School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Human Action Recognition

Zonglin Ji

Submitted in accordance with the requirements for the degree of MSc Advanced Computer Science (Data Analytics)

2018/2019

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Deliverable 1	Report	SSO & VLE (04/09/2019)
Deliverable 2	Software code URL	Supervisor & Assessor (04/09/2019)

Type of project: **Exploratory Software**

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) [illegible]

Summary

The area of computer vision is not only satisfied with object recognition and image classification. Another essential yet challenging task called human action recognition has also brought to the table. By extracting spatial and temporal features from frames to model and analyse evolutions of different actions, computers can identify and classify human behaviours. It is beneficial in the real world in multiple areas such as babysitting, elderly well-being or even counter-terrorism.
The goal of this project is to develop a classifier model that can distinguish and classify different actions on human skeleton based dataset. This report introduces the relevant research in this application field, followed by a CRISP-DM (Cross-industry standard process for data mining) procedure to apply the experiment. To be specific, there are several divided objectives:

1. Identify an appropriate data set and make data understanding on it.
2. Data preprocessing and preparation for the model experiment. This task is challenging and will be done more than one time in order to improve coping with models.
3. Build appropriate classification models.
4. Experiments and evaluations.

Acknowledgements

The experiment in this paper used the NTU RGB+D Action Recognition Dataset made available by the ROSE Lab at the Nanyang Technological University, Singapore. Also, thanks to Dr Jun Liu, who is very generous that answered my questions.
First, I would like to express my appreciation to my supervisor, Dr Andy Bulpitt, for all his patience and support throughout the process of this project. We had a lot of invaluable discussions that inspired me. This project could not be done without his wisdom. Under his guidance, I have learnt and improved my knowledge in this area as well as my skills.
I am also grateful to my assessor Dr Vania Dimitrova. I was super stressful and had no confidence when I had the progress meeting with her. She gave me positive feedback and encouragement that is a relief to me. She also gave me very practical and constructive suggestions for next steps.
There were many downtimes while doing this project. My parents are the solid back stones that kept encouraging me when I was stressed and frustrating. They are the ones that motivated me through the dark times.
Finally, I want to thank a special colleague to me, Miss Xiangyin Chen. She accompanied me this whole time and brought me happiness in this period of depression.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Human action recognition plays a vital role in video understanding. It applies to many occasions such as intelligent surveillance like babysitting, or video understanding and interpretation. To achieve a goal like this, the spatial and temporal evolutions in the actions would both be playing a critical part and thus need to be considered together. Extracting discriminative spatial-temporal features from action videos and training with an appropriate model is the key to this task. Generally, there are two approaches to the

Figure 1.1: An example of human actions

task. One focuses on the information from RGB videos, however, due to its excessive dependence on the clarity of videos, and the capture of human action in a frame is 2 dimensional which would miss some information because of the overlap when actions are taking place in 3-dimensional space. That means it relies on good captions, which is not possible for the real-world scenario; in other words, its applicability is not good [17]. This project focuses on another approach, which is recognition from skeleton data that represents a person by the 3-dimensional coordinate of key joints. By using this representation, the recognition would gain robustness against the evolution of depth movement as well as background noise.

1.2 Aim and Objectives

This project aims to develop a classifier that can classify different actions represented by skeletal movements as numerical 3-dimensional coordinates of body joints. Since it is a multi-class classification task and the data volume is also massive. Deep learning is more suitable than traditional machine learning algorithms like Support Vector Machine.
The project can be divided into the following steps:

1. **Data Acquisition** This project is not provided with data, so the first step is to find a suitable dataset. The dataset this project requires is several classes of actions with reasonable 3-dimensional coordinates that represent joints of a body, preferably with a fixed number of joints and varieties of actors on the same action.

2. **Data Processing** If the data acquired with a well-explained document, the data can be processed based on that. The steps of processing the data include extraction from original files, sorting it into the desired shape, and then writing the data into separate files with the dividing of training and testing set.

3. **Modelling and Evaluations** Building several models in different approaches, evaluate each model to make sure it works. Compare performance differences among the models, find the best one.

1.3 Problem Statement

There are two representative networks in deep learning. Recurrent Neural Networks (RNN) can learn the temporal pattern, such as the grammar in the sentence of a natural language. Convolutional Neural Networks (CNN), on the other hand, can learn the spatial features in a single image.
However, in our task, actions have two evolution tracks, time sequence and spatial change. For example, a person is raising his hand. In the period of this action, his movement takes a certain time to complete that a single image in the action video cannot represent its entire actions, considering there are more complicated actions than raise a hand. On the other hand, even we consider the time sequence, we cannot identify an action just because of how many time has passed with rough information inside the video.
In conclusion, a classifier should consider both temporal orders as well as spatial evolutions within action videos.

1.4 Academic Relevance

This project is conducted based on the knowledge of MSc students in Advanced Computer Science. The relevant modules that have taught in this year of master study are COMP3611 Machine Learning and COMP5870M Image Analysis. However, the taught materials in these modules are only reached to simple image classification using CNN. RNN is not included in the whole year of study. Also, because this task requires to consider both temporal and spatial evolutions, and two different methods cannot simply combined. The difficulty of this project should be considered as way beyond the taught materials.

1.5 Project Planning

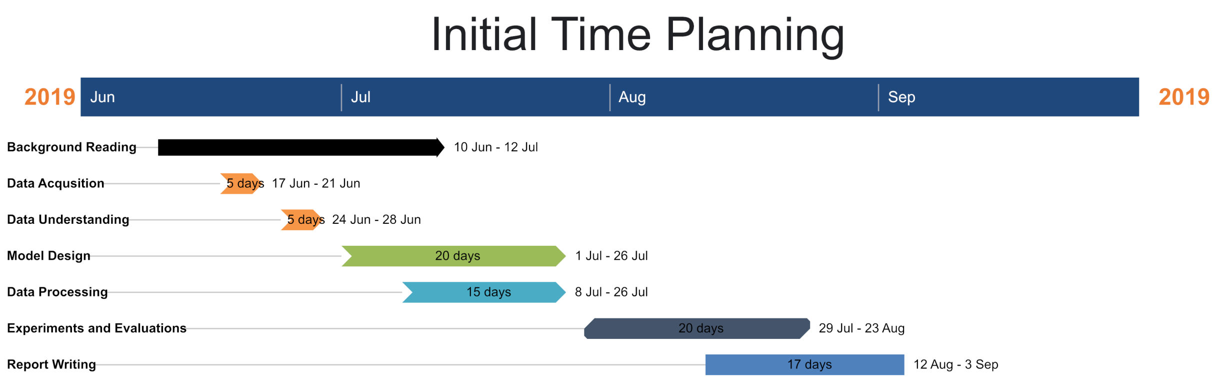


Figure 1.2: Initial Time Planning

The initial time plan in Fig 1.2 is made based on the information in April 2019, where a data was promised to provide to this project. However, when the project starts, the data is not provided due to some reasons. So data acquisition took longer than expected because of the negotiations and authorisations taking in finding new data. Therefore all of the implementation work has postponed. The rest of the time has to be squeezed in order to guarantee to finish the project on time. The actual timetable is shown in Fig 1.3.

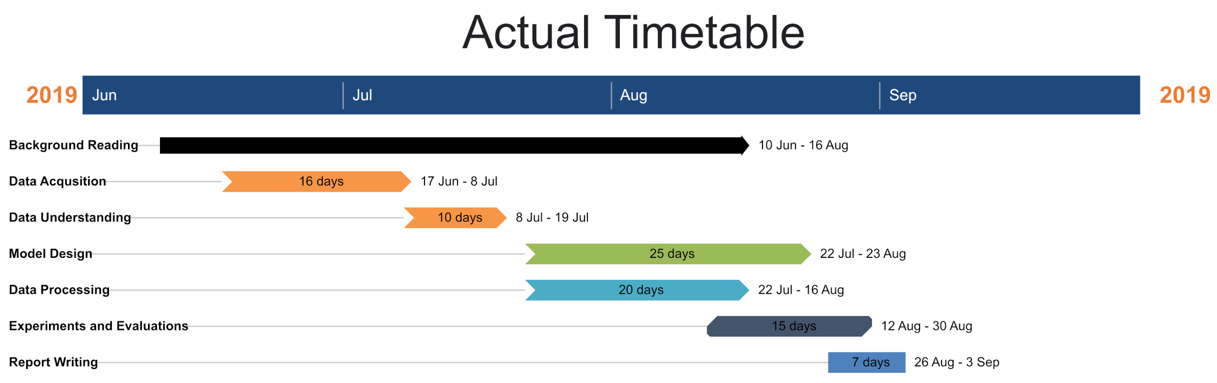


Figure 1.3: Actual Timetable

1.6 Report Structure

This project report has 7 chapters outlined as following:

- **Chapter 1** General introduction and background of the project with initial time planning and actual timetable.
- **Chapter 2** Background research with a literature review about the state-of-arts concepts, theories and methods.
- **Chapter 3** Data understanding with a discussion of the nature of data and proposal on how to use and prediction of experimental outcomes.
- **Chapter 4** Model design under the understanding of data in the last chapter, building several different models with different approaches identified in background research in order to compare for the best.
- **Chapter 5** Data processing based on the understanding of data and the model built in the last chapter, although the processed data should comply with the model, it should not sacrifice its unique features.
- **Chapter 6** Experiments on models, design a practical evaluation method and use for comparing between models.
- **Chapter 7** Project conclusion and discussion about its limitations and future improvements along with personal reflection.

Chapter 2

Background Research

2.1 3D Skeleton Representation

Human action recognition has become an important research area in computer vision in the past decades, but most of the researches before are studied on RGB-based approaches rather than 3D skeleton-based approaches [18] because RGB-based data is easier to acquire and process, where skeleton extraction would be much more difficult to produce therefore the data would be expensive. However, with the development of new skeleton capture systems like Microsoft Kinect that can capture human actions even without wearable gears shown in Fig 2.1 available in recent years, the skeleton data became cheaper, and large scale datasets in 3D skeleton representation are accessible in these days [4]. Therefore attracted many interests a lot of researches has taken on data in skeleton representation since it is way more effective than RGB-based videos on robustness to noise and background.

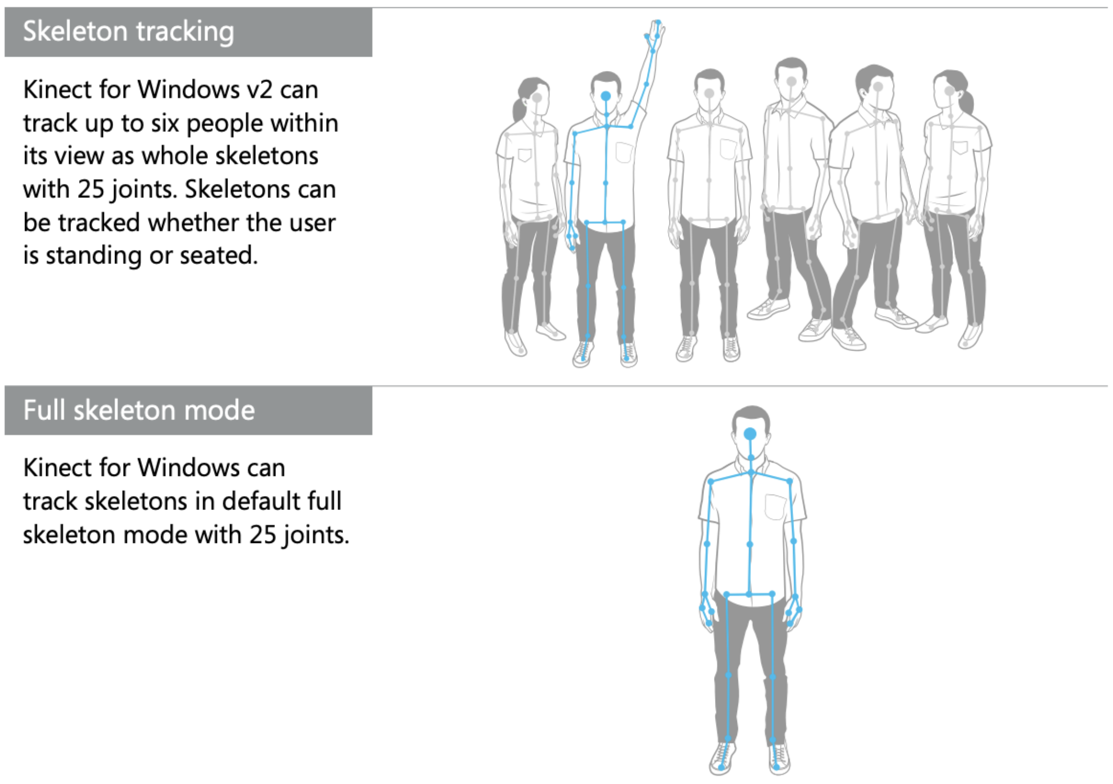

Skeleton tracking
Kinect for Windows v2 can track up to six people within its view as whole skeletons with 25 joints. Skeletons can be tracked whether the user is standing or seated.
Full skeleton mode
Kinect for Windows can track skeletons in default full skeleton mode with 25 joints.

Figure 2.1: Microsoft Kinect V2 Cameras capturing human skeleton actions [7]

This project aims to build a classifier that can classify the human actions captured in 3D skeletons. Deep learning method is employed because of its advantages in dealing with this kind of tasks. In this chapter we are introducing the concepts of deep learning and how should it work with the task to classify human actions.

2.2 Machine Learning

Before introducing deep learning, we need to get acquaintance with machine learning which deep learning is the subfield of.
In traditional statistics, a result is usually calculated by an algorithm with a clearly indicated direction. However, if the data changes in two ways, old-fashioned calculation could be powerless:

1. The amount of data has increase largely that is not possible to be observed by human eyes.

2. Wide varieties of data that would be hard to find the association rules between them.

Sampling is the most common way to deal with large data before, but sampling needs time-consuming preparation task, and it does not guarantee the correctness of the result. Brexit is the recent most famous proof of how traditional polling got it wrong. They have carefully selected the representative of different demographics within the electorate in a small group and ask questions, then gauge fluctuations in opinions to the whole country [8]. This method went wrong this time because they have ignored the actual changes in a large number of people.
Machine learning, on the other hand, it is the scientific study of statistical models that relies on patterns and inference in the data. It can build a mathematical model based on a large scale of samples known as the training set, and there are no explicit calculations programmed except the core algorithms. So every predictions and decision purely depend on data itself. [13].

2.2.1 Deep Learning

Machine learning is proposed in the 1950s when computers are not really powerful. Modern computer technologies are no longer the same as that time. Also, in the 21st century, the data is growing rapidly that beyond anyone could imagine before. An era of big data has come. Although traditional machine learning algorithms are sufficient for dealing with the relatively large number of data, it is not enough fast and effective in dealing with modern days big data and hard to improve the result.
Therefore, deep learning is coming to the stage. Most of the deep learning models are based on artificial neural networks, which initially trying to simulate the process of the human brain. It uses multiple layers to increase the complexity of networks and the performance of the models. With the computing power grows these years, deep learning

repeatedly shows its superiority over the traditional machine learning. Not only the processing speed, but deep learning is also better in the input varieties, data diversities, data scales and more, one example comparison is shown in Fig 2.2.

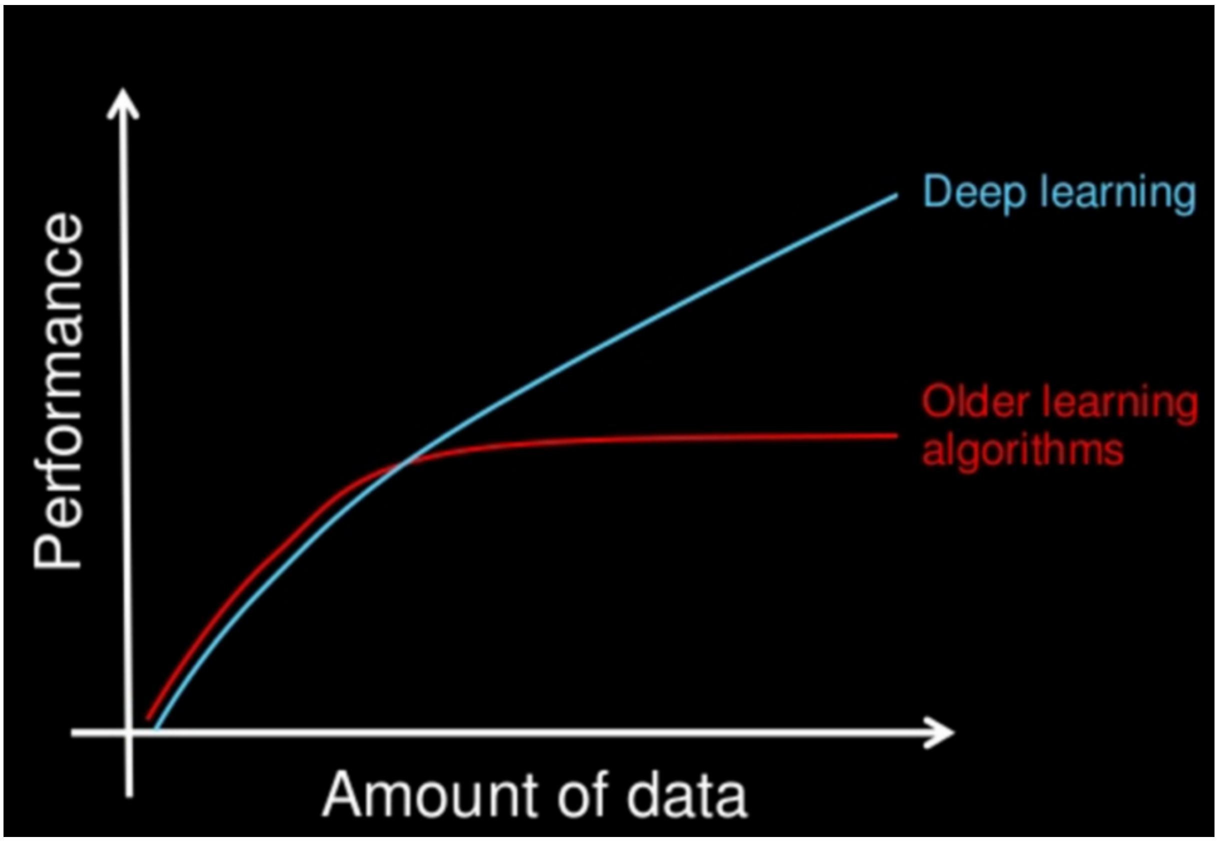


Figure 2.2: Deep Learning VS Machine Learning on data scales

Moreover, traditional machine learning algorithms require feature engineering to reduce the dimension of data. Deep learning can load the data directly to the network, and because of its multi-layer structure, the data normalisation and feature learning can be done internally, which eliminates the challenges of feature engineering and dimensionality reduction.
Also, deep learning is easy to transfer, because the model structure is separated from data inputs, a well-trained model can be transferred to a new data in similar applications.

