of data through the cell, which is pivotal for catching long-term conditions. [3]

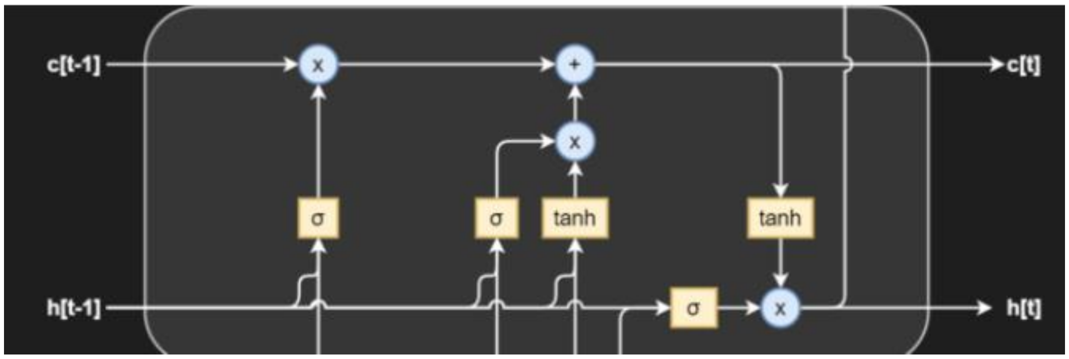


Fig 3.13 LSTM Architecture

Cell

A cell alludes to an independent module that plays out a particular procedure on the input succession and outputs a secret state, which is passed to the following cell in the grouping. The cell contains a few parts, including a memory cell, an input gate, an output gate, and a forget gate, that cooperate to specifically control the progression of data through the cell and update its memory over the long haul.

Every cell in the LSTM network has its own memory cell and gates, which permit it to catch long-term conditions in the input succession. By specifically refreshing and outputting data from the memory cell, the LSTM cell can recall significant data over a longer timeframe, making it especially successful for undertakings that require displaying consecutive information with long-term conditions [18].

By and large, the cell is a vital part of the LSTM architecture and empowers the network to specifically control the progression of data and update its memory over the long haul, making it a useful asset for displaying complex groupings.

Gates

In LSTM (Long Short-Term Memory), there are three sorts of gates: input gate, forget gate, and output gate. These gates control the progression of data into and out of the memory cell, which is the primary part of the LSTM cell. Each gate utilizes a sigmoid enactment capability to output a worth somewhere in the range of 0 and 1, which addresses the amount of the data ought to be permitted to go through the gate [18].

1. Input gate: The input gate determines the amount of the new input ought to be added to the memory cell. It takes as input the ongoing input and the past output of the network and outputs a worth somewhere in the range of 0 and 1, which is duplicated by the up-and-comer enactment to determine the amount of the new data ought to be added to the memory cell.

2. Forget gate: The forget gate determines which data to dispose of from the memory cell. It takes as input the ongoing input and the past output of the network and outputs a worth somewhere in the range of 0 and 1, which is duplicated by the past condition of the memory cell to determine the amount of the data ought to be neglected.

3. Output gate: The output gate manages the progression of data from the memory cell to the remainder of the network. It takes as input the ongoing input and the past output of the network, as well as the present status of the memory cell, and outputs a worth somewhere in the range of 0 and 1, which is duplicated by the present status of the memory cell to determine the amount of the data ought to be output.

Each gate is constrained by a bunch of teachable loads that are refreshed during preparing utilizing backpropagation through time (BPTT), which is a variation of the standard backpropagation calculation that is intended for recurrent networks. By specifically controlling the progression of data through the gates, the LSTM network can specifically hold and update data over a longer timeframe, making it especially viable for errands that require demonstrating successive information with long-term conditions.

In general, the gates are an essential part of the LSTM architecture and empower the network to specifically control the progression of data and update its memory over the long run, making it an incredible asset for displaying complex groupings.

3.4.2 Why does LSTM outperform RNN?

LSTM beats RNN in many assignments that require displaying consecutive information since it can all the more likely catch long-term conditions in the information. One of the critical explanations behind this is that LSTM has a memory cell that permits it to store and recover data over a longer timeframe specifically. This is accomplished using gates, which specifically control the progression of data into and out of the memory cell. By permitting the network to specifically refresh and output data from the memory cell, LSTM can recollect significant data over a longer timeframe, making it especially compelling for undertakings that require displaying successions with long-term conditions.

Interestingly, RNNs experience the ill effects of the issue of disappearing and gradient descent where the angles of the misfortune capability concerning the boundaries either become excessively little or excessively huge, making it challenging to successfully prepare the network [18]. This is on the grounds that the slopes in RNNs are processed by taking the result of numerous networks, which can prompt the angles either rotting or descending over the long haul.

LSTM resolves this issue by utilizing a more perplexing architecture that incorporates different layers of calculation, which permits it to specifically control the progression of data and update its memory after some time without experiencing the gradient descent issue.

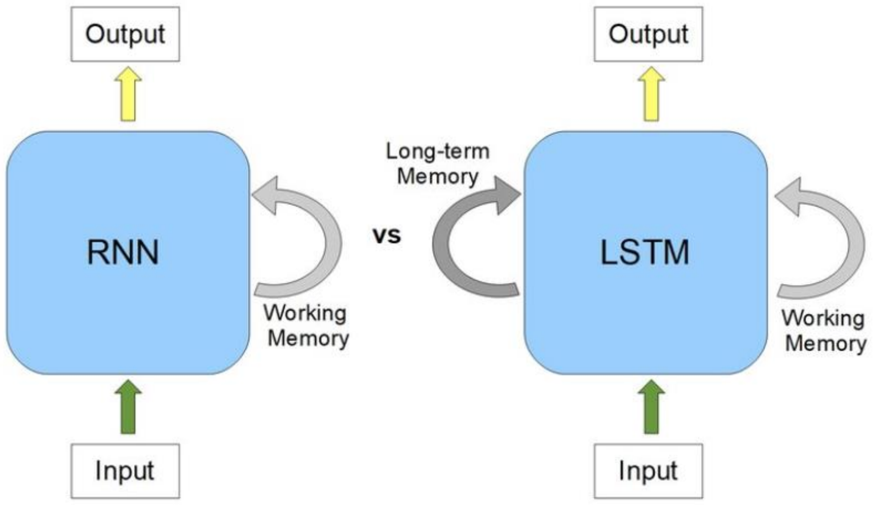


Fig 3.14 LSTM and RNN comparison

By and large, LSTM can beat RNN in many undertakings that require demonstrating consecutive information due to its capacity to specifically store and recover data over a longer timeframe and its capacity to address the evaporating and gradient descent issue.

3.4.3 LSTM for human action detection

LSTM can be utilized for human activity location in recordings by modeling the worldly conditions between video outlines and anticipating the activities being acted in the video. [4]

The interaction for involving LSTM for human activity discovery ordinarily includes the accompanying advances:

1. Data collection and pre-handling: Initial, a dataset of recordings is gathered and pre-handled. This includes sectioning the recordings into shorter clasps containing a solitary activity, removing features from each casing of the clasps utilizing strategies like optical stream, and changing over the features into a

grouping of input vectors [18].

2. Model design and training: Next, a LSTM model is designed and trained on the input succession of component vectors. The LSTM model regularly comprises of a few LSTM layers with different units in each layer, trailed by at least one completely associated layers to create the last output.

3. Validation and testing: When the model is trained, it is approved and tried on a different arrangement of recordings to assess its presentation. This includes working out measurements like exactness, accuracy, and review to gauge the model's capacity to accurately order the activities being acted in the recordings.

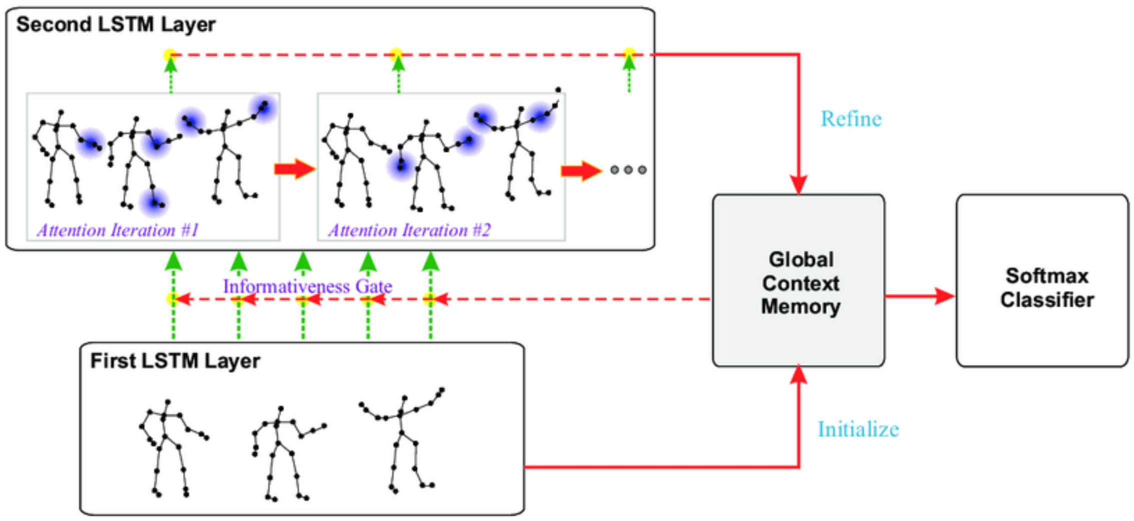


Fig 3.15 Block diagram of LSTM for stroke/human action detection

During training, the LSTM model figures out how to foresee the activities being acted in the video in light of the input grouping of component vectors. The LSTM model accomplishes this by specifically refreshing and outputting data from its memory cell over the long haul, permitting it to catch long-term conditions in the video outlines and recognize the activities being performed [18].

