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

School of Computer Science and Statistics

Human Activity Recognition, Tracking and Evaluation – A Machine Learning Approach

Manu Prasannakumar

MSc Computer Science - Data Science

August 19, 2022

A dissertation submitted in partial fulfilment of the requirements for the degree of
MSc Computer Science - Data Science

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work. A section of this study has been researched upon as a part of Security and Privacy (CS7NS5) module. I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at http://www.tcd.ie/calendar.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at http://tcd-ie.libguides.com/plagiarism/ready-steady-write.

Signed: Manu Prasannakumar Date: 19-08-2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Signed: Manu Prasannakumar Date: 19-08-2022

Abstract

Human Action Recognition is very significant in the field of computer vision as it allows the machines (computers) to recognize what humans are doing and can take necessary actions in response. Although Human Action Recognition involves a lot of challenges, it has gained huge importance over the years and these algorithms are already employed in various fields such as Human Computer Interaction, Smart Homes, security, transportation, medication management and education. Because of its positive impact these techniques have been deployed for video processing for human action analysis in many more fields. However, the existing algorithms based on videos needs to improve for it to be employed reliably for applications such as assistive robotics and augmented reality gaming. Here, we are trying to achieve the same for a specific set of actions such as Jog on the spot; Jump forward; Jump high; Jump sideways. This work is focussed on researching various algorithmic approaches with machine learning models to not only recognise these actions but also calculate the results of these actions based on specific criteria. The criteria for Jump High and Jump Forward is the distance jumped; for Jump Sideways and Jog is the number of Jumps or Jogs respectively. This is addressed by two approaches. The first approach focusses on deriving joint based features of the human for each frame in the videos and training classification algorithms on all the frames and calculating the results as per criteria. The second approach focusses on directly running deep convolutional neural networks with and without transfer learning on all frames of the videos and classify them as per their labels and calculate the results based on the criteria. The video dataset used for this experiment is collected as part of a research conducted in [19]. The experimental results shows that the first approach is more accurate, more efficient and less complex.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Inmaculada Arnedillo Sanchez, for all the guidance and support she has provided me throughout this project. I am grateful to the Computer Science and Statistics Department of Trinity College Dublin for providing me with the necessary resources to complete my thesis.

I would also like to thank Benoît Bossavit for providing help with the original research related to the data set used in the thesis. I am also grateful to my second reader, Prof. Jason Wyse for giving useful insights which helped me improve my thesis.

Last but not the least, I would like to thank my parents Prasannakumar M and Dr. S Geethamony Amma, and my brother Gokul Prasannakumar for supporting me throughput my master's degree.

Contents

List of Tables

List of Figures

1 Introduction

In this era, when technology is advancing at a fast pace, there is a huge amount of video data being generated from various avenues such as CCTV, social media, smartphones, etc. It becomes necessary to analyse and extract useful information from these videos. The recognition of human actions would be one of the most important applications, which can be utilized in various fields such as autonomous vehicles, sports analysis, shopping behaviour analysis, surveillance systems, etc. Hence, the field of Human Action Recognition has gained lot of attention in the recent years.

Human Action Recognition is the process of analysing and detecting actions performed by humans from video sequences. The core objective would be to analyse videos and label each section of the videos according to the behaviour of the human. This involves many sub tasks such as categorizing what action is performed and locating where the action is performed in the video. Complex activities need to be further broken down into simpler activities and recognised using a combination of machine learning and pattern recognition systems. There are challenges involved such as group activities, multi subject interactions and complex backgrounds.

Human Action Recognition has been more effective in the recent years with the advancements in deep learning. The recent human action recognition methods [1] [2] [3] utilized deep Convolutional Neural Networks (CNNs), Graph Convolutional Neural Networks (GCNs) [4] [5] and Recurrent Neural Networks (RNNs) [6] [7]. All these methods can recognize different sets of human actions with a good accuracy once trained on enough videos.

1.1 Motivation

As deep learning and computer vision techniques are advancing at a fast pace, the applications of human action recognition have been extended to many new domains such as anti-terrorist and anti-crime systems, life logging and assistance services, etc. So, it has become necessary not only to recognise actions from these videos but also to calculate the result of these actions so that these systems can run autonomously with minimal human intervention. This becomes complex as different actions might have different results based on the scenarios in which the action is performed. In this paper we are focussed on four actions – Jump High, Jump Forward, Jump Sideways and Jog on the Spot. The criteria we have set for calculating results are distance jumped for Jump

High and Jump Forward, and number of repetitions for Jump Sideways and Jog on the Spot.

This study can help in the advancement of automation of systems as machines become aware of what action is performed, what are the results of these actions and can take necessary responses based on situations.

1.2 Research Question

How can different machine learning models such as Random Forest, XGBoost, Deep Convolutional Neural Networks and Long Short Term Memory Networks be used in combination with pattern recognition algorithms to detect four specific human actions (Jump High, Jump Forward, Jog on the Spot and Jump Sideways) and calculate the results of these actions such as the distance jumped and the number of repetitions of the actions, from video data collected as part of the research conducted in [19]?

1.3 Research Objective

The objectives of this research to address the research question are the following:

1. Classify the videos into four actions using Convolutional Neural Networks and Long Short-Term Memory Networks.

2. Label each frame of the videos to an intermediate stage of the action as defined by a heuristic.

3. Train four classification models based on different approaches using joint features and direct images on these labelled frames corresponding to each of the four actions.

4. Use the trained classification models to predict the intermediate stage of each of the videos.

5. Calculate the necessary criteria such as distance jumped and the number of repetitions for each action using the defined functions.

6. Evaluate the criteria calculated by the algorithm with the criteria values previously noted during the research.

1.4 Research Challenges

1. The identification of heuristics to label the intermediate stages is difficult.

2. The dataset contains many bad videos (videos where children are not performing the actions mentioned properly).

3. The hyperparameter tuning is challenging as it requires a balance between underfitting and overfitting.

4. Evaluation of the criteria for each action is difficult as it cannot be automated.

1.5 Thesis Overview

This research focusses on recognizing actions and calculating the results of these actions based on certain criteria. The video dataset used for this experiment is collected as part of a research conducted in [19]. The four actions which are evaluated in this work are Jump High, Jump Forward, Jog on the Spot/Run and Jump Sideways. The result measured for Jump High and Jump Forward is the distance jumped and for Jog and Jump Sideways is the number of jogs or jumps.

The videos are classified into the four different actions using Convolutional Neural Networks and Long Short-Term Memory Networks. The four actions are further analysed separately for calculating the criteria values using two approaches. In the first approach, the joint features of the human are extracted and machine learning models such as Random Forests and XGBoost are used to train on the joint features of all the frames of all videos with their intermediate labels as outputs. The intermediate labels correspond to the position the human is assuming in the frame. The trained classifier is used to predict the labels of the frames of a new video and specifically designed algorithms are used to calculate the result based on criteria from these predicted labels. In the second approach, all the frames are directly passed through a deep convolutional neural network and are trained to classify the intermediate labels. A convolutional network designed from scratch and another network implementing transfer learning using EfficientNetB7 is both trained and evaluated. The designed algorithms are applied on the predicted labels of each video to calculate their results as per the criteria. The results of the two approaches are compared with the criteria values recorded for the research for all the four actions.

1.6 Thesis Structure

The thesis is organized as follows. In Chapter 2, the background and related works of the thesis are discussed. Chapter 3 discusses in brief the working of the machine learning methods used in this project. Chapter 4 explains the implementation of the project including data pre-processing and the system architecture. Chapter 5 presents the security and privacy considerations involved in this project. The evaluation of the models and the project results are discussed in Chapter 6. The study concludes with Chapter 7 which discusses conclusion and future works.

2 Background and Related Work

A review of literature on the various techniques and technologies involved in Human Action Recognition is conducted in this chapter. Since the methods used in this paper employ joint detection (pose detection) and Neural Network architectures, some relevant researches in Human Joints (Pose) Detection and Human Action Recognition using deep learning architectures are discussed below.

2.1 Human Pose Detection

The most widely used and accurate libraries for joint (pose) detection are OpenPose and MediaPipe. The studies behind these libraries are discussed below.

A real time approach for detecting poses of multiple people in an image which is implemented in OpenPose is discussed in [8]. OpenPose works by using Part Affinity Fields (PAFs) to detect human posture. The pairwise relationship between different body parts is encoded as a set of flow fields and these are the PAFs. The dataset used for this study is MPII human multi-person dataset, COCO key point challenge dataset and a custom foot dataset which is a subset of 15k images annotated from the COCO key point dataset. The image is first passed through the first 10 layers of a pretrained model called VGG-19 to extract its features. These features are then passed through a set of Part Confidence Maps. These are 2D confidence maps for body part locations and a map of each joint location is generated. The features are also passed on to a set of Part Affinity Fields. These are a set of 2D vector fields which will capture the association between the body parts. The output from Part Confidence Maps and Part Affinity Fields are processed by a greedy algorithm to get the joint locations connected which can be used to estimate the pose of the person. OpenPose can hence process and image and output the 2D coordinates and confidence of 18 body joints in this manner. The sample model of 18 body joints is shown in Figure 2.1.

MediaPipe is a library developed by google research that offers customizable Machine Learning Solutions for live and streaming media. The functionality relevant to this research is the MediaPipe Pose, which is a solution for body pose tracking and inference of 33 3D body landmarks as shown in Figure 2.2. This is based on a new approach to

human body pose estimation called BlazePose[1] which is utilized to create 3D human modelling pipelines in [9]. This approach has two stages, the first stage detects the region of interest (ROI) within the frame and the second stage extracts the 33 key point from this ROI by predictions. The detection stage is an extension of the face detection model called BlazeFace which is discussed in [10]. A regression model approach is used to predict these key points as shown in Figure2.3. A heatmap and offset loss is used to train the first two parts of the network and the heatmap output is removed while training the third part of the network to get the embeddings for the joint locations. The model was evaluated on geo-diverse datasets of various poses and the Percent of Correct Points with 20% tolerance (PCK@0.2) was 97.2. The tolerance indicates the allowed 2D Euclidean error in distance from the predicted values as a percentage of the person's torso size.

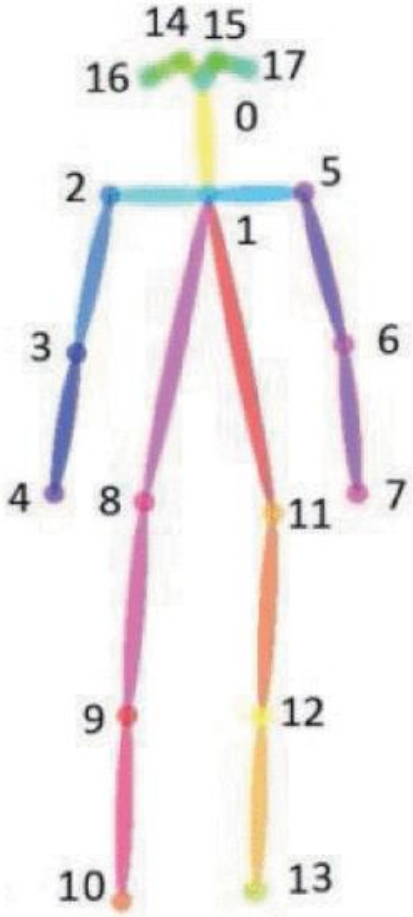

Figure 2.1: Model of 18 joints given by OpenPose

[1] https://ai.googleblog.com/2020/08/on-device-real-time-body-pose-tracking.html

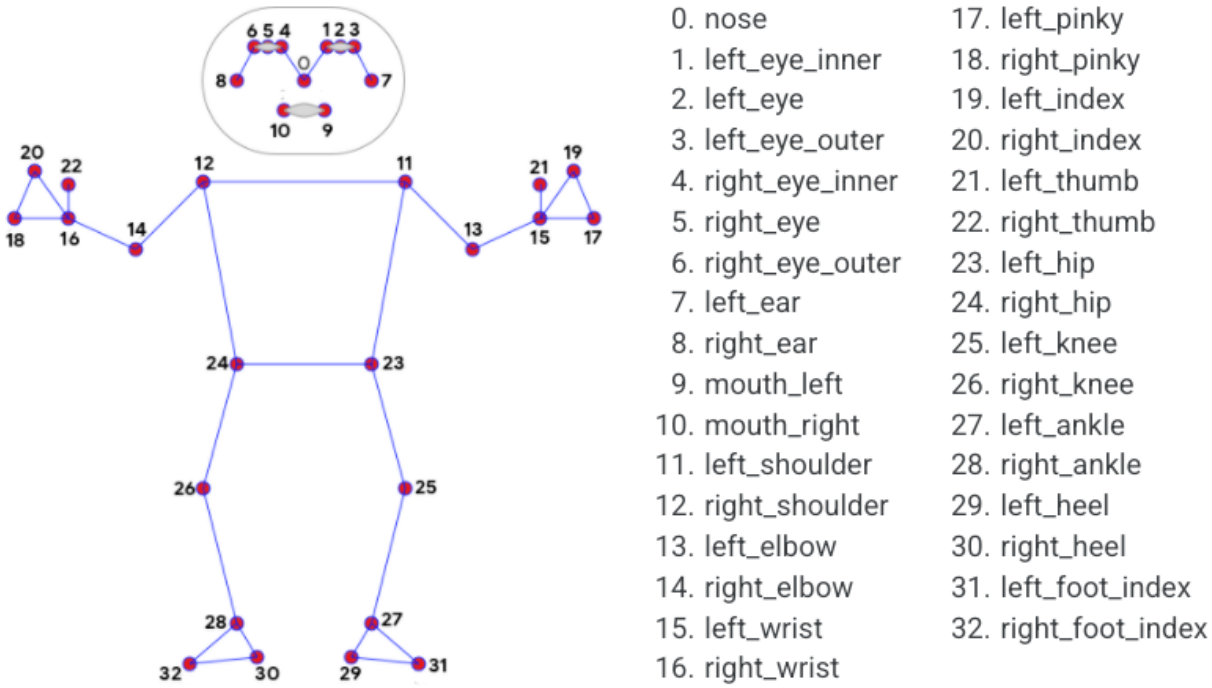

Figure 2.2: 33 Pose Landmarks given by MediaPipe

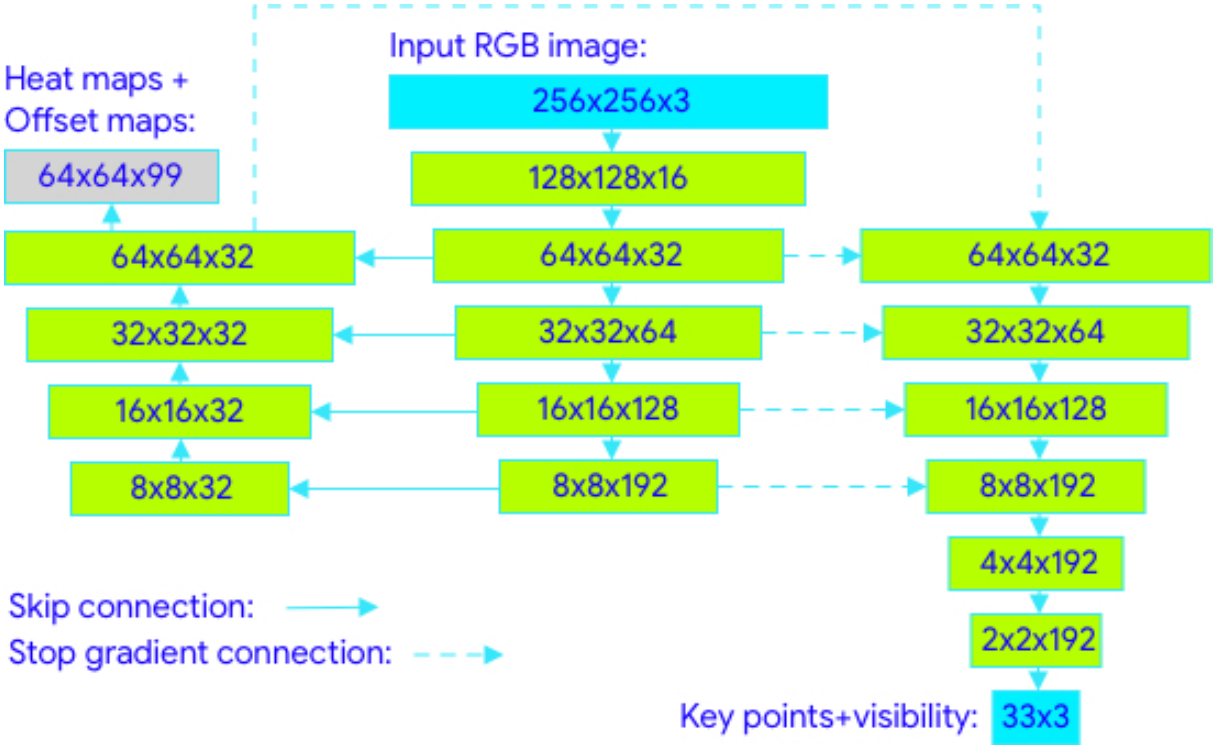

Figure 2.3: Architecture of Tracking network in BlazePose

2.2 Human Action Recognition

There are several researches conducted in the field of Human Action Recognition. This is because of the huge potential of its applications in various fields. Few of the latest and relevant works in this regard are discussed below.

A study of an action recognition system, which is skeletal based and uses computer vision feature and joint trajectory images to model human action, is carried out in [11]. The dataset used in this study is the UCF11 dataset. This dataset contains 11 action categories. Each video in the dataset was processed using OpenPose and the position information of skeletal joints in all frames were stored in a JSON file in the same order as in the video. This joint data is further normalised and filtered to rectify errors and deviations. The data is converted into a matrix which is used to form a character graph which traces the trajectory of the skeletal joints. The trajectory of skeletal joints is drawn by generating Histogram of Oriented Gradients feature vectors. These features are further passed as input for training to a linear Support Vector Machine Classifier which uses a radial basis function as the kernel. The trained SVM model can classify the video into 11 action categories with 60% accuracy.

One of the biggest challenges of human action recognition is cluttered backgrounds. To recognise human actions after removing cluttered backgrounds a new method which estimates human poses by making the actions predictions aware of the context is discussed in [12]. The dataset used in this study is the NTU RGB+D [13] dataset. This dataset contains 58000 videos with 60 action categories. The four main components of the method described in this study are pose estimation module (P-module), skeletal based action recognition module (S-module), appearance-based action recognition module (A-module) and the aggregation module. The block diagram of this architecture is shown in Figure 2.4. The input RGB videos to the system are subsampled to get 32 frames and these frames are fed to the Pose Estimation Network which will generate the 3D body join coordinates. This is fed to the skeletal based action recognition network. The convolutional features and the body joint heatmaps generated during the process of pose estimations are fed to the appearance-based action recognition network after applying the Kronecker product operation. The results from the appearance based and skeletal based action networks are fused by the aggregation module using element wise summation, element wise multiplication and feature map concatenation to generate the prediction. This method reached accuracy up to 95.41% and 91.76% on cross view and cross subject evaluation methods as described in [13].