2.2.2 Supervised Learning

For deep learning models, learning can be supervised, semi-supervised or unsupervised. In this project, since all the data is already human crafted, it is typical supervised learning. Therefore we skip the rest of two learning methods, only introduce supervised learning.
As shown in Fig 2.3, supervised learning means learning is based on example input-output pairs [15]. In other words, for every input sample, there is one label in the

corresponding to it. Supervised learning is mostly using for classification tasks. So more samples that are provided, the more accurate the model gets when testing.

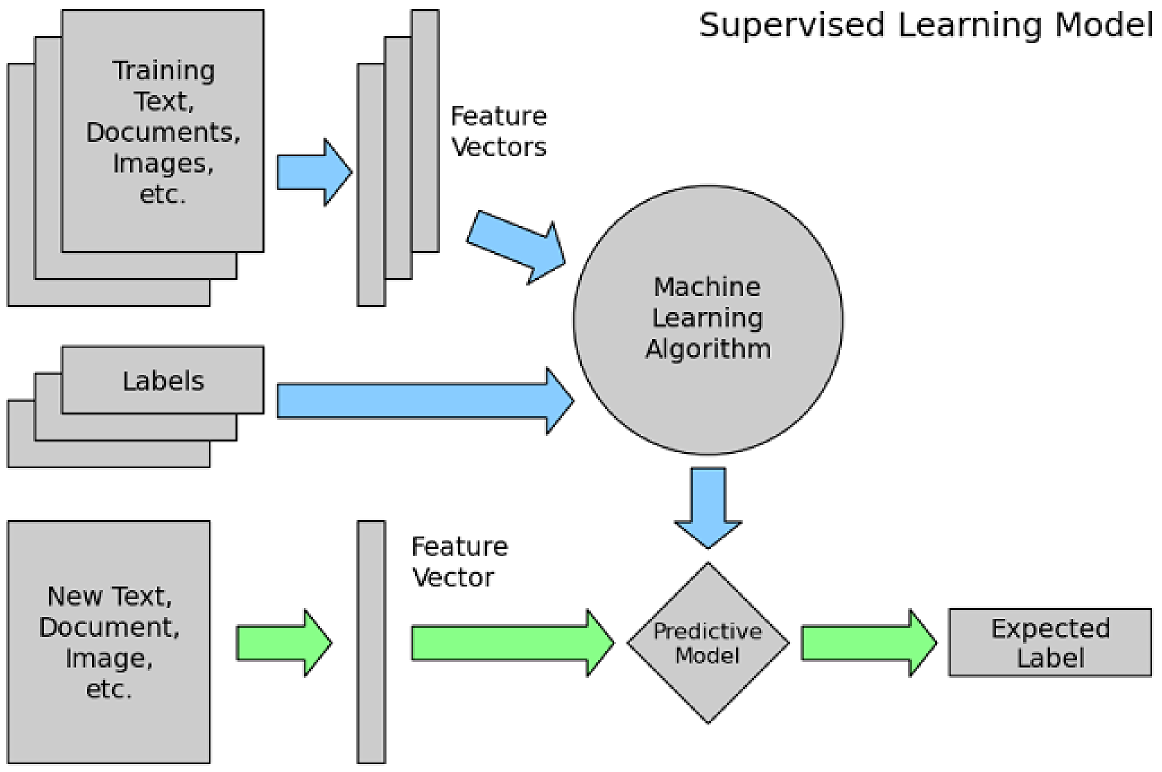

Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

Figure 2.3: Supervised Learning Model [3]

2.3 Multi-Layer Perceptron

The basic artificial neural network in deep learning is Multi-Layer Perceptron (MLP), the structure is shown in Fig 2.4. The simplest MLP has three layers. The number of neurons in the input layer usually would show how many dimensions are there in the input data, and the output layer for classification tasks usually only has one neuron which is the calculated predicted label. In the middle, there is a hidden layer section, the number of hidden layers, as well as the number of neurons in each layer, can be defined by the user. Generally speaking, the more complex the problem, the more hidden layers and neurons is needed. However, with a more complex network, the training of the network would require more time, but the performance would not have significant improvement after reaching some point.
However, the basic MLP would reach the bottleneck when dealing with a particular task like this project. In this project, the actions are stored in videos which generally are series of sequential images so that the input length would be different, that is hard to process when fully connected neural network has fixed input and output shape. Moreover, all the neurons in the same layer of MLP are not connected. In this project,

we need feature sharing within the same layer, which is impossible for MLP neurons.

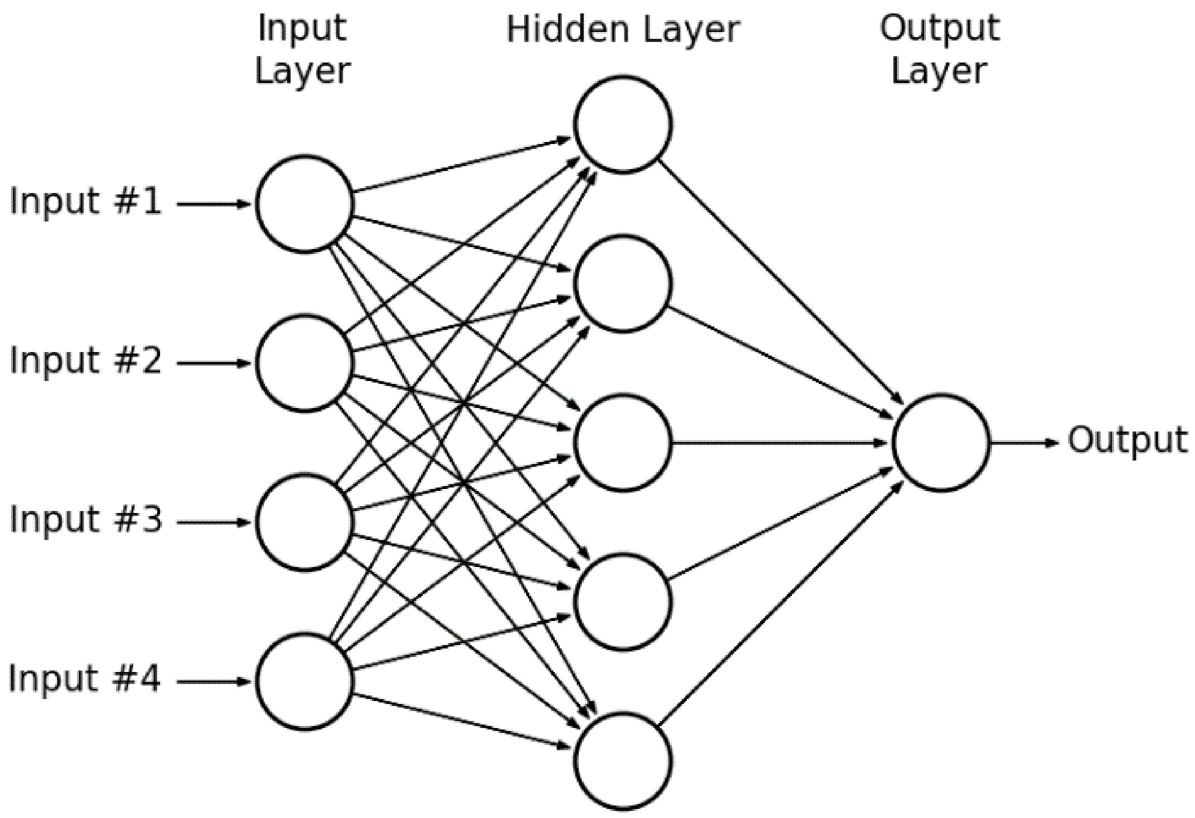


Figure 2.4: Multi-Layer Perceptron Structure

2.4 Recurrent Neural Network

In this project, since we are dealing with time series input. In order to break the limitations of basic MLP, we employed Recurrent Neural Network (RNN) for this task. Fig 2.5 shows a classic structure of RNN:

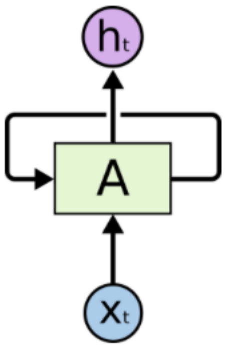


Figure 2.5: Classic RNN Cell Structure

For RNN, the most important concept is time sequence. The RNN gives an output for each moment's input combined with the state of the current model. In Fig 2.5, the input of the main structure A of the RNN at time t is not only from current input layer x_t, there is also a hidden state from time $t-1$ that would affect the output of this time h_t. RNN can be viewed as the result of replicating the same neural network structure in time series like the concept graph shows in Fig 2.6.

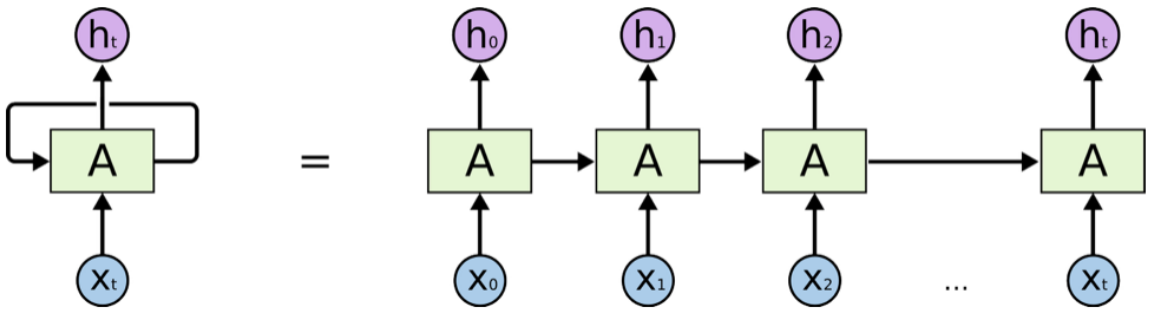


Figure 2.6: An Unrolled RNN (Looks Like An replication for one Network)

Therefore, it is not hard to understand why RNN is best for dealing with time related problems.
As the main structure of RNN, A is also replicate multiple time in corresponding to the time series, Fig 2.7 shows the inside structure of A.

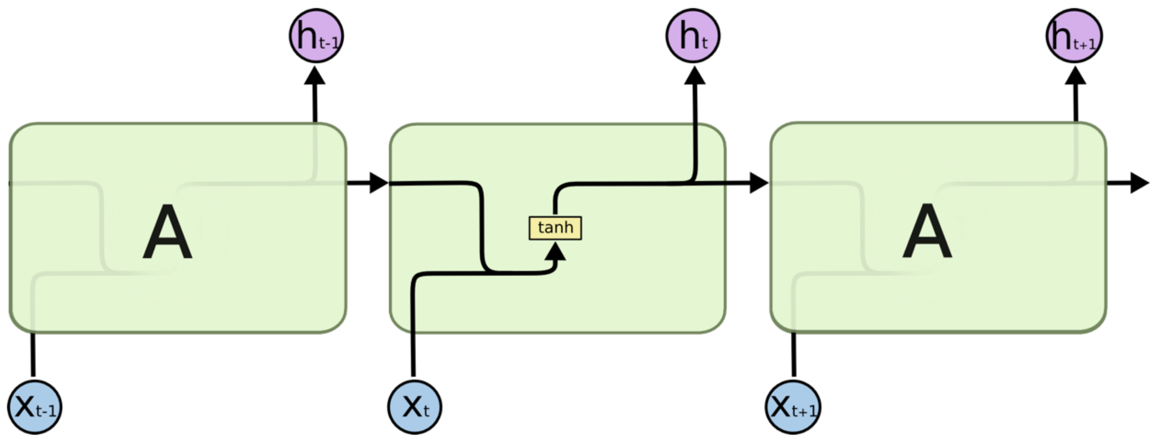


Figure 2.7: Loop Body in RNN structure

In time t, loop body A has input of x_t and the hidden state h_{t-1} from time $t-1$. The calculation of this time is shown in Equation 2.1.

$$\begin{aligned} h_t &= softmax\,(V \cdot s_t) \\ s_t &= tanh\,(U \cdot x_t + W \cdot s_{t-1}) \end{aligned} \tag{2.1}$$

2.4.1 Long Short-Term Memory

Although theoretically, RNN can support any length sequence. However, in practice, there may have a huge gap between the relevant information and the point where it is needed, and unfortunately due to the single update sequence in RNN structure, as the gap grows, RNN cannot connect the information that is so far away. This problem is called long-term dependencies.
Long Short-Term Memory (LSTM) is a modified RNN network, capable of dealing with long-term dependencies. By using LSTM, remembering information for long periods of time is possible. It introduces gates concept into RNN instead of having a single neuron in the loop, the structure is shown in Fig 2.8.

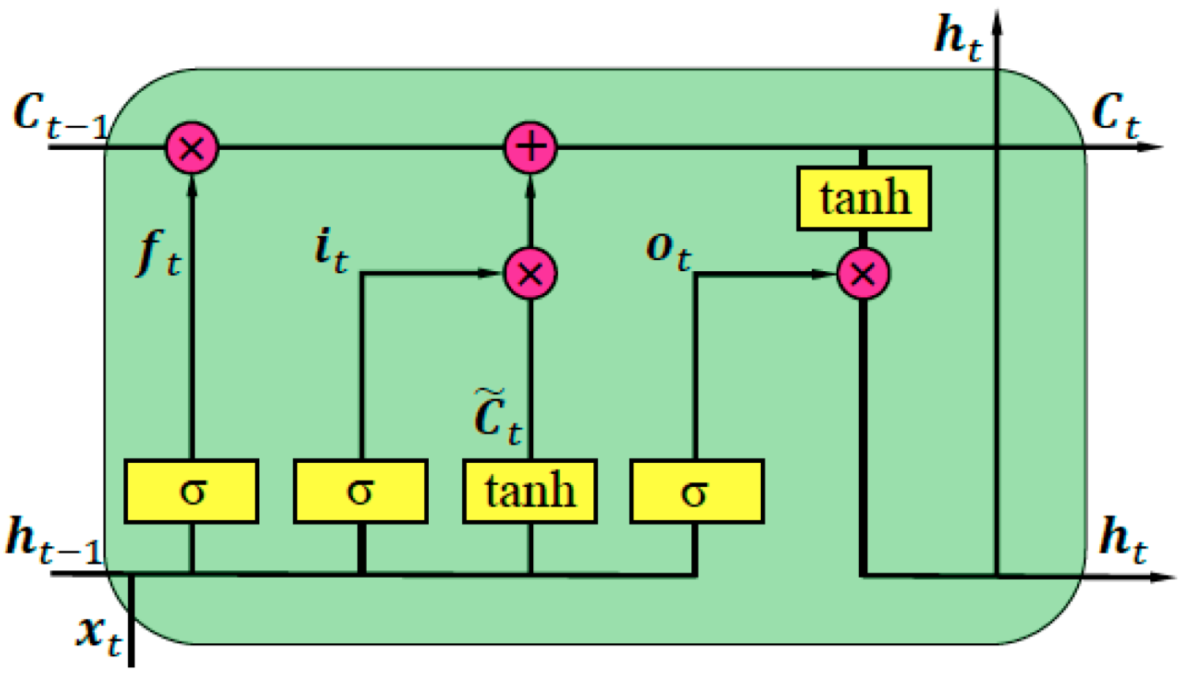


Figure 2.8: Loop Body in LSTM with Four Gates

There are four gates in an LSTM cell shown in the Fig 2.8, the processing steps introduce in the following:

- **Forget Gate** Input with h_{t-1} which is the last hidden state and input of present x_t, output a number between 0 and 1 for each number in the cell state C_{t-1}. 1 stands for keep all information while 0 stands for completely forget. Calculation is shown in 2.2.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{2.2}$$

- **Input Gate** This gate decide the values to be updates. The calculation is shown in Equation 2.3

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{2.3}$$

- **Input Modulation Gate** This gate creates a candidate value by Equation 2.4 for adding to the state. After generate a candidate, update the new cell state C_t with the result from input gate i_t.

$$\begin{aligned} \tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{2.4}$$

- **Output Gate** Calculate the final output value using Equation 2.5 based on the cell state and the previous hidden state with input.