In this way, involving LSTM for human activity recognition includes planning and training a LSTM model on a grouping of video edges to anticipate the activities being acted in the video, and approving and testing the model to assess its presentation. By modeling the transient conditions between video outlines, LSTM can actually catch long-term conditions and precisely distinguish the activities being acted in the video [18].

CHAPTER 4
EXPERIMENTS AND RESULTS

4. EXPERIMENTS AND RESULTS

4.1 Data Collection and Preprocessing (extracting coordinates, flattening, augmentation)

Data collection and preprocessing are very important steps in any machine learning or deep learning project. In this section, we have discussed how collection and preprocessing of the data for the project on stroke recognition using any video feed by using computer vision and deep learning.

The initial step in a deep learning / machine learning project is collection and preprocessing of data. In our project, we have collected video data of stroke persons and normal persons walking and we have used pose estimation algorithms for extracting the 2D pose points of various body parts, such as the shoulders, wrists, and elbows, from each frame of the video. Specifically, we made use of the MediaPipe library, which is an open-source software that can accurately and efficiently estimate human pose key points in real-time that can run on low architecture devices [13]. The points that were extracted were then flattened into a 1D array, resulting in a dataset of size 100 x 2112 for the LSTM model. And for the 12 channel 1D-CNN model our dataset size is 100 x 176 x 12. We made use of data augmentation techniques, like random horizontal flipping and random rotation, to increase the diversity of our dataset in order to prevent overfitting.

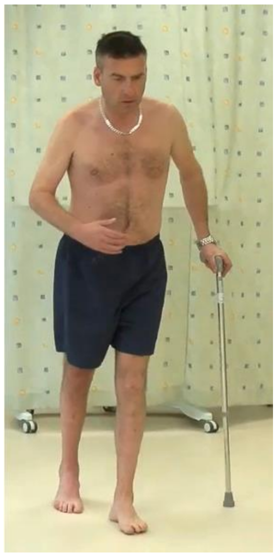


Fig 4.1 An image of a stroke patient from dataset

Our dataset comprised of 100 examples, each containing a solitary individual performing one of strolling. To separate valuable data from these recordings, we initially expected to remove the spatial directions of the individual's joints. We utilized MediaPipe, an open-source library for present assessment, to extricate the joint directions from every video outline. MediaPipe gives a bunch of pre-prepared models for different errands, including present assessment, which made it simple for us to remove the joint directions without the requirement for extra preparation.

Once we had taken out the pose points coordinates, we have then flattened them into a 1D array. This allowed us to treat each video frame as a single data point with 2112 features (12 joints (pose points), each expressed by an x and y coordinate) [13]. After that we concatenated the flattened arrays for each video frame to create a 2D array for each video, with the shape (num_frames, 2112). To enhance the diversity of our models and to reduce overfitting, we have made use of data augmentation techniques as discussed earlier. Data augmentation creates new training data by putting various transformations to our existing data. For our project, we have used two types of data augmentation: random cropping and random flipping.

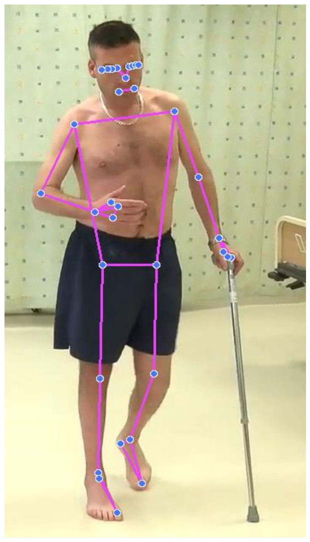
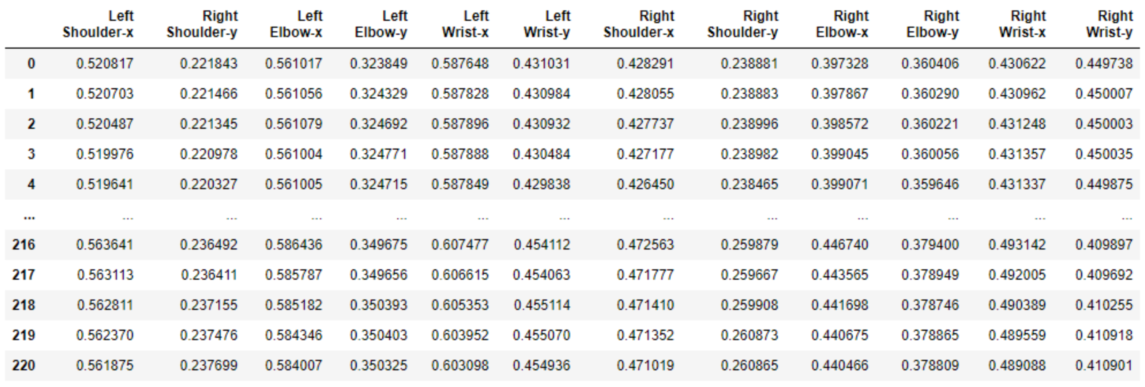


Fig 4.2 Mediapipe detecting points from a video of a stroke patient

Random cropping involves selecting a portion of the video frames randomly and resizing it to the original size. This technique creates contrast variation in our input data, that will help the model training to be more robust to various spatial configurations of the pose points [13]. Random flipping includes horizontally flipping the video frames, that creates a mirror image of the pre-existing data. This is helpful in model learning to be invariant to left-right orientation, that is important for actions such as walking. We have also randomly flipped each video frame with a probability of 0.5.



	Left Shoulder-x	Right Shoulder-y	Left Elbow-x	Left Elbow-y	Left Wrist-x	Left Wrist-y	Right Shoulder-x	Right Shoulder-y	Right Elbow-x	Right Elbow-y	Right Wrist-x	Right Wrist-y
0	0.520817	0.221843	0.561017	0.323849	0.587648	0.431031	0.428291	0.238881	0.397328	0.360406	0.430622	0.449738
1	0.520703	0.221466	0.561056	0.324329	0.587828	0.430984	0.428055	0.238883	0.397867	0.360290	0.430962	0.450007
2	0.520487	0.221345	0.561079	0.324692	0.587896	0.430932	0.427737	0.238996	0.398572	0.360221	0.431248	0.450003
3	0.519976	0.220978	0.561004	0.324771	0.587888	0.430484	0.427177	0.238982	0.399045	0.360056	0.431357	0.450035
4	0.519641	0.220327	0.561005	0.324715	0.587849	0.429838	0.426450	0.238465	0.399071	0.359646	0.431337	0.449875
...	...	...	...	...	...	...	...	...	...	...	...	...
216	0.563641	0.236492	0.586436	0.349675	0.607477	0.454112	0.472563	0.259879	0.446740	0.379400	0.493142	0.409897
217	0.563113	0.236411	0.585787	0.349656	0.606615	0.454063	0.471777	0.259667	0.443565	0.378949	0.492005	0.409692
218	0.562811	0.237155	0.585182	0.350393	0.605353	0.455114	0.471410	0.259908	0.441698	0.378746	0.490389	0.410255
219	0.562370	0.237476	0.584346	0.350403	0.603952	0.455070	0.471352	0.260873	0.440675	0.378865	0.489559	0.410918
220	0.561875	0.237699	0.584007	0.350325	0.603098	0.454936	0.471019	0.260865	0.440466	0.378809	0.489088	0.410901

Fig 4.3 12 Pose Points extracted from a video of stroke patient

After data augmentation is done and there is variety in our dataset, we have splitted our dataset into training and testing sets. We use 80% for training and 20% for testing. We use the training set to train our models and testing set for evaluation of the models performance. We have also ensured that there were no common videos in our training and testing sets to avoid any uncertainty in our model evaluation. This ensured that all features have a similar scale and help our model connect faster during training. We have used the mean and standard deviation of the training as feature in SVM model to train it inorder to derive contrasting accuracy results between our existing models.

To sum up, gathering data and analyzing it are essential aspects in every machine learning and deep learning. In our project on action recognition in videos, we used MediaPipe to extract joint coordinates, flattened them into a 1D array [13], and used data augmentation to create new training data. We also standardized our data and split it into training and validation sets. These steps helped us to create a high-quality dataset for training our deep learning models.

4.2 Data Analysis

The pose points collected from the videos provides a time series of x, y coordinates for each point. In order to extract meaningful features and patterns from out data, it is vital to perform both time and frequency domain analysis.

4.2.1 Time domain analysis

The time domain analysis involves deriving the statistical features such as mean, standard deviation etc. for each point'   s x and y coordinates over a given time interval. These derived features provide a general overview of the movement patterns of each joint during performing a certain action, in our case it is walking.