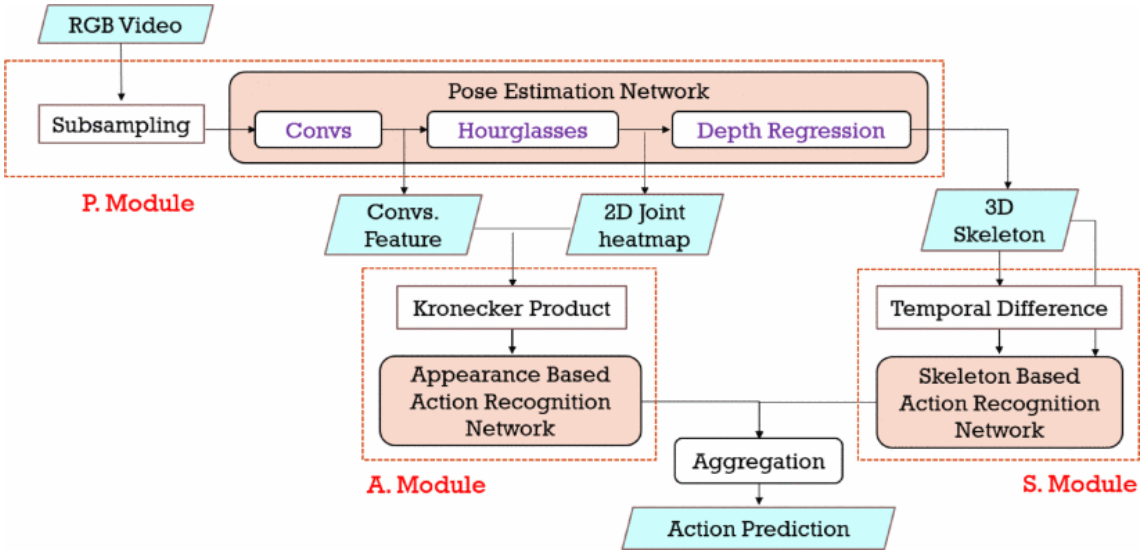

Figure 2.4: Network Architecture for Action Recognition

A more recent study [14] approaches the human action recognition problem as a process of representing an action as a collection of representative frames called exemplars and actions are modelled by the distance to those representative frames. This study is conducted on the KTH dataset. This dataset consists of 600 videos with 6 types of actions (boxing, handclapping, handwaving, jogging, running, and walking). The video input is divided into its frames and each of these frames is converted to grayscale images. These grayscale images are further converted to its binary form. The frame background value is subtracted from the whole image and a blob of the required human is cut out. An example of this is shown in Figure 2.5. The distances of these frames with the exemplars of each action are calculated using silhouettes or edges and the exemplars closest to the frames in the videos on an average is the action category for that video. The neural network model is trained to minimise the distance between all frames in a video and the exemplars thereby finding the correct exemplar for categorising each video into different actions. This model was able to classify all actions efficiently where temporal information was not required. In actions where temporal information is required such as getting up and sitting down, where both actions would look the same without the order of frames, this method fails.

One of the major issues in video surveillance systems is that action recognition in such harsh environments such as low light environments is challenging. This has been tackled in [15]. The dataset used for this study is KETI RGB+D. This dataset consists of 1000 videos and 13 classes of actions which are based on video surveillance applications. The deep neural network architecture used for this study is shown in Figure 2.6. The Cov3D layers apply 3D convolutional filters on the inputs and extract different features as the

model trains on the data. The MaxPooling layers will average the values from the convolutional layers across different regions enabling the model to detect local features at different parts in each frame of the videos. These are performed in the spatiotemporal domain. There are multiple such blocks of Conv3D and MaxPooling layers which will extract all the localized features from the frames and finally feed it to dense layers which will convert the large number of features to the number of categories to which the actions are to be classified, the weights of which will be learned during the training process. The model was able to achieve an accuracy of about 61.5%.

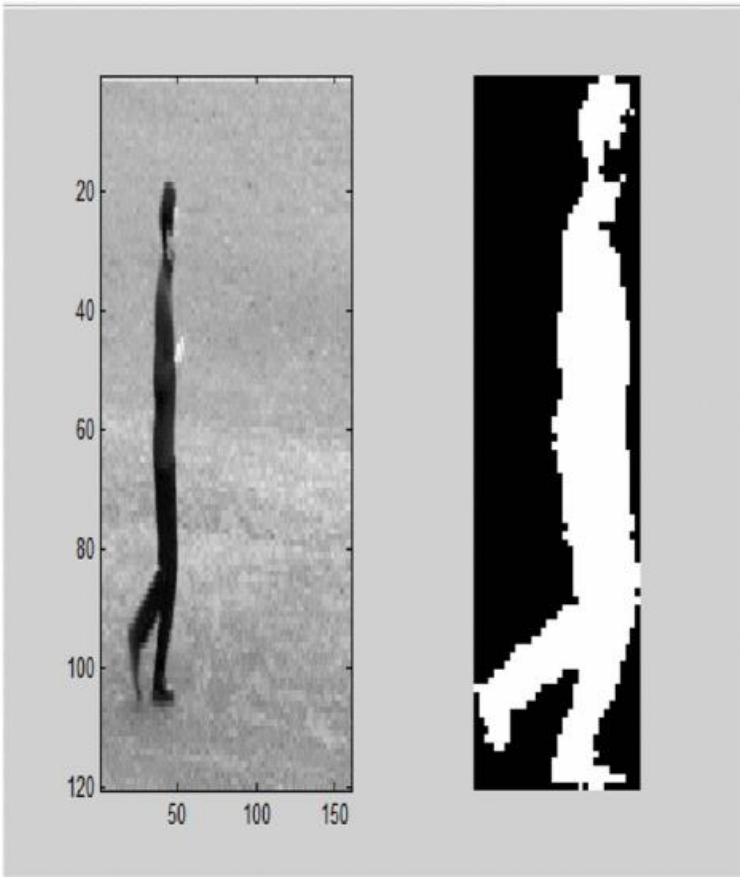

Figure 2.5: Background removal

An image based human action recognition system using Convolutional Neural Network (CNN) architecture and transfer learning is discussed in [1]. This study has been conducted on the Stanford 40 and PPMI datasets. The Stanford 40 dataset contains 9532 images with 40 different human action categories. The PPMI dataset contains images of humans using musical instruments of 12 different types. The process defined in this study is divided into two stages – 1) Data Augmentation and Image Pre-processing and 2) Transfer learning. In the first stage the image is resized to a standard size based on the input size to the model in the transfer learning phase. The resized image is then

passed as input to a data augmentation pipeline. In the pipeline many transforms such as perspective skewing, rotation, vertical flipping, horizontal flipping and gaussian noise are applied to the image to generate multiple augmented images. The augmented images are further normalised to keep the values close to 0. The next stage is the transfer learning stage where pretrained networks such as ImageNet, ResNet or VGG are used. The last layers of these pretrained models are removed and the augmented images are passed through these pretrained networks to generate features from the images. These features are further passed to two blocks of Convolutional and MaxPooling layers, followed by a Dense layer and Softmax layer. The Softmax layer will classify the images into different categories. The network using the ResNet model achieved the best results on both the datasets with an accuracy of about 84.615%, precision of 85.036% and recall of 84.4% on the PPMI dataset and an accuracy of about 95.98%, precision of 96.19% and recall of 95.75% on the Stanford 40 dataset.

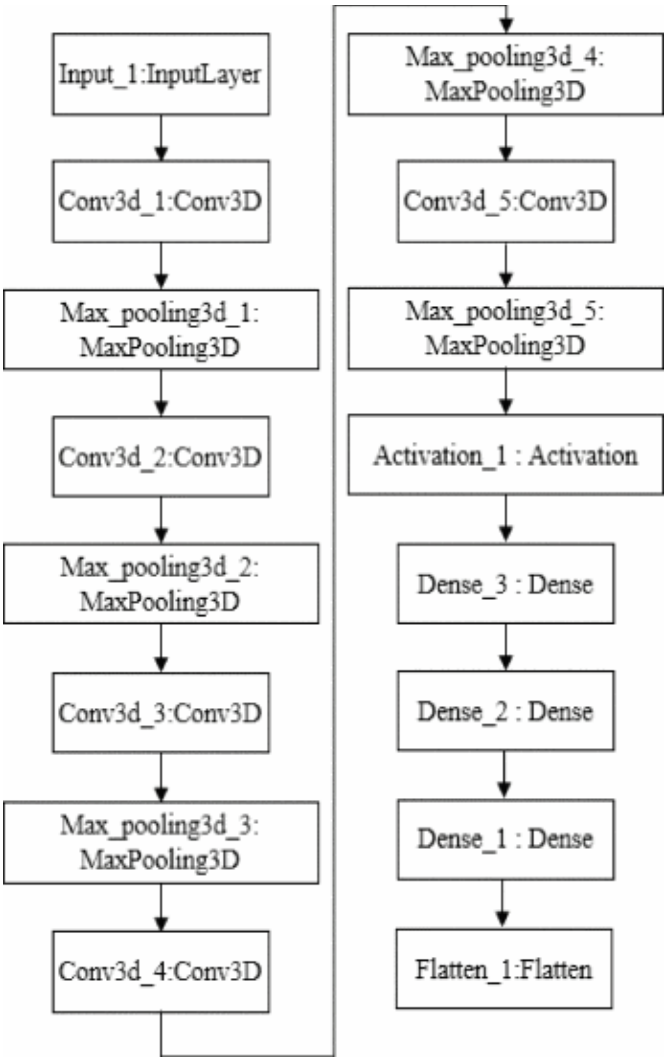

Figure 2.6: Network Architecture for 3D Action Recognition

One of the challenges in deep convolutional networks is information loss occurring due to increased depth of the network. [16] handles this challenge by implementing a multi-level feature fusion mechanism which utilizes the outputs of the middle CNN layers in the final output so that the information loss from those layers is minimal. The study is conducted on the KTH dataset. This dataset consists of 600 videos with 6 types of actions (boxing, handclapping, handwaving, jogging, running, and walking). The novel concept in this paper is the multi-level feature extraction and features at each level are extracted using a pretrained network called VGG16. The features from the middle layer VGG16 blocks are passed through a Convolutional Autoencoder models (CAE). The CAE model encodes the middle layer features, maps it to another feature space and decode it while minimizing the error. This makes the process more efficient by allowing the model to cache only the CAE feature errors during training process and not the complete features from the middle layers. The reconstructed (decoded) outputs from the middle layers and the output from the final layer are fused and the results are minimized during training using a join optimization module. The joint optimization module trains the CAE model where the loss function is the error between the features from middle layers and the reconstructed features. This module also trains the final classification model based on the fused inputs and cross entropy function. The schematic diagram of the architecture is shown in Figure 2.7. This architecture attained an average accuracy of about 92.54% for all the actions.

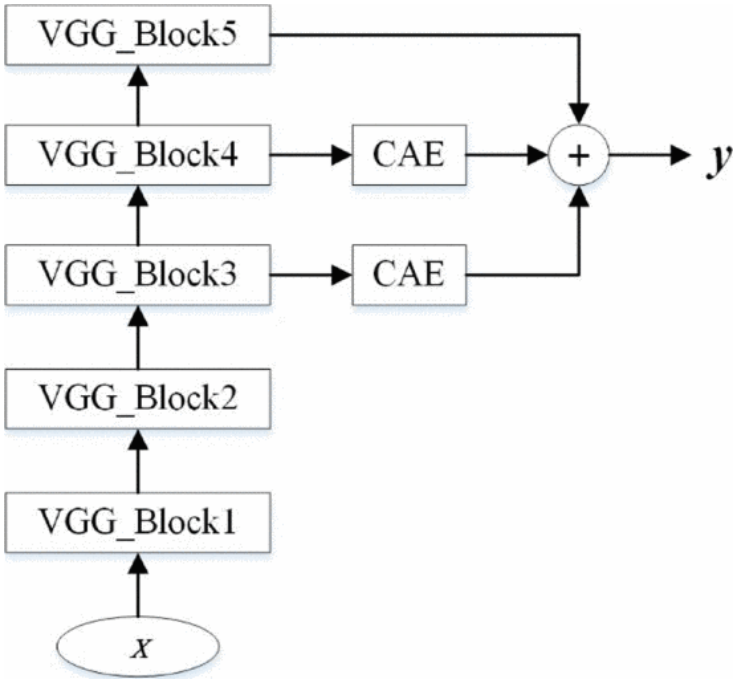

Figure 2.7: Schema of fusion based architecture

2.3 Summary

A combination of LSTM and CNN layers in a neural network to capture the framewise features and temporal features is used to classify the videos into the four actions (Jump High, Jump Sideways, Jump Forward and Jog on the Spot). Each of the actions is further analysed separately to calculate the results of the actions. For this framewise image-based recognition systems are utilized. Each frame is classified into a predetermined label class which shows the current intermediate stage of the action being performed. These classes are learned from the images using two approaches. In the first approach MediaPipe is used to generate the skeletal join features of the people in the frames and classification model such as RandomForest and XGBoost are trained on these joint features with the intermediate labels as outputs. In the second approach, the frames are directly trained either on a custom deep convolutional neural network or a combination of pretrained network such as EfficientNet and convolutional neural networks with the intermediate labels as outputs. The labelled frames are further processed by algorithms to detect the start and end of actions and the distance covered by the action or the number of repetitions of the action based on the criteria of each action.

3 Methods

3.1 Convolutional Neural Networks

A convolutional neural network is a type of neural network that has significant advantages in processing data such as images. A sample architecture of a convolutional neural network is shown in Figure 3.1. A typical architecture consists of convolutional layers, pooling layers and fully connected layers.

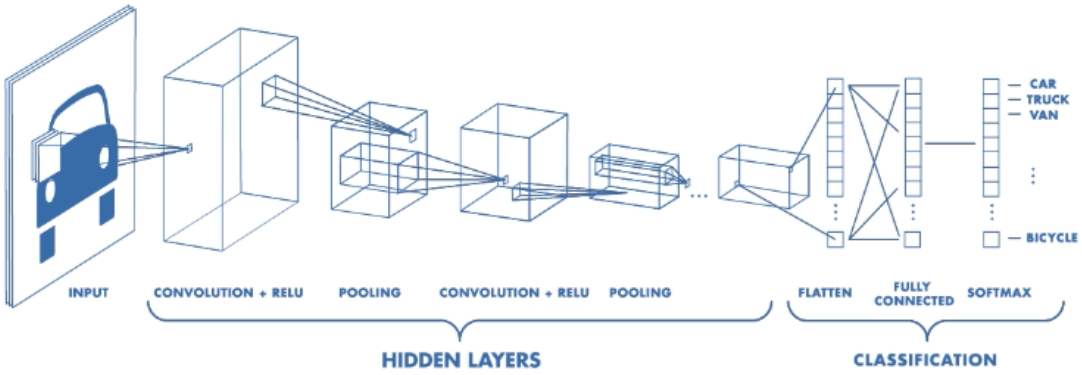

Figure 3.1: Sample Convolutional Neural Network Architecture

A convolution layer performs the convolution operation which is the dot product of two matrices where one matrix is the convolutional filter (this is the kernel whose values will be trained during the model training process) and the other is the receptive field (the region of the image on which the convolution operation is performed at each step) of the image. The filter slides across the image during the training forward pass and converts the image pixel values to an activation map based on the dot product result. An example of the operation is shown in Figure 3.2. There are multiple such filters in a convolutional layer which will capture multiple features from the images. A pooling layer can reduce the computations required by reducing the size of the outputs. This is done by replacing the outputs at pixel locations by the summary statistics of the neighbouring pixels. This is also done using filters of smaller sizes (size depends on the region we need the pooling applied on). The summary statistics used for pooling can be functions such as average, weighted average, maximum, etc. A fully connected layer is a layer of neurons which has connection with all the neurons in the previous and next layers. The result of these layers is the product of matrix multiplication of the previous layer outputs

and the weights associated with the connections between the layers. These weights are learned during the training process.