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\ h_t &= o_t * \tanh\left(C_t\right) \end{aligned} \tag{2.5}$$

2.4.2 Part-Aware LSTM

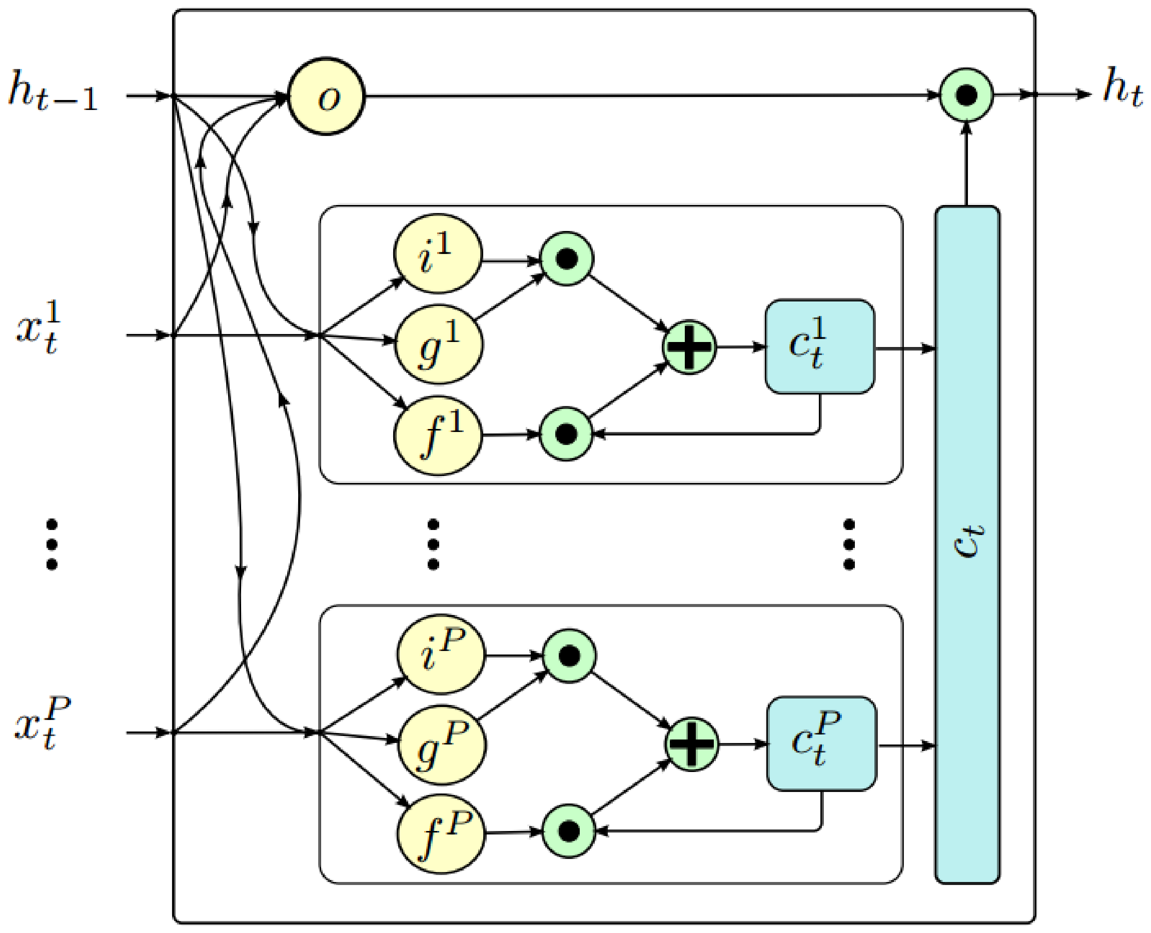


Figure 2.9: Unit Illustration Part-Aware LSTM Cell Proposed by NTU paper [9]

According to recent skeleton-based action recognition literature, RNN is mainly used to model long-term information across the temporal dimension. However, the spatial

domain is also essential to recognise the actions [6]. In order to improve the performance of interpreting body motions, Jun Liu and his colleagues proposed a Part-Aware LSTM in their paper [9].
In their proposal, instead of make one LSTM cell to learn the entire body, they have split the body joints into several part-based cells [9]. The concept graph is shown in Fig 2.9. Each cell has an LSTM structure without an output gate, the input to each cell would be a divided body part, after calculating the update cell state for each cell, concatenate them and go through the shared output gate with the whole body input to generate the final output.
Therefore, the calculation of models would be modified as Equation 2.6:

$$\begin{aligned}
f_t^P &= \sigma\left(W_f^P \cdot \left[h_{t-1}, x_t^P\right] + b_f^P\right)\\
i_t^P &= \sigma\left(W_i^P \cdot \left[h_{t-1}, x_t^P\right] + b_i^P\right)\\
\tilde{C}_t^P &= \tanh\left(W_C^P \cdot \left[h_{t-1}, x_t^P\right] + b_C^P\right)\\
C_t^P &= f_t^P * C_{t-1}^P + i_t^P * \tilde{C}_t^P\\
o_t &= \sigma\left(W_o\left[h_{t-1}, (x_t^1 + \ldots + x_t^P\right] + b_o\right)\\
h_t &= o_t * \tanh\left(\left[C_t^1 + \ldots + C_t^P\right]\right)
\end{aligned} \tag{2.6}$$

The reason why modifying LSTM cell like this rather than loading a list of parts to the same cell is to maintain each parts weights and bias because each part of the body may have its own motion. If loading all parts into one cell, it would break the continuity of calculations by parts since the cell needs to update by the situation of each part. It would be no different than a normal LSTM cell.

2.5 Convolutional Neural Network

Convolutional Neural Networks (CNN) are regularised versions of MLP, take advantage of a hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns [12]. Therefore CNN is well suitable for extracting features from images and has achieved great success in many image recognition tasks.

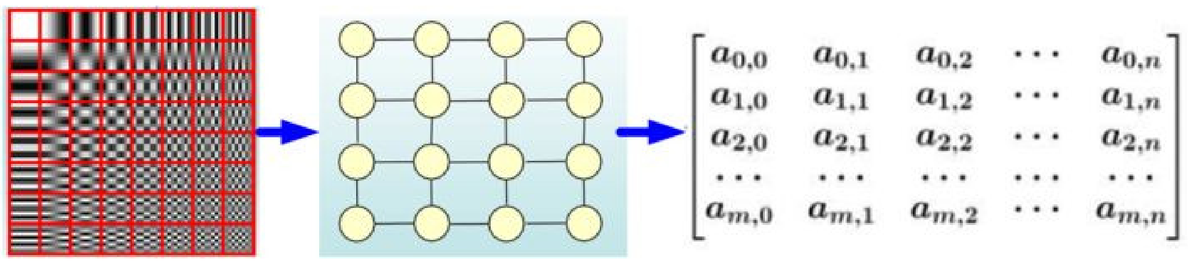

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $\cdots$ $a_{0,n}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $\cdots$ $a_{1,n}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $\cdots$ $a_{2,n}$
$a_{m,0}$ $a_{m,1}$ $a_{m,2}$ $\cdots$ $a_{m,n}$

Figure 2.10: Extracting Features in CNN (Euclidean Structure)

As mentioned above, although LSTM approaches can learn the temporal evolutions from actions efficiently, learning spatial features is not the advantage of it. CNN, on the other hand, is the best method for images. However, since skeleton data shown in Fig 2.11 is only coordinated rather than full-scale images that CNNs typically learn as shown in Fig 2.10, it is so-called non euclidean structure which requires preprocessing before the model can handle.

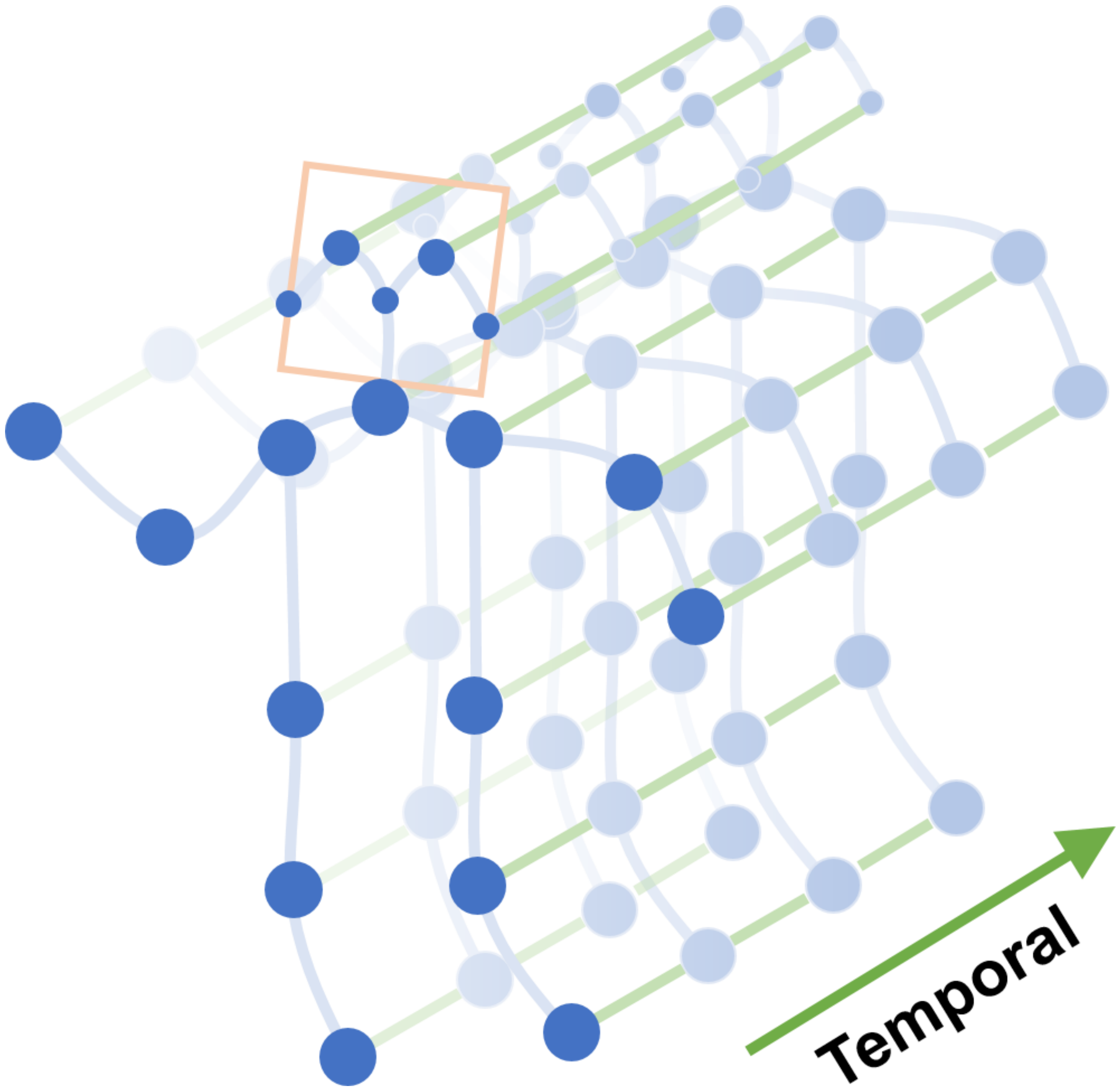


Figure 2.11: Spatial Temporal Graph of a Skeleton Sequence in the work of ST-GCN (Non Euclidean Structure)[17]

2.5.1 Graph Convolutional Network

Recently, generalised CNN to graphs of arbitrary structures have gain huge success, this modified network called Graph Convolutional Networks (GCN) can deal with non euclidean structure by using spectral graph theory to learn the association within graph nodes. In this project, we use GCN to construct spatial-temporal graph on the skeleton sequences in two steps. First link body joints according to the connectivity of human body structure in single frame, and then each joint connect to the same joint in the following frames [17].

2.5.2 Spatial-Temporal GCN

Sijie Yan and his colleagues proposed a Spatial-Temporal GCN (ST-GCN) in their paper [17]. By construct the spatial-temporal graphs as explained above, a partition strategy can be used on the graphs for learning the patterns. For example, one of the partitioning strategy shown in Fig 2.12 is to divide body joints in single frame into parts and label them. Then, the network would learn the distance of each frames to model its spatial evolutions among time in an action video.

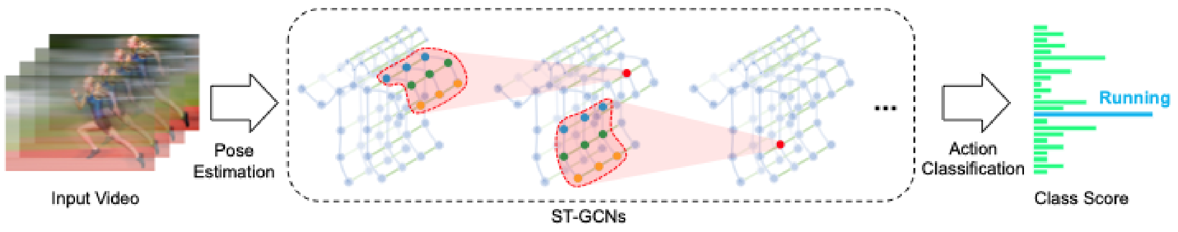


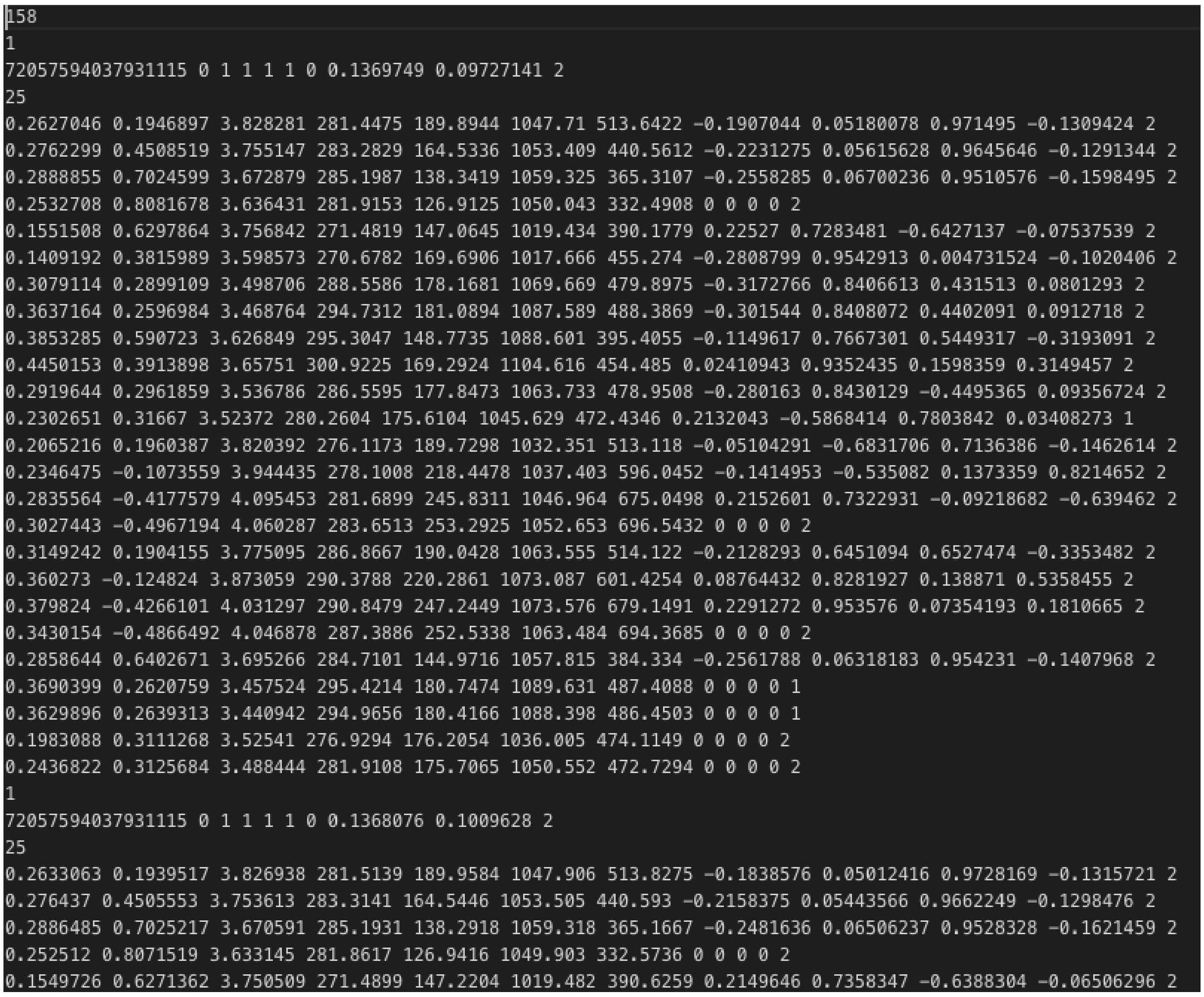
Figure 2.12: Partition Strategy Plays a Vital Role of the Whole Process of ST-GCN