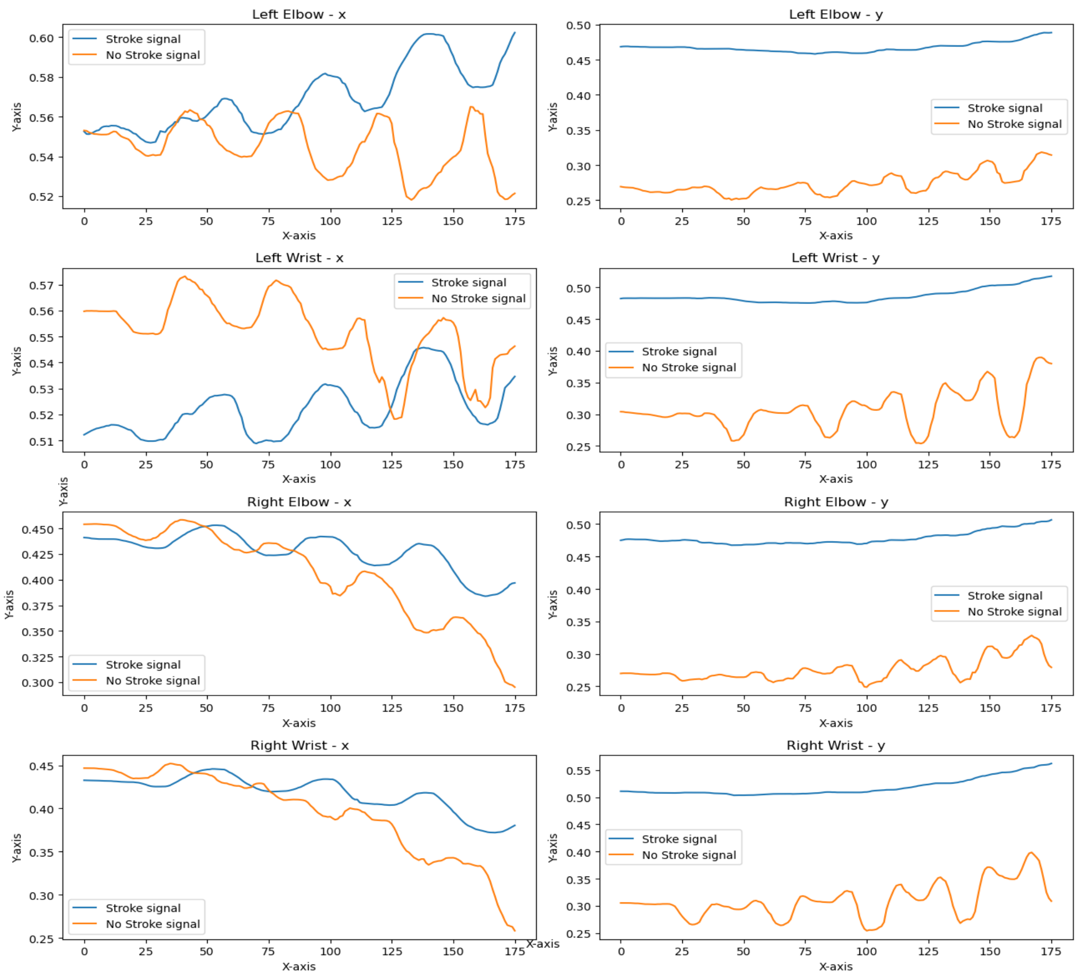


Fig 4.4 Time Domain Analysis of various joints

The above graph shows the time domain analysis of a stroke vs no stroke person walking. The time domain features are extracted for 8 points as seen above for the following points-

- Left Shoulder - X Coordinate
- Left Elbow - X Coordinate
- Left Wrist - X Coordinate
- Right Shoulder - X Coordinate
- Right Elbow - X Coordinate
- Right Wrist - X Coordinate
- Left Shoulder - Y Coordinate
- Left Elbow - Y Coordinate
- Left Wrist - Y Coordinate
- Right Shoulder -Y Coordinate
- Right Elbow - Y Coordinate
- Right Wrist - Y Coordinate

The graph shows the contrast difference between movements of a stroke person with that of a normal person.

4.2.2 Frequency domain analysis

The frequency domain analysis includes changing the time series data into the frequency domain using the FFT (Fast Fourier Transform). Therefore, this allows us to extract features and observe patterns related to the frequency content of the movement of our hands. We can then extract features like the dominant frequency, that represents the highest common frequency of movement for each part/ joint.

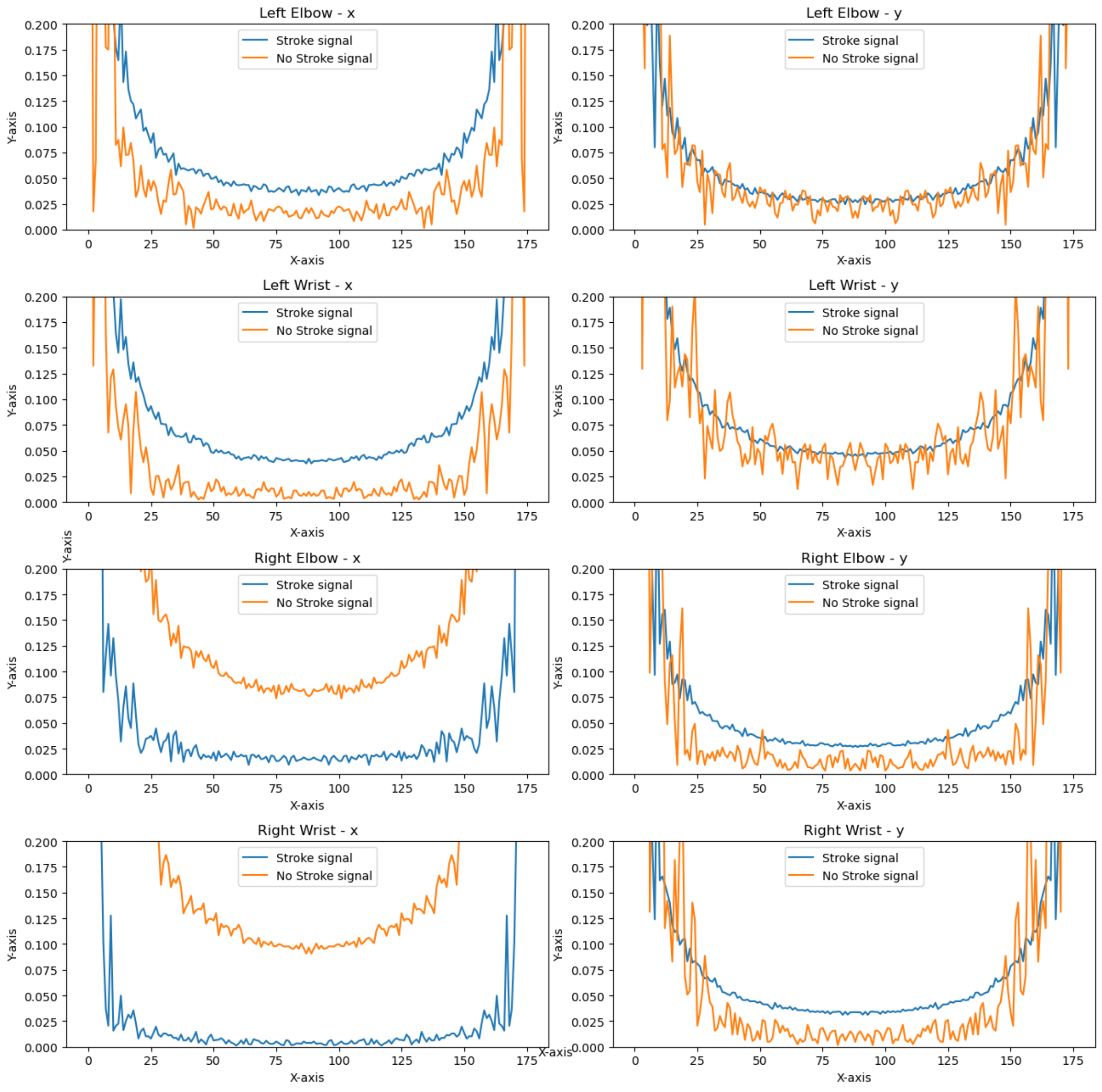


Fig 4.5 Frequency Domain Analysis of various joints

By combining the features extracted from both time and frequency domains, we can derive a more comprehensive representation of the movement patterns for each joint over time. These extracted features can be then used as input to train machine learning models such as SVM for improving the accuracy.

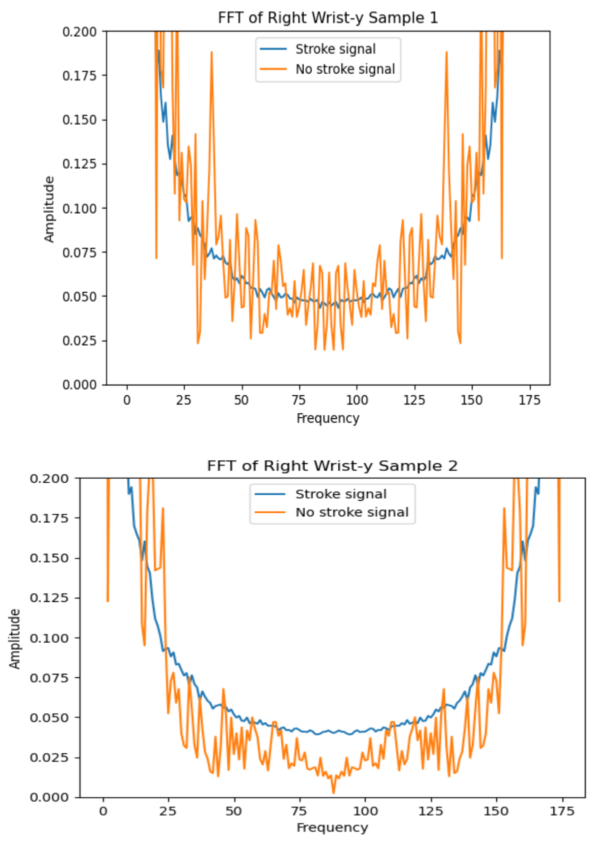


Fig 4.6 High Frequency Components seen in Right Wrist Y coordinate across all videos- sample 1, sample 2

For example, in this project, the time and frequency domain features extracted from the pose points were used as input to train LSTM and 12D CNN models for binary action recognition. The models were able to achieve high accuracy, indicating the effectiveness of the features extracted from the time and frequency domains. Overall, time and frequency domain analysis of pose points are an essential step in preprocessing and feature extraction for action recognition in videos. The features extracted can provide valuable insights into the movement patterns of different joints and can be used to train machine learning models for various applications.