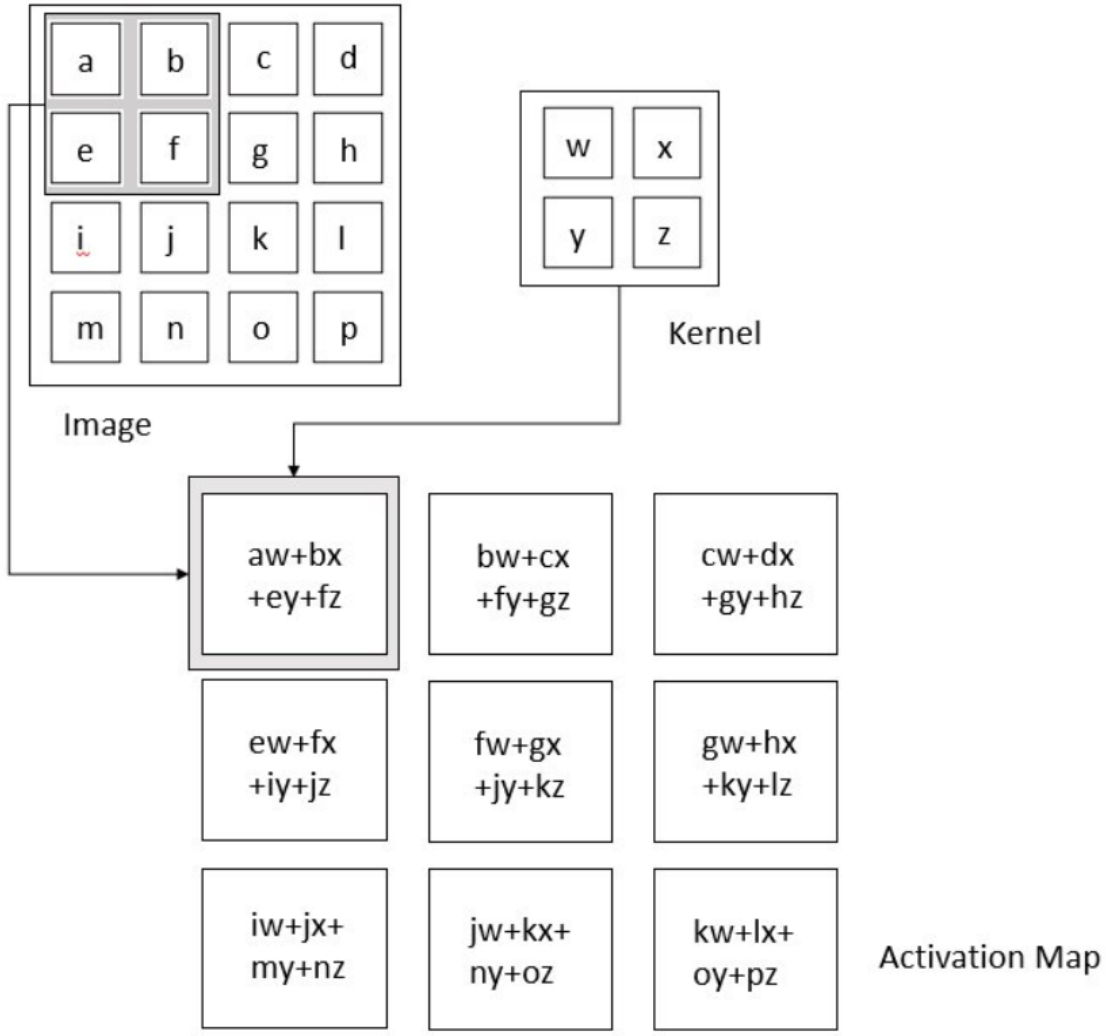

Figure 3.2: Convolution Operation Example

3.2 Long Short-Term Memory Networks

Long Short-Term Memory (LSTM) network is a type of Recurrent neural networks which uses LSTM layers as part of its hidden layers to capture sequence information from sequence data such as audio, video, etc. LSTM networks are one of the most powerful Recurrent Neural networks since LSTMs can capture long term dependencies between sequence information such as different frames of a video. An LSTM unit consists of three parts as shown in Figure 3.3. The Forget Gate is the part which will decide whether the information from the input is relevant or not and consequently retain it if it is relevant. A function f_t is calculated using the formula $f_t = \sigma (x_t * U_f + H_{t-1} * W_f)$. Here, x_t represents the input from current values in the sequence, U_f represents the weights of the inputs, H_{t-1} represents the hidden state values of the previous input in the sequence, W_f is the

weights of the hidden state and σ represents the application of sigmoid function over the calculated result. The f_t will have a value between 0 and 1. The product of this value and the cell state value of previous input in the sequence is calculated. If this value is 0, the information from current input in the sequence is forgotten and if it is 1, the value will be retained for use in the final outputs. The input gate extracts new information from the input to be used by the cell. The equation governing the input gate is $i_t = \sigma (x_t * U_i + H_{t-1} * W_i)$. Here, x_t represents the input from current values in the sequence, U_i represents the weights of the inputs, H_{t-1} represents the hidden state values of the previous input in the sequence, W_i is the weights of the hidden state and σ represents the application of sigmoid function over the calculated result. Based on this new information is calculated as $N_t = \tanh (x_t * U_c + H_{t-1} * W_c)$. This new information used to update the cell state as $C_t = f_t * C_{t-1} + i_t * N_t$. Here, C_t and C_{t-1} represents the cell state values from the current and previous inputs in the sequence respectively. The output gate will pass the updated information from current input in the sequence to the next input in the sequence. The output gate equation is $o_t = \sigma (x_t * U_o + H_{t-1} * W_o)$. The current hidden state value will be calculated using the formula $H_t = o_t * \tan (C_t)$. hence this hidden state has managed to function as a feature for long term memory. The output of the current input in the sequence is calculated as Output = Softmax (H_t).

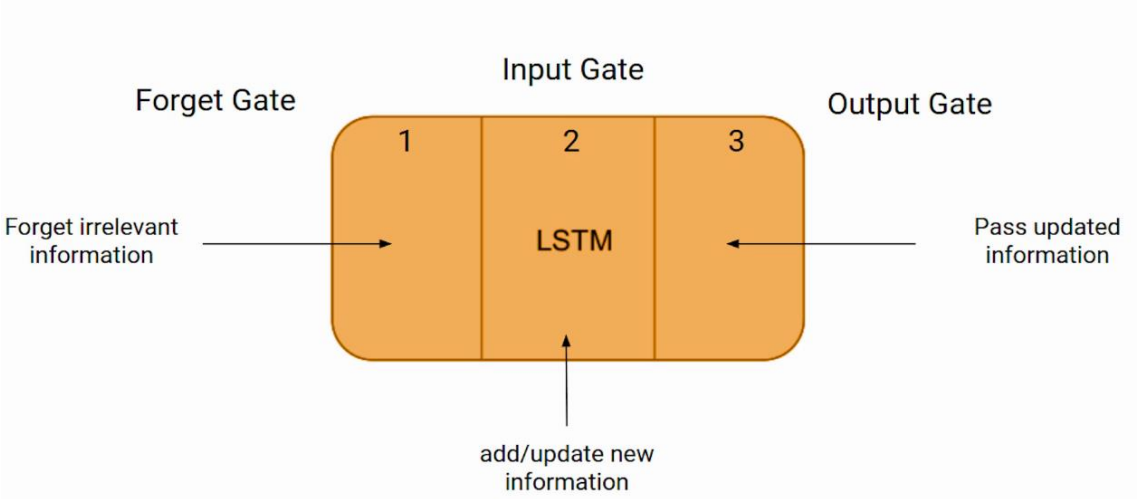

Figure 3.3: LSTM Unit

3.3 Transfer Learning

Transfer Learning is the process of using pretrained neural networks trained for some classification tasks as part of the neural network for another classification task. In this approach, the final layers of the pretrained model will be removed as these will be specific to the classification task being performed and hence cannot be used for another classification task. The trained middle layers would have extracted useful features from the input (such as edges of images, objects, etc) which can be used as input to other convolutional or fully connected layers to classify for the new classification task. This will reduce the time and computational resources required for training deep neural networks as we have eliminated many middle layers in the neural network by using the

fixed weights and layers from the pretrained model. The pretrained model used in this study is EfficientNetB7. This is a pretrained model developed by Google AI. This model has higher accuracy (84.3% on ImageNet dataset), efficiency (the number of parameters is 8.4x lesser) and speed (6.1x faster) as compared to the other pretrained models. This network is a convolutional neural network and has a scaling factor which compounds uniformly with the size of the input image.

3.4 Random Forest Classifier

Random Forest classifier is an ensemble machine learning model. This means that it uses a group of models rather than a single model. Random Forest uses an ensemble of Decision Trees. A decision tree is a type of machine learning model that takes all the input data points and tries to split into branches based on the Gini Index. The decision tree makes new branches until most of the data points are classified into the required classes or the hyperparameters are satisfied. An example of a decision tree classification is shown in Figure 3.4.

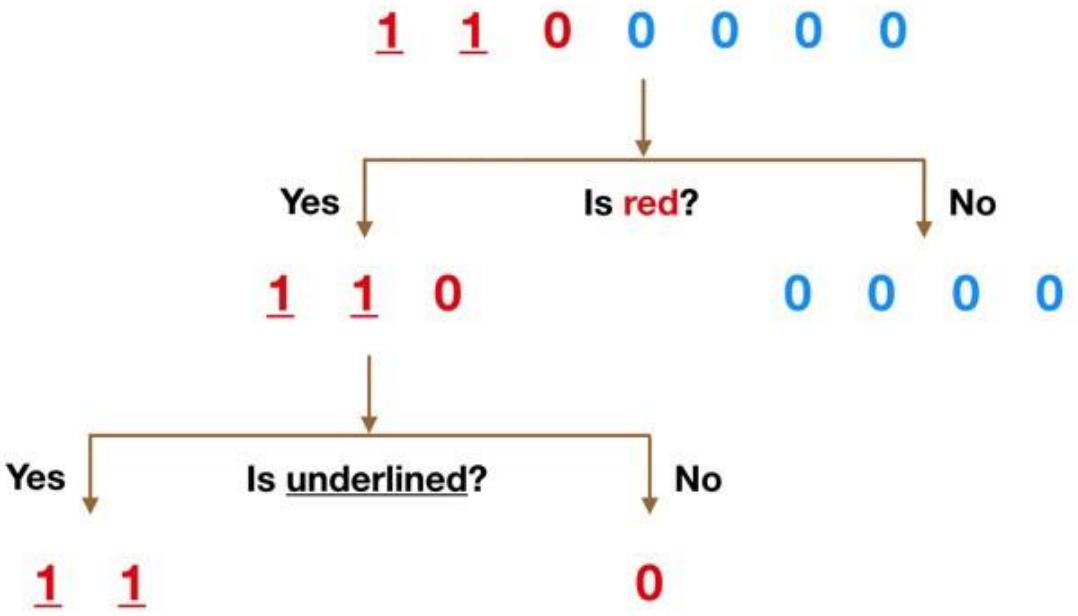

Figure 3.4: Decision Tree Example

In Random Forest, a group of decision trees will be trained on the data with the bagging approach. Decision trees are prone to overfitting and tree structures are sensitive to the input data. This is resolved using the bagging approach. Bagging is the method of sampling a random subset of data from the whole data set with replacement. This means that each decision tree in the Random Forest will be trained on a subset of data fetched from the original dataset with replacement. This ensures that each of the decision tree in the random forests are uncorrelated to each other. Another approach called feature randomness is also used by random forests to further improve the variation among the decision trees. This means that each decision tree will select only a random set of features from feature space for training. These two approaches ensures that all the decision trees in the ensemble have variations among them. All the decision

trees will be trained in this way and each of the model will classify the input data. The final output class predicted from the random forest will be the class predicted by most of the decision trees. The training process will try to minimise the number of misclassifications by the ensemble rather than each decision tree.

3.5 XGBoost Classifier

XGBoost Classifier is also an ensemble of decision trees. This model uses gradient boosting technique. In boosting, a strong classifier is built using a large number of weak classifiers in a sequential manner. The first model is trained on the training data. It predicts the classes for all the training data. The second model then tries to minimise the errors of the first model. Then new models are built in the same way until all data points are predicted correctly by the ensemble or the hyperparameter for maximum models is satisfied. Gradient boosting uses the boosting technique where each model will try to minimize the residual errors from the predictions using the gradient descent algorithm. In the XGBoost algorithm, decision trees are built in a sequential manner with weights assigned to independent variables in the data. The first decision tree makes the predictions and weights for the predictions of the first model which are incorrect are increased for the second model. This process is repeated for all the models in the sequence until the hyperparameters are satisfied. The final prediction will be made by the considering the outputs of all the individual models with different models having different weights depending on the error of their predictions. The working of XGBoost algorithm is illustrated in Figure 3.5.

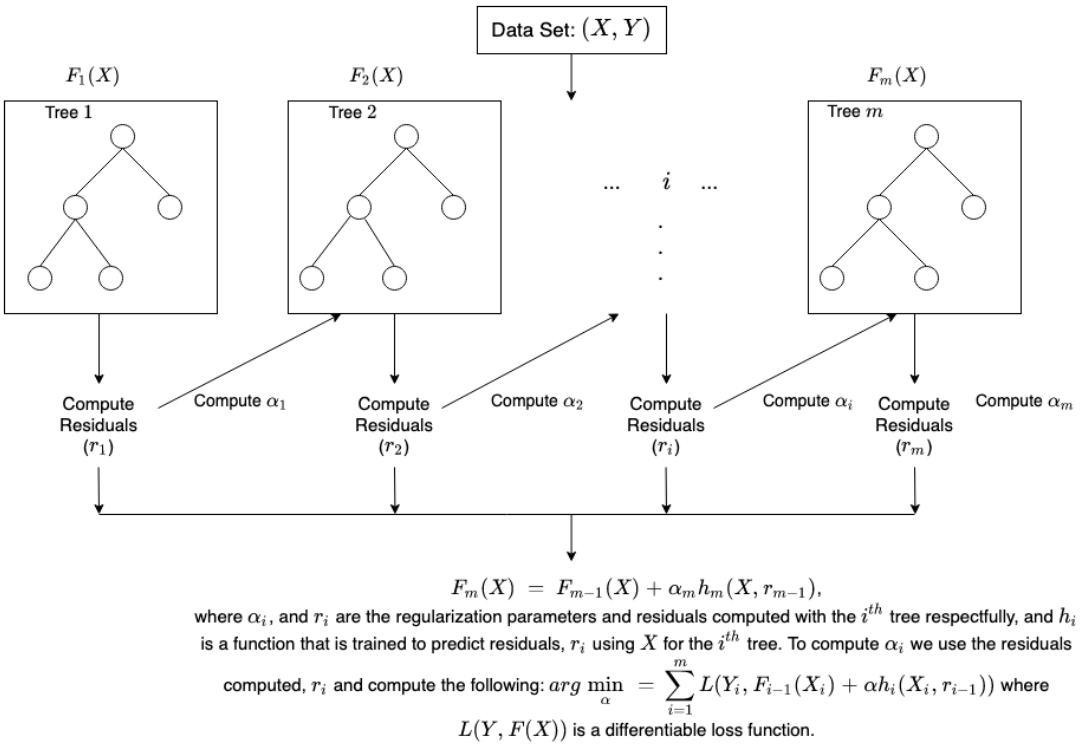

Figure 3.5: XGBoost Workflow

4 Implementation

The different stages of the project implementation are discussed in detail in the below sections.

4.1 Data Collection

There are four different actions that are being analysed in this project. The actions are Jump High, Jump Forward, Jump Sideways and Jog on the Spot. There are 500 videos for each of the four actions. These videos were recorded as part of a research ([19]) conducted for studying the learning capabilities of children at different ages. These videos are the dataset used for this project. Since the video were already collected as part of the mentioned study, there is no separate data collection processes implemented as part of this project.

4.2 Data Pre-processing

There are two stages in the pipeline of the project. The first is a Video Based Human Action Classifier which classifies all the video data into the four categories. The second is a set of four Image Based Video Frames Classifiers for each of the four actions. These classify all the frames of the videos into intermediate statuses for calculating the results of the criteria. The different processing steps are discussed in the below sections.

4.2.1 Video Based Human Action Classifier

The deep learning architecture used for classifying the video into four actions is a combination of Convolutional layers and LSTM layers. The LSTM layers capture the temporal information across the frames. The Time Distributed layer which passes information to the LSTM layers needs to have a shape parameter which captures the number of frames in the video, so that it can pass all the frames of the video to the LSTM layers. This means that all the video should be of the same number of frames. This is not the case as all video are of different durations. So, we fixed the number of frames we have to generate from the videos as 100 for this experiment. To achieve this, all the frames from each of the videos are fetched and depending on the total number of frames a skip factor is calculated as the total number of frames divided by the required

number of frames (100). The frames which have indices that are multiples of these skip factor are moved to a folder with the name of the video. After this processing, all the videos are converted to frames images in their corresponding folders with 100 uniformly spaced frames from the respective video. The videos with frames less than 100 are removed from this analysis. These folders are further split into a train and validation folders with 195 videos in the validation folder and 1745 videos in the train folder. A train.csv file is created with the names of the videos and their action tags in the train folder and a val.csv file is created with the names of the videos and their action tags in the validation folder.

4.2.2 Image Based Video Frames Classifiers

In this approach, each of the four actions are analysed separately. Each of the frames are trained with their intermediate labels. Since this labelling is not already available some processing steps based on defined heuristics are required to label the frames. These steps based on heuristics are discussed for each of the actions in detail. After labelling, other pre-processing steps are required before the frames can be used as inputs to the models. These are discussed separately in the Modelling section for both the Skeletal Joints Based Approach and Frame Images Based Approach.

Each of the frames from the videos is extracted using the OpenCV library. These are passed through the MediaPipe Pose functionality and this will extract all the joint features. The output of the MediaPipe Pose will be a set of x-axis, y-axis and z-axis coordinates corresponding to each of the joints as marked in Figure 2.2. These features are further analysed separately for each of the four actions as follows.

Jump High

The focus of this action is to measure how high the person has jumped vertically. All the videos are of people jumping vertically. Since this action is measured based on the jump above the ground, the joint features of interest from the MediaPipe Pose are 27, 28, 29, 30, 31 and 32 which refer to the points on the foot and ankle. Since the movement in vertical direction is the criteria for this action, the coordinates of these points along the y-axis for all frames in a video are plotted to analyse patterns emerging from the actions. The plots of y-axis values for left leg (27, 29, 31) and right leg (28, 30, 32) of one video are shown in Figures 4.1 and 4.2 respectively.

The plots clearly show a drop in all the values (a valley) whenever a jump is performed (which is four times in this case). These valleys are our region of interest. We need to define thresholds based on these valleys so that we can label frames whenever a Jump

High is made. These features have different range of values in different videos depending on the initial position of the person in the frame. We need to find the frames the feet take off the ground. To account for these, the y-axis coordinates of the ground is calculated as a new feature. The ground value is calculated for each video as the average of the mean of y-axis values of left and right leg for the first 20 frames. This assumes that people are standing on the ground at the beginning of the videos. The features 29_y, 30_y, 31_y and 32_y show a great range of variation in and across videos and hence cannot be used accurately for setting the ground value. The features 27_y and 28_y are more smooth and show lesser variations across videos and hence are used to set the ground value. Hence, the average of the distance of 27_y and 28_y from the ground value is calculated for all the frames of the videos and added as a new feature. The plot of this feature is shown in Figures 4.3.

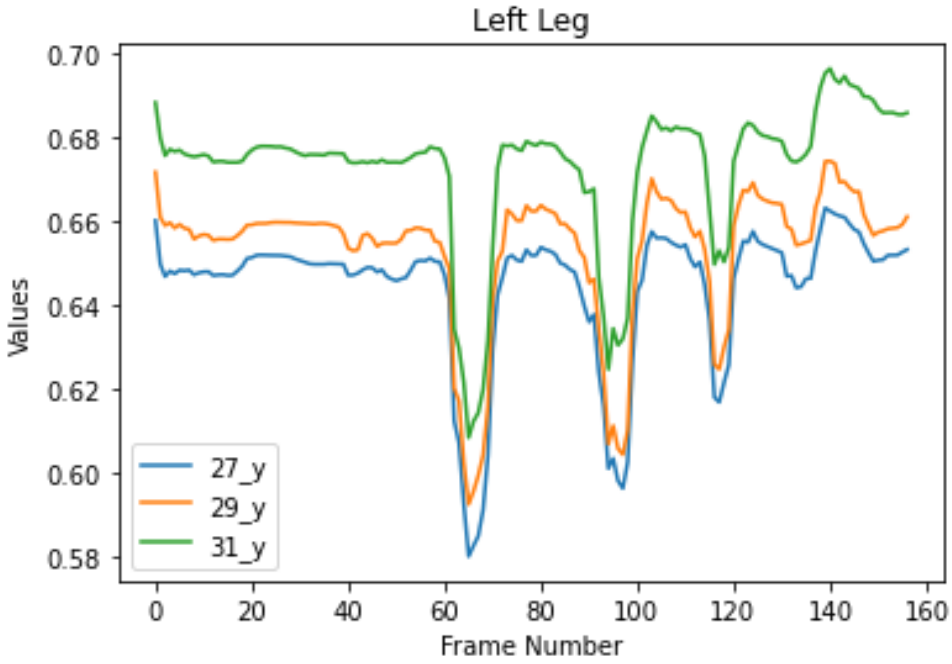

Figure 4.1: Joint Features of Left Leg across Frames

The distance from the ground feature is in the same range across videos and can be used for setting the thresholds. A threshold of 0.01 is set for this feature based on experimentation on videos randomly selected from the sample. When the value of this feature is equal to or greater than 0.01, the frames are labelled as “Jump High” and when the value is below 0.01, the frames are labelled as “Standing”. This is the defined heuristics for the action- Jump High. This labelled data has some frames which are incorrectly labelled “Standing” in the middle of the Jump. This is because of the fluctuations in the joint features from the MediaPipe library. To smoothen this, a rolling

window of 5 frames is taken across all the frames of a video and when there is a frame with label “Standing” in the middle of a window with most of other frames labelled as “Jump High”, this is changed to “Jump High”.