Chapter 3

Data Understanding

3.1 Data Acquisition

The data this project uses for its experiments is acquired from the Rapid-Rich Object Search Lab at Nanyang Technological University in Singapore. It is currently the largest publicly available dataset for skeleton action recognition. The data contains 3-dimensional skeletal data videos for each sample, which is captured by three Microsoft Kinect V2 cameras concurrently. The data is provided only by pre-request for authorisation, and it should not be upload to the Internet personally.
There is no document along with the dataset, only a few brief introductions on Github. However, some of the information regarding the data could be found on the paper the data provider has published.



```
158
1
72057594037931115 0 1 1 1 1 0 0.1369749 0.09727141 2
25
0.2627046 0.1946897 3.828281 281.4475 189.8944 1047.71 513.6422 -0.1907044 0.05180078 0.971495 -0.1309424 2
0.2762299 0.4508519 3.755147 283.2829 164.5336 1053.409 440.5612 -0.2231275 0.05615628 0.9645646 -0.1291344 2
0.2888855 0.7024599 3.672879 285.1987 138.3419 1059.325 365.3107 -0.2558285 0.06700236 0.9510576 -0.1598495 2
0.2532708 0.8081678 3.636431 281.9153 126.9125 1050.043 332.4908 0 0 0 0 2
0.1551508 0.6297864 3.756842 271.4819 147.0645 1019.434 390.1779 0.22527 0.7283481 -0.6427137 -0.07537539 2
0.1409192 0.3815989 3.598573 270.6782 169.6906 1017.666 455.274 -0.2808799 0.9542913 0.004731524 -0.1020406 2
0.3079114 0.2899109 3.498706 288.5586 178.1681 1069.669 479.8975 -0.3172766 0.8406613 0.431513 0.0801293 2
0.3637164 0.2596984 3.468764 294.7312 181.0894 1087.589 488.3869 -0.301544 0.8408072 0.4402091 0.0912718 2
0.3853285 0.590723 3.626849 295.3047 148.7735 1088.601 395.4055 -0.1149617 0.7667301 0.5449317 -0.3193091 2
0.4450153 0.3913898 3.65751 300.9225 169.2924 1104.616 454.485 0.02410943 0.9352435 0.1598359 0.3149457 2
0.2919644 0.2961859 3.536786 286.5595 177.8473 1063.733 478.9508 -0.280163 0.8430129 -0.4495365 0.09356724 2
0.2302651 0.31667 3.52372 280.2604 175.6104 1045.629 472.4346 0.2132043 -0.5868414 0.7803842 0.03408273 1
0.2065216 0.1960387 3.820392 276.1173 189.7298 1032.351 513.118 -0.05104291 -0.6831706 0.7136386 -0.1462614 2
0.2346475 -0.1073559 3.944435 278.1008 218.4478 1037.403 596.0452 -0.1414953 -0.535082 0.1373359 0.8214652 2
0.2835564 -0.4177579 4.095453 281.6899 245.8311 1046.964 675.0498 0.2152601 0.7322931 -0.09218682 -0.639462 2
0.3027443 -0.4967194 4.060287 283.6513 253.2925 1052.653 696.5432 0 0 0 0 2
0.3149242 0.1904155 3.775095 286.8667 190.0428 1063.555 514.122 -0.2128293 0.6451094 0.6527474 -0.3353482 2
0.360273 -0.124824 3.873059 290.3788 220.2861 1073.087 601.4254 0.08764432 0.8281927 0.138871 0.5358455 2
0.379824 -0.4266101 4.031297 290.8479 247.2449 1073.576 679.1491 0.2291272 0.953576 0.07354193 0.1810665 2
0.3430154 -0.4866492 4.046878 287.3886 252.5338 1063.484 694.3685 0 0 0 0 2
0.2858644 0.6402671 3.695266 284.7101 144.9716 1057.815 384.334 -0.2561788 0.06318183 0.954231 -0.1407968 2
0.3690399 0.2620759 3.457524 295.4214 180.7474 1089.631 487.4088 0 0 0 0 1
0.3629896 0.2639313 3.440942 294.9656 180.4166 1088.398 486.4503 0 0 0 0 1
0.1983088 0.3111268 3.52541 276.9294 176.2054 1036.005 474.1149 0 0 0 0 2
0.2436822 0.3125684 3.488444 281.9108 175.7065 1050.552 472.7294 0 0 0 0 2
1
72057594037931115 0 1 1 1 1 0 0.1368076 0.1009628 2
25
0.2633063 0.1939517 3.826938 281.5139 189.9584 1047.906 513.8275 -0.1838576 0.05012416 0.9728169 -0.1315721 2
0.276437 0.4505553 3.753613 283.3141 164.5446 1053.505 440.593 -0.2158375 0.05443566 0.9662249 -0.1298476 2
0.2886485 0.7025217 3.670591 285.1931 138.2918 1059.318 365.1667 -0.2481636 0.06506237 0.9528328 -0.1621459 2
0.252512 0.8071519 3.633145 281.8617 126.9416 1049.903 332.5736 0 0 0 0 2
0.1549726 0.6271362 3.750509 271.4899 147.2204 1019.482 390.6259 0.2149646 0.7358347 -0.6388304 -0.06506296 2
```

Figure 3.1: An Example of Original Skeleton Files

3.2 Data Structures

There is in total 56880 action samples in this dataset — each skeletal action is stored in one single skeleton file which can be opened as a text file. The total dataset files size is 5.8GB. The name of each file is in the format of SsssCcccPpppRrrrAaaa (e.g., S001C003P001R001A100), in which sss is the setup number, ccc is the camera ID, ppp is the performer (subject) ID, rrr is the replication number (1 or 2) and aaa is the action class label according to the explanation from data provider.
Inside each skeleton file, is the information collected by the Microsoft Kinect V2 cameras. An example is shown in Fig 3.1. Unfortunately, there is not any explanation of details in files from the data provider. Therefore the Microsoft Kinect V2 manual and online resources were acquired and learned to understand the information in order to use it.

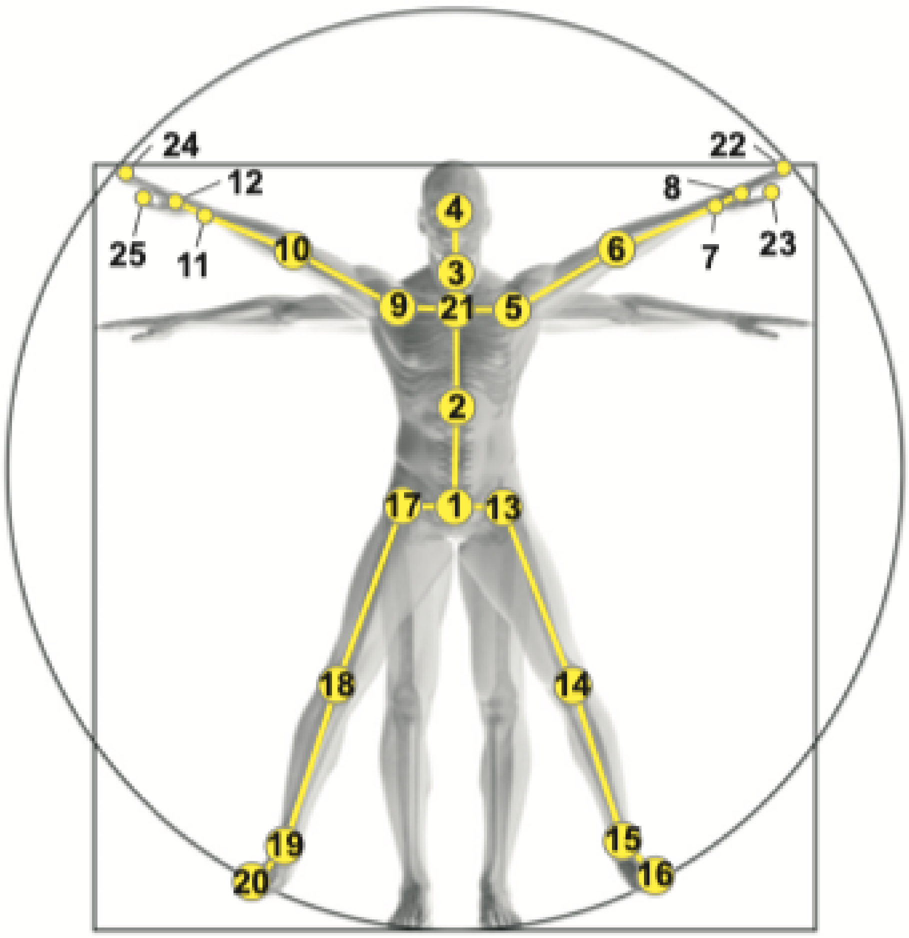


Figure 3.2: Configuration of 25 Body Joints in Dataset

The first figure of each file is the total number of frames in one action video, followed by a performer's ID and other information of this caption. Then there is a 25 row matrix showing the 3-dimensional locations of 25 major body joints for a single frame, for each row shows a list of parameters for general 3D coordinates, depth coordinates, orientation parameters and tracking state. In this task, only general 3D coordinates are required, for all 25 body joints shown in the Fig 3.2, and the labels of joints shown in the Table 3.1.

Table 3.1: Labels of 25 Body Joints in a Skeleton [9]

1: base of spine	2: middle of spine	3: neck	4: head	5: left shoulder
6: left elbow	7: left wrist	8: left hand	9: right shoulder	10: right elbow
11: right wrist	12: right hand	13: left hip	14: left knee	15: left ankle
16: left foot	17: right hip	18: right knee	19: right ankle	20: right foot
21: spine	22: tip of left hand	23: left thumb	24: tip of right hand	25: right thumb

3.3 Data Attributes

3.3.1 Action Classes

There are 60 action classes in total in the dataset. They are in three major categories, 40 classes of actions concluded in daily actions such as pick up or stand up; 9 classes of medical conditions like a sneeze or blow nose. There are also 11 Mutual actions or two-person interactions like punching each other. This project only focuses on basic one-person action recognition, since the medical conditions are subtle movements, features would be difficult to learn by the classifier. Therefore, the medical conditions, as well as mutual actions, are removed during the preprocessing step, leaving 40 action classes for the task. All the actions descriptions are shown in Table 3.2.

Table 3.2: Action Classes List [5]

A1: drink water	A2: eat meal	A3: brush teeth	A4: brush hair
A5: drop	A6: pick up	A7: throw	A8: sit down
A9: stand up	A10: clapping	A11: reading	A12: writing
A13: tear up paper	A14: put on jacket	A15: take off jacket	A16: put on a shoe
A17: take off a shoe	A18: put on glasses	A19: take off glasses	A20: put on a hat
A21: take off a hat	A22: cheer up	A23: hand waving	A24: kicking
A25: reach into pocket	A26: hopping	A27: jump up	A28: phone call
A29: play with phone	A30: type on a keyboard	A31: point on something	A32: taking a selfie
A33: check time on watch	A34: rub two hands	A35: bow	A36: shake head
A37: wipe face	A38: salute	A39: put palms together	A40: cross hands in front

From the Table 3.2 we can summarise some of the characteristics in the actions.

1. **Different actions with similar motion trajectories** Some different actions have similar motions. For example, action 'sit down' in the Fig 3.3 has the same motions with action 'stand up'. Only the time sequence is different.

2. **Subtle differences between actions** There are some different actions that looks similar, such as 'reading' and 'writing' or 'taking a selfie' and 'point on something', the only difference presents on the skeleton is the hand.

3. **Same action with different time continuity** Action 'jump up' means a person jump once whereas action 'hopping' may have involving jumping several times. Same with 'rub two hands' and 'put palms together', some different actions only have a difference when checking the time continuity as known as Temporal evolution.

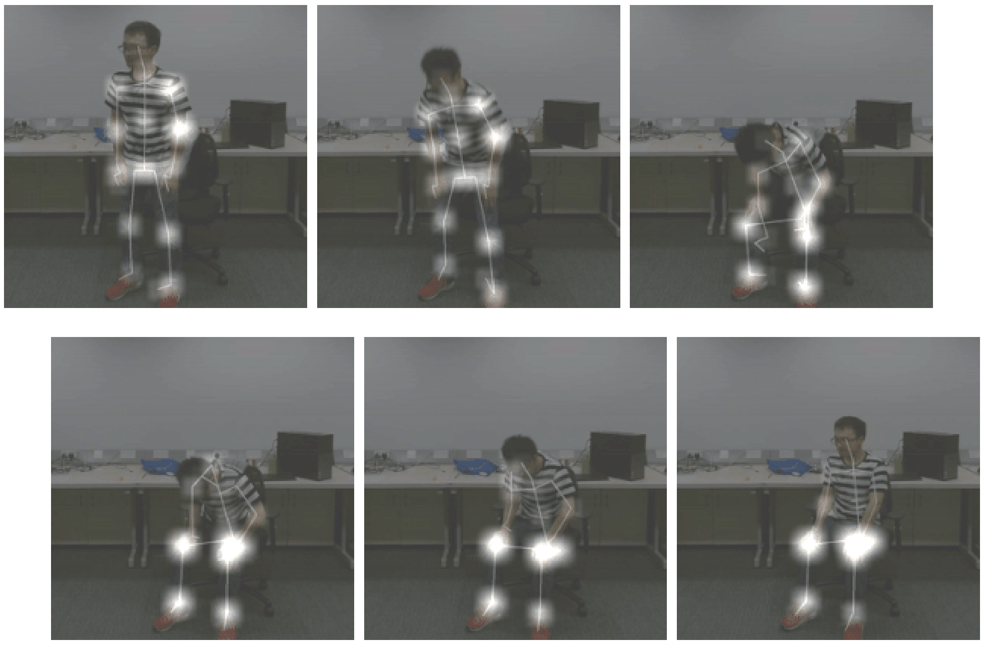


Figure 3.3: Example of an Action (Sit Down) from data provider

3.3.2 Subjects

There are 40 distinct subjects in the dataset with a balance of ages, gender and height. The data provider instructed the dataset should be split in order to use the cross-subject evaluation. According to them, 40 subjects are split into training and testing groups where each group consists of 20 subjects. There are 40320 samples in the training set, whereas 16560 samples in the testing set. The IDs of training subjects are: 1, 2, 4, 5, 8, 9, 13, 14, 15, 16, 17, 18, 19, 25, 27, 28, 31, 34, 35, 38, and the remaining subjects are for testing [9].

3.3.3 Views

According to the data provider, action videos are shot by three cameras simultaneously from different but horizontal views. The angles are fixed: -45°, 0° and +45°. Each subject performed each action twice, once towards the left and another towards the right. In that way, two front views, one left side view, one right side view, one left side 45° view and one right side 45° view are captured. Because the camera IDs are fixed,

which means Camera 1 shot all the 45° views while Camera 2 and 3 shot front and side views [9].
Besides views, there are 17 setups shown in Table 3.3. For each setup the height and distance of the cameras are changed as shown in the Table 3.3, but for every setup all of the cameras always remain horizontally.
According to the instruction of the data provider, for cross-view evaluation, samples shot from Camera 2 and 3 should be used for training and samples from Camera 1 are for testing because it did not shoot any front views but has 45° views for both sides of a subject.

Table 3.3: Parameters of Every Setup

Setup	Height	Distance	Setup	Height	Distance
1	1.7m	3.5m	2	1.7m	2.5m
3	1.4m	2.5m	4	1.2m	3.0m
5	1.2m	3.0m	6	0.8m	3.5m
7	0.5m	4.5m	8	1.4m	3.5m
9	0.8m	2.0m	10	1.8m	3.0m
11	1.9m	3.0m	12	2.0m	3.0m
13	2.1m	3.0m	14	2.2m	3.0m
15	2.3m	3.5m	16	2.7m	3.5m
17	2.5m	3.0m			

Chapter 4

Model Design

The model this project aims to design should achieve the objective to classify action videos into 40 classes. According to the previous literature review in Chapter 2, this model should be able to deal with the temporal evolution while its spatial pattern should also be taken care of, the model is designed based on this requirement.

4.1 Experiment Platform

4.1.1 Programming Language

All the coding work in this project has done in Python 3.7, which is one of the most popular programming languages in the world, and it is widely using in the data science and artificial intelligence area. Python is a cross-platform language that can compile and run on a variety of operating systems including MacOS, Linux and Windows, let alone it is also the most open source programming language that every programmer can make their own contribution to the community.
Python also has one of the largest package repositories that extend the use of a single programming language for nearly every part of computer science. All the technical requirements for this project involving deep learning, computer vision, data visualisation are satisfied in Python. So instead of using one tool to process the data, then using another tool to analyse the data, and finally visualising it with a tool that is different from the other two and generating a lot of troubles and waste of money because of the authorisation of the tools, Python can do all the work in one format of code and totally free.

4.1.2 Deep Learning Framework

There are a lot of deep learning frameworks these days. Most of the frameworks are open source and support Python. Fig 4.1 from an online posting [1] shows the power scores of some most popular frameworks based on criteria such as GitHub activities, Google search volume, ArXiv related articles and online job listings. This project uses Keras to implement the experimental model. The reason is as quote in its documentation: "Keras is an API designed for human beings, not machines [2]." So it is designed to be straightforward and consistent that can reduce lots of user workload, it is also easy to learn and use, under this designing idea that even new user can build a model very fast which suits a scientific research.

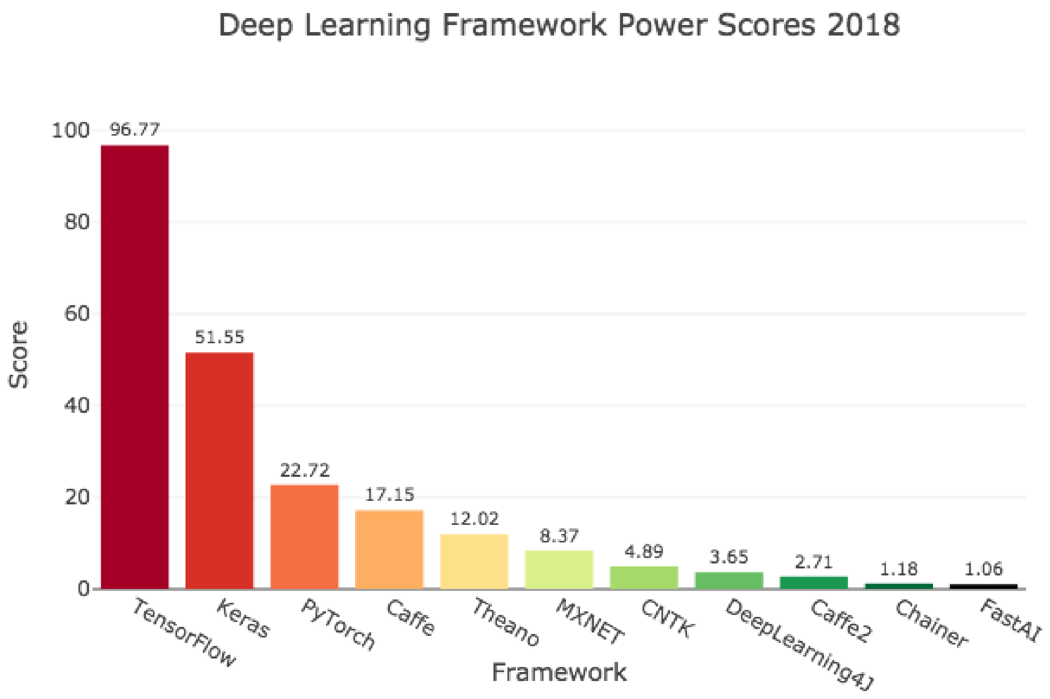


Figure 4.1: Deep Learning Frameworks Popularity

Moreover, it does not sacrifice flexibility because of its low learning threshold. It integrates low-level deep learning frameworks like Tensorflow, Theano and CNTK and also some of the functions from basic Python packages like Numpy. It enables any pre-built model in a base language like Python or Tensorflow. According to Fig 4.1, it is stronger than others except for Tensorflow, which Keras is the official frontend of it. Besides all that, a Keras model is easy to deploy into an actual product with just some system support.