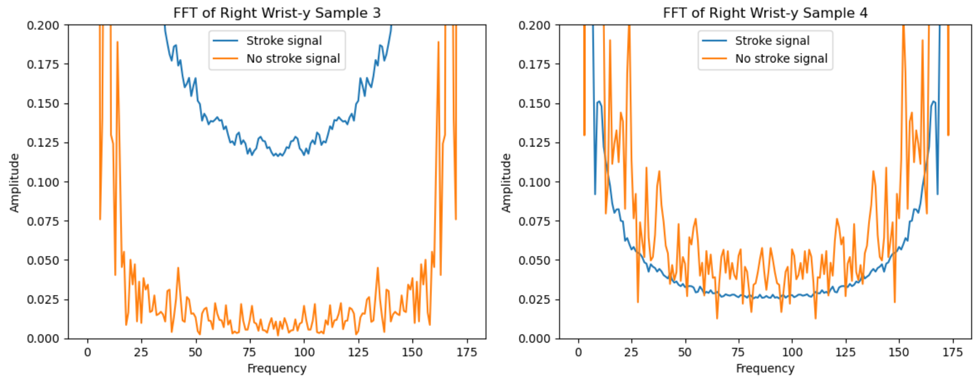


Fig 4.7 High Frequency Components seen in Right Wrist Y coordinate across all videos - sample 3, sample 4

Various statistical features like mean and standard deviation for each coordinate point for both stroke and non-stroke patients were computed from the time domain analysis of the pose points. An observation that there were drastic differences in the standard deviation and mean values of certain coordinates between the stroke patients and normal people [12].

The mean values of the coordinates were generally higher than in stroke patients in non-stroke patients [12]. This could be compared with the fact that non-stroke patients have normal hands and legs movement and can freely move their hands without any restrictions. On the other hand, stroke patients have constricted movements, and their hands are usually paralyzed or weaker, leading to a decrease in the standard deviation and mean values of the coordinates [12].

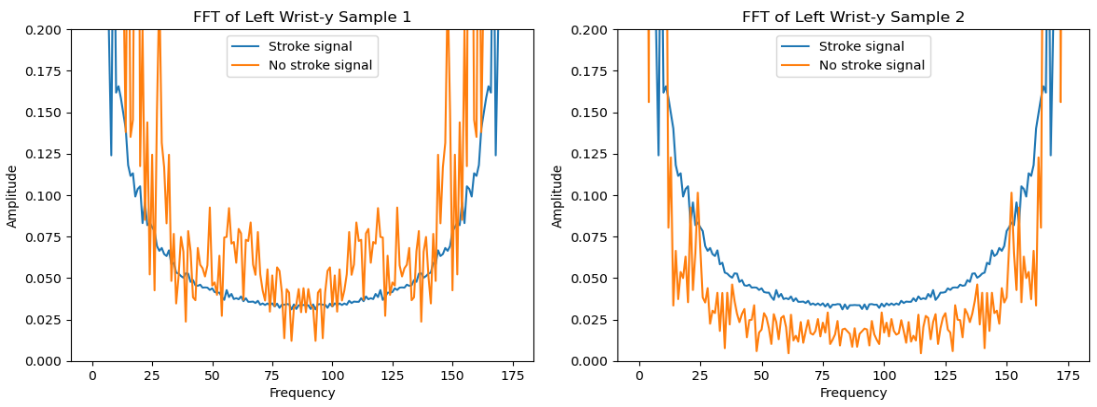


Fig 4.8 High Frequency Components seen in Left Wrist Y coordinate across all videos- - sample 1, sample 2

In Addition, we have also observed that the values of standard deviation of the coordinates were drastically higher in non-stroke patients than that in stroke patients. This might be due to the increased flexibility and variability in movement in hands and legs in non-stroke patients [12]. Their movements can be more varied and unpredictable than in stroke patients, who have restricted limb movement since they have a normal limb movement.

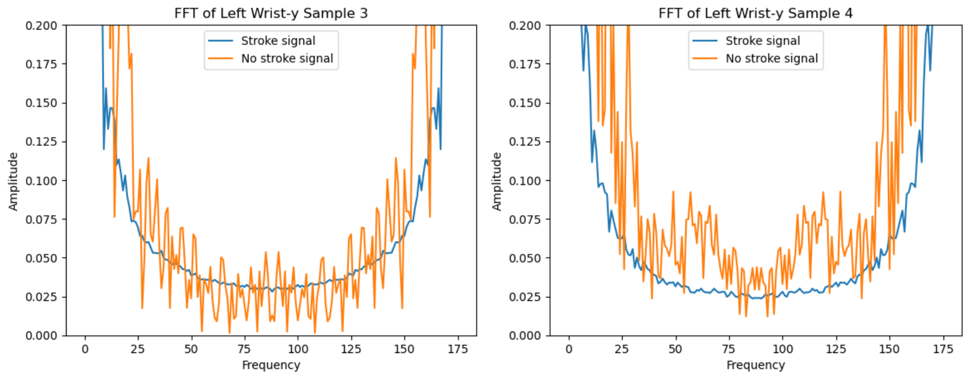


Fig 4.9 High Frequency Components seen in Left Wrist Y coordinate across all videos- - sample 3, sample 4

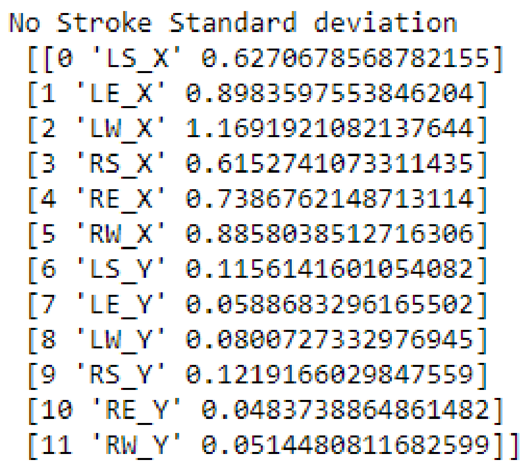
We have also observed that there were drastic differences in the frequency distribution between stroke and non-stroke patients in the frequency domain analysis we using FFT. Specifically, we have observed that there were higher frequency components in non-stroke patients than that in stroke patients. This result may be explained by the fact that non-stroke patients have higher range of motion in their hands and legs and are able to carry out a wider variety of actions. On the other hand, stroke patients have very constricted movement in their hands and legs, that could result in less variation in the frequency components of their movements.

Another Observation was made where in the differences in frequency distribution were more noticeable in the following coordinates, such as the right and left elbow's y coordinate [12]. This recommends that these coordinates could be especially used in separating among stroke and non-stroke patients.

Summary: Significant differences between stroke and non-stroke patients were found in the time and frequency domain analysis of the posture points. While the frequency domain analysis revealed differences in the frequency distribution of the movements, the time domain analysis brought out differences in the mean and standard deviation values of certain coordinates. These observations could be useful in developing a pose point’s based model for stroke detection.

4.2.3 Statistical features

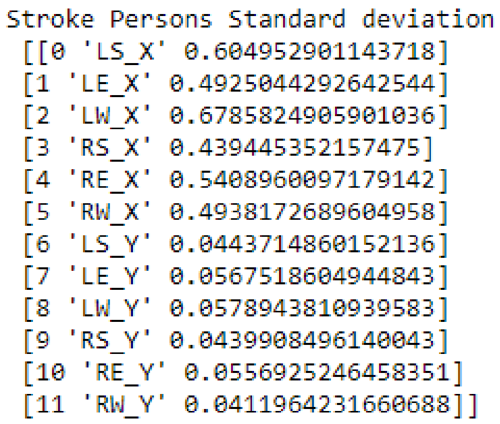
In addition to the time and frequency domain analysis, statistical features were also derived from the pose points. The standard deviation and root mean square (RMS) value were calculated for each coordinate. These statistical features provide information on the variability and amplitude of the movements, respectively.



```
No Stroke Standard deviation
 [[0 'LS_X' 0.6270678568782155]
 [1 'LE_X' 0.8983597553846204]
 [2 'LW_X' 1.1691921082137644]
 [3 'RS_X' 0.6152741073311435]
 [4 'RE_X' 0.7386762148713114]
 [5 'RW_X' 0.8858038512716306]
 [6 'LS_Y' 0.1156141601054082]
 [7 'LE_Y' 0.0588683296165502]
 [8 'LW_Y' 0.0800727332976945]
 [9 'RS_Y' 0.1219166029847559]
 [10 'RE_Y' 0.0483738864861482]
 [11 'RW_Y' 0.0514480811682599]]
```