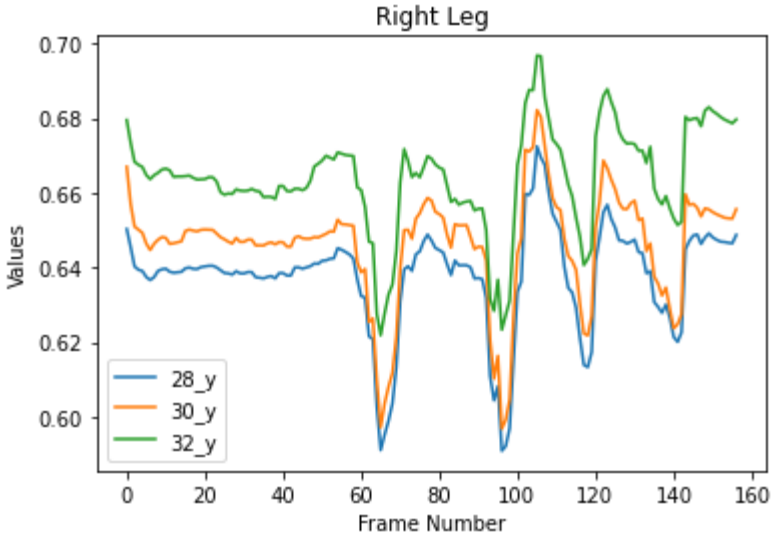

Figure 4.2: Joint Features of Right Leg across Frames

All the joint features generated from the MediaPipe and the status of the frame as labelled earlier are written into a csv file. This is the processed dataset for the skeletal joints-based machine learning approach for the action- Jump High. The frames are saved into a different folder with a name (combination of name of video and frame number). The frame names and the status of the frame as labelled above are saved into a different csv file. This is the processed dataset for the frame images-based machine learning approach for the action- Jump High.

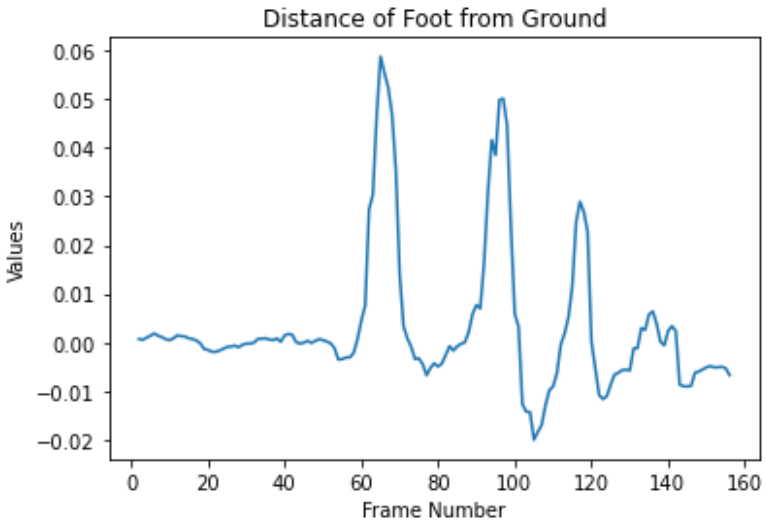

Figure 4.3: Distance of Feet from Ground across Frames

Jump Forward

The focus of this action is to measure how much distance the person has jumped forward. All the videos are of people jumping forward. Since this action is measured based on jumping forward and moving closer to the camera, the joint features of interest from the MediaPipe Pose are 11, 12, 23, 24, 25, 26, 27 and 28 which refer to the points on the torso and legs. Since the movement in vertical direction (with respect to the frame) is the criteria for this action, the coordinates of this points along the y-axis for all frames in a video are plotted to analyse patterns emerging from the actions. The plots of y-axis values for right leg (12, 24, 26, 28) and left leg (11, 23, 25, 27) of one video are shown in Figures 4.4 and 4.5 respectively.

The plots clearly show a rise followed by a drop and again a rise in all the values (a peak, followed by valley, followed by a peak) whenever a jump is performed (which is once in this case). This is our region of interest. Thresholds need to be defined based on these regions so that we can label frames whenever a Jump Forward is made. These features have different values in different videos depending on the initial position of the person in the frame. Additionally, all these features show a great range of variation across videos of people of different heights. Hence, these features cannot be used directly for setting the threshold.

Six new features are introduced to further analyse the videos and find a measure suitable for all videos. These are “Hip to Knee Distance”, “Ankle to Knee Distance” and “Shoulder to Hip Distance” for both right and left legs separately. The “Hip to Knee Distance” for right leg is calculated by taking the difference of 24_y from 26_y and the same for left leg is calculated by taking the difference of 23_y from 25_y. The “Ankle to Knee Distance” for right leg is calculated by taking the difference of 26_y from 28_y and the same for left leg is calculated by taking the difference of 25_y from 27_y. The “Shoulder to Hip Distance” for right leg is calculated by taking the difference of 12_y from 24_y and the same for left leg is calculated by taking the difference of 11_y from 23_y. A heuristic from each leg is calculated as 0.2 * (Hip to Knee Distance) + 0.4 *(Ankle to Knee Distance) + 0.4 * (Shoulder to Hip Distance). The average of this heuristic value from both the legs are taken as the final heuristic value. The plot of this final heuristic value is shown in Figure 4.6. This smoothened out a lot of fluctuations in the joint values from the MediaPipe library.

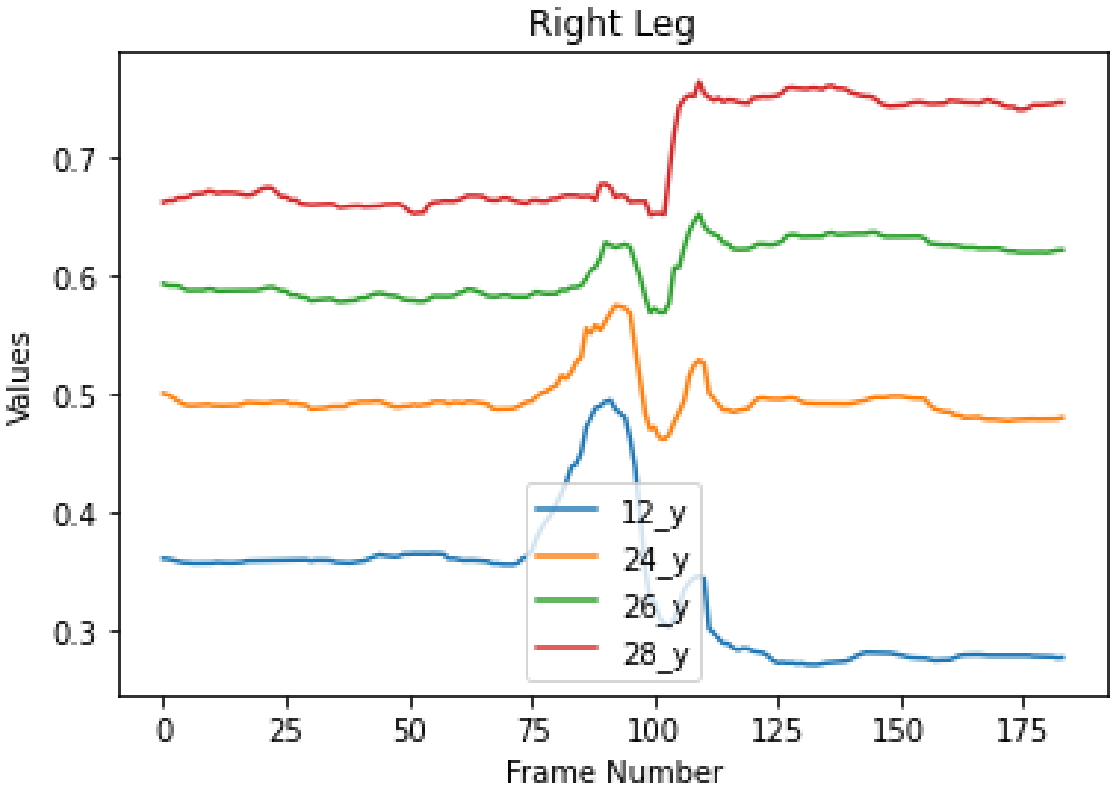

Figure 4.4: Joint Features of Right Leg across Frames

Furthermore, a new feature which measures the change in this heuristic value is generated. This is calculated as the heuristic value of previous frame subtracted from the heuristic value of the current frame. This feature gives a clearer picture on the when the Jump starts. The plot of this feature is shown in Figure 4.7. This feature which measures change in heuristic value is in the same range across videos and for people of different heights. Hence, this can be used for setting the thresholds. A threshold range of [-0.005, 0.005] is set for this feature based on experimentation on videos randomly selected from the sample. When the value of this feature is in the range of [-0.005, 0.005], the frames are labelled as “Standing” and when the value is outside this range, the frames are labelled as “Jump Progress”. This labelled data has some frames which are incorrectly labelled “Standing” in the middle of the Jump. This is because of the fluctuations in the joint features from the MediaPipe library. To smoothen this, a rolling window of 4 frames is taken across all the frames of a video and when there is a frame with label “Standing” in the middle of a window with most of other frames labelled as “Jump Progress”, this is changed to “Jump Progress”. This smoothening is repeated two more times to get the smoothened and correct labels for the frames.

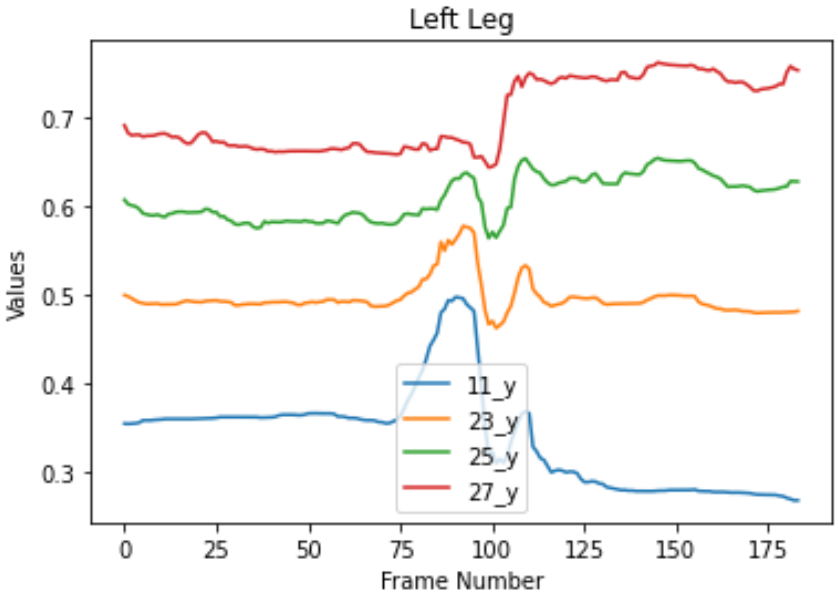

Figure 4.5: Joint Features of Left Leg across Frames

All the joint features generated from the MediaPipe and the status of the frame as labelled earlier are written into a csv file. This is the processed dataset for the skeletal joints-based machine learning approach for the action- Jump Forward. The frames are saved into a different folder with a name (combination of name of video and frame number). The frame names and the status of the frame as labelled above are saved into a different csv file. This is the processed dataset for the frame images-based machine learning approach for the action- Jump Forward.

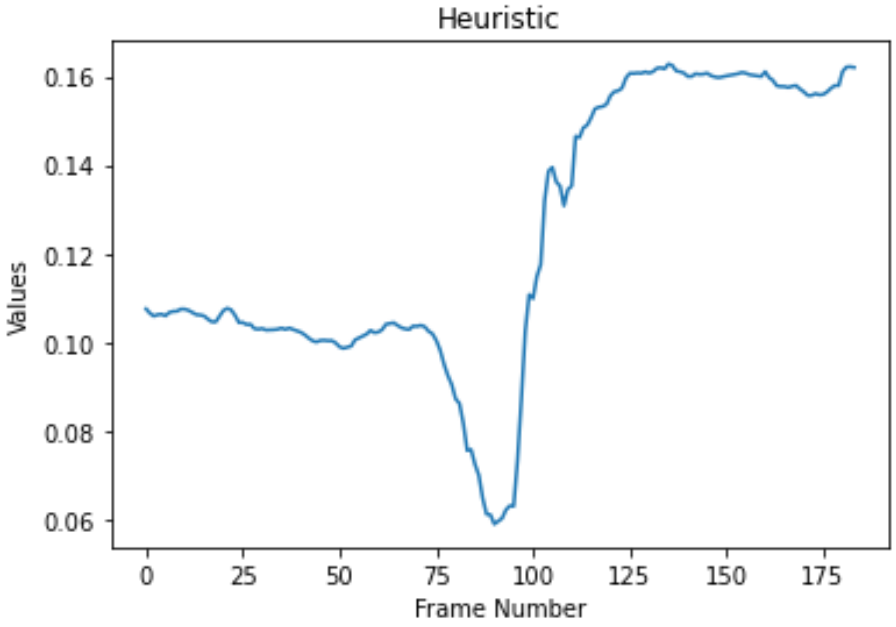

Figure 4.6: Heuristic Value across frames

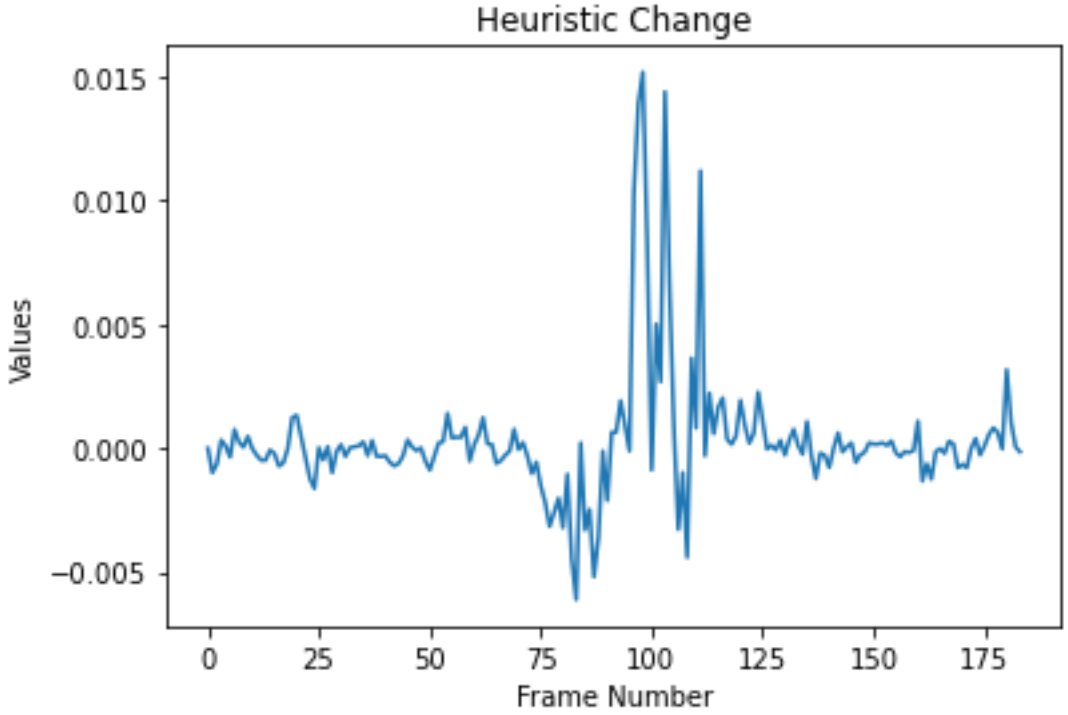

Figure 4.7: Heuristic Value Change across frames

Jump Sideways

The focus of this action is to measure how many jumps the person has made sideways where one full lateral jump is a combination of one jump left and one jump right. All the videos are of people jumping sideways. Since this action is measured based on the jump sideways, the joint features of interest from the MediaPipe Pose are 25, 26, 27 and 28 which refer to the points on the knees and ankles. Since the movement in horizontal direction (with respect to the frame) is the criteria for this action, the coordinates of this points along the x-axis for all frames in a video are plotted to analyse patterns emerging from the actions. The plots of x-axis values for right leg (26, 28) and left leg (25, 27) of one video are shown in Figures 4.8 and 4.9 respectively.

The plots clearly show a drop in all the values (a valley) whenever a jump is performed (which is eight times in this case). These valleys are our regions of interest. Thresholds need to be defined based on these regions so that we can label frames whenever a Jump left or jump right is made. These features have different values in different videos depending on the initial position of the person in the frame. Additionally, all these features show a large range of variation across videos of people of different heights. Hence, these features cannot be used directly for setting the threshold.

A new heuristic feature is generated. The heuristic value is calculated as 0.2 * (26_x) + 0.2 *(25_x) + 0.3 * (28_x) + 0.3 * (27_x). The plot of this heuristic value is shown in Figure 4.10. This smoothened out a lot of fluctuations in the joint values from the MediaPipe library.