4.2 Implementation

There are two ways to build a Neural network in Keras, Sequential API and Functional API. The sequential model is easier to build when a user only needs to consider which layer should be adding without considering how the data input and flows in the model, the limitation of the sequential model is lack of flexibility, layers cannot be reuse and inputs and outputs are restricted.
Since our task has more complicated inputs than usual, this project uses functional API to create models. In this way, a layer does not have to connect its adjacent layer, and multiple inputs or even splicing and concatenating data is feasible.

4.2.1 LSTM

As explained in the literature review, Recurrent Neural Networks with Long Short-Term Memory cells were born to learn the temporal relations in the data as its structure and function explained in Chapter 2. So, the first and baseline approach is to use Keras API to build an LSTM model. Fig 4.2 shows the order to build a model using the Keras functional API.

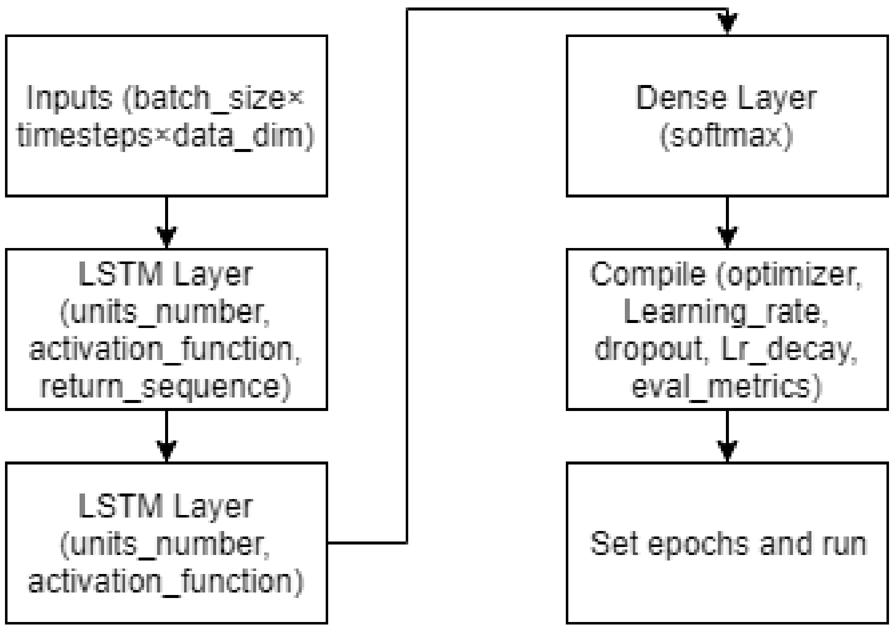


Figure 4.2: Steps of Defining a model Layer by Layer using Functional API in Keras

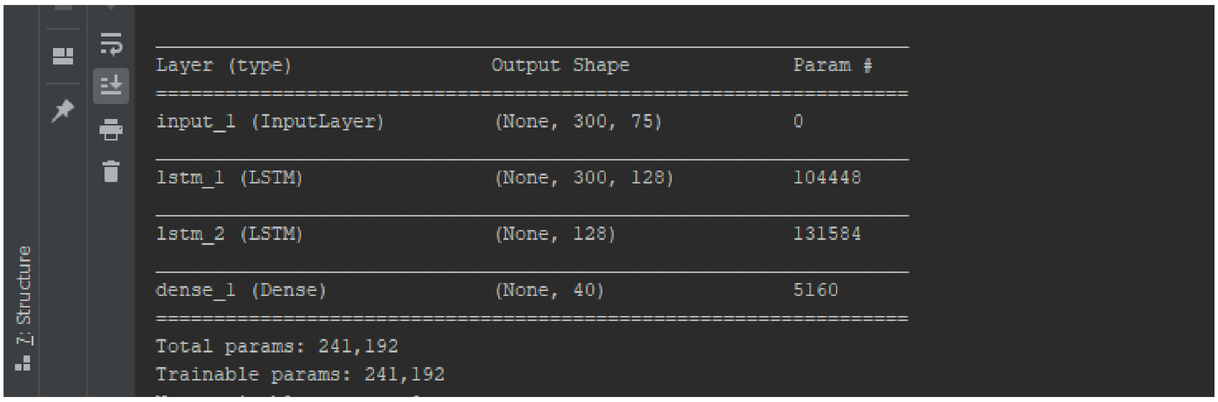
1. One of the unique points and quite critical to our task for the functional API is that defining an input layer is compulsory. The functional model must know the input data shape, in our case it is a 3-dimensional tensor with fixed batch size. In the API we can simply point the input as the variable we set to iterate the data by batch which will be discussed later in Chapter 5.

2. After the definition of input layer, there comes the main part of the model, the functional layers. For each layer the number of hidden units in the layer must be defined, the activation function is optional, but for the first LSTM layer the batch input shape must be defined, and the return sequence must set to True to pass the hidden states (not cell states) to the next layer in order to make learning process consistent.

3. After the LSTM layers, the output must fit the label to give a result. Labels are supposed to be processed in one-hot encoding in the data processing step which is discussed in Chapter 5, this would make the label to a list full of 0, only the index of the number of label is marked as 1. Defining a dense layer with softmax would do the trick to make a prediction to the most probable class by setting only the max probability to 1 and set others to 0 to fit the one-hot encoding of original label.

4. When all layers are defined, the user must define the model itself and compile it. In the compile API we can defined other hyper parameters like learning rate and its decay over epochs, dropout rates and evaluation metrics which are categorical metrics in this task since we have more than 10 classes of actions.

5. Final step is to set an epoch number and run. The parameters counting in this model is shown in Fig 4.3.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 300, 75)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 300, 128)          104448
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense_1 (Dense)              (None, 40)                5160
=================================================================
Total params: 241,192
Trainable params: 241,192
```

Figure 4.3: Parameters Counting in Basic 2-Layer LSTM

4.2.2 Part-Aware LSTM

The data provider proposed Part-Aware LSTM as an approach to improve the performance of a basic LSTM. However, the explanation in their paper is pretty vague about how to implement this network, only a proposal on structure level in Fig 2.9. There are no other credible online resources that is discussing this implementation. However, given that the body joints are fixed in the data, meaning the tensor for each timestep can be divided by certain rules, the implementation of this network is feasible.

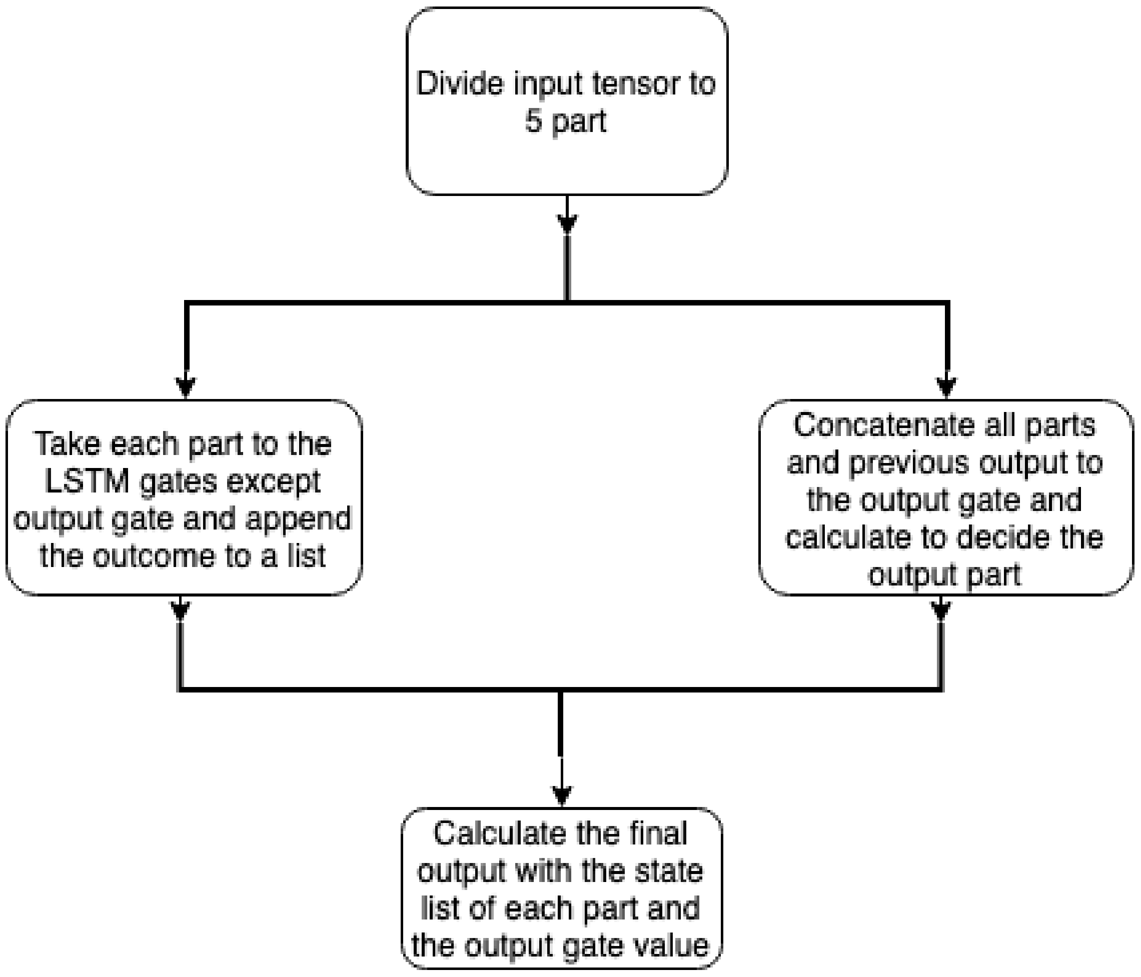
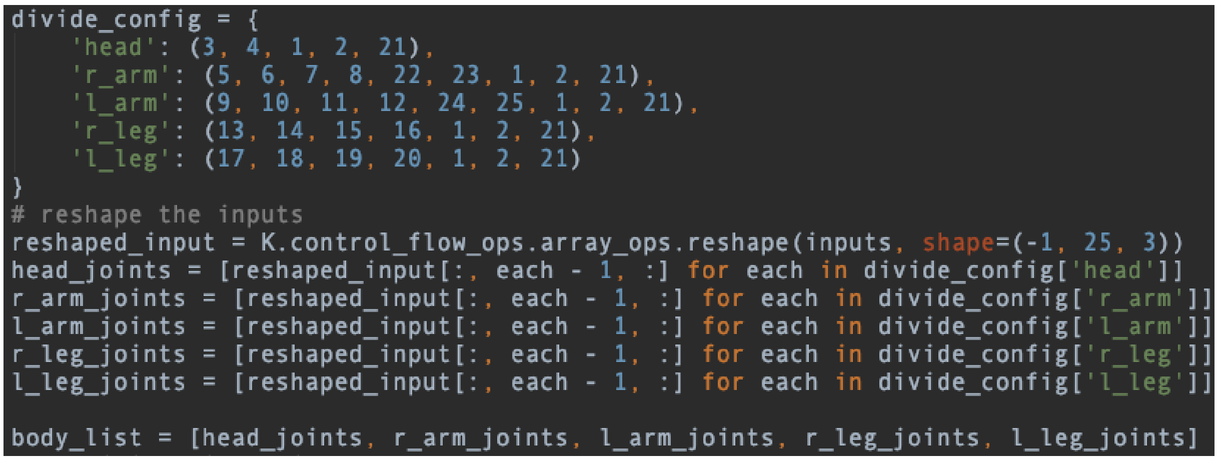

Divide input tensor to 5 part
Take each part to the LSTM gates except output gate and append the outcome to a list
Concatenate all parts and previous output to the output gate and calculate to decide the output part
Calculate the final output with the state list of each part and the output gate value

Figure 4.4: Part-Aware LSTM Cell Customisation Theory

With the customisation support of Keras, this proposed model can be implemented by modifying the source code of Keras LSTM cell and create a customised layer to call the cell. According to the paper of Part-Aware LSTM, the cell can be modified to divide the input tensor into 5 body part, the introduction about how to load the data is explained in Chapter 5. At this moment, there should be only one tensor from a frame in actions is loaded into the cell. The processing theory is shown in 4.4.
After dividing the input tensor into five-part groups in Fig 4.5, each group calculates their own forget value, input value and candidate known as the forget gate, the input gate and the modulation gate to finally get a new unit state and append all five to a cell



```
divide_config = {
    'head': (3, 4, 1, 2, 21),
    'r_arm': (5, 6, 7, 8, 22, 23, 1, 2, 21),
    'l_arm': (9, 10, 11, 12, 24, 25, 1, 2, 21),
    'r_leg': (13, 14, 15, 16, 1, 2, 21),
    'l_leg': (17, 18, 19, 20, 1, 2, 21)
}
# reshape the inputs
reshaped_input = K.control_flow_ops.array_ops.reshape(inputs, shape=(-1, 25, 3))
head_joints = [reshaped_input[:, each - 1, :] for each in divide_config['head']]
r_arm_joints = [reshaped_input[:, each - 1, :] for each in divide_config['r_arm']]
l_arm_joints = [reshaped_input[:, each - 1, :] for each in divide_config['l_arm']]
r_leg_joints = [reshaped_input[:, each - 1, :] for each in divide_config['r_leg']]
l_leg_joints = [reshaped_input[:, each - 1, :] for each in divide_config['l_leg']]

body_list = [head_joints, r_arm_joints, l_arm_joints, r_leg_joints, l_leg_joints]
```

Figure 4.5: Actual Input Dividing Configuration

state list. Since there is only one output gate, we concatenate all input part list we divided before back to one input. and calculate with the previous output to decide what parts of the cell state that is going to output. The final step is to calculate the final output and concatenate the Cell state. The detailed algorithm pseudocode shows below in Algorithm 1 [9].

Algorithm 1 Part-Aware LSTM Cell

```
Input: Current Input Tensor $x_t$, previous output $h_{t-1}$, previous carry state $C_{t-1}$;
Output: Current carry state $C_t$ & Output Tensor $h_t$;
  Divide the input tensor $x_t$ into 5 part base on the joints representation and put in a list $x_t\_list$;
  for $i = 0 \rightarrow 4$ do
    Forget gate: $f_t[i] = sigmoid(W_f \cdot [h_{t-1}, x_t\_list[i]] + bias_f)$;
    Input gate: $i_t[i] = sigmoid(W_i \cdot [h_{t-1}, x_t\_list[i]] + bias_i)$;
    Candidate vector: $\widetilde{C}_t[i] = tanh(W_g \cdot [h_{t-1}, x_t\_list[i]] + bias_g)$;
    Update single unit: $C_t[i] = f_t[i] \times C_{t-1} + i_t[i] \times \widetilde{C}_t[i]$;
    Append $C_t[i]$ to $C_t\_list$;
  end for
  Output gate: $o_t = sigmoid(W_o \cdot [h_{t-1}, x_t\_list] + bias_o)$;
  New output: $h_t = o_t \times tanh(C_t\_list)$;
  New state: Concatenate $C_t\_list$ and $h_t$ to $C_t$;
  return $h_t$, $C_t$
```

In this way, the Part-Aware LSTM cell can divide the input and calculate itself without any support from the layer, meaning there is no need to change the input shape of the layer as well as the output. Thus, any other configuration functions can remain the same, reducing much work for customising the layer. The customisation to the layer can be done by simply following the instruction of Keras documentation [2]. Due to the time shortages of this project, the custom cell has some limitations that will be discussed in Chapter 7.

Chapter 5

Data Processing

5.1 Steps of Processing

As introduced in Chapter 3, skeletal action videos are stored in skeleton files separately. In order to use the data in the model experiments, several steps of processing the data must be taken, as there is no explanation from the data provider, the data processing task is challenging. The brief steps are shown in Fig 4.1 and the details explained below.

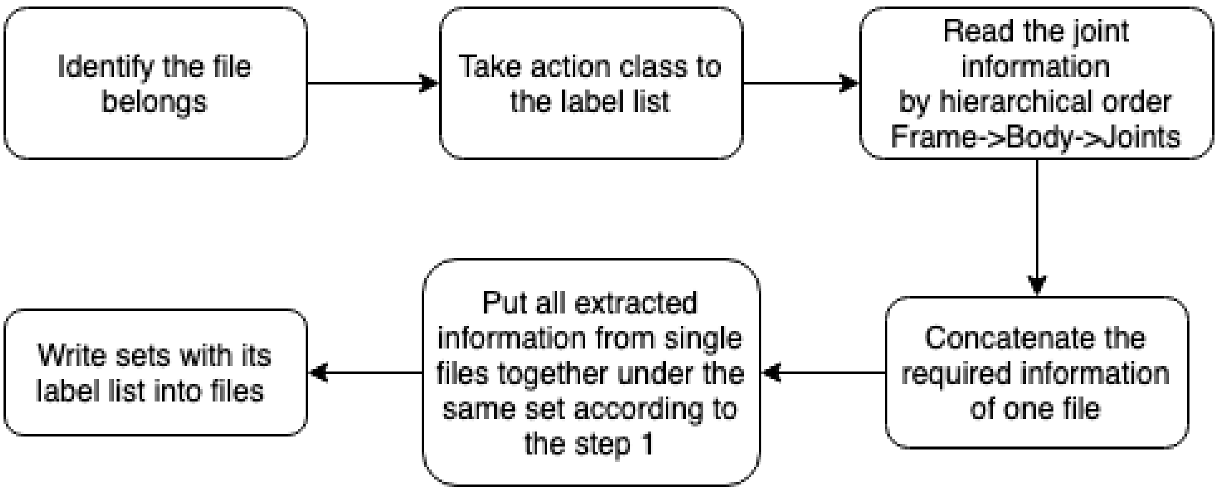

Identify the file belongs
Take action class to the label list
Read the joint information by hierarchical order Frame->Body->Joints
Concatenate the required information of one file
Put all extracted information from single files together under the same set according to the step 1
Write sets with its label list into files

Figure 5.1: Processing Steps

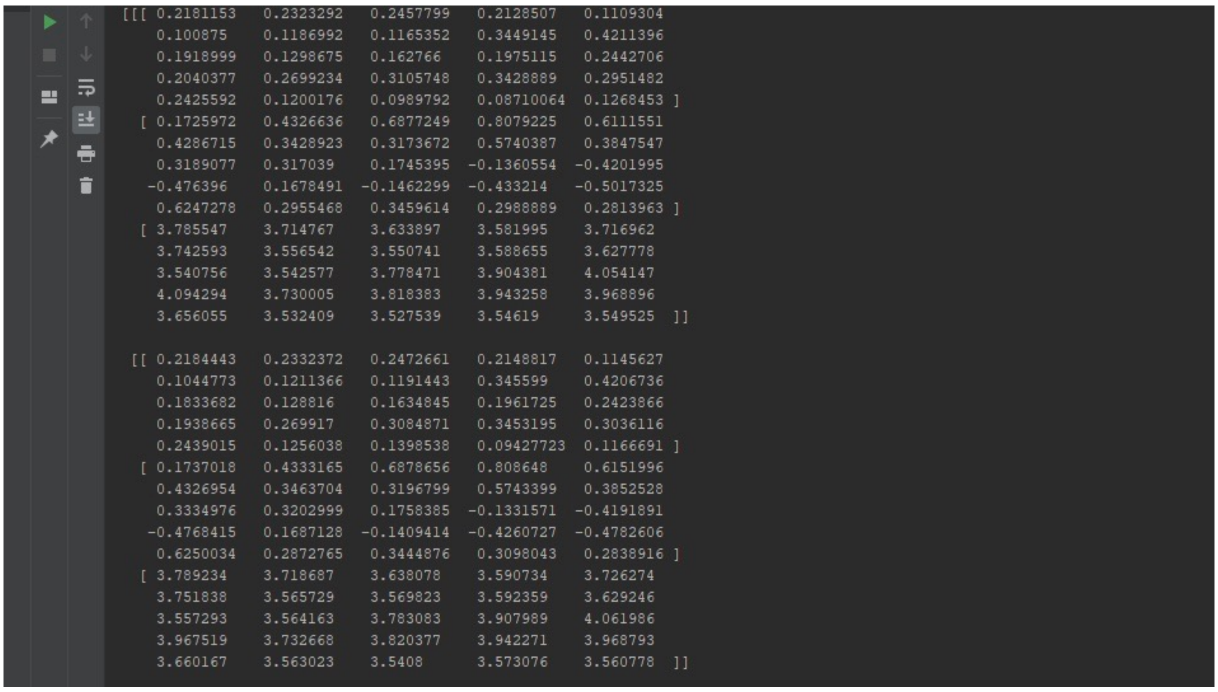
1. Select one file and identify the file belongs since each file name contains subject ID, camera ID, we can identify whether this file should be in the training set or the validation set based on the introduction of evaluation method in Chapter 3.

2. Take the number after the 'A' in the filename, minus integer 1 since the index should start with 0, append the number to the label list.

3. Open the file as reading mode and read the file line by line. Since the file started the first line with a number shows how many frames are in this video, followed a body ID and the body information line above each frame. In this project, we do not consider two-person actions so that we can skip the body information. However, to make the extraction code extensible (expandable to two-person) and avoid any influence of data recording errors, we still read the body information first, then read all the frames under the same body ID.

4. The original joint information contains multiple parameters. We only need to use the general 3-dimensional coordinates, which is every first three numbers of each line. Write the information into an n-dimensional array by a number of frames, the number of channels for coordinates which is 3 and the number of joints which is 25.

5. Build an multi-dimensional array with zeros by the number of files, scale the number of frames to 300 in order to make all the inputs for the model unified, the rest of two dimensions are the number of channels and the number of joints which are the same to the last step. write every previously built array for single file under the same set (e.g. Cross-Subject Training set) into this array.