Fig 4.10 Derived Values of Standard Deviation for No Stroke Patients

On analysis, we have observed that individuals without stroke have had higher magnitude of standard deviation values than those with stroke [14]. This result may be explained by the fact that non-stroke patients have higher range of motion in their hands and legs and are able to carry out a wider variety of actions [12]. On the other hand, individuals with stroke often have limited movement in certain joints or limbs, resulting in less variability in their movements. This trend was noticeable consistently across all the videos analyzed.



```
Stroke Persons Standard deviation
 [[0 'LS_X' 0.604952901143718]
 [1 'LE_X' 0.4925044292642544]
 [2 'LW_X' 0.6785824905901036]
 [3 'RS_X' 0.439445352157475]
 [4 'RE_X' 0.5408960097179142]
 [5 'RW_X' 0.4938172689604958]
 [6 'LS_Y' 0.0443714860152136]
 [7 'LE_Y' 0.0567518604944843]
 [8 'LW_Y' 0.0578943810939583]
 [9 'RS_Y' 0.0439908496140043]
 [10 'RE_Y' 0.0556925246458351]
 [11 'RW_Y' 0.0411964231660688]]
```

Fig 4.11 Derived Values of Standard Deviation for Stroke Patients

Similarly, the RMS value was found to be higher in individuals without stroke compared to those with stroke. This is because non-stroke patients are able to perform larger and more diverse movements, resulting in a greater range of motion and higher amplitude of movements [12]. In contrast, individuals with stroke may have limited range of motion and decreased strength, resulting in smaller and less intense movements.

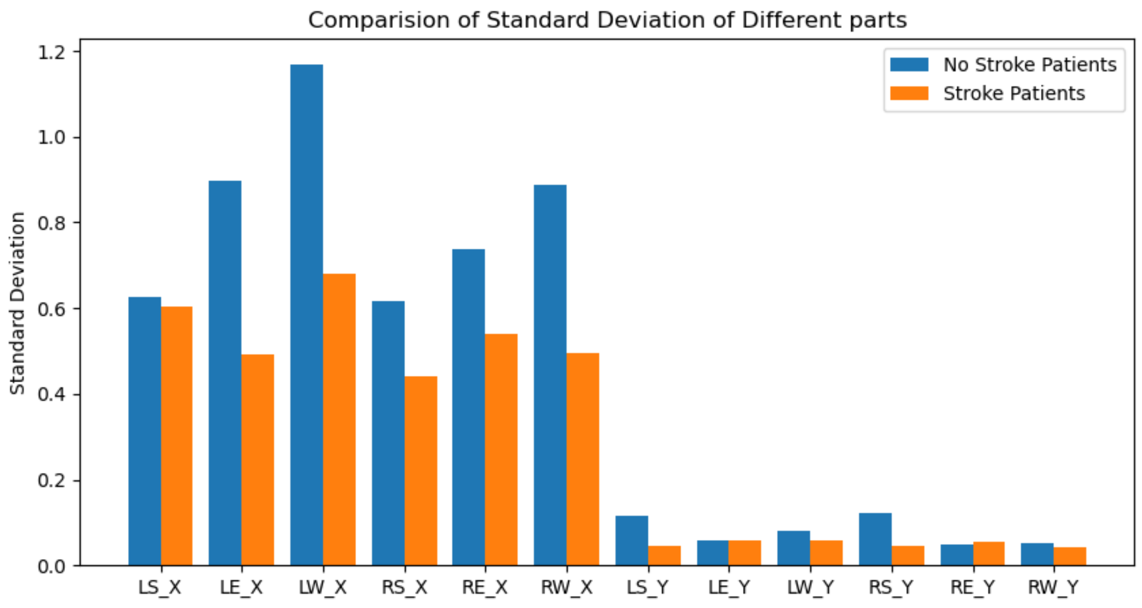


Fig 4.12 Contrasting difference in Standard deviation values of different parts for stroke and no stroke patients.

These statistical features can provide valuable insights into the differences in movement patterns between individuals with and without stroke. By identifying these differences, it may be possible to develop more accurate and effective methods for detecting stroke and monitoring recovery progress. Furthermore, these statistical features can also be used as input features in machine learning models for stroke detection and classification.

```
Stroke RMS =  0.604952901143718 No Stroke RMS=  0.6270678568782155
```

Fig 4.13 difference in RMS values for stroke and no stroke patients.

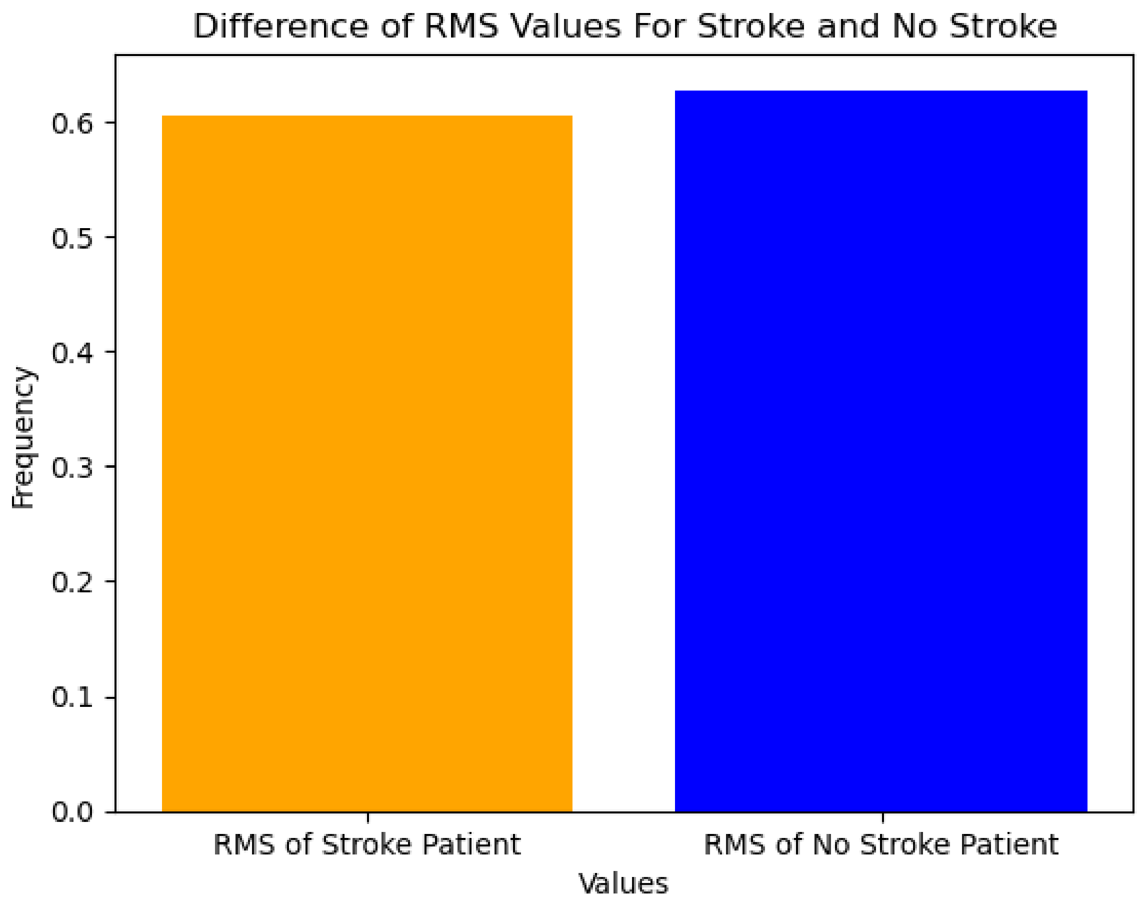


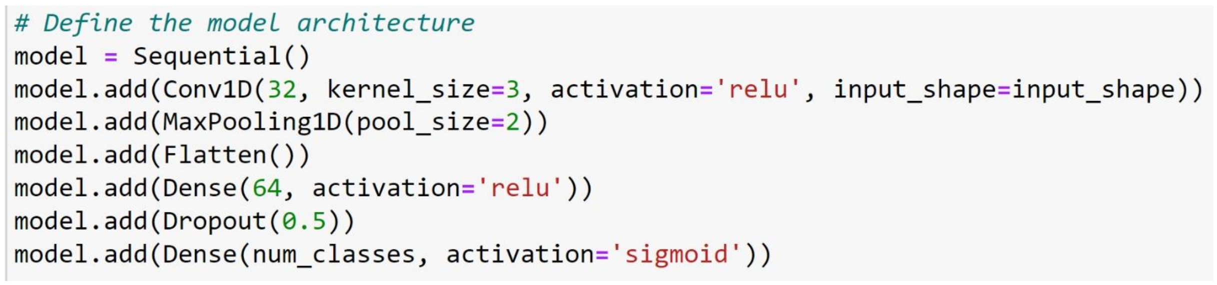
Fig 4.14 Contrasting difference in RMS values of different parts for stroke and no stroke patients.

4.3 Architectural Design and Hyper parameters

The architecture of a CNN/LSTM model characterizes the construction of the neural network. Included a few layers are liable for feature extraction, spatial separating, and classification. Hyperparameters are values that characterize the way of behaving of the model during training. The decision of hyperparameters can fundamentally influence the exhibition of the model. Some of the hyper parameters include batch size, number of epochs, dropout rate, and size of the filters [14].