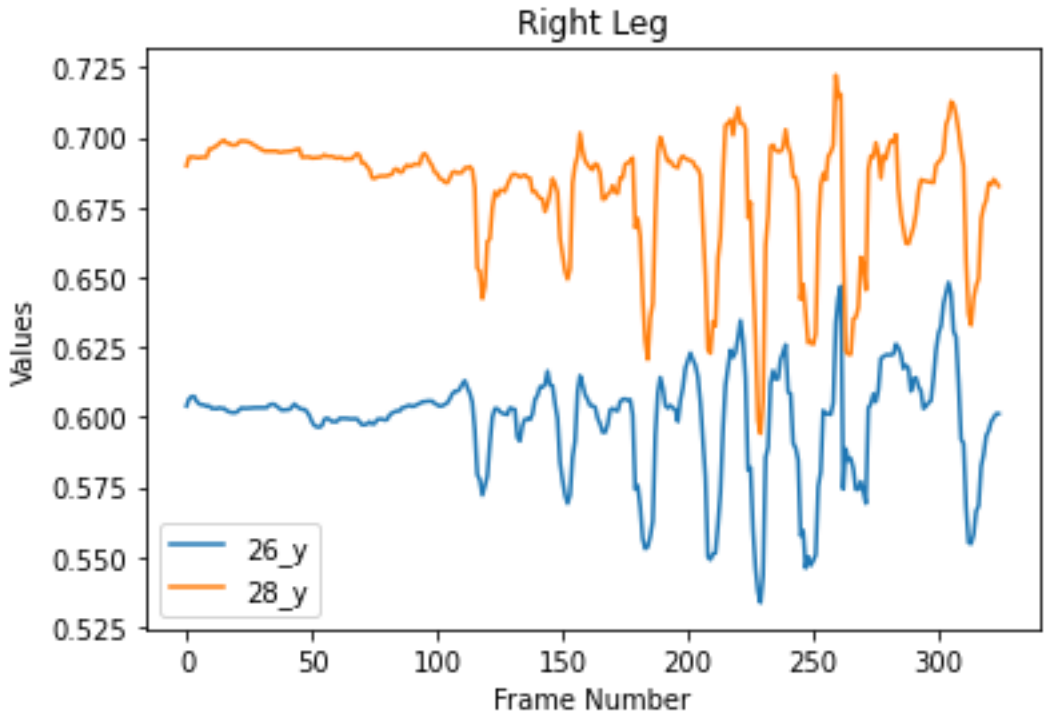

Figure 4.8: Joint Features of Right Leg across Frames

Furthermore, a new feature which measures the change in this heuristic value is generated. This is calculated as the heuristic value of previous frame subtracted from the heuristic value of the current frame. This feature gives a clearer picture on the when the Jumps start. The plot of this feature is shown in Figure 4.11. This feature which measures change in heuristic value is in the same range across videos and for people of different heights. Hence, this can be used for setting the thresholds. A threshold range of [-0.008, 0.008] is set for this feature based on experimentation on videos randomly selected from the sample. When the value of this feature is in the range of [-0.008, 0.008], the frames are labelled as “Standing”. When the value is greater than 0.008, the frames are labelled as “Jump Left” and when the value is lesser than -0.008, the frames are labelled as “Jump Right”. This labelled data has some frames which are incorrectly labelled “Standing” in the middle of the Jump. This is because of the fluctuations in the joint features from the MediaPipe library. To smoothen this, a rolling window of 4

frames is taken across all the frames of a video and when there is a frame with label “Standing” in the middle of a window with most of other frames labelled as “Jump Left” or “Jump Right”, this is changed to “Jump Left” or “Jump Right” respectively. This smoothening is repeated two more times to get the smoothened and correct labels for the frames.

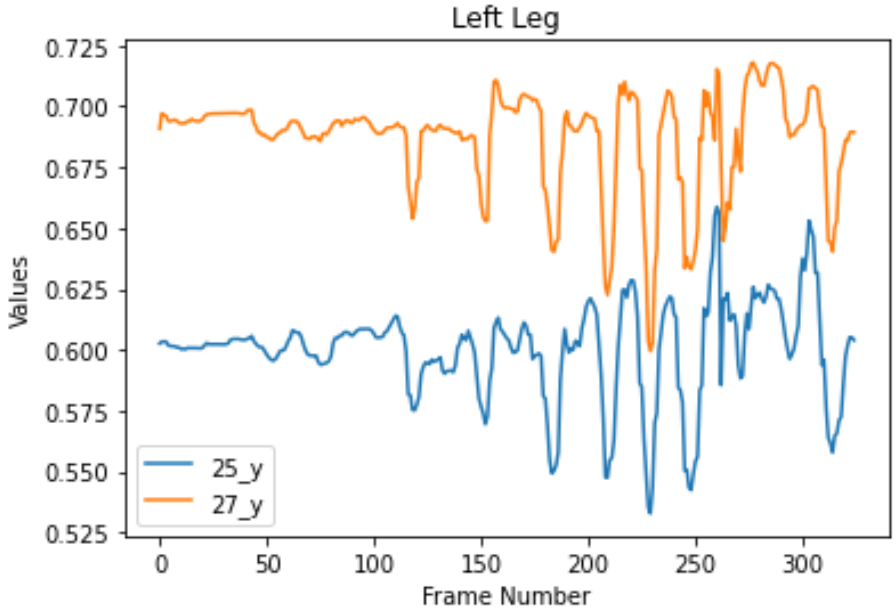

Figure 4.9: Joint Features of Left Leg across Frames

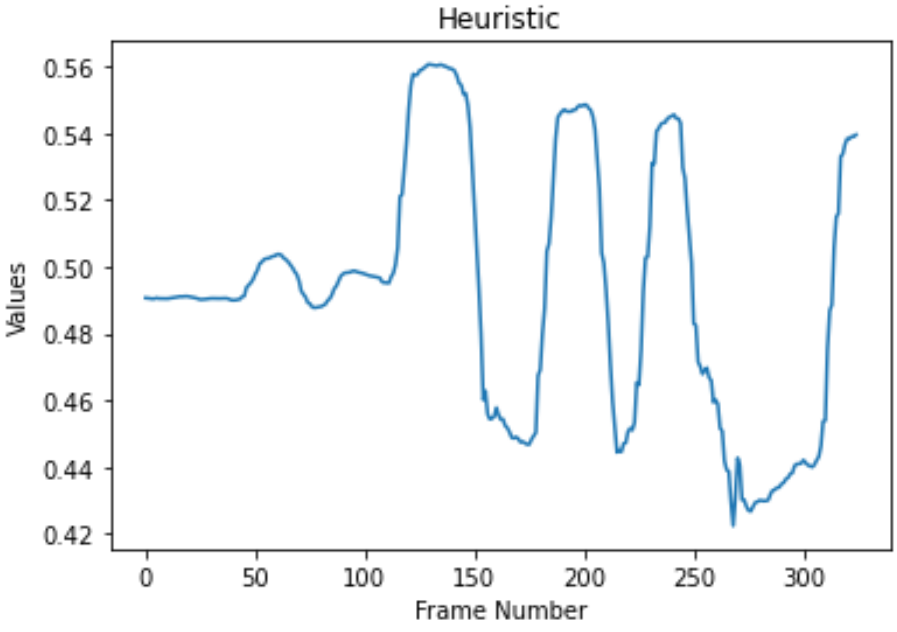

Figure 4.10: Heuristic Value across frames

Now, all the joint features generated from the MediaPipe and the status of the frame as labelled earlier are written into a csv file. This is the processed dataset for the skeletal joints-based machine learning approach for the action- Jump Sideways. The frames are saved into a different folder with a name (combination of name of video and frame number). The frame names and the status of the frame as labelled above are saved into a different csv file. This is the processed dataset for the frame images-based machine learning approach for the action- Jump Sideways.

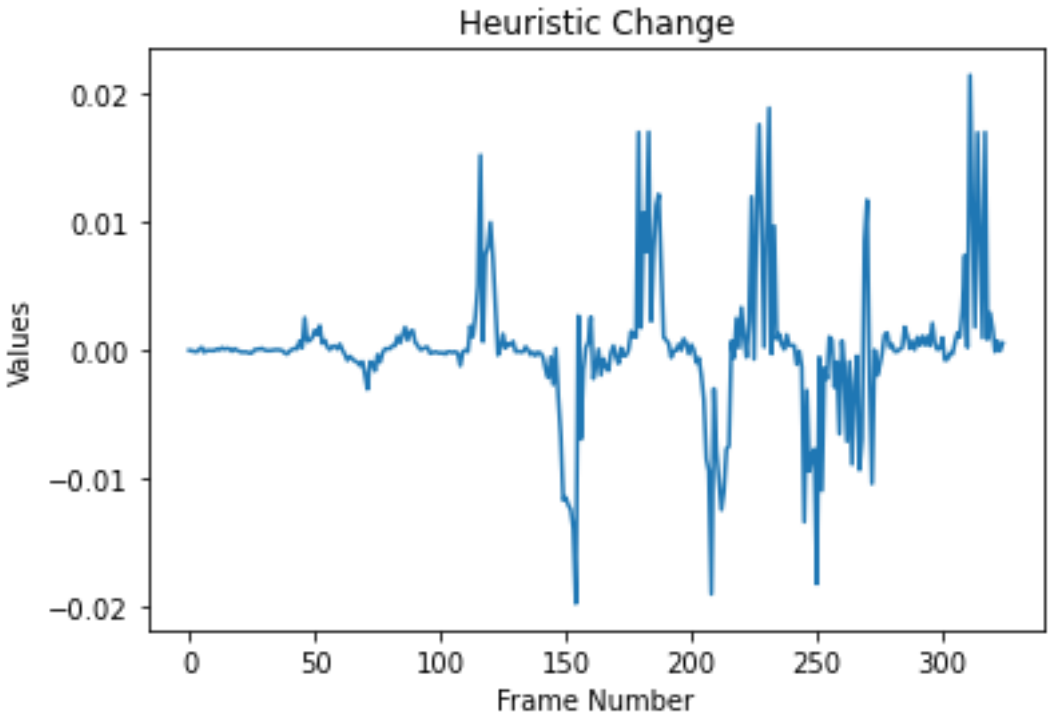

Figure 4.11: Heuristic Value Change across frames

Jog on the Spot

The focus of this action is to measure how many jogs the person has made with both the legs. All the videos are of people jogging. Since this action is measured based on the jogging action, the joint features of interest from the MediaPipe Pose are 23, 24, 27 and 28 which refer to the points on the hips and ankles. Since the movement in vertical direction (with respect to the frame) is the criteria for this action, the coordinates of this points along the y-axis for all frames in a video are plotted to analyse patterns emerging

from the actions. The plots of y-axis values for right leg (24, 28) and left leg (23, 27) of one video are shown in Figures 4.12 and 4.13 respectively.

The plots clearly show a drop in all the values (a valley) whenever a jog is performed. These valleys are our regions of interest. Thresholds need to be defined based on these regions so that we can label frames whenever a Jog is done. These features have different values in different videos depending on the initial position of the person in the frame. Additionally, all these features show a large range of variation across videos of people of different heights. Hence, these features cannot be used directly for setting the threshold.

Two new features are introduced to further analyse the videos and find a measure suitable for all videos. This is the “Hip to Ankle Distance” for both right and left legs separately. The “Hip to Ankle Distance” for right leg is calculated by taking the absolute difference of 24_y from 28_y and the same for left leg is calculated by taking the absolute difference of 23_y from 27_y. This smoothened out a lot of fluctuations in the joint values from the MediaPipe library. These features still showed a large range of variation across videos where people are of different heights.

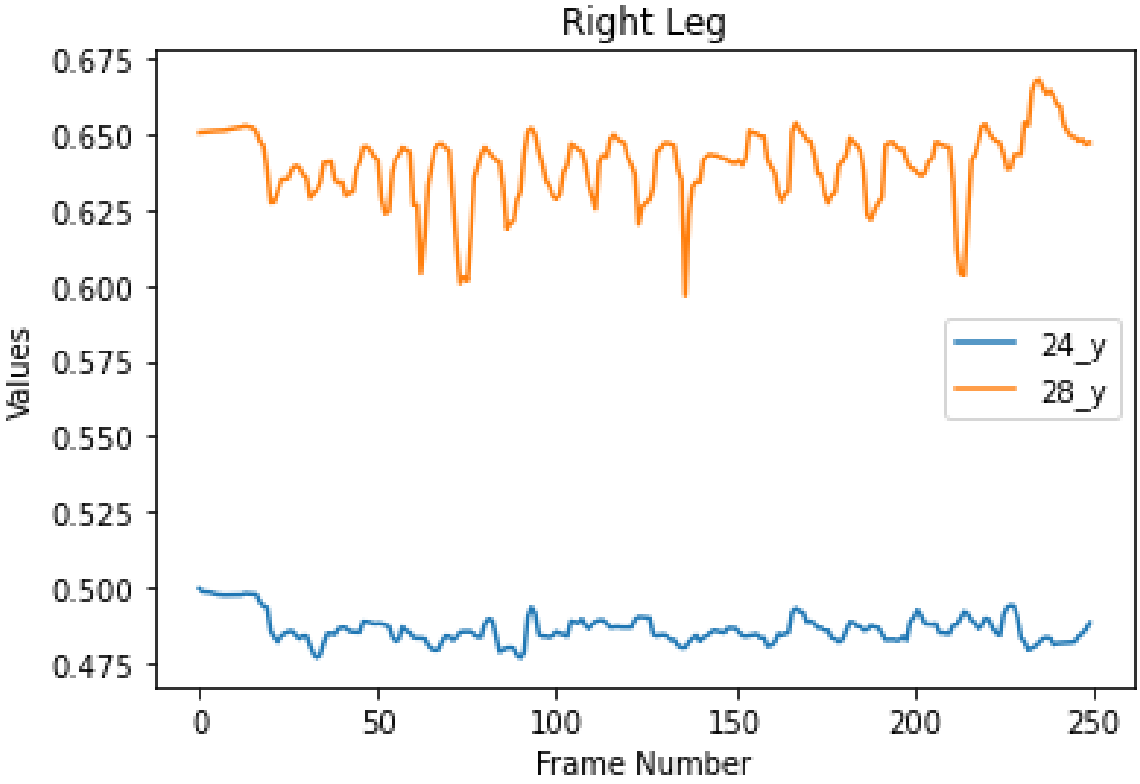

Figure 4.12: Joint Features of Right Leg across Frames

To normalize the variations a new feature called “Height” is calculated as 0.5 * ((27_y – 11_y) + (28_y – 12_y)). The “Hip to Ankle Distance” values for both the legs are normalized by dividing by the ‘Height” feature. So, there are two heuristic values for this action. These heuristic features give a clearer picture on the when the Jogs start. The plots of these heuristic values are shown in Figures 4.14 and 4.15 respectively. These features are in the same range across videos and for people of different heights. Hence, these can be used for setting the thresholds. The thresholds are selected based on experimentation on videos randomly selected from the sample. When the value of right leg heuristic is less than 0.57 and the value of left leg heuristic is greater than or equal to 0.57, the frames are labelled as “Jog Right”. When the value of left leg heuristic is less than 0.57 and the value of right leg heuristic is greater than or equal to 0.57, the frames are labelled as “Jog Left”. When the value of left leg heuristic is greater than or equal to 0.57 and the value of right leg heuristic is greater than or equal to 0.57, the frames are labelled as “Standing”. When the value of left leg heuristic is less than 0.57 and the value of right leg heuristic is less than 0.57, the frames are labelled as “Standing”.

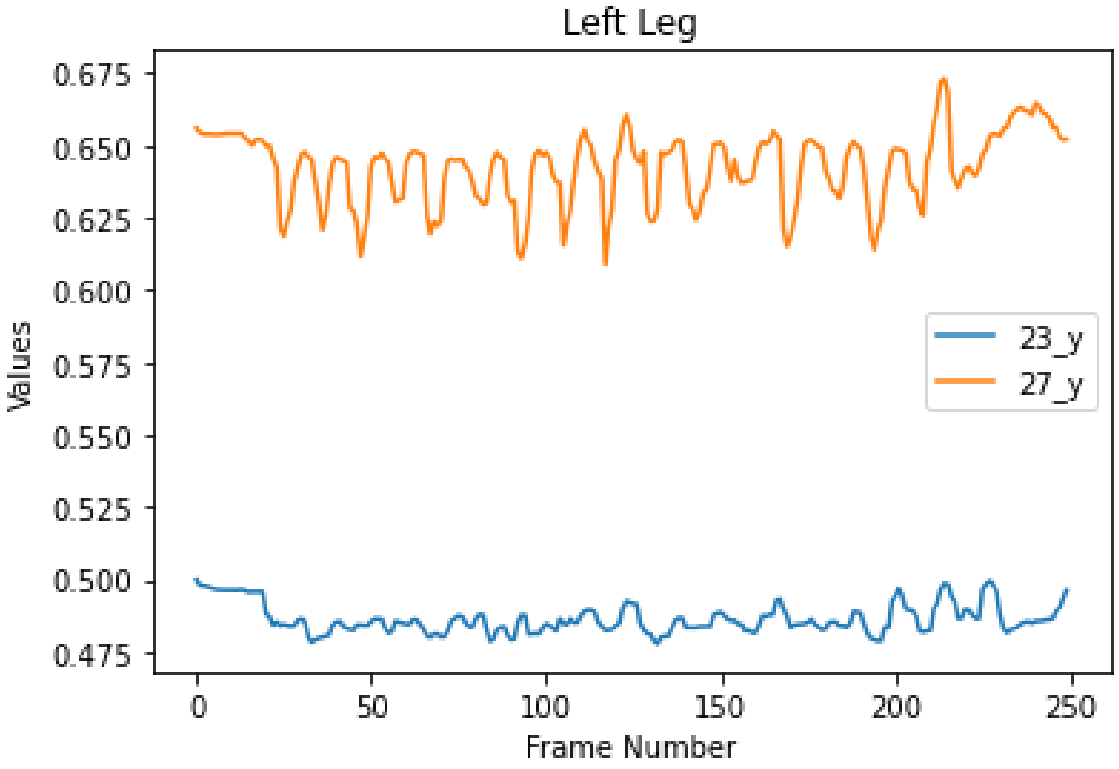

Figure 4.13: Joint Features of Left Leg across Frames

Now, all the joint features generated from the MediaPipe and the status of the frame as labelled earlier are written into a csv file. This is the processed dataset for the skeletal joints-based machine learning approach for the action- Jog on the Spot. The frames are

saved into a different folder with a name (combination of name of video and frame number). The frame names and the status of the frame as labelled above are saved into a different csv file. This is the processed dataset for the frame images-based machine learning approach for the action- Jog on the Spot.