```
[[[ 0.2181153   0.2323292   0.2457799   0.2128507   0.1109304
    0.100875    0.1186992   0.1165352   0.3449145   0.4211396
    0.1918999   0.1298675   0.162766    0.1975115   0.2442706
    0.2040377   0.2699234   0.3105748   0.3428889   0.2951482
    0.2425592   0.1200176   0.0989792   0.08710064  0.1268453 ]
  [ 0.1725972   0.4326636   0.6877249   0.8079225   0.6111551
    0.4286715   0.3428923   0.3173672   0.5740387   0.3847547
    0.3189077   0.317039    0.1745395  -0.1360554  -0.4201995
   -0.476396    0.1678491  -0.1462299  -0.433214   -0.5017325
    0.6247278   0.2955468   0.3459614   0.2988889   0.2813963 ]
  [ 3.785547    3.714767    3.633897    3.581995    3.716962
    3.742593    3.556542    3.550741    3.588655    3.627778
    3.540756    3.542577    3.778471    3.904381    4.054147
    4.094294    3.730005    3.818383    3.943258    3.968896
    3.656055    3.532409    3.527539    3.54619     3.549525  ]]

 [[ 0.2184443   0.2332372   0.2472661   0.2148817   0.1145627
    0.1044773   0.1211366   0.1191443   0.345599    0.4206736
    0.1833682   0.128816    0.1634845   0.1961725   0.2423866
    0.1938665   0.269917    0.3084871   0.3453195   0.3036116
    0.2439015   0.1256038   0.1398538   0.09427723  0.1166691 ]
  [ 0.1737018   0.4333165   0.6878656   0.808648    0.6151996
    0.4326954   0.3463704   0.3196799   0.5743399   0.3852528
    0.3334976   0.3202999   0.1758385  -0.1331571  -0.4191891
   -0.4768415   0.1687128  -0.1409414  -0.4260727  -0.4782606
    0.6250034   0.2872765   0.3444876   0.3098043   0.2838916 ]
  [ 3.789234    3.718687    3.638078    3.590734    3.726274
    3.751838    3.565729    3.569823    3.592359    3.629246
    3.557293    3.564163    3.783083    3.907989    4.061986
    3.967519    3.732668    3.820377    3.942271    3.968793
    3.660167    3.563023    3.5408      3.573076    3.560778  ]]
```

Figure 5.2: Processed Data (Dimensions: Frames-Channels-Joints)

6. After the processing, the data should be formatted as Fig 5.2. For the entire data set, it should be stored in a 4-dimensional array. The four dimensions from the outside to the inside are the number of actions in this set, the number of frames (which is scaled to a fixed number: 300), the number of channels which is 3, the number of joints which is 25. To avoid this processing step each time when we run the code, these divided and processed datasets should be stored in files for future use.

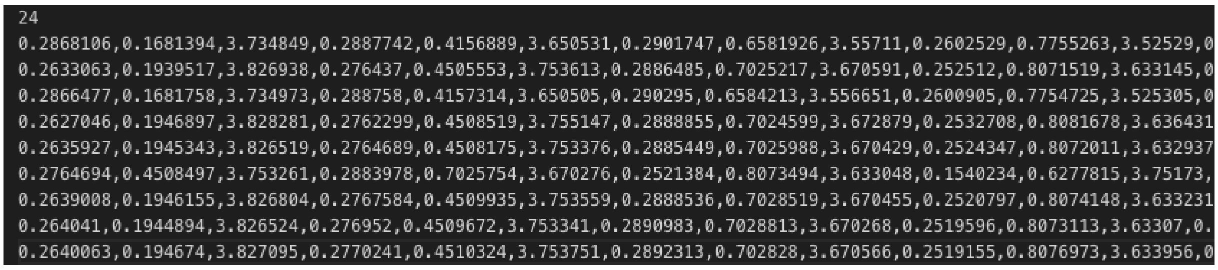
5.2 Dataset Storage

5.2.1 Data File Type

Since the nature of this data is in a numerical format, multiple file types can be chosen for storage of the dataset. Commonly there are several data file types which are suitable for the model to load. In this project, because the volume of the data is big, the speed and stability of data loading needs to be considered. There are 3 data file types that were compared:

CSV Files

CSV, comma-separated values files, are delimited text file that generally uses a comma to separate values. A CSV file stores tabular data in plain text and each line of the file is a data record. In each record, there are one or more fields separated by commas [11]. Our data fits the format that can be stored in CSV files, for each action, the label can be its start, after the label, store the frames line by line with stretched joints coordinates like shown in Fig 5.3.



```
24
0.2868106,0.1681394,3.734849,0.2887742,0.4156889,3.650531,0.2901747,0.6581926,3.55711,0.2602529,0.7755263,3.52529,0
0.2633063,0.1939517,3.826938,0.276437,0.4505553,3.753613,0.2886485,0.7025217,3.670591,0.252512,0.8071519,3.633145,0
0.2866477,0.1681758,3.734973,0.288758,0.4157314,3.650505,0.290295,0.6584213,3.556651,0.2600905,0.7754725,3.525305,0
0.2627046,0.1946897,3.828281,0.2762299,0.4508519,3.755147,0.2888855,0.7024599,3.672879,0.2532708,0.8081678,3.636431
0.2635927,0.1945343,3.826519,0.2764689,0.4508175,3.753376,0.2885449,0.7025988,3.670429,0.2524347,0.8072011,3.632937
0.2764694,0.4508497,3.753261,0.2883978,0.7025754,3.670276,0.2521384,0.8073494,3.633048,0.1540234,0.6277815,3.75173,
0.2639008,0.1946155,3.826804,0.2767584,0.4509935,3.753559,0.2888536,0.7028519,3.670455,0.2520797,0.8074148,3.633231
0.264041,0.1944894,3.826524,0.276952,0.4509672,3.753341,0.2890983,0.7028813,3.670268,0.2519596,0.8073113,3.63307,0.
0.2640063,0.194674,3.827095,0.2770241,0.4510324,3.753751,0.2892313,0.702828,3.670566,0.2519155,0.8076973,3.633956,0
```

Figure 5.3: CSV storage screenshot

The advantages of CSV files are as follows:

- Human readable and easy to edit manually.
- Simple to implement and parse in Python.
- Provides a straightforward information schema.
- Easy to generate.

However, in this task, human readability and manual editing are not necessary since it represents coordinates that are not supposed to be editing or reading in hand. Moreover, the way of loading CSV files is like loading any plain text files, and the machine is going to load the files into the memory then extract it to use which performs poorly in a large dataset. Let alone its stability is influenced by file loading. If the memory is insufficient for reading the whole file, the machine can crash down. Therefore in this project, we abandoned this file type to store our data.

Numpy File

Numpy is a scientific computation library for Python which supports large, multi-dimensional arrays and high-level mathematical functions for operating these arrays [14]. With this library, there is file storage provided, which is the NPY files. NPY files store all the information required to reconstruct an array even it is multi-dimensional. The advantages of NPY files are as follows:

- Like CSV files, NPY files are simple to implement and parse in Python.
- Since NPY files can store multi-dimensional arrays in a easy way, a user does not need to convert the high-dimensional arrays, NPY files can store and extract automatically. To be specific in our task, there is no need to concatenate coordinates for each frame from a 3×25 matrix into a single one-dimensional array with 75 figures.
- By using the function numpy.memmap, a large file can be assessed in small segments rather than reading the entire file into memory at once. In this way, the stability of file loading can be secured without sacrificing the speed because inside the file the data is in its native binary form, which would be not readable directly, but in this task, it does not matter.

There is a limitation to NPY files that only one array can be stored when using the numpy.save function, so the actions and labels cannot be stored in single file, labels need to save in a separate file which would increase the processing steps when loading data into models. Despite that, in most of the situations, NPY files are suitable for this task.

TFRecord File

Although NPY files are enough for this task, it is not the optimal choice. Given that we have already chosen Keras as the deep learning framework for this project, to read data more efficiently and also help serialise the data. TFRecord format of files is used in this project. This format is provided by Tensorflow which Keras is the official frontend to it, meaning TFRecord files can use on any Keras models just as they run on Tensorflow. Due to its binary file format nature, a TFRecord file contains a sequence of records that can only be read sequentially [10]. Therefore, with a proper pipeline, TFRecord file type is the fastest and the most suitable format for Tensorflow and Keras models. Not only that, the TFRecord has been optimised that made multiple datasets can be integrated and import using a single pipeline.
However, there is a challenge using TFRecord files. Except for official document from Tensorflow [10], there are not many of tutorials online about the practical coding steps for writing the data into TFRecord files and more importantly how to build a pipeline and feed the data from files out to the model. However, the difficulty is not an excuse for

not getting it done, because TFRecord files are the best plan to store the datasets, after reading the official document and trying countless time, it is finally done. Details about how to write datasets into files and pipeline to load the data explain in next sections.

5.2.2 Writing Datasets into Files

Tensorflow gives an API TFRecordWriter for creating a file. The input sample should store as a dictionary: a key as the action label and values as the action itself. Unlike previous explains in the Section 5.1, because of the serialisation mechanism of TFRecord, the dictionary works as a pipeline, writes actions into files one by one, therefore as shown in Fig 5.4, instead of a whole dataset in a 4-dimensional array, each time the dictionary contains one action with a 3-dimensional arrays of frames, channels and joints. After writing in the file, the dictionary updates for the next action.

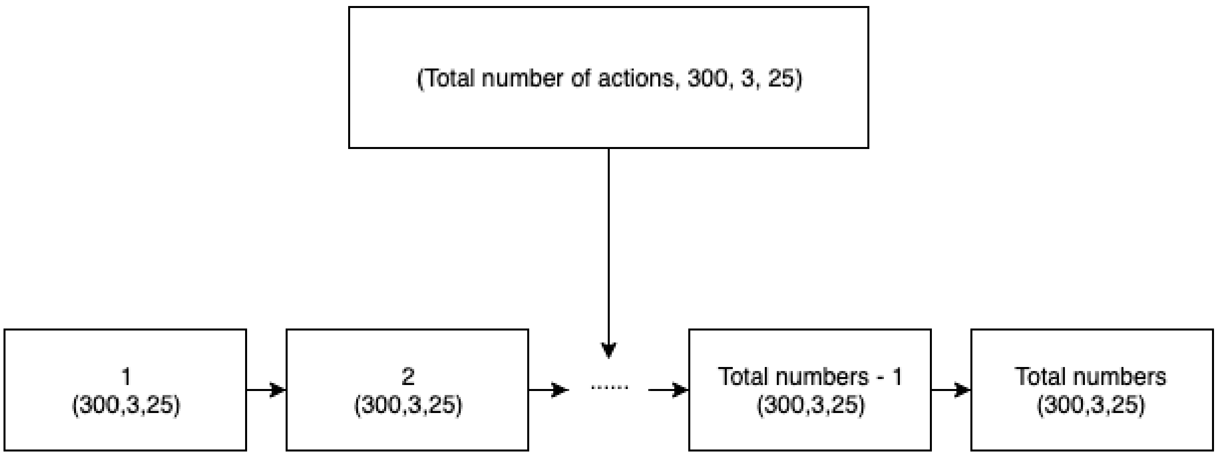


Figure 5.4: Instead the whole dataset in one 4-dimensional array, separate into a list of 3-dimensional arrays for the serialisation

According to the official document, the accepted input shape is only lists with the format of integer, float or string. So we need to transform our data from a 3-dimensional tensor to a list as a sample and serialise. However, doing that would make the data lose its shape. The official document does not mention anything about this. After a long time of online searching and countless attempts and trying, the solution that this report presents is quite inspiring but straightforward, which is to write the shape of original tensor to a separate list and store along with the transformed tensor to the file. The label list can also be stored in the same file with the data, so there is no need to read two files at the same time when loading one dataset into models. After writing all the information of one action into the dictionary, put the dictionary into several processes to make it a Tensorflow example and serialise it to a string, then write the serialised sample into the file and repeat, the steps are shown in Fig 5.5. When reaching the end of the list of action arrays, close the file writer to save the file.

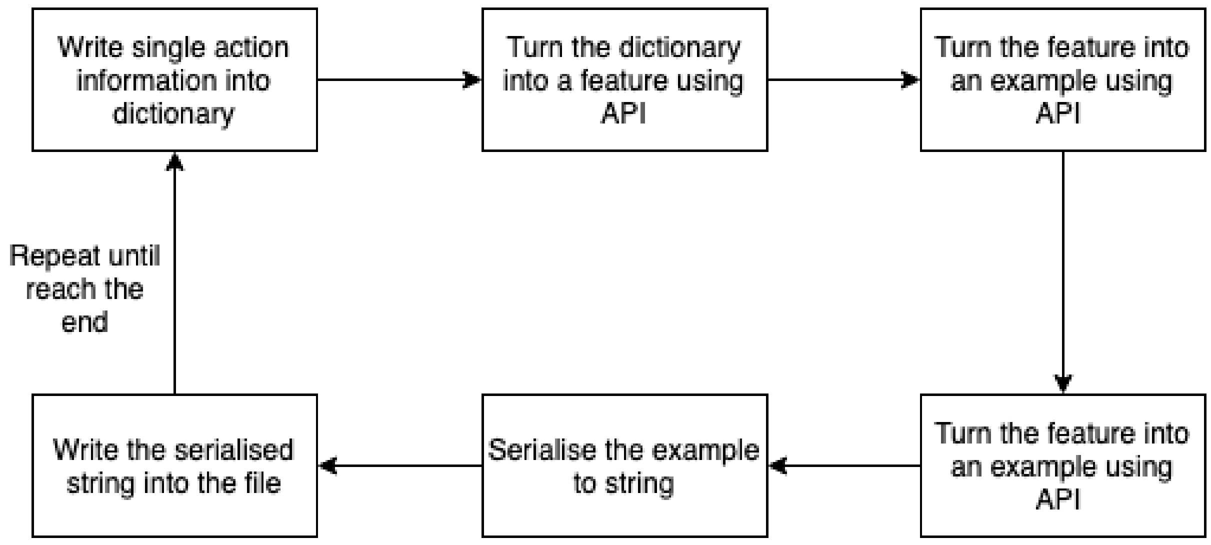

Write single action information into dictionary
Turn the dictionary into a feature using API
Turn the feature into an example using API
Turn the feature into an example using API
Serialise the example to string
Write the serialised string into the file
Repeat until reach the end

Figure 5.5: Process of writing actions one by one into TFRecord files

5.2.3 Dataset Loading

Like the previous steps, there are also APIs for reading the TFRecord dataset. However, the APIs are only for an operation, not for the entire process. Therefore, we need to build an iterator as in Fig 5.6 for data to flow into the model we have built in Chapter 4.

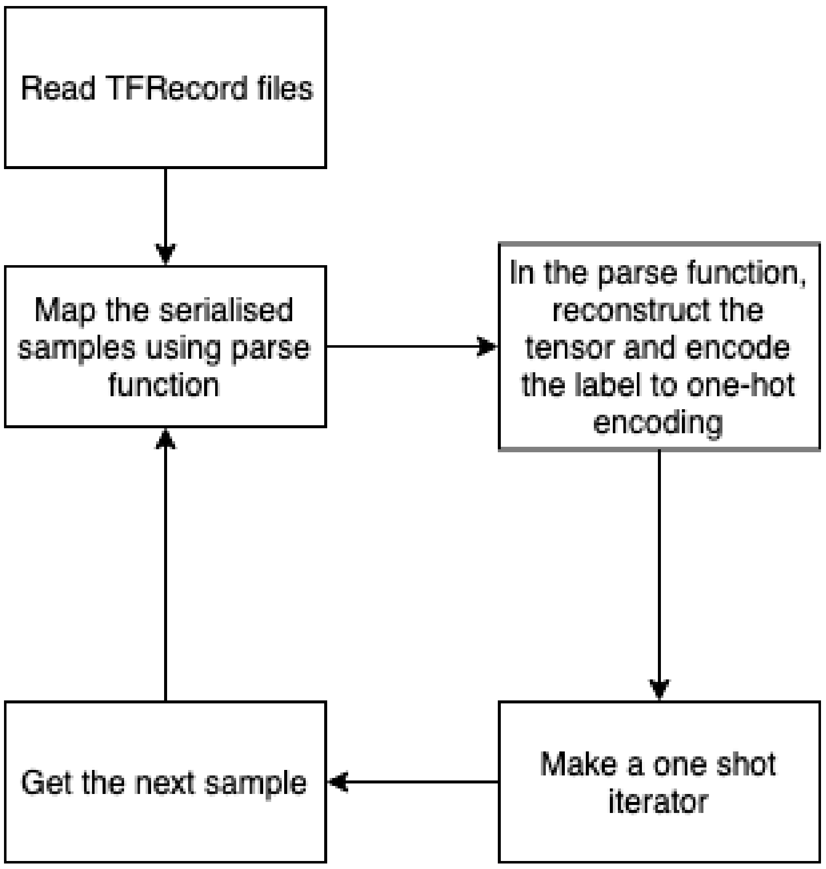
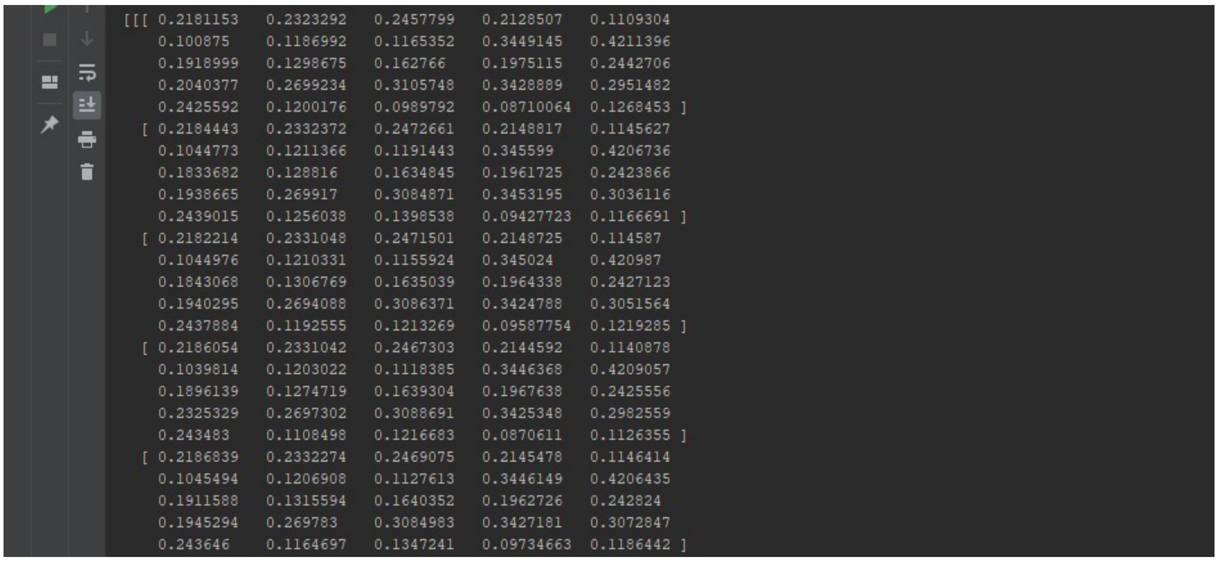

Read TFRecord files
Map the serialised samples using parse function
In the parse function, reconstruct the tensor and encode the label to one-hot encoding
Make a one shot iterator
Get the next sample

Figure 5.6: Process of loading TFRecord files and feed models through iterator

This step is designed to be the feeder to models, so after this process, the data is prepared directly to load into Keras models.
Under this concept, two functions are defined, one is the parse function which is to reconstruct the 3-dimensional tensor based on the information extracted from serialised files and to encode the label in one-hot encoding for its use in classification output as discussed in Chapter 4. The action data can also reconstruct to any shape that is required for the model, such as putting all the joints coordinates in one channel of the videos shown in Fig 5.7
Another function is loading function. This function would read from files and call the parse function, then split the dataset into batches and flow through the iterator. There are several kinds of iterators can be built under the instruction of official document, but due to our data is well-formatted and only one file for one dataset, there is no need to make it complicated. A one-shot iterator is defined to load one dataset at a time.