4.3.1 Convolution 1D 12 channel architecture and hyperparameters

We used sequential CNN architecture, A Sequential CNN is a Convolutional Neural Network (CNN) model architecture that is designed as a sequence of layers. In this type of architecture, each layer is associated with the past layer, framing a linear pipeline of tasks. Basically a sequential CNN includes input layer, convolution layer, pooling layer, activation layer and fully connected layer.



```python
# Define the model architecture
model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='relu', input_shape=input_shape))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='sigmoid'))
```

Fig 4.15 Convolution 1D 12 channels model architecture

Epoch determines the number of times the model is trained on the entire dataset, so more the epoch size more the accuracy, we set it as 1000, and means entire model is trained 1000 times over a dataset given.

Convolution 1D is used as it is best suited for one- dimensional data such as time series. As in our case data changes from frame to frame this model is efficient. Input shape is assigned as 100*176*12

Batch size is the number of samples that are passed through the neural network at a time we used a smaller batch size of 32 because smaller the batch size more accurate the gradient estimate, even though its takes little more time.

Used a flattening layer for dimensionality reduction, followed by a dense layer with batch size 64, ReLU (Rectified Linear Unit) activation is used to replace negative values with zero without modifying positive values.

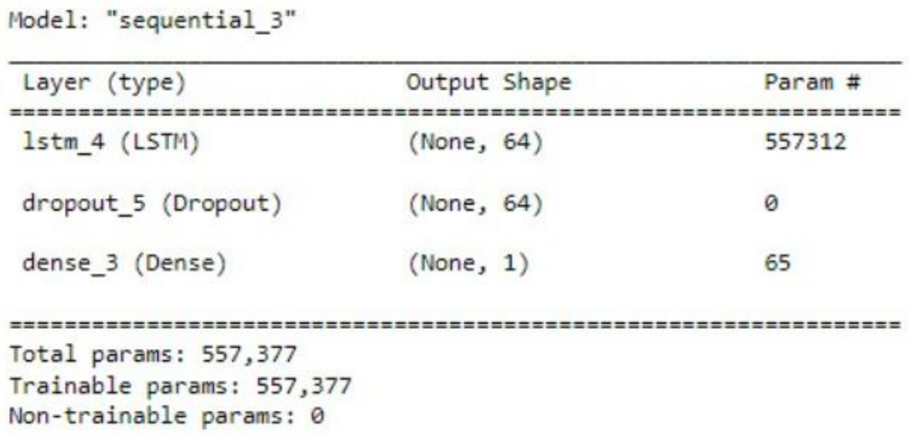
The Dropout Layer works by randomly exiting (i.e., setting to nothing) a fraction of the output units of the past layer during training. This implies that the network is compelled to learn more vigorous features that are not excessively reliant upon any single input unit.

By exiting a portion of the units during each training emphasis, the Dropout Layer diminishes the co-variation of element finders and assists with forestalling overfitting. We configured dropout as 0.6 means, it will drop 60% data at each iteration.

Finally used a dense layer, for getting an output/prediction, number of classes for is binary (0 or 1) stroke or no stroke, so used sigmoid activation which is best suited for binary output.

4.3.2 LSTM architecture and hyperparameters

We used sequential LSTM architecture, A Sequential LSTM designed as a sequence of layers. In this type of architecture, each layer is associated with the past layer, framing a linear pipeline of tasks. Basically a sequential LSTM includes LSTM layer, dropout layer and dense layer



```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 64)                557312

 dropout_5 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 557,377
Trainable params: 557,377
Non-trainable params: 0
```

Fig 4.16 LSTM model architecture

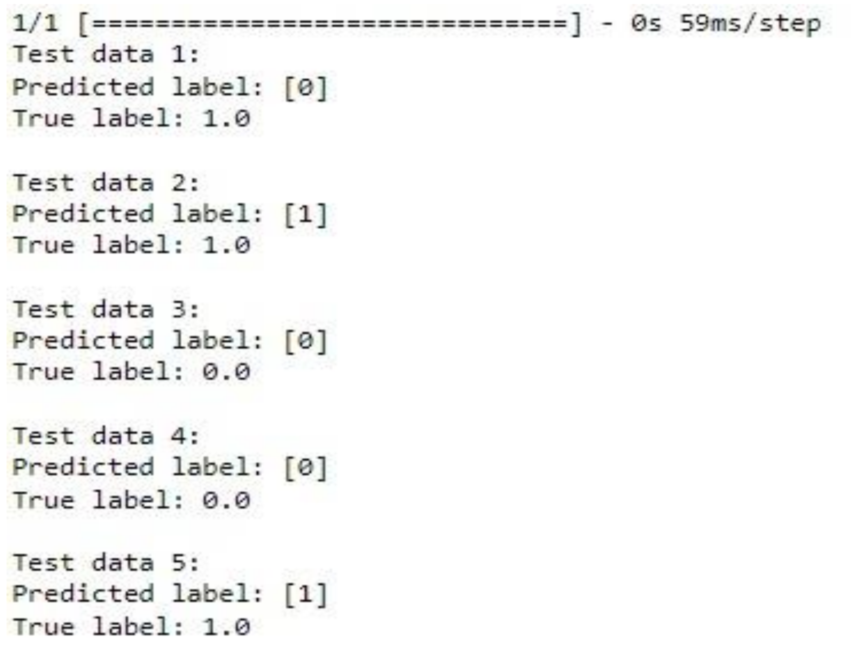
Input shape is flattened like 100*176*12, used a dropout layer and a dense layer is added to get output.

4.4 Training and validation process

To train the 12-channel CNN, we made use of the Keras library in Python. The model was defined with a convolutional layer with 32 filters and a kernel size of 3, trailed by max pooling and a flatten layer. The result of the flatten layer was then fed to a dense layer with 64 units and a ReLU activation function, after that it was followed by a dropout layer with a dropout rate of 0.4. At long last, a dense layer with a 1 unit and a sigmoid activation function was utilized as the result layer to make binary predictions. Compilation of the model was done with binary cross-entropy loss. We made use of Adam optimizer. We also used accuracy and val_accuracy as metric to monitor the performance of the model during training.

During training, we made use of early stopping to avoid overfitting of the model. The early stopping rules were set to screen the validation loss, and training was stopped on the off chance that the approval misfortune didn't improve for 10 sequential ages. The model was prepared on the preparation set with 32 batches for each epoch, and the training process was recurrent for 100 epochs. To avoid overfitting of the model, we have also incorporated a testing split of 10% of the training data.

In general, the preparation and approval process was done successfully, and also we successfully trained the 12-channel CNN that accomplished high accuracy in predicting stroke from present information.

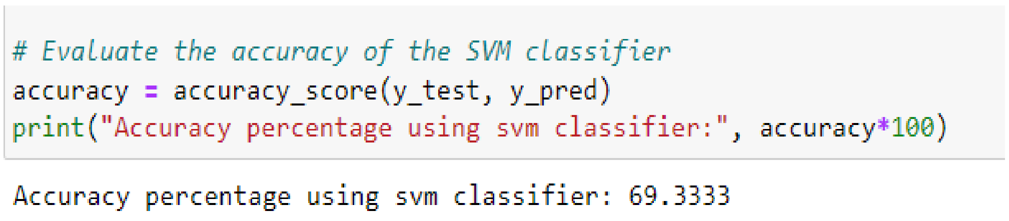


```
1/1 [==============================] - 0s 59ms/step
Test data 1:
Predicted label: [0]
True label: 1.0

Test data 2:
Predicted label: [1]
True label: 1.0

Test data 3:
Predicted label: [0]
True label: 0.0

Test data 4:
Predicted label: [0]
True label: 0.0

Test data 5:
Predicted label: [1]
True label: 1.0
```

Fig 4.17 Realtime Detection of No Stroke

4.5 Model Comparison

During the timespan of our project, we utilized various robust algorithms in order to evaluate their performance in terms of assessing whether a patient has stroke or not. We also compared these algorithms to one and another so we could make an informed decision on which algorithm is best suited for our dataset. The algorithms that we considered for our project were Support Vector Machine (SVM), Central Neural Network (CNN) and long short-term memory (LSTM). First and foremost, we utilized the media-pipe framework for pose estimation and identified and selected the necessary semantic key points for our system. Then we labelled the points using inbuilt tools and visualized the pose estimation in real time. After this, we collected the coordinates from the pose points and exported them to a CSV file and performed time and frequency domain analysis. We then found the standard deviation of each of the points to understand and quantify the variability of the set of data to evaluate how much the individual data points deviate from the mean of the data set. A dataset was created with standard deviation values of all points of patients with stroke and no stroke. Initially we passed these values to the SVM classifier to evaluate the accuracy and decide whether we would utilize this as a feature based on its performance.

