Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

by Luke McClure
12 May 2020

Tracking human movement to prevent resistance training injuries using generalised Artificial Intelligence systems.

Project supervisor: Dr Heather Packer
Second examiner: Dr Maurits De Planque

A project report submitted for the award of
MEng Computer Science with Artificial Intelligence

0.1 Abstract

There are little to no tools or methods to classify a human motion pattern as fitting a movement or not, a task particularly prevalent in the world of resistance training where a lifter can fail a repetition of a movement if they are not within the boundaries of following a movement pattern. This project aims to firstly provide a system that is able to class a repetition of a resistance training movement between failure and success and secondly provide a general method for developing systems to classify the correctness of movements using a machine learning model that is able to be dropped in place into the system.
Developing a general method for this task is key to provide a framework to push the boundaries on this type of problem, and to explore what may be possible with this type of analysis.

0.2 Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence,

references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/-code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

Contents

List of Figures

1. Body

1.1 Introduction

1.1.1 Problem Statement

There is a lack of computer systems that can classify if motion of a human correctly follows a movement pattern, or if the movement is being performed incorrectly. Developing a method and system to generalise classifying full body movements will help further development of classifiers for many differing types of movement.

1.1.2 Motivation

Keeping correct form is crucial to performing any exercise safely and it is incredibly easy to sustain injuries if not doing so. The problem with this is that it is hard to ensure a movement is being performed correctly, especially under heavy load, as it is difficult to get an external viewpoint of performing the movement. This is also compounded as correct form can vary between lifters from how someone has learnt to perform the movement and biomechanics of an individual. As a result, there aren't many tools available to ensure correct form of people during resistance movements.

1.1.3 Goals

This project aims to show a computer system can classify if a specific movement pattern is being performed correctly by a human, in this example a resistance training setting performing a squat. The system framework should be general enough to be trained on different movement patterns while still maintaining accuracy.

1.1.4 Scope

During initial development of this system, the focus will be on a single movement to show the system is capable of classifying movement correctly. The squat was chosen as there is a lot of variation in how it's performed safely with lots of unsafe variations of the movement. To focus on this goal, there will be constraints on the format of video including where the camera will be placed, a single lifter will be used and there will be the only person in frame.

1.2 Background and Related Work

1.2.1 Background

Several systems have been created for gyms that incorporate the motivating features behind this project, but track bar path rather than tracking user movements.

SmartSpot

SmartSpot[1] uses a single camera to track features of a performed set including data about depth, bar path, tilt and velocity but also data about the user similar to what is proposed here. Using joint tracking the system can measure back stability, curvature, weaknesses and imbalances.

Perch

Perch[2] is similar to SmartSpot but focuses on bar tracking, being able to generate velocity, power and bar path data and predict performances from a single camera unit attached to a rack in front of the user.

Profile Estimation

This is a paper aimed at introducing CV techniques to resistance training[3] specifically on trying to measure many of the values that are also found in SmartSpot and Perch. The bar & weight plates are used to create features to track throughout the video. This gives the benefit of being able to accurately measure the bar path, and from extrapolating a single point on the bar can measure velocities within the movements performed but will not allow for analysis of the lifter's fine movements.

1.2.2 Computer Vision

When working with human movement, abstracting video into a skeletal model of the movement with only key joints and connections can be more useful than raw

[1] http://www.smartspot.io/
[2] https://www.perch.fit/

input[4]. There has been a lot of research into 2D pose estimation from camera feeds, but due to lack of training data limited progress has been made recognising 3D form from single video sources.

2D Pose Estimation

2D pose estimation libraries have been created using multiple different methods, OpenPose[5] is one such but under explicit statement in the license the library is not to be used for any sports applications. AlphaPose[6] is another library which focuses on real-time pose estimation, AlphaPose could be applied to this project for fast results if accuracy is not dependant on 3D estimation.

VNect

VNect[7] uses a single RGB camera to perform 3D human pose estimation.

FIGURE 1.1: A demonstration of VNect software

VNect is significantly useful for this application due to extensive back joint modelling and separate joints modelling the shoulders allowing for rotation and back bending to be tracked. VNect has issues with self-occlusion, but this can be avoided by placing the camera to open up as much of the body to the camera as possible.

SMPLify

SMPLify[8] models soft tissues of the physical human form to generate 3D pose and skeletal models, with the aim being that more accurate predictions will be

made that do not violate any biomechanics. Whilst this method may be more accurate due to modelling of more accurate bodies, this system does not provide a simple skeletal model.

Kinect