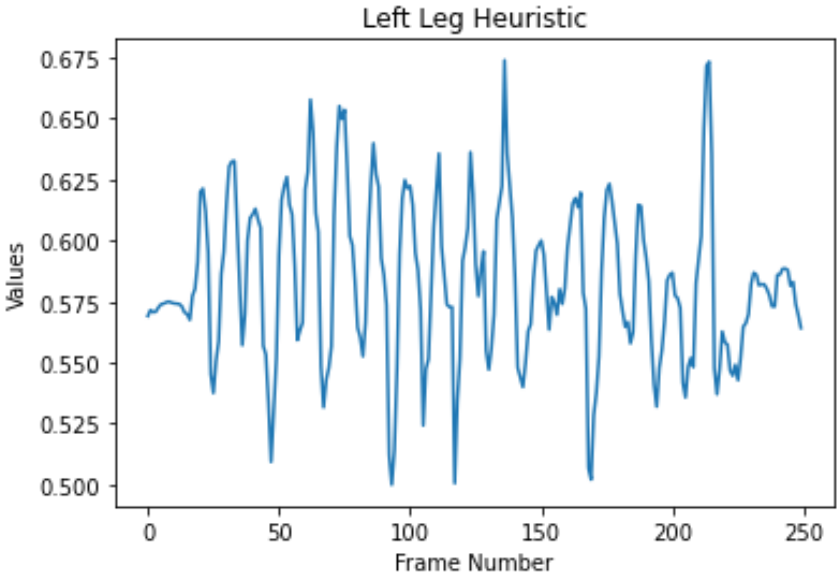

Figure 4.14: Heuristic Value across frames for Left Leg

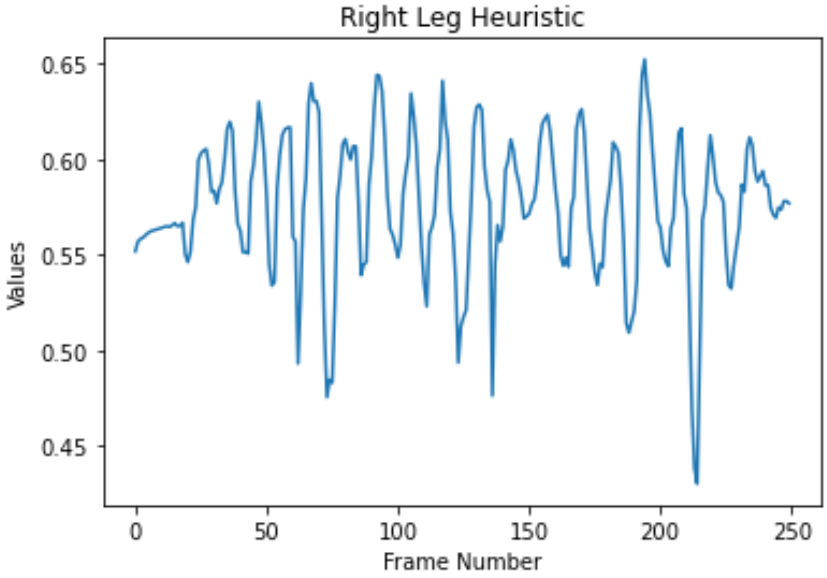

Figure 4.15: Heuristic Value across frames for Right Leg

4.3 System Architecture

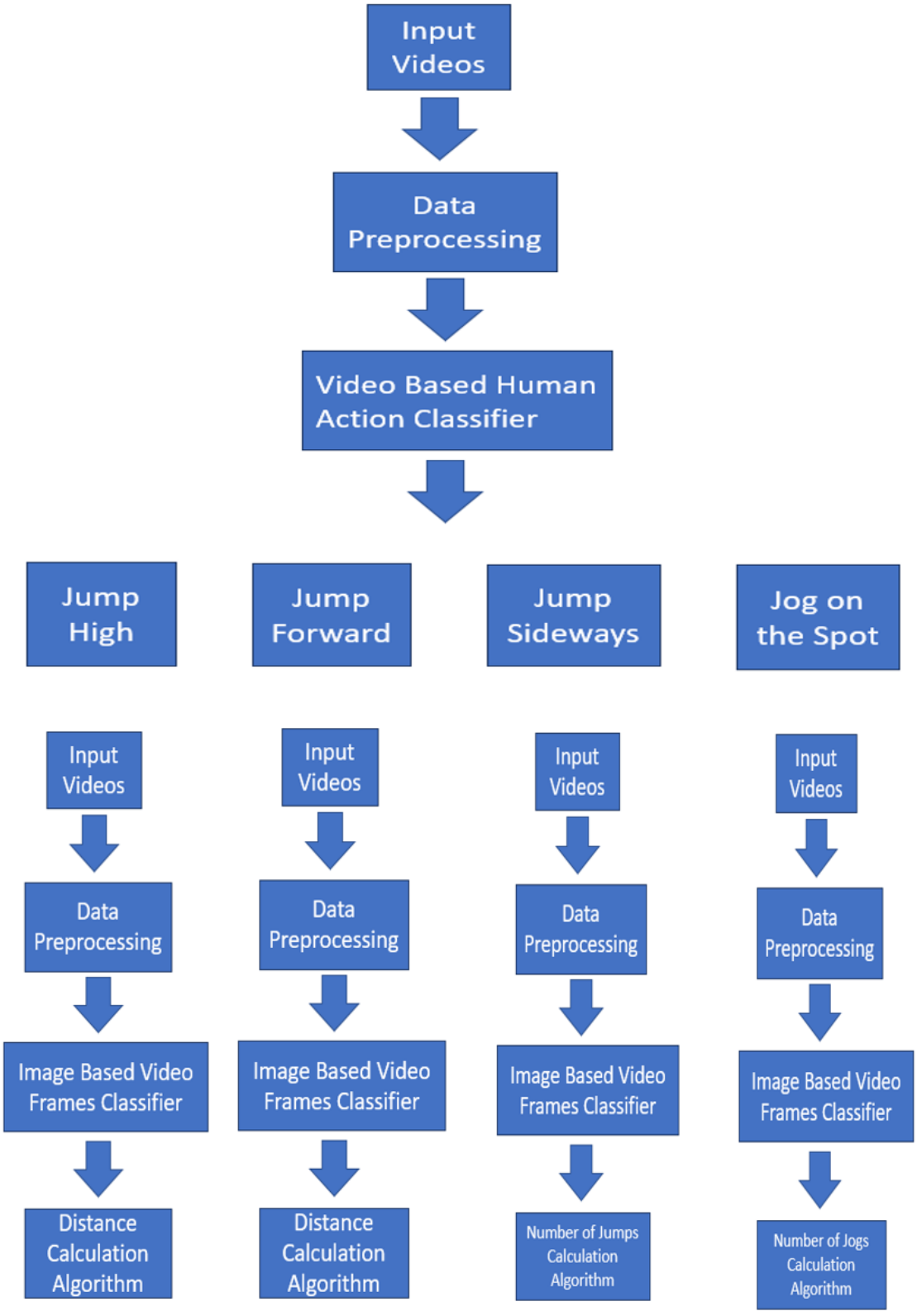

Figure 4.16: Workflow diagram of the project

The complete workflow of the Human Action Recognition System proposed in this dissertation is shown in Figure 4.16. There are two main stages in this workflow. The first is a Video Classification stage. The input to the system will be videos of people performing any of the four actions among Jump High, Jump Forward, Jump Sideways or Jog on the Spot. These input videos will be pre-processed as discussed earlier in data processing section for Video Based Human Action Classifier. Each video input will be converted to a series of 100 frames after this pre-processing. The data are fetched from train and validation folders with train.csv and val.csv referencing the labels for each video. These are fed to the Video Based Human Action Classifier, which is a deep neural network based on convolutional and LSTM layers.

Since the number of frames to be loaded is too large per video, the machine memory cannot handle all the data together in one shot. A custom data generator is designed for this purpose. Inside the generator all the 100 frames corresponding to a video are loaded into an array. These are further reshaped to the size (120,120) to reduce the load on the model. Multiple such sets of 100 frames are loaded together depending on the batch size. The batch size used here was 2 and hence 2 sets of 100 frames from 2 videos were resized and loaded together into an array which will be fed as input to the model in one feedforward run. The labels corresponding to these two videos will also be fed to the model by the generator.

The architecture of the deep neural network used by this classifier is shown in two parts. The first part is a feature extractor which extracts features from each frame. Each frame is passed through a pretrained neural network called “EfficientNetB7”. The output layers of this pretrained network is removed and we use the layers before the output layer to extract the features as identified by EfficientNetB7. The output from this pretrained network is passed through a flatten layer which changes the shape of the output to a single dimension. This is again passed through a dense layer with 64 neurons and a relu activation which will condense the output to 64 features or a vector of shape (64,1) corresponding to each frame of a video. The relu function outputs the same value whenever the neuron value is positive and 0 otherwise. The architecture of the same is shown in Figure 4.17.

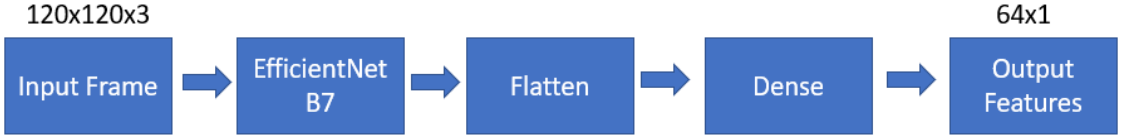

Figure 4.17: Architecture of the Frames Feature Extractor

The video classifier has 100 such feature extractor blocks to handle all the 100 frames from a video simultaneously. So, 64 features from all the 100 frames will be extracted using the feature extractor blocks and fed to an LSTM layer which has 32 units. This layer will capture the relationships between the features of the 100 frames to understand the temporal or sequential information embedded in the video. Since there 32 LSTM units in that layer, this layer will output a shape 100x32. This layer is followed by another LSTM layer with 16 units to further capture sequential information at a higher level. The output from this layer will be 16. This output is passed through a Dropout layer of 0.5 which will dropout 50 % of the connections with the next layer randomly during training to avoid overfitting. This is followed by a Dense layer with 8 neurons with a relu activation. This will condense the 16 features from the LSTM layer to 8 features. Another Dense layer with 4 neurons and a softmax activation is the last layer in this neural network. This will convert the features to the 4 values showing the probability of the video being belonging to the 4 categories (Jump High, Jump Forward, Jump Sideways and Jog on the Spot). The architecture of the same is shown in Figure 4.18. This deep learning model will be trained on the processed video frames from the train folder and validated using the video frames from the validation folder with the frames of videos as inputs and the action label as the predicted output.

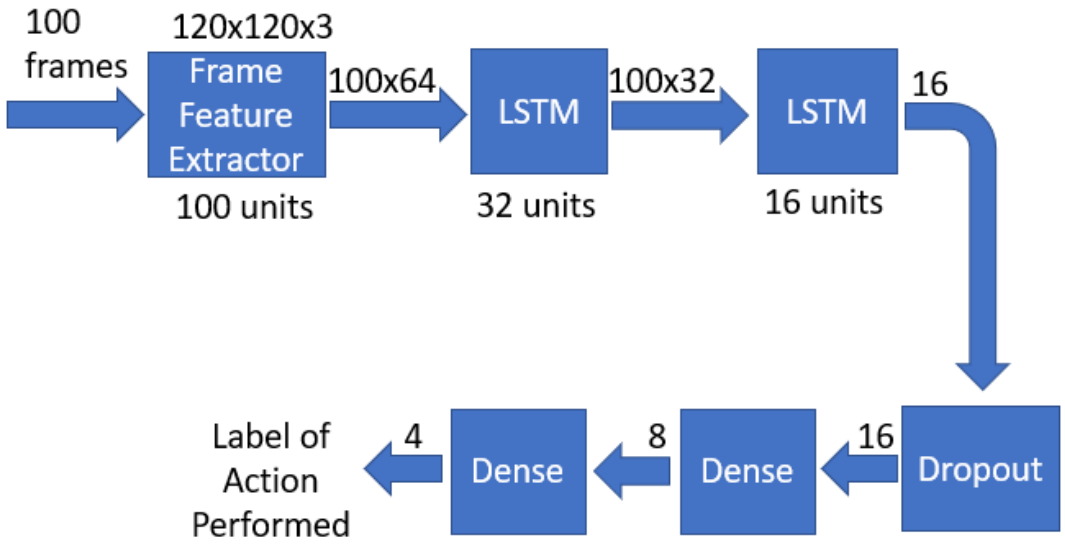

Figure 4.18: Architecture of the Video Based Human Action Classifier

The second stage of the Human Action Recognition System is different for the four categories of actions. After categorizing the video into one of the four categories, the workflow corresponding to the categorized video class is selected.

4.3.1 Jump High

Skeletal Joints Based Model

In this approach, the input videos will be broken down into individual frames and the frames are pre-processed by data pre-processing module as explained in the Data Pre-processing section. So, frames will be converted to skeletal joint values by the MediaPipe library and written into a csv file after processing by the Data Pre-processing module. The Image Based Video Frames Classifier module uses this csv file as its input. It loads the csv file and splits the data into train and test sets where all the joint values act as the input (X) and the status (intermediate labels as labelled in data processing) of the frames as output. With these values two types of classification models (one bagging model and one boosting model) are trained to predict the labels of each of the frames as “Jump High” indicating the person is in the process of jumping vertically (off the ground) and “Standing” indicating the person is on the ground. The first model used to train on this data is a RandomForestClassifier. This model is trained on the train data set with n_estimators = 100 (100 decision trees are used in the ensemble) and class_weight = ”balanced” (compensates for the class imbalance as most of the labels are “Standing”) and max_depth = 20 (maximum depth the decision trees can grow). The class imbalance occurs because people are standing on the ground for a large amount of time on the videos. The second model used to train on the data is an XGBoostClassifier. This model is trained on the train data set with eval_metric = 'mlogloss' (Metric used for training is the Multi class log loss evaluated for the predicted and true values), alpha = 0.05 (L2 regularization term on weights) and reg_lambda = 0.99 (L1 regularization term on weights). The regularization terms prevent the model from overfitting to the training data. Grid Search Cross Validation was performed on a set of hyperparameter values and the hyperparameters with the lowest cross validation errors were selected for the final models. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Distance Calculation Algorithm”.

Frame Images Based Model

In this approach, the input videos will be broken down into individual frames and the frames are pre-processed by data pre-processing module as explained in the Data Pre-processing section. Since this data is too large to be loaded into the memory in one shot ImageDataGenerator class of keras library is used to load the images into the memory in batches during training. For this purpose, the frames are divided into train, test and

validation folders with each folder having two folders – “Jump High” and “Standing” corresponding to the two status labels for the frames. These two folders will have only frames with the corresponding status labels. These ImageDataGenerator will load frames from these folders for train and validation sets during the model training. Two types of deep neural networks are used to build the classification models. The first one is a Convolutional Neural Network. The detailed architecture with shapes of inputs and outputs of each layer is shown in Figure 4.19. The Conv2D layers perform convolution operation on the inputs based on the size of the filter to derive the features and create a feature map. The MaxPooling2D layers aggregates the values among a pool of neighbouring values to give the maximum value from that pool. This will highlight the most present feature in each patch of the image as defined by the pool size. The Dropout layers will disconnect certain connections between two layers randomly during training to avoid the model from overfitting to the training data. The Flatten layer will flatten all the input values to a single dimension. The Dense layer is a set of neurons all of which are connected to the output values from the previous layer. The second type of neural network is an application of Transfer learning. Here, a pretrained neural network called EfficientNetB7 is used without its final layers to derive features from the images. These features are further passed through a designed convolutional neural network. During the training, only the weights of the designed convolutional neural networks are allowed to change. The weights of the layers of pretrained network remain the same. The detailed architecture with shapes of inputs and outputs of each layer is shown in Figure 4.20. The functional layer denoted in the architecture is the pretrained model (EfficientNetB7 with last layer removed) followed by a Dense layer which extracts 64 features from the output of the pretrained model. The images will be fed to both these models from the ImageDataGenerator for training. The models are compiled with “categorical cross entropy” as the loss function and Adam as the optimizer. The categorical cross entropy metric calculates the difference between the predicted classes and true classes considering all class distributions and not just the overall accuracy. The Adam optimizer has learning_rate=0.001, beta_1=0.9 and beta_2=0.99. The training will be done for 50 epochs with a batch size of 20 using the train data set and will be validated using the validation data set. The class_weight parameter is set as {0: 0.85, 1: 0.15} (0 is “Jump High” and 1 is “Standing”) during training to account for the class imbalance. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Distance Calculation Algorithm”.

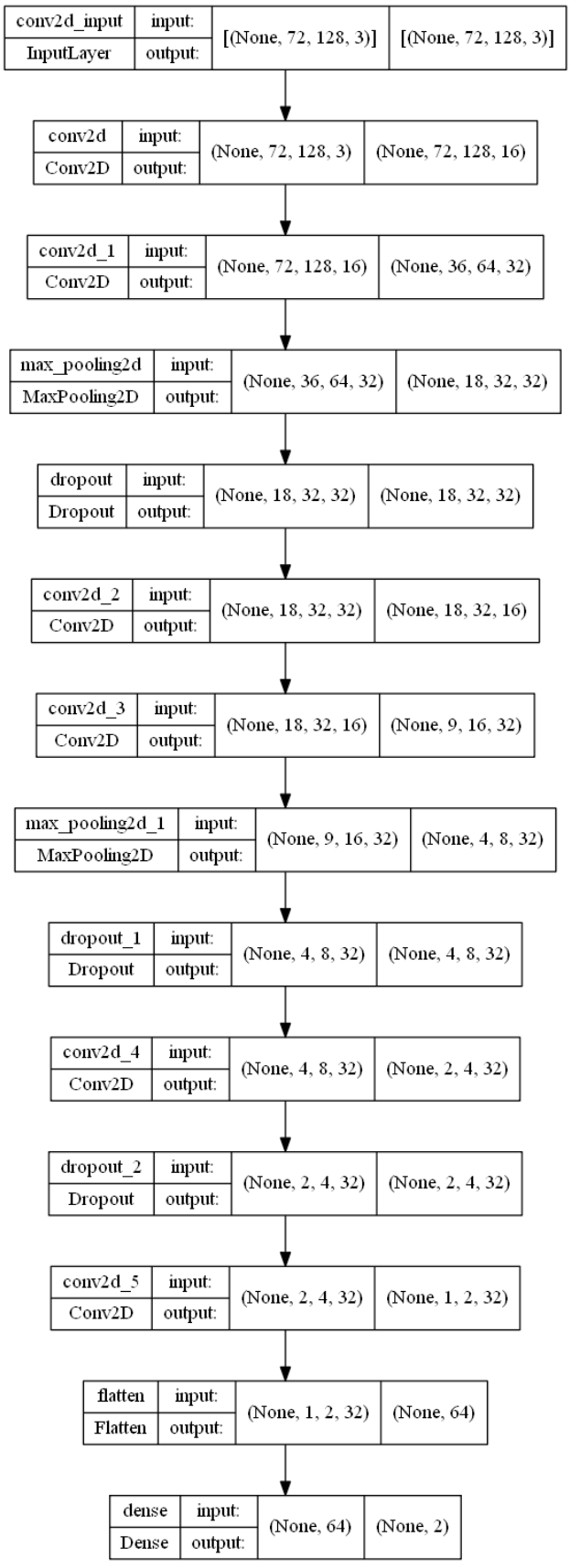

Figure 4.19: Architecture of the Convolutional Neural Network

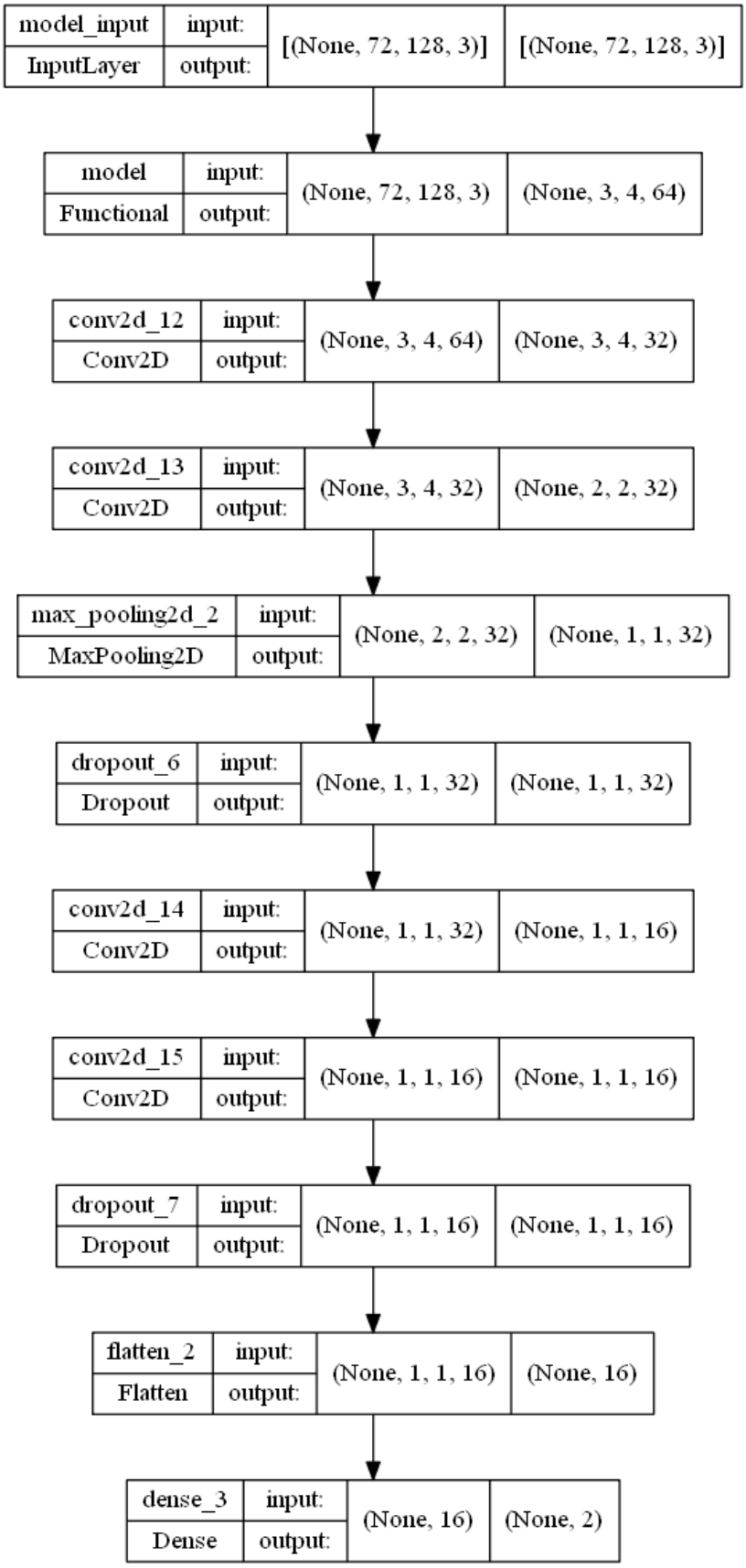

Figure 4.20: Architecture of the Transfer Learning Network

Distance Calculation Algorithm

The input to this algorithm will be all the frames of the video labelled as either “Standing” or “Jump High”. This algorithm finds out the frame in which the status changes from a standing position to a jumping position and marks that frame as the jump start frame. This is done by checking if one frame status is “Standing” and there are more than 4 “Jump High” statuses in the next 7 frames. Similarly, the algorithm finds out the frame in which the status changes from a jumping position to a standing position and marks that frame as the jump end frame. This is done by checking if one frame status is “Jump High” and there are more than 4 “Standing” statuses in the next 7 frames. The distance jumped is calculated as the average of the difference of maximum and minimum joint values of the ankle in the frames between jump start and jump end for the left (27_y) and right (28_y) legs.

4.3.2 Jump Forward

Skeletal Joints Based Model

The approach is similar to the skeletal joints-based approach used for Jump High action with some changes in the model hyperparameters. The pre-processed data will be available as a csv file. The Image Based Video Frames Classifier module uses this csv file as its input. It loads the csv file and splits the data into train and test sets where all the joint values act as the input (X) and the status (intermediate labels as labelled in data processing) of the frames as output. With these values two types of classification models (one bagging model and one boosting model) are trained to predict the labels of each of the frames as “Jump Progress” indicating the person has started the process of jumping forward (towards the camera) and “Standing” indicating the person is in the normal standing posture. The first model used to train on this data is a RandomForestClassifier. This model is trained on the train data set with n_estimators = 110 (110 decision trees are used in the ensemble) and class_weight = ”balanced” (compensates for the class imbalance as most of the labels are “Standing”) and max_depth = 22 (maximum depth the decision trees can grow). The second model used to train on the data is an XGBoostClassifier. This model is trained on the train data set with eval_metric = 'mlogloss' (Metric used for training is the Multi class log loss evaluated for the predicted and true values), alpha = 0.01 (L2 regularization term on weights) and reg_lambda = 1 (L1 regularization term on weights). Grid Search Cross Validation was performed on a set of hyperparameter values and the hyperparameters with the lowest cross validation errors were selected for the final models. These trained

models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Distance Calculation Algorithm”.

Frame Images Based Model

This approach is similar to the Frame Images Based approach for Jump High action with differences in the hyperparameters of the models. These ImageDataGenerator will load frames from the folders for train and validation sets (having labels of “Jump Progress” or “Standing”) during the model training. Two types of deep neural networks are used to build the classification models. The first one is a Convolutional Neural Network. The architecture is the same as the one used for Jump High action and is shown in Figure 4.19. The second type of neural network is an application of Transfer learning. The architecture is the same as the one used for Jump High action and is shown in Figure 4.20. The images will be fed to both these models from the ImageDataGenerator for training. The models are compiled with “categorical cross entropy” as the loss function and Adam as the optimizer. The Adam optimizer has learning_rate=0.002, beta_1=0.8 and beta_2=0.9. The training will be done for 52 epochs with a batch size of 20 using the train data set and will be validated using the validation data set. The class_weight parameter is set as {0: 0.75, 1: 0.25} (0 is “Jump Progress” and 1 is “Standing”) during training to account for the class imbalance. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Distance Calculation Algorithm”.

Distance Calculation Algorithm

The input to this algorithm will be all the frames of the video labelled as either “Standing” or “Jump Progress”. This algorithm finds out the frame in which the status changes from a standing position to a jumping position and marks that frame as the jump start frame. This is done by checking if one frame status is “Standing” and there are more than 4 “Jump Progress” statuses in the next 7 frames. Similarly, the algorithm finds out the frame in which the status changes from a jumping position to a standing position and marks that frame as the jump end frame. This is done by checking if one frame status is “Jump Progress” and there are more than 4 “Standing” statuses in the next 7 frames. The distance moved in pixel values is calculated as the average of the difference of the ankle joint value in the jump start frame and jump end frame for the left (27_y) and right (28_y) legs. The distance jumped is calculated by multiplying this value with a scaling factor. The scaling factor is calculated using a reference distance in the video divided by the same distance in frame pixel values.

4.3.3 Jump Sideways

Skeletal Joints Based Model

The approach is similar to the skeletal joints-based approach used for Jump Forward action with some changes in the model hyperparameters. The pre-processed data will be available as a csv file. The Image Based Video Frames Classifier module uses this csv file as its input. It loads the csv file and splits the data into train and test sets where all the joint values act as the input (X) and the status (intermediate labels as labelled in data processing) of the frames as output. With these values two types of classification models (one bagging model and one boosting model) are trained to predict the labels of each of the frames as “Jump Left” indicating the person has started the process of jumping left, “Jump Right” indicating the person has started the process of jumping right and “Standing” indicating the person is in the normal standing posture. The first model used to train on this data is a RandomForestClassifier. This model is trained on the train data set with n_estimators = 120 (120 decision trees are used in the ensemble) and class_weight = ”balanced” (compensates for the class imbalance as most of the labels are “Standing”) and max_depth = 21 (maximum depth the decision trees can grow). The second model used to train on the data is an XGBoostClassifier. This model is trained on the train data set with eval_metric = 'mlogloss' (Metric used for training is the Multi class log loss evaluated for the predicted and true values), alpha = 0.02 (L2 regularization term on weights) and reg_lambda = 0.98 (L1 regularization term on weights). Grid Search Cross Validation was performed on a set of hyperparameter values and the hyperparameters with the lowest cross validation errors were selected for the final models. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Number of Jumps Calculation Algorithm”.

Frame Images Based Model

This approach is similar to the Frame Images Based approach for Jump Forward action with differences in the model architecture and hyperparameters of the models. These ImageDataGenerator will load frames from the folders for train and validation sets (having labels of “Jump Left”, “Jump Right” or “Standing”) during the model training. Two types of deep neural networks are used to build the classification models. The first one is a Convolutional Neural Network. The detailed architecture with shapes of inputs and outputs of each layer is shown in Figure 4.21. The second type of neural network is

an application of Transfer learning. The detailed architecture with shapes of inputs and outputs of each layer is shown in Figure 4.22. The images will be fed to both these models from the ImageDataGenerator for training. The models are compiled with "categorical cross entropy" as the loss function and Adam as the optimizer. The Adam optimizer has learning_rate=0.003, beta_1=0.7 and beta_2=0.99. The training will be done for 55 epochs with a batch size of 20 using the train data set and will be validated using the validation data set. The class_weight parameter is set as {0: 0.4, 1: 0.45, 2: 0.15} (0 is "Jump Left", 1 is "Jump Right" and 2 is "Standing") during training to account for the class imbalance. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the "Number of Jumps Calculation Algorithm".

Number of Jumps Calculation Algorithm

The input to this algorithm will be all the frames of the video labelled as either "Standing", "Jump Left" or "Jump Right". This algorithm maintains a list called order_queue which is empty in the beginning. This algorithm checks if there are more than 4 "Jump Left" statuses in a sliding window of 7 frames, and if it finds such a window it appends "Jump Left" to the order_queue. Similarly, the algorithm checks if there are more than 4 "Jump Left" statuses in a sliding window of 7 frames, and if it finds such a window it appends "Jump Left" to the order_queue. But, once a "Jump Left" is appended to order_queue, only "Jump Right" will be appended next. Similarly, once a "Jump Right" is appended to order_queue, only "Jump Left" will be appended next. This is done by maintaining two switch flags for "Jump Left" and "Jump Right", which gets switched off when the corresponding label is added to the queue and switched on when the other label is added to the queue. These switch flags are checked while appending values to the queue. This is done because the criteria for one complete jump is a combination of "Jump Left" and "Jump Right" in any order. So, once the algorithm goes through the all the frame statuses, it will have an order_queue with "Jump Left" and "Jump Right" tags occurring alternatively in any order. The total number of jumps is calculated as the length of the queue divided by two (as the criteria for one complete jump is a combination of "Jump Left" and "Jump Right" in any order).

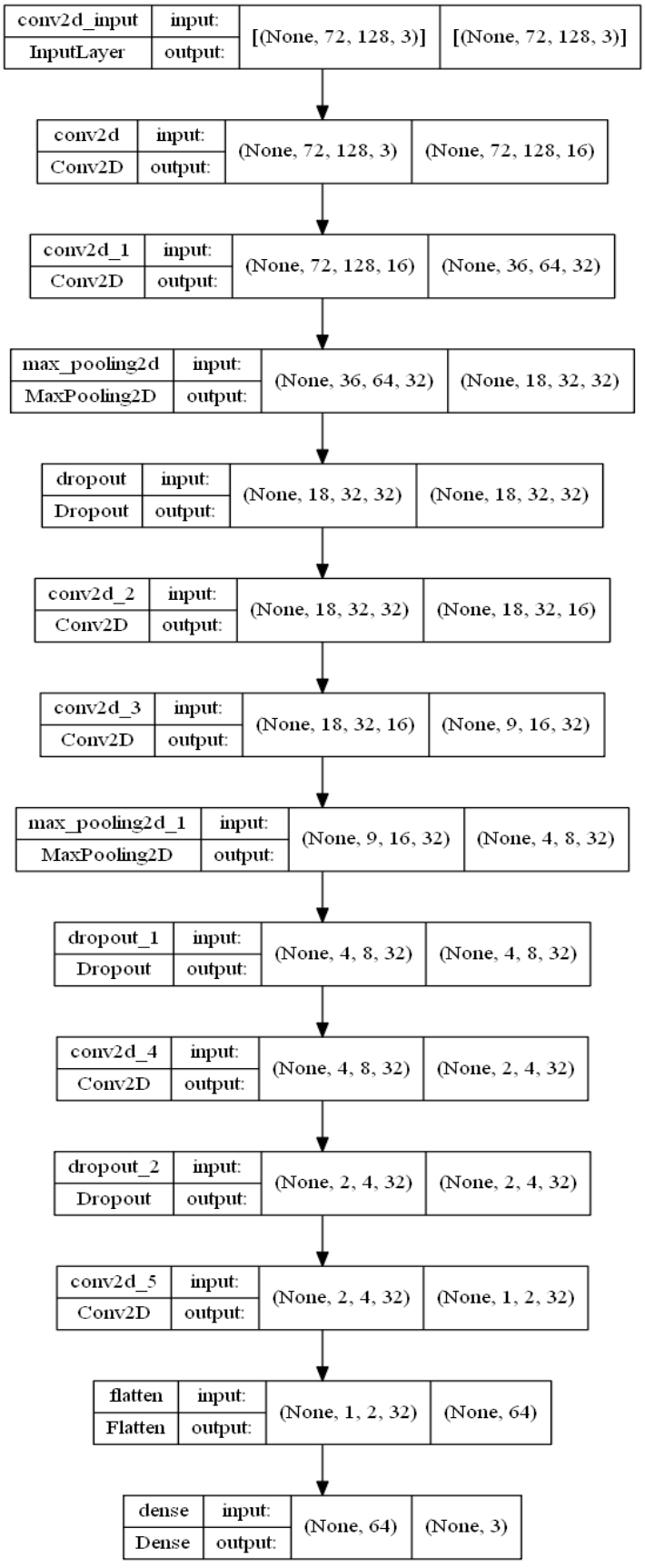

Figure 4.21: Architecture of the Convolutional Neural Network

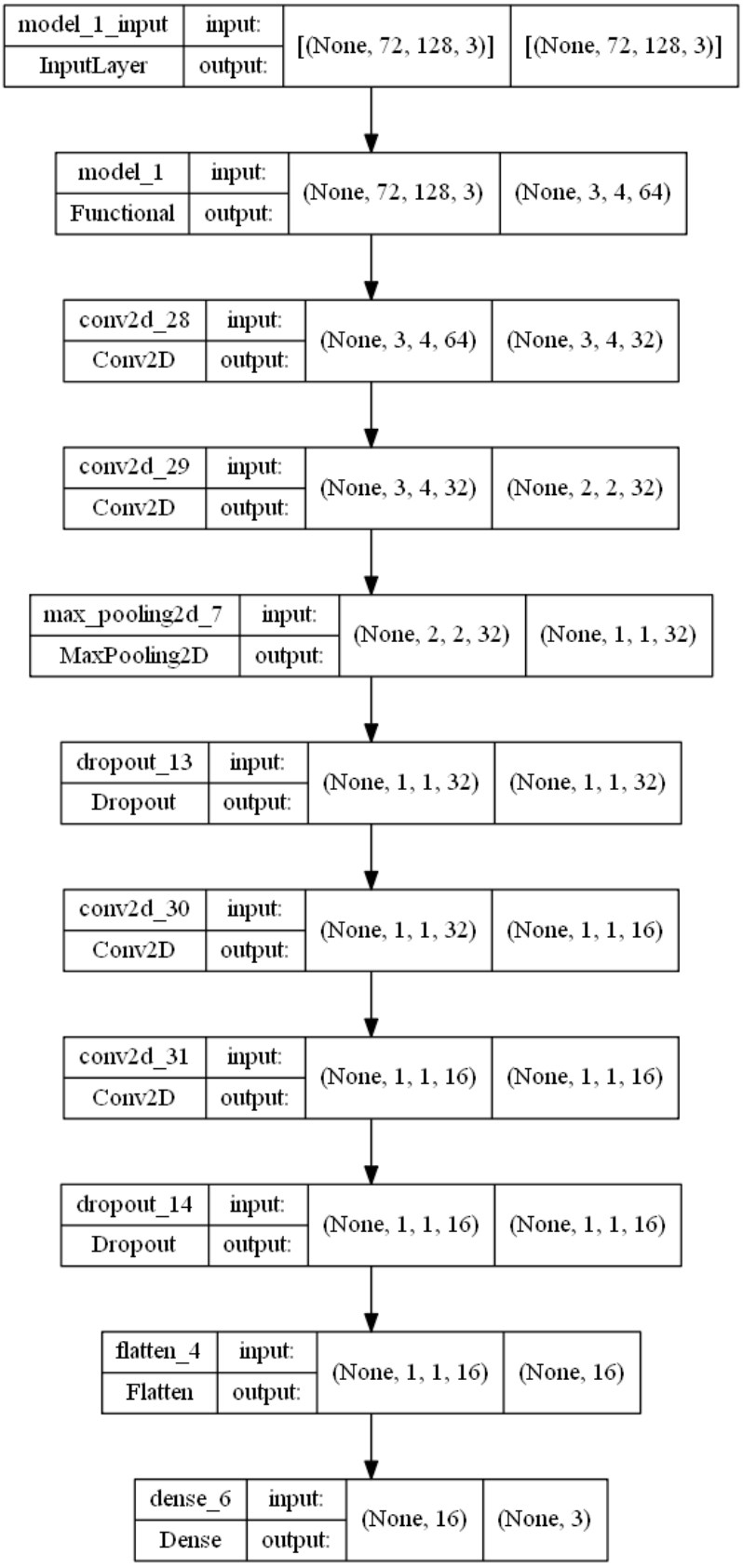

Figure 4.22: Architecture of the Transfer Learning Network

4.3.4 Jog on the Spot

Skeletal Joints Based Model

The approach is similar to the skeletal joints-based approach used for Jump Sideways action with some changes in the model hyperparameters. The pre-processed data will be available as a csv file. The Image Based Video Frames Classifier module uses this csv file as its input. It loads the csv file and splits the data into train and test sets where all the joint values act as the input (X) and the status (intermediate labels as labelled in data processing) of the frames as output. With these values two types of classification models (one bagging model and one boosting model) are trained to predict the labels of each of the frames as “Jog Left” indicating the person has raised his/her left leg above the ground, “Jog Right” indicating the person has raised his/her right leg above the ground and “Standing” indicating the person is in the normal standing posture or has both legs on the ground. The first model used to train on this data is a RandomForestClassifier. This model is trained on the train data set with n_estimators = 122 (122 decision trees are used in the ensemble), class_weight = ”balanced” (compensates for the class imbalance as most of the labels are “Standing”) and max_depth = 22 (maximum depth the decision trees can grow). The second model used to train on the data is an XGBoostClassifier. This model is trained on the train data set with eval_metric = 'mlogloss' (Metric used for training is the Multi class log loss evaluated for the predicted and true values), alpha = 0.05 (L2 regularization term on weights) and reg_lambda = 0.9 (L1 regularization term on weights). Grid Search Cross Validation was performed on a set of hyperparameter values and the hyperparameters with the lowest cross validation errors were selected for the final models. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Number of Jogs Calculation Algorithm”.