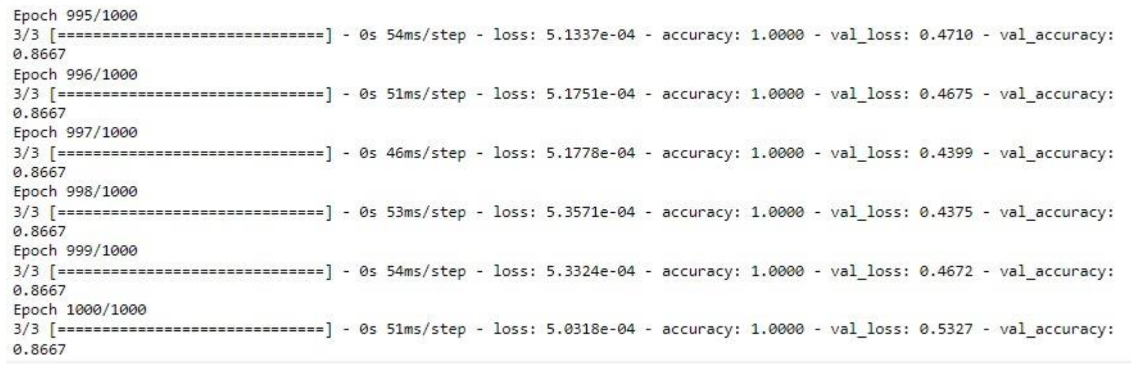


```
# Evaluate the accuracy of the SVM classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy percentage using svm classifier:", accuracy*100)
```

```
Accuracy percentage using svm classifier: 69.3333
```

Fig 4.18 SVM accuracy

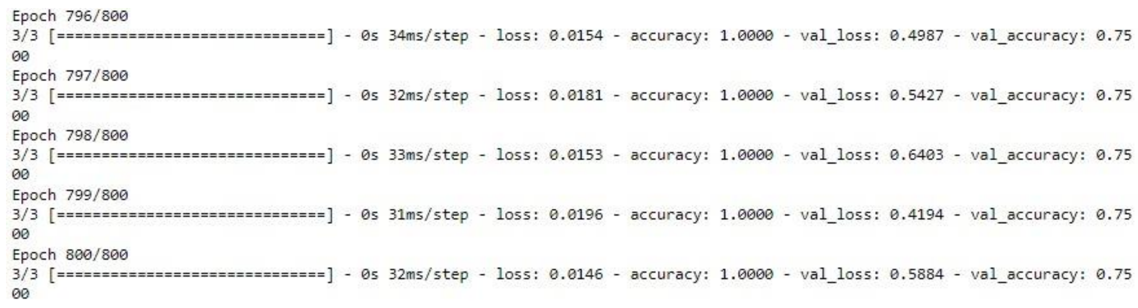
We attained an accuracy of 69.33%, in order to increase the accuracy of our model we decided to feed the data to a convolutional neural network and also implement LSTM network for the same data. After feeding the data into a 1D CNN model consisting of Conv1D, Pooling, Flattening and Dense layers, we evaluated the model. We tested various validation splits including 0.1,0.2,0.3 and concluded that a validation split of 0.2 (CNN) gave us the most accuracy of 0.8667.



```
Epoch 995/1000
3/3 [==============================] - 0s 54ms/step - loss: 5.1337e-04 - accuracy: 1.0000 - val_loss: 0.4710 - val_accuracy:
0.8667
Epoch 996/1000
3/3 [==============================] - 0s 51ms/step - loss: 5.1751e-04 - accuracy: 1.0000 - val_loss: 0.4675 - val_accuracy:
0.8667
Epoch 997/1000
3/3 [==============================] - 0s 46ms/step - loss: 5.1778e-04 - accuracy: 1.0000 - val_loss: 0.4399 - val_accuracy:
0.8667
Epoch 998/1000
3/3 [==============================] - 0s 53ms/step - loss: 5.3571e-04 - accuracy: 1.0000 - val_loss: 0.4375 - val_accuracy:
0.8667
Epoch 999/1000
3/3 [==============================] - 0s 54ms/step - loss: 5.3324e-04 - accuracy: 1.0000 - val_loss: 0.4672 - val_accuracy:
0.8667
Epoch 1000/1000
3/3 [==============================] - 0s 51ms/step - loss: 5.0318e-04 - accuracy: 1.0000 - val_loss: 0.5327 - val_accuracy:
0.8667
```

Fig 4.19 CNN accuracy

The following procedure was repeated for LSTM and we attained an accuracy of 0.75, hence we concluded that CNN gave better accuracy than LSTM and SVM which was evaluated earlier.



```
Epoch 796/800
3/3 [==============================] - 0s 34ms/step - loss: 0.0154 - accuracy: 1.0000 - val_loss: 0.4987 - val_accuracy: 0.75
00
Epoch 797/800
3/3 [==============================] - 0s 32ms/step - loss: 0.0181 - accuracy: 1.0000 - val_loss: 0.5427 - val_accuracy: 0.75
00
Epoch 798/800
3/3 [==============================] - 0s 33ms/step - loss: 0.0153 - accuracy: 1.0000 - val_loss: 0.6403 - val_accuracy: 0.75
00
Epoch 799/800
3/3 [==============================] - 0s 31ms/step - loss: 0.0196 - accuracy: 1.0000 - val_loss: 0.4194 - val_accuracy: 0.75
00
Epoch 800/800
3/3 [==============================] - 0s 32ms/step - loss: 0.0146 - accuracy: 1.0000 - val_loss: 0.5884 - val_accuracy: 0.75
00
```

Fig 4.20 LSTM accuracy

4.6 Real-time Human stroke detection

After completing the data collection, preprocessing, and analysis, we have developed a real-time working system for our model. The system takes input for 4 to 5 seconds and extracts pose points and predicts the movements of the patients in real-time.

The system makes use of a pre-trained deep learning model to classify the pose points of a person as either having a stroke or not. The model is trained on a large dataset of stroke and non-stroke patients and uses the features extracted from the pose points to classify the patients. Fig 4.21 and Fig 4.22 depicts the live detection of stroke as a human starts walking.

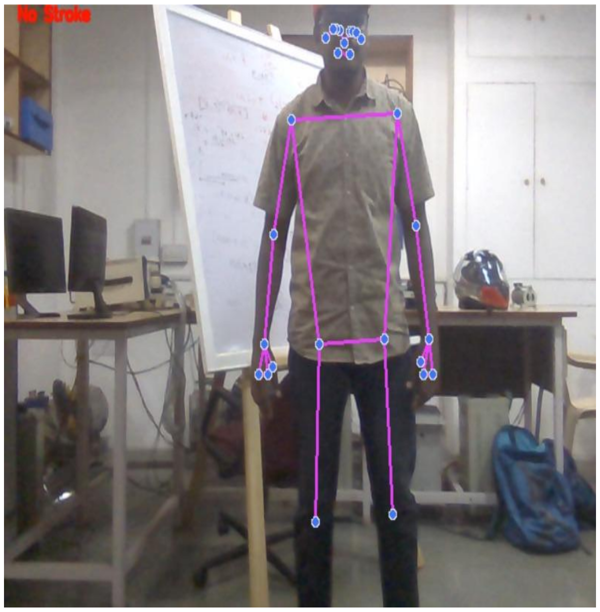


Fig 4.21 Real-time Detection of No Stroke

The system continuously extracts pose points in the time interval of 5 seconds from the video that is running using a web-cam. The model predicts if the person has a stroke or not based on these extracted points. This system can be used in rehabilitation centers or hospitals where continuous monitoring of stroke patients is required.

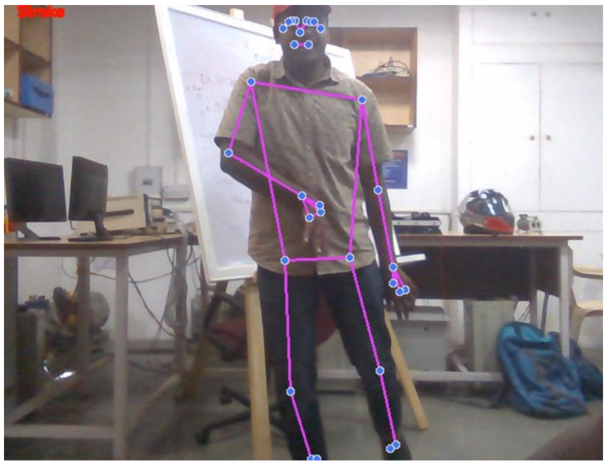


Fig 4.22 Real-time Detection of Stroke in real-time

This real-time working system can be a noteworthy progress in the field of stroke rehab as it has the capacity to monitor the patients continuously and provide real-time feedback on their movements. This feedback can be used to help the patients to improve their movements and can also help the doctors to monitor the progress of the patients. As and then we are monitoring the patients affected by stroke, we can collect dataset from them in the form of videos. This video data collected can not only be then used for improving the accuracy of the model but also will bring more diversity in our data to detect stroke in finer movements.

Generally, the framework is not difficult to utilize, and the predictions are exact. The framework can be utilized by specialists or recovery focuses to screen their patients' developments and give customized input to every patient in view of their developments.

CHAPTER 5
CONCLUSION AND FUTURE SCOPE

5. CONCLUSION AND FUTURE SCOPE

In conclusion, the commonality and detrimental consequences of stroke make it an issue of utmost prevalence in the healthcare sector. Stroke is a neurological disease that is caused by the intervention of blood supply to the brain, it affects people of all ages, making it a leading cause of disability and death globally. It is also quintessential to note that relatives of patients with stroke often struggle with depression, anxiety, and post-traumatic stress disorder (PTSD). Although healthcare has advanced to a significant extent, there are many blatant problems when it comes to diagnosing and treating a stroke patient which must be addressed. A lack of skilled neurologists to properly diagnose patients and provide high-quality patient care serves as a major obstacle in treating patients with stroke. Patients who reside in rural areas do not have access to skilled neurologists or healthcare facilities. Numerous healthcare specialists fail to diagnose neurological diseases like stroke early on, hence exacerbating the consequences and making the conditions worse. Progress and feedback of treatment are not monitored properly as skilled neurologists cannot observe the minute regressions or progressions of their patients' conditions.