```
[[[ 0.2181153   0.2323292   0.2457799   0.2128507   0.1109304
    0.100875    0.1186992   0.1165352   0.3449145   0.4211396
    0.1918999   0.1298675   0.162766    0.1975115   0.2442706
    0.2040377   0.2699234   0.3105748   0.3428889   0.2951482
    0.2425592   0.1200176   0.0989792   0.08710064  0.1268453 ]
  [ 0.2184443   0.2332372   0.2472661   0.2148817   0.1145627
    0.1044773   0.1211366   0.1191443   0.345599    0.4206736
    0.1833682   0.128816    0.1634845   0.1961725   0.2423866
    0.1938665   0.269917    0.3084871   0.3453195   0.3036116
    0.2439015   0.1256038   0.1398538   0.09427723  0.1166691 ]
  [ 0.2182214   0.2331048   0.2471501   0.2148725   0.114587
    0.1044976   0.1210331   0.1155924   0.345024    0.420987
    0.1843068   0.1306769   0.1635039   0.1964338   0.2427123
    0.1940295   0.2694088   0.3086371   0.3424788   0.3051564
    0.2437884   0.1192555   0.1213269   0.09587754  0.1219285 ]
  [ 0.2186054   0.2331042   0.2467303   0.2144592   0.1140878
    0.1039814   0.1203022   0.1118385   0.3446368   0.4209057
    0.1896139   0.1274719   0.1639304   0.1967638   0.2425556
    0.2325329   0.2697302   0.3088691   0.3425348   0.2982559
    0.243483    0.1108498   0.1216683   0.0870611   0.1126355 ]
  [ 0.2186839   0.2332274   0.2469075   0.2145478   0.1146414
    0.1045494   0.1206908   0.1127613   0.3446149   0.4206435
    0.1911588   0.1315594   0.1640352   0.1962726   0.242824
    0.1945294   0.269783    0.3084983   0.3427181   0.3072847
    0.243646    0.1164697   0.1347241   0.09734663  0.1186442 ]
```

Figure 5.7: Parse the tensor to whatever needed (this figure groups all the x coordinates)

5.3 Visualisation

The skeletal data can be visualised. In normal circumstances, the process of visualisation for one picture is to reshape the skeleton coordinates into a map with $255 \times 255 \times 255$ scale canvas and plot it using cv2, a Python library developed by OpenCV (Open Source Computer Vision). However there are actions containing hundreds of frames, one frame can be plotted as an image, that means for each action there are 300 images. Luckily, The visualisation in videos can be achieved by using the library scikit-video in Python. After turning skeletons into images, this package can be used to parse images into a video and output. An example shows in 5.8.

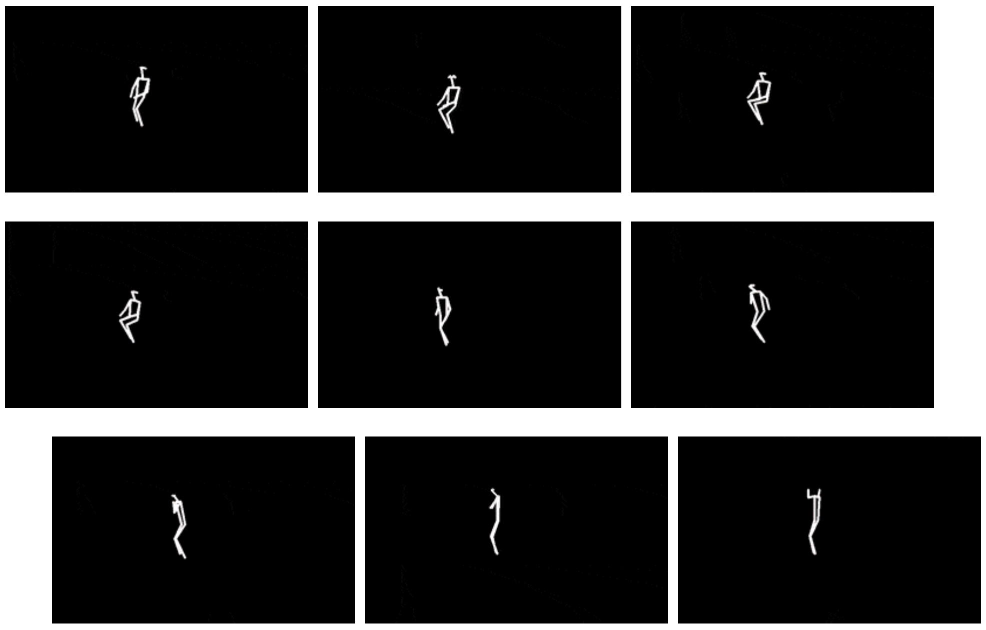


Figure 5.8: Example of an Action Visualisation (Jump up)

Chapter 6

Experiments and Evaluations

After the data processing in Chapter 5, the data should load into the model built in Chapter 4. We would perform several experiments and evaluate on them. Details would be introduced in this Chapter.

6.1 Experimental Setup

6.1.1 Hyperparameter Tuning

When building the model in Chapter 4, we set up some hyperparameters for the models. Unlike regular parameters in neural networks that would be calculated and updated automatically by its algorithms like cell weights and bias, a hyperparameter is a parameter whose value is set before the learning process begins. For example, in our case, the number of classes for actions is a hyperparameter that is pre-defined as 40. Others like batch size, number of epochs, learning rate or even the optimiser algorithm are all hyperparameters.
Although the model does not automatically adjust hyperparameters like adjusting weights and bias, optimizing hyperparameters is still an essential step in achieving a good model. The tuning strategies for some critical hyperparameters along with the evaluations are introduced below.

Number of Epochs

One epoch is when an entire dataset is imported into the network once, so the number of epochs in practical means how many times the network needs to learn. Just like a student needs to learn the same content multiple times before really master it, the artificial neural network needs to study the same dataset for more than one time to learn the features and patterns within it. So, theoretically, with more epochs, the performance of the model is better. However, the running time would also increase. In this project, we set several numbers of epochs to experiment on the basic 2-layer LSTM with cross-view dataset and the batch size to 512, Fig 6.1 shows the accuracy performance with the increasing of the number of epochs settings in Table 6.1, showing that after some point, even the number of epochs increasing further, the performance would not continue to go up significantly, but with the running time grows that is not to be ignored, keep increasing the number of epochs when the accuracy growing is tend to slow down may not be wise. Obviously, there are trade-off relations between the performance and the running time cost that needs to be considered. Generally, with a more diverse dataset like our dataset, the number of epochs would need to set larger to make it perform better.

Table 6.1: Epochs Experimental Setup

Number of Epochs	Test Accuracy
10	4.87%
50	8.29%
100	12.33%
200	19.45%
500	27.38%
1000	39.24%
2000	53.33%
5000	56.21%
10000	58.21%

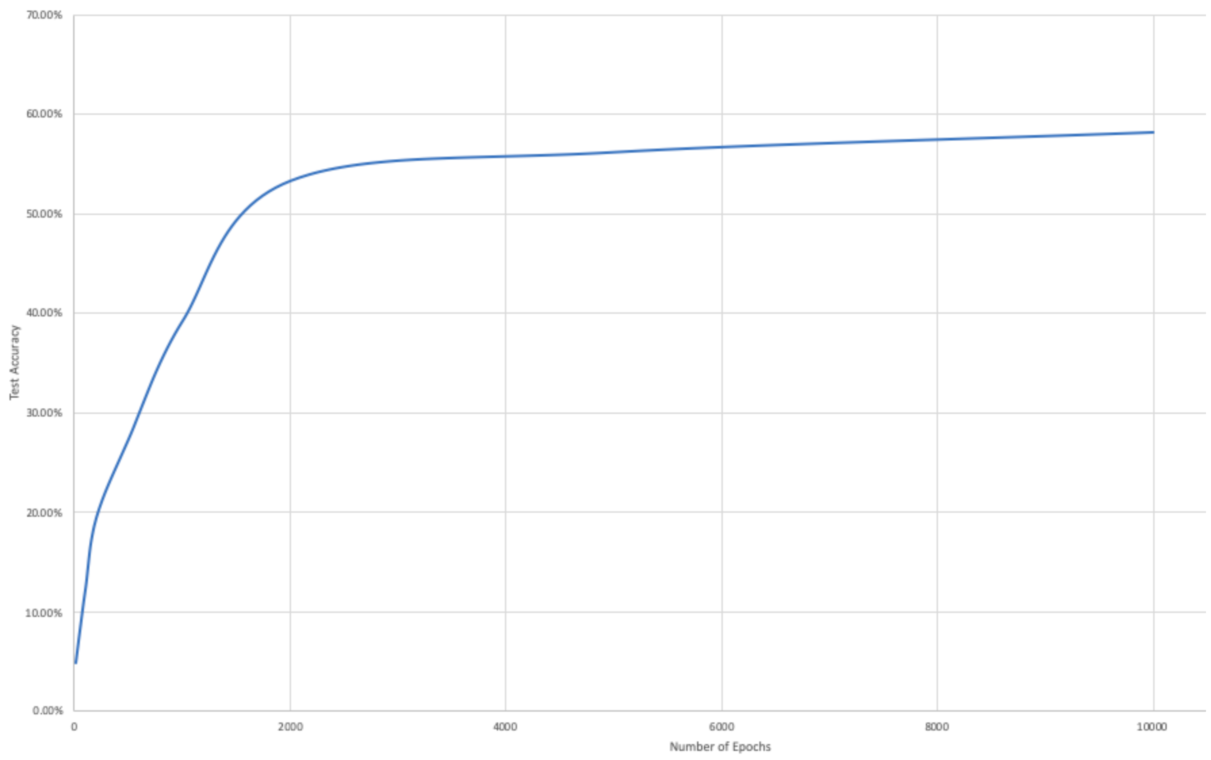


Figure 6.1: The performance on basic LSTM with the different number of epochs

Batch Size

As explained in Chapter 5, the entire dataset is too big to pass it into the model at once, so we divide the dataset into several batches. Practically in the code, this is present by the setting of the batch size which means how many examples in a batch we want to load in a single batch, the process we load a single batch is called iteration. In the Dataset Loading section, we built an iterator to perform this iterate action. In short, the number of iterations the number of batches needed to complete one epoch set by the batch size. However, an iteration is not only just a loading process. When the model has finished the calculation of one batch, it will calculate the gradient descent with error functions to perform a backpropagation and update its weights and bias. So the number of iterations also means how many times the network updates itself to improve in a training epoch. Like the epoch experiment, we set different batch size in Table 6.2with the number of epochs to 2000 in 2-layer LSTM. The results show in Fig 6.2. Interestingly, the accuracy is not always going up with increasing batch size, proven that when the batch size is over some point, the performance would go down. The reason behind that would because the iteration times, when the batch size increase the iteration times decrease, lead the weights and bias did not frequently update as it should be, which influences the learning outcome, just like a student who learnt too much in one day may result in remembering less.

Table 6.2: Batch Size Experiment Setup

Batch Size	Test Accuracy
16	3.53%
32	5.22%
64	15.73%
128	29.13%
256	44.21%
512	53.33%
1024	56.29%
2048	52.67%

Layers

Although it is not compulsory for LSTM to add more layers, stacking more layers on the network would make each sample to be exploited on its extracted features, this would definitely improve the overall performance. However, just like the number of epochs, there is a trade-off relation between the number of layers and run time. Because of the time restraint and hardware limit, this project did not have the chance to experiment on the LSTM network more than 2 layers. The performance is shown in Table 6.3 in the Experimental Evaluations section.

6.1.2 Transfer Learning

The implementation of this project concentrates on Recurrent Neural Network. However, Graph Convolutional Neural Network approaches are becoming very popular in recent years of research in video recognition. In order to compare the performance on the general direction, this approach inherited the approach of Spatial-Temporal Graph

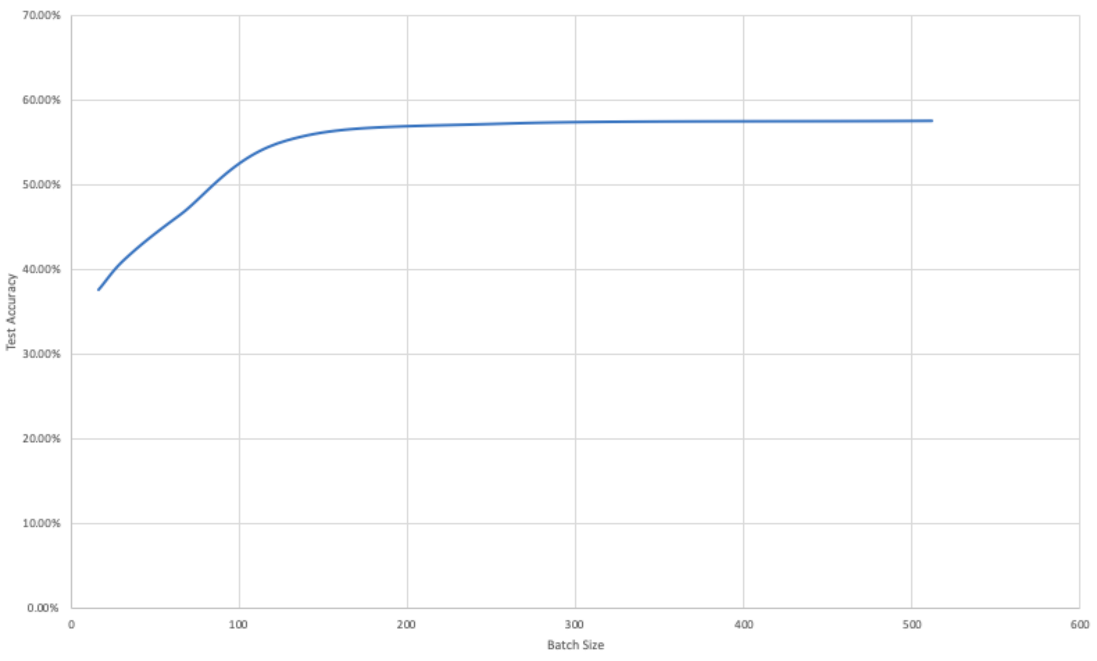


Figure 6.2: The performance on basic LSTM with the different batch size

Convolutional Network from the GitHub of its author who works at the Multimedia Laboratory in The Chinese University of Hong Kong [17]. The code they posted is well-formatted, and there is a specified instruction about how to run it and test, the author tends to make it an open-source toolbox for skeleton-based human understanding tasks.

The inherited code is used for transfer learning in this project. Transfer Learning is a research problem in the machine learning area that focuses on deploying experiments faster when dealing with a different but related problem [16]. metaphorically, this is like a teaching activity. A teacher can teach students how to play, even though the teacher cannot teach students every solution to all scenarios, but from the experience students learnt from the teacher, they can still improvise based on the tricks taught by the teacher and the actual situation. This teaching activity is the information transfer from teacher to students, and students learn this pattern from the transfer to perform on new cases. In this project we inherited the Spatial-Temporal Graph Convolutional Network from a well-done project as they are the teacher in this case, this network is also used for human 3-dimensional skeletal actions recognition that same to our project, although the data is not precisely the same, some of the parameters they present can be an inspiration for tuning in our case.

6.2 Experimental Evaluations

Unlike any usual classification task that has binary or 10 classes that can be evaluated confusion matrix because a multi, this project has 40 classes which cannot be assessed by standard True/False Positive/Negative index. The only way to assess the performance of each experiment is the categorical accuracy for testing. Luckily, as explained in Chapter 3, the data provider provides an evaluation method on data itself, cross-subject and cross-view. We can assess the overall network performance by comparing its test accuracy on each evaluation method, make sure the model is practical rather than just lucky.
All of the following results in Table 6.3 is obtained with the balance of the performance and the time cost. Due to the hardware limit, more kinds of models cannot be performed. From the result, we can see the Part-Aware LSTM has a significantly improved from the basic LSTM because it divided one input as 5 parts so the network could know which part is moving instead of searching in the whole body to learn the patterns.
As a result of experiments and the comparing trade-off relation with the practical runtime and model performance, For LSTM methods, the number of epochs was set to 2000, with the batch size of 512 samples. As for the GCN method was experimenting on the hyperparameter list that they suggested. On the other hand, ST-GCN as a

Table 6.3: The Evaluations of the two settings with different models

Model	Cross-Subject Accuracy	Cross-View Accuracy
1 Layer LSTM	40.19%	50.71%
2 Layer LSTM	49.4%	53.33%
1 Layer PLSTM	60.28%	67.27%
2 Layer PLSTM	62.37%	69.34%
ST-GCN	75.79%	79.22%

comparison method achieves the highest performance. Although it is inherited transfer learning, which is nearly optimum while the model we built from scratch is lack of time for tuning, the advantages in modified graph convolutional networks are significant. This is likely because the Convolutional Neural Network can study more features from single frames, and if it can be recorded in the time stream, the performance would be better than LSTM approaches.