Frame Images Based Model

This approach is similar to the Frame Images Based approach for Jump Sideways action with differences in the model hyperparameters. These ImageDataGenerator will load frames from the folders for train and validation sets (having labels of “Jog Left”, “Jog Right” or “Standing”) during the model training. Two types of deep neural networks are used to build the classification models. The first one is a Convolutional Neural Network. The architecture is the same as the one for the action Jump Sideways and is shown in Figure 4.21. The second type of neural network is an application of Transfer learning.

The architecture is the same as the one for the action Jump Sideways and is shown in Figure 4.22. The images will be fed to both these models from the ImageDataGenerator for training. The models are compiled with “categorical cross entropy” as the loss function and Adam as the optimizer. The Adam optimizer has learning_rate=0.001, beta_1=0.85 and beta_2=0.95. The training will be done for 60 epochs with a batch size of 20 using the train data set and will be validated using the validation data set. The class_weight parameter is set as {0: 0.35, 1: 0.35, 2: 0.3} (0 is “Jog Left”, 1 is “Jog Right” and 2 is “Standing”) during training to account for the class imbalance. These trained models will be used to predict the labels of all the frames of the video and these labelled frames will be passed on to the “Number of Jogs Calculation Algorithm”.

Number of Jogs Calculation Algorithm

The input to this algorithm will be all the frames of the video labelled as either “Standing”, “Jog Left” or “Jog Right”. This algorithm maintains a list called order_queue which is empty in the beginning. This algorithm checks if there are more than 4 “Jog Left” statuses in a sliding window of 7 frames, and if it finds such a window it appends “Jog Left” to the order_queue. Similarly, the algorithm checks if there are more than 4 “Jog Left” statuses in a sliding window of 7 frames, and if it finds such a window it appends “Jump Left” to the order_queue. But, once a “Jog Left” is appended to order_queue, only “Jog Right” will be appended next. Similarly, once a “Jog Right” is appended to order_queue, only “Jog Left” will be appended next. This is done by maintaining two switch flags for “Jog Left” and “Jog Right”, which gets switched off when the corresponding label is added to the queue and switched on when the other label is added to the queue. These switch flags are checked while appending values to the queue. This is done because the person must use left leg and right leg alternatively while jogging. So, once the algorithm goes through the all the frame statuses, it will have an order_queue with “Jog Left” and “Jog Right” tags occurring alternatively in any order. The total number of jumps is calculated as the length of the queue (as the criteria for a jog is a leg raising from the ground).

5 Security and Privacy Considerations

5.1 Security Considerations

The human action detection system proposed in this paper when deployed as applications could face security threats in two stages. The first stage is during model training process. The second stage is after the training process is completed.

The first stage is where the hackers target training data as discussed in [16]. This is done in the following ways:

Data Poisoning: This can occur when the training data used in a machine learning model is polluted by a hacker. They achieve this by tampering with the data. Data poisoning has two types of impacts on data which are impact on confidentiality and impact on trustworthiness. Most of the time, the training video data contains sensitive and confidential information. The data loses its confidentiality by these poisoning attacks. Since data is gathered from many sources such as cameras or online sources, the chance of data getting poisoned is high. This impacts the machine learning engineer's confidence on the data and thus reduces its trustworthiness. Two attacks that come under the umbrella of data poisoning are label flipping and gradient descent attack. Label flipping occurs in cases where the output to be learned in the training data (labels) is changed or flipped so that the model learns wrong associations. Gradient descent attack is of two types. One type is where the actual output or label is continuously changed, and the model continues performing gradient descent as it thinks it has not reached the answer and ends up training for a very long time. The other one is where the model is inaccurately forced to complete gradient descent algorithm as it is in the belief that it reached the correct answer and thereby ending up with incompletely trained model. Data Poisoning can be prevented by using outlier detection techniques.

Stealthy Channel Attack: Data quality plays an important part in developing pipelines for machine learning systems. For building models which are applicable in the real world, the video data is gathered from many sources and the data collected must be relevant. The hacker will be able to insert large amounts of inaccurate or fraudulent data into the machine learning system and compromise the whole pipeline even before model is created. This is called stealthy channel attack and can be prevented by following due diligence during the data collection process.

The second stage attacks are done in the following ways:

Transfer Learning Attack: In human actions recognition, we might have to use pre-trained models to reduce training time, which are further trained and included as part deep neural network architecture. This is exploited by a hacker by replacing the original model with another malicious one. This is hard to figure out as the pretrained model are used without much inspection. We need to be cautious in using the models by referring to reliable sources.

Adversarial Example: These are also known as evasion attacks. In this, the data input to the models are tampered to force the machine learning models to make incorrect predictions. The integrity and confidence of the complete system is affected in such attacks. The systems which overfits to its training data are most susceptible to these attacks. A good preventive measure for these attacks is to evaluate the model against as many adversarial examples as we can before deployment and training of the models to avoid these examples (adversarial training).

Output Integrity Attack: This can occur when the results are manipulated by a hacker between the interface which displays the results and the model. The integrity of output is compromised by this attack. A good preventive measure for these attacks is to use some strong cryptographic security algorithms between the interface and the model so that the hacker cannot easily access the results.

System Manipulation: The machine learning systems will continue to learn even after being deployed by taking feedback based on the new predictions. A hacker can exploit this as in [17] and provide tampered data as the model inputs and cause the degradation of the model performance over time rather than gradual improvement. A good preventive measure for these attacks is to implement outlier detection techniques.

5.2 Privacy Considerations

5.2.1 Implicit considerations

Privacy is an important factor to consider for the human action recognition system. Here video data of users performing actions is used for training and hence these are sensitive data containing identity information about the users. So, the identity of users in the video should be masked for privacy. We are building our machine learning system pipeline including a step that blurs the faces before the model learning the actions so that the person performing the action in the video cannot be identified at any stage. This protects the privacy and identity of users. This will be done after getting approval from the college ethics committee.

5.2.2 Explicit considerations

In explicit considerations we are concerned about the privacy attacks similar to the ones discussed in [18] and their preventive measures. Privacy attacks are of two types -model extraction and retrieval of training data.

Model Extraction: A hacker can send various inputs to the model and check the outputs and tries to figure out the equations governing the model and thereby decoding the model weights and biases. If the hacker succeeds in this, it means that the human action recognition model is extracted or stolen and can be utilized by the hacker.

Retrieval of the Training Data: There are two ways in which retrieval of the training data is done. First being membership inference in which the hacker can figure out if a particular record was used in the training of the model. The second being model inversion in which a hacker can figure out information pertaining to the model with pieces of information about the data used for training and generated outputs from the model.

Good preventive measures for these attacks include various methods such as adding noise intentionally to the data (differential privacy), computing weights of model after encrypting the data (homomorphic encryption) and multiple parties jointly performing actions with partial access to datasets (multi-party computation).

6 Evaluation

To evaluate the performance of the models used in this study, the predicted outputs and the true outputs are compared using suitable metrics. F1 Score is used as a metric for evaluation of the models in the first and second stages of the workflow. F1 Score is the harmonic mean of precision and recall, where precision is the number of true positives/ (number of true positives + number of false positives) and recall is the number of true positives/ (number of true positives + number of false negatives). This a good metric for classification problems especially when class imbalances are present in the data. The F1 Score on train data and test data for the Video Based Human Action Classifier in workflow stage 1 is 0.64 and 0.62 respectively.

The Image Based Video Frames Classifier in second stage of the workflow is implemented using Skeletal Joint Based models (Random Forest Classifier and XGBoost Classifier) and Frame Images Based models (CNN Model and Transfer Learning + CNN model). The F1 Scores of these models on the training data is shown in Table 6.1. The F1 Scores of these models on the test data is shown in Table 6.2. The Skeletal Joint Based model outperformed the other approach on train and test data. To further assess the generalizing capabilities of these Skeletal Joint Based models cross-validation was performed on all these models. The cross-validation scores of the models are shown in Table 6.3. The Random Forest Classifier performs the best among all the models for all the actions with good generalizing capabilities. Hence, this model will be selected as the final model for the second stage of the workflow.

Each action is further evaluated with the results calculated by algorithms corresponding to each action. The results of all actions (except Jump High) are available in a doc format, but it is too unstructured to be read by the program. Since true values for the action Jump High is not available, it is not possible to evaluate the results calculated by the distance calculation algorithm. For the action Jump Forward, the distance jumped was recorded manually during the recording of video. We compared this result to the results calculated by the distance calculation algorithm for 50 random video and the algorithm measured the distance with an average error of 10cm. For the action Jump Sideways, the number of complete lateral jumps was recorded manually during the recording of video. We compared this result to the results calculated by the number of jumps calculation algorithm for 50 random video and the algorithm measured the number with an average error of 1 jump. For the action Jog on the Spot, the number of jogs was recorded manually during the recording of video. We compared this result to the results

calculated by the number of jogs calculation algorithm for 50 random video and the algorithm measured the number with an average error of 2 jogs.

	Skeletal Joint Based		**Frame Images Based**	
	Random Forest Classifier	XGBoost Classifier	CNN Model	Transfer Learning + CNN
Jump High	0.99	0.98	0.79	0.73
Jump Forward	0.99	0.94	0.54	0.61
Jump Sideways	0.99	0.90	0.57	0.57
Jog on the Spot	0.99	0.95	0.31	0.35

Table 6.1: F1 Scores of Different Image Based Video Frames Classifier Models on Train Data

	Skeletal Joint Based		**Frame Images Based**	
	Random Forest Classifier	XGBoost Classifier	CNN Model	Transfer Learning + CNN
Jump High	0.95	0.95	0.79	0.72
Jump Forward	0.94	0.91	0.54	0.61
Jump Sideways	0.87	0.83	0.56	0.56
Jog on the Spot	0.89	0.90	0.31	0.35

Table 6.2: F1 Scores of Different Image Based Video Frames Classifier Models on Test Data

	Skeletal Joint Based	
	Random Forest Classifier	XGBoost Classifier
Jump High	0.95	0.95
Jump Forward	0.94	0.92
Jump Sideways	0.87	0.84
Jog on the Spot	0.88	0.90

Table 6.3: Cross Validation Scores of Different Image Based Video Frames Classifier Models

7 Conclusion and Future Work

7.1 Conclusion

In this study a system where videos are classified into the action being performed and then analysed frame by frame to track the action throughout the video and calculate the results of the actions based on the specified criteria is proposed. The Video classifiers will categorise the videos into one of the four actions (Jump High, Jump Forward, Jump Sideways and Jog on the Spot) and then track the actions across the frames throughout the video using the frame images classifiers to evaluate the results of the actions. The video classifiers are LSTM networks combined with pretrained neural network (EfficientNetB7). These are trained on the available dataset from the research conducted in [19] and was able to get an F1 score of 0.62 on the test data. This shows that the network has classification power but needs more training data for it capture all the information related to each action. Each frame was labelled with an intermediate status using a heuristic defined based on the skeletal joint values based on the MediaPipe Library. This was further used to train different types of classification models either with the skeletal joint values as the input or the frame images as the input. From the experiment results, the models based on Skeletal Joint Value Inputs (Random Forest Classifier and XGBoost Classifier) outperformed the models based on Frame Image Inputs. The Random Forest Classifier achieved the best results with average F1 scores of 0.95, 0.94, 0.87 and 0.89 for the actions Jump High, Jump Forward, Jump Sideways and Jog on the Spot respectively. The distance calculation algorithm for the action Jump High was able to find the height of the jump relative to the frame size but was unable to scale it to a real-world distance because of the lack on any reference distance in the videos. The distance calculation algorithm for the action Jump Forward was able to find the distance jumped as there was reference distance marking along the horizontal direction in the videos which is used to scale the frame distance to a real-world distance. This distance was measured with an average error of 10 cm. For the action Jump Sideways, the number of jumps were calculated with an average error of 1 jump. The number of jogs for the action Jog on the Spot was calculated with an average error of 2 jogs.

7.2 Future Work

This study has focussed only on four human actions and the dataset used was limited to the videos recorded as part of the research conducted in [19]. The same systems and methods can be extended to various actions and generic video datasets with changes in the heuristics and the criteria calculation algorithms. For future work, this study can be adapted to other actions involving repetition of the same movements by figuring out the intermediate stages of the actions to track based on heuristics. Similarly, for actions where distance needs to be calculated, additional steps to record a reference distance from the video environment need to be included to make the system more invariant to various video data sources. Furthermore, the systems proposed in this study can be made more autonomous by letting the machine learn the heuristics on its own rather than specifying the parameters by training machine learning model which can capture the patterns emerging from the tracking of skeletal joint values.

Bibliography

[1] Chakraborty, S., Mondal, R., Singh, P. K., Sarkar, R. & Bhattacharjee, D. Transfer learning with fine tuning for human action recognition from still images. *Multimed. Tools Appl.* **80**, 20547–20578 (2021).

[2] Zhang, P. et al. Semantics-guided neural networks for efficient skeleton-based human action recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 1112–1121 (2020).

[3] Guha, R., Khan, A. H., Singh, P. K., Sarkar, R. & Bhattacharjee, D. Cga: A new feature selection model for visual human action recognition. *Neural Comput. Appl.* **33**, 5267–5286 (2021).

[4] Ye, F. et al. Dynamic gcn: Context-enriched topology learning for skeleton-based action recognition. In Proceedings of the 28th ACM International Conference on Multimedia, 55–63 (2020).

[5] Huang, L., Huang, Y., Ouyang, W. & Wang, L. Part-level graph convolutional network for skeleton-based action recognition. Proc. AAAI Conf. Artif. Intell. 34, 11045–11052 (2020).

[6] W. Zhu, C. Lan, J. Xing, W. Zeng, Y Li, L. Shen, et al., "Co-occurrence Feature Learning for Skeleton based Action Recognition using Regularized Deep LSTM Networks", *2016 Association for the Advancement of Artificial Intelligence (AAAI)*, pp. 3697-3703.

[7] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue and N. Zheng, "View Adaptive Recurrent Neural Networks for High Performance Human Action Recognition from Skeleton Data", *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2136-2145.

[8] Z. Cao, G. Hidalgo, T. Simon, S. -E. Wei and Y. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields," in IEEE

Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 1, pp. 172-186, 1 Jan. 2021, doi: 10.1109/TPAMI.2019.2929257.

[9] H. Xu, E. G. Bazavan, A. Zanfir, W. T. Freeman, R. Sukthankar and C. Sminchisescu, "GHUM & GHUML: Generative 3D Human Shape and Articulated Pose Models," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 6183-6192, doi: 10.1109/CVPR42600.2020.00622.

[10] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K. and Grundmann, M., 2019. Blazeface: Sub-millisecond neural face detection on mobile gpus. arXiv preprint arXiv:1907.05047.

[11] X. Liang, H. -B. Zhang, Y. -X. Zhang and J. -L. Huang, "JTCR: Joint Trajectory Character Recognition for human action recognition," 2019 IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE), 2019, pp. 350-353, doi: 10.1109/ECICE47484.2019.8942672.

[12] Y. -Y. Jhuang and W. -J. Tsai, "DeepPear: Deep Pose Estimation and Action Recognition," 2020 25th International Conference on Pattern Recognition (ICPR), 2021, pp. 7119-7125, doi: 10.1109/ICPR48806.2021.9413011.

[13] A. Shahroudy, J. Liu, T. Ng and G. Wang, "NTU RGB+D: A Large-Scale Dataset for 3D Human Activity Analysis", *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1010-1019.

[14] N. S. A. Latha and R. K. Megalingam, "Exemplar-based Learning for Recognition & Annotation of Human Actions," 2020 9th International Conference System Modeling and Advancement in Research Trends (SMART), 2020, pp. 91-93, doi: 10.1109/SMART50582.2020.9337151.

[15] S. Park and D. Kim, "Study on 3D Action Recognition Based on Deep Neural Network," 2019 International Conference on Electronics, Information, and Communication (ICEIC), 2019, pp. 1-3, doi: 10.23919/ELINFOCOM.2019.8706490.

[16] Sehatbakhsh, Nader and Daw, Ellie and Savas, Onur and Hassanzadeh, Amin and McCulloh, Ian, “Security and Privacy Considerations for Machine

Learning Models Deployed in the Government and Public Sector", arXiv, 2020. Available at: https://arxiv.org/ftp/arxiv/papers/2010/2010.05809.pdf .

[17] J. Chou, E. Al-Masri, S. Kanzhelev and H. Fattah, "Detecting Security and Privacy Risks in Microservices End-to-End Communication Using Neural Networks," 2021 IEEE 4th International Conference on Knowledge Innovation and Invention (ICKII), 2021, pp. 105-110, doi: 10.1109/ICKII51822.2021.9574757.

[18] H. Xie, L. Wei and F. Fang, "Research on Privacy Protection Based on Machine Learning," 2021 International Wireless Communications and Mobile Computing (IWCMC), 2021, pp. 1003-1006, doi: 10.1109/IWCMC51323.2021.9498632.

[19] Bossavit, B., Arnedillo-Sánchez, I. (2020). Designing Digital Activities to Screen Locomotor Skills in Developing Children. In: Alario-Hoyos, C., Rodríguez-Triana, M.J., Scheffel, M., Arnedillo-Sánchez, I., Dennerlein, S.M. (eds) Addressing Global Challenges and Quality Education. EC-TEL 2020. Lecture Notes in Computer Science(), vol 12315. Springer, Cham. https://doi.org/10.1007/978-3-030-57717-9_37

Appendix

Abbreviations

PAF	Part Affinity Field
ROI	Region of Interest
PCK@0.2	Percent of Correct Points with 20% tolerance
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory Network
CAE	Convolutional Autoencoder models