Additionally, it is extremely challenging to detect stroke in infants and young children as the lack of observable symptoms and communication hinders our ability to diagnose them accurately and efficiently. Invasive procedures like MRI, are not appropriate for young children or infants with neurological diseases. In this project, we have utilized the convolutional neural network (CNN) as well as the mediapipe library for stroke detection and have solved the impending issues listed above in a non-invasive fashion by leveraging the mediapipe framework to detect whether a patient has stroke or not by detecting the semantic key points and evaluating if there is an abnormality in the distance between the respective joint coordinates.

The following observation were made as well:

- High frequency components in no stroke, low frequency components in stroke.
- Contrasting difference was seen in the frequency domain of Y coordinates of

both wrists.

- The magnitude of standard deviation was higher for no stroke patients and slightly lower for patients with stroke.
- The magnitude of RMS was higher for no stroke patients and slightly lower for patients with stroke.

After observing the following, we have concluded that patients with stroke have limited movement in certain joints or limbs, which results in less variability in their movements. Whereas non-stroke patients have a higher range of motion in their hands and legs, hence they can perform a wider variety of actions. This trend was consistent across all the videos analyzed. We have successfully created 3 stroke detection models with the following results which include:

- SVM: Accuracy of 69%
- LSTM: Accuracy of 75%
- 12 Channel – Accuracy of 85%

This highly functional and versatile model developed could be refined and improved further. Fine movements like facial expressions and other small movements could be considered for greater accuracy as we have selected 12 semantic key points. Since we have incorporated a holistic model, we have attained 12 semantic key points, including other mediapipe models we could also detect smaller traces of stroke such as tremors in our fingers or tremors in our face. Progression in the recovery process could be analyzed and valuable insights would be given, this could help healthcare professionals and skilled neurologists to diagnose numerous other diseases not just stroke. This model could also be used to diagnose other diseases with different training data as well.

REFERENCES

1. Han, Yull Kyu, and Young Bok Choi. "Human action recognition based on LSTM model using smartphone sensor." In 2019 Eleventh International Conference on Ubiquitous and Future Networks (ICUFN), pp. 748-750. IEEE, 2019.
2. Liu, Jun, Gang Wang, Ling-Yu Duan, Kamila Abdiyeva, and Alex C. Kot. "Skeleton-based human action recognition with global context-aware attention LSTM networks." IEEE Transactions on Image Processing 27, no. 4 (2017): 1586-1599.
3. Orozco, Carlos Ismael, Maria Elena Buemi, and Julio Jacobo Berlles. "Towards an attention mechanism LSTM framework for human action recognition in videos." In 2020 IEEE Congreso Bienal de Argentina (ARGENCON), pp. 1-6. IEEE, 2020.
4. Deep, Samundra, and Xi Zheng. "Hybrid model featuring CNN and LSTM architecture for human activity recognition on smartphone sensor data." In 2019 20th international conference on parallel and distributed computing, applications and technologies (PDCAT), pp. 259-264. IEEE, 2019.
5. Kazemimoghadam, Mahdieh, and Nicholas P. Fey. "An Activity Recognition Framework for Continuous Monitoring of Non-Steady-State Locomotion of Individuals with Parkinson’s Disease." *Applied Sciences* 12, no. 9 (2022): 4682.
6. Luo, Yue, Jimmy Ren, Zhouxia Wang, Wenxiu Sun, Jinshan Pan, Jianbo Liu, Jiahao Pang, and Liang Lin. "Lstm pose machines." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5207-5215. 2018.
7. Ismail, Hafsa, Ibrahim Radwan, Hanna Suominen, and Roland Goecke. "Gait estimation and analysis from noisy observations." In 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 2707-2712. IEEE, 2019.
8. Sarshar, Mustafa, Sasanka Polturi, and Lutz Schega. "Gait phase estimation by using LSTM in IMU-based gait analysis—Proof of concept." Sensors 21, no. 17 (2021): 5749.
9. Paragliola, Giovanni, and Antonio Coronato. "A deep learning-based approach for the classification of gait dynamics in subjects with a neurodegenerative disease." In

Intelligent Systems and Applications: Proceedings of the 2020 Intelligent Systems Conference (IntelliSys) Volume 3, pp. 452-468. Springer International Publishing, 2021.

10. Elkholy, M. E. Hussein, W. Gomaa, D. Damen and E. Saba, "Efficient and Robust Skeleton-Based Quality Assessment and Abnormality Detection in Human Action Performance, " in IEEE Journal of Biomedical and Health Informatics, vol. 24, no. 1, pp. 280-291, Jan. 2020, doi: 10.1109/JBHI.2019.2904321.
11. Eltoukhy, Moataz A., Christopher Kuenze, Jeonghoon Oh, and Joseph F. Signorile. "Validation of static and dynamic balance assessment using Microsoft Kinect for young and elderly populations." IEEE journal of biomedical and health informatics 22, no. 1 (2017): 147-15
12. Sanchez, Robert J., Jiayin Liu, Sandhya Rao, Punit Shah, Robert Smith, Tariq Rahman, Steven C. Cramer, James E. Bobrow, and David J. Reinkensmeyer. "Automating arm movement training following severe stroke: functional exercises with quantitative feedback in a gravity-reduced environment." *IEEE Transactions on neural systems and rehabilitation engineering* 14, no. 3 (2006): 378-389.
13. Kritsis, Kosmas, Maximos Kaliakatsos-Papakostas, Vassilis Katsouros, and Aggelos Pikrakis. "Deep convolutional and lstm neural network architectures on leap motion hand tracking data sequences." In 2019 27th European Signal Processing Conference (EUSIPCO), pp. 1-5. IEEE, 2019.
14. Carpinella, Ilaria, Elisa Gervasoni, Denise Anastasi, Tiziana Lencioni, Davide Cattaneo, and Maurizio Ferrarin. "Instrumental assessment of stair ascent in people with multiple sclerosis, stroke, and Parkinson’s disease: a wearable-sensor-based approach." IEEE Transactions on Neural Systems and Rehabilitation Engineering 26, no. 12 (2018): 2324-2332.
15. Cai, Zhuohao, Yi Yang, and Lan Lin. "Human action recognition and art interaction based on convolutional neural network." In 2020 Chinese Automation Congress (CAC), pp. 6112-6116. IEEE, 2020.
16. Yu, Xiaodong, Ta Wen Kuan, Yuhan Zhang, and Taijun Yan. "YOLO v5 for SDSB Distant Tiny Object Detection." In 2022 10th International Conference on Orange Technology (ICOT), pp. 1-4. IEEE, 2022.

17. Chen, Yangsen, Rongxi Du, Kaitao Luo, and Yuheng Xiao. "Fall detection system based on real-time pose estimation and SVM." In 2021 IEEE 2nd international conference on big data, artificial intelligence and internet of things engineering (ICBAIE), pp. 990-993. IEEE, 2021.
18. Yin, Jun, Jun Han, Ruiqi Xie, Chenghao Wang, Xuyang Duan, Yitong Rong, Xiaoyang Zeng, and Jun Tao. "Mc-lstm: Real-time 3d human action detection system for intelligent healthcare applications." IEEE Transactions on Biomedical Circuits and Systems 15, no. 2 (2021): 259-269.
19. Amrutha, K., P. Prabu, and Joy Paulose. "Human Body Pose Estimation and Applications." In 2021 Innovations in Power and Advanced Computing Technologies (i-PACT), pp. 1-6. IEEE, 2021.
20. Ho, King Chung, William Speier, Haoyue Zhang, Fabien Scalzo, Suzie El-Saden, and Corey W. Arnold. "A machine learning approach for classifying ischemic stroke onset time from imaging." IEEE transactions on medical imaging 38, no. 7 (2019): 1666-1676.
21. C. Zheng, C. Hu, Y. Chen and J. Li, "A Self-Learning-Update CNN Model for Semantic Segmentation of Remote Sensing Images," in *IEEE Geoscience and Remote Sensing Letters*, vol. 20, pp. 1-5, 2023, Art no. 6004105, doi: 10.1109/LGRS.2023.3261402.
22. N. Haouchine, P. Juvekar, M. Nercessian, W. M. Wells III, A. Golby and S. Frisken, "Pose Estimation and Non-Rigid Registration for Augmented Reality During Neurosurgery," in *IEEE Transactions on Biomedical Engineering*, vol. 69, no. 4, pp. 1310-1317, April 2022, doi: 10.1109/TBME.2021.3113841.
23. J. Torner, S. Skouras, J. L. Molinuevo, J. D. Gispert and F. Alpiste, "Multipurpose Virtual Reality Environment for Biomedical and Health Applications," in IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 27, no. 8, pp. 1511-1520, Aug. 2019, doi: 10.1109/TNSRE.2019.2926786.
24. K. Kritsis, M. Kaliakatsos-Papakostas, V. Katsouros and A. Pikrakis, "Deep Convolutional and LSTM Neural Network Architectures on Leap Motion Hand Tracking Data Sequences," 2019 27th European Signal Processing Conference

(EUSIPCO), A Coruna, Spain, 2019, pp. 1-5, doi: 10.23919/EUSIPCO.2019.8902973.

25. G. Zhang, J. Liu, H. Li, Y. Q. Chen and L. S. Davis, "Joint Human Detection and Head Pose Estimation via Multistream Networks for RGB-D Videos," in IEEE Signal Processing Letters, vol. 24, no. 11, pp. 1666-1670, Nov. 2017, doi: 10.1109/LSP.2017.2731952.
26. Parameswari, V., and S. Pushpalatha. "Human activity recognition using SVM and deep learning." *European Journal of Molecular & Clinical Medicine* 7, no. 4 (2020):