Also, for two evaluation methods comparison, we can see in all models are performed better with cross-view dataset than the cross-subject dataset. This is because the deep learning methods focus on the shape of the subject rather than its angle since the subject is in 3-dimensional space because different subjects have unique shapes of their own, the model may find confused, therefore dropped more features from learning. In the literature review in Chapter 2, in the paper by the data provider, some of the

baseline method they used which is old-fashioned in these days, have better accuracy on cross-subject than cross-view, even though the accuracy overall is twice lower than deep learning methods.
For more overall discussion and conclusion about the project, and the top misclassified actions presents in Chapter 7.

Chapter 7

Conclusion

7.1 Project Conclusion

This project uses deep learning networks to classify human actions into multi classes. The implemented networks are Recurrent Neural Networks that are perfect for learning the time series data like a video. For comparison with this method, we inherited one of the Convolutional Networks implementation from a published paper since the Convolutional Networks are better for learning features from images. According to the evaluation in Chapter 6, all of the implementations have done the job by achieving an acceptable test accuracy. However, the evaluation did show that the modified Graph Convolutional Networks are better than Recurrent Neural Networks. This result may be because the Recurrent Neural Networks implemented in this project did not pay attention to the detailed features in the actions. In the literature review some proposed recurrent models can do better with the modification of the geometry vectors. Unfortunately, this project did not have the time to implement all the proposed models.

Table 7.1: Top 5 Misclassified Actions in different Approaches

PLSTM	ST-GCN
Reading → Writing	Clapping → Tear Up Paper
Phone Call → Salute	Put on Jacket → Take off Jacket
Wipe Face → Rub Two Hands	Put on a Shoe → Take off a Shoe
Drink Water → Eat Meal	Hopping → Jump up
Brush Teeth → Brush Hair	Put on Glasses → Take off Glasses

The experiments prove the hypothesis we made in Chapter 3 about the similarity of actions may confuse models in some way. Table 7.1 shows the top misclassified actions for different models. In comparison, it clearly shows the flaws in both models. The LSTM approaches are hard to deal with the minor feature differences in the actions like a Phone call and Salute action would both raise the hand near the ear, but the place and the position of the arm would be slightly different. So even though the ST-GCN has been modified for time series images, it still makes mistakes about the order within the videos which would lead to misclassification. Fig 7.1 and Fig 7.2 show a visualisation of a pair of two actions that would be misclassified as one, hopping and jump up.

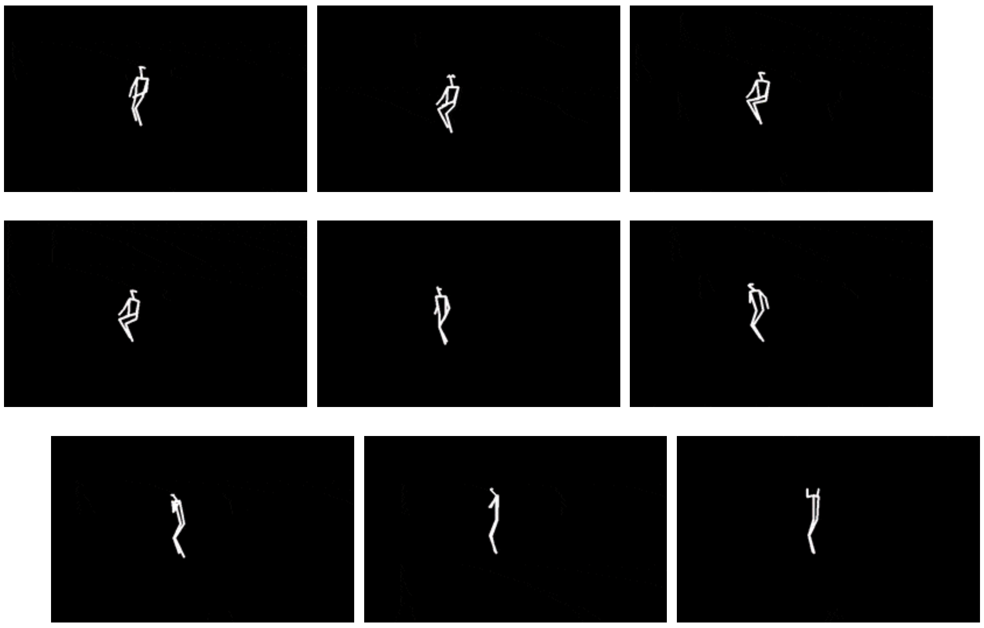


Figure 7.1: Visualisations of One Misclassified Actions Pair (Jump up)

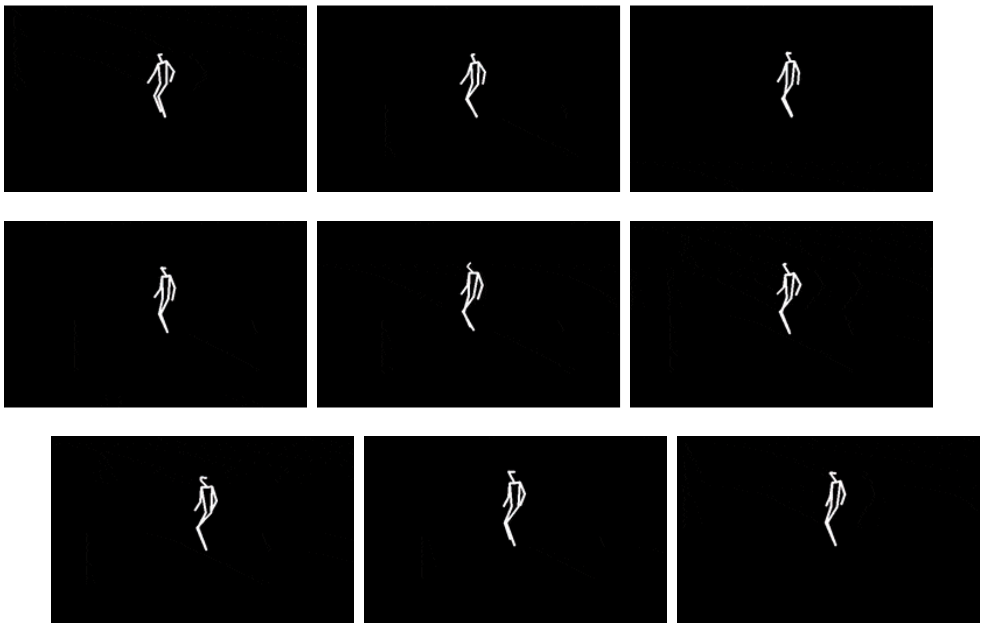


Figure 7.2: Visualisations of One Misclassified Actions Pair (Hopping)

7.2 Challenges

There are several significant challenges and difficulties in the process of the project.

1. **Data Hunting** This project was supposed to be provided with data by a partner of the university. However, due to some unforeseen reasons, the data did not pass through. As an alternative, the project can only find qualified data by itself. Contacting the data sources and acquiring for authorisation needs time, which brought the next challenge.

2. **Shortage of Time** As a result of Data Hunting, The time for overall implementation phase was postponed. There was not enough time to try more models than those presents in the report.

3. **Lack of Information** The data is provided without a thorough document about how to use it and explanation of data attributes. There are only a few papers published by the data provider can be used as a reference. The sample code they provided for data extraction is written in Matlab, which is not learnt before. By reading their papers and the manual their cameras for shooting the videos, learning the Matlab code, some inspiration was gained for understanding and processing the data.

4. **Lack of Instruction** The practical problem for this project is that there are no examples to learn from. In the human action recognition tasks, most of the recent proposals are Graph Convolutional Networks related and written in PyTorch, while our task is to implement the Recurrent Neural Networks, similar implementations on the Internet are either from years ago and written in other format like Matlab code or Lua code that used on an outdated software, or some newly published papers that only have a theoretical proposal without codes available. In short, this implementation is done from scratch. Since the nature of the coordinates is a 3×25 tensor while an LSTM layer can only accept a list in one timestep. Multiple attempts have carried out before success. Also, there is no instruction on how to build the Part-Aware LSTM, which is already discussed in Chapter 4. Besides the model building, datasets building is also hard because of the vague instruction and shortage of tutorials online for Tensorflow file type, which is discussed in Chapter 5.

7.3 Project Limitation and Future Work

This project is limited to some conditions that are explain below following a proposal to work on it in the future.

1. **Data Varieties** This project only uses one-person daily actions. The medical condition actions and two-person actions are discarded since it would be more difficult to classify. If there is more time in the future, these two types of action classes can be used in the model with better parameter tuning. Moreover, the data provider has a larger dataset for 120 action classes that can also be used if there is enough time.

2. **Customised Cell Limitation** Due to the shortage of time, The Part-Aware LSTM this project developed is not refactored for general use yet. It is strictly limited to the data processing outcome, even a slight change or different dataset

would cause the model failed to run. In the future, the customised cell can be refactored for more kinds of data. In the best-case scenario, it can be made into an API for future transfer learning.

3. **Model Varieties** More models that introduced in the literature review did not have the time to implement, especially the Graph Convolutional Networks. Also, the implemented models are not fine-tuned, there is space for optimisation. If there is enough time, more models can be implemented for comparison, and the evaluation would be more comprehensive.

7.4 Personal Reflection

The task in this project is very challenging to me. In this year of master study, the only module I have learnt is image analysis that taught simple image classification using Convolutional Neural Networks. Before this project, I have only a few learning experiences in deep learning, mostly in Multi-Layer Perceptrons and Convolutional Neural Networks. Recurrent Neural Networks are a new knowledge to me.

The beginning of this project did not go well. Although the task of this project is well understood very quickly, the data understanding and processing took me longer than expected because of the lack of information. However, throughout the struggling and questioning myself, when the process is finally done, I have learnt a lot from it, and I believe if there is more time or there is a new task, I can do better.

In the latest papers published by masters in the field of study, I found myself very interested in the models they built and the principles and ideas behind them. If I have an opportunity to enter a PhD program, I would like to study more deeply in this field, so that one day I can present my own theories and ideas.

Overall, in the process of this project, although there were many difficulties and setbacks, I finally overcame everything and completed the objectives. The task was completed on time, and I also gained invaluable experience.

References

[1] J. Hale. Deep learning framework power scores 2018, September 2018.

[2] Keras. *Keras Documentation*, 2018.

[3] A. Kumar. Introduction to machine learning, 2018. [Online; accessed 2-September-2019].

[4] C. Li, Q. Zhong, D. Xie, and S. Pu. Skeleton-based action recognition with convolutional neural networks. In *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pages 597–600. IEEE, 2017.

[5] J. Liu, A. Shahroudy, M. L. Perez, G. Wang, L.-Y. Duan, and A. K. Chichung. Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[6] J. Liu, A. Shahroudy, D. Xu, A. C. Kot, and G. Wang. Skeleton-based action recognition using spatio-temporal lstm network with trust gates. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):3007–3021, 2017.

[7] Microsoft. *Kinect for Windows – Human Interface Guidelines*. Microsoft, 2.0 edition.

[8] A. New. Brexit: The uncivil war showed us how the eu referendum was won with data science, 2019. [Online; accessed 2-September-2019].

[9] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang. Ntu rgb+ d: A large scale dataset for 3d human activity analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1010–1019, 2016.

[10] Tensorflow. *Tensorflow Documentation*, 2019.

[11] Wikipedia contributors. Comma-separated values — Wikipedia, the free encyclopedia, 2019. [Online; accessed 29-August-2019].

[12] Wikipedia contributors. Convolutional neural network — Wikipedia, the free encyclopedia, 2019. [Online; accessed 2-September-2019].

[13] Wikipedia contributors. Machine learning — Wikipedia, the free encyclopedia, 2019. [Online; accessed 2-September-2019].

[14] Wikipedia contributors. Numpy — Wikipedia, the free encyclopedia, 2019. [Online; accessed 29-August-2019].

[15] Wikipedia contributors. Supervised learning — Wikipedia, the free encyclopedia, 2019. [Online; accessed 2-September-2019].

[16] Wikipedia contributors. Transfer learning — Wikipedia, the free encyclopedia, 2019. [Online; accessed 29-August-2019].

[17] S. Yan, Y. Xiong, and D. Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] P. Zhang, C. Lan, J. Xing, W. Zeng, J. Xue, and N. Zheng. View adaptive neural networks for high performance skeleton-based human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

Appendices

Appendix A

Code on GitHub

The code of this project can be acquired by opening the URL below:
`https://github.com/sc18zj/UniLeeds-MSc-Project`

Appendix B

External Material

Keras: This library is used to build model and customise cells and layers.
Link: `https://keras.io`

Tensorflow: This library is used to create TFRecord files and as the backend of Keras.
Link: `https://www.tensorflow.org`

Scikit-Video: This library is used to visualise the data.
Link: `http://www.scikit-video.org`

Numpy: This library is used to implement array operations.
Link: `https://numpy.org`

LaTeX: This report is written in LaTeX, a document preparation system that designed for the production of technical and scientific documentation.
Link: `https://www.latex-project.org`

Appendix C

Ethical Issues Addressed

The data used in this project belongs to Nanyang Technological University, Singapore. In order to protect copyright of their work, the data is not shared with any other organization or individual, nor uploaded on the Internet. The use of data is strictly limited to academic research activities and does not capture any commercial benefits. The ethical issues within the data, such as the portrait rights of the actors, are the responsibility of the data provider and should be taken care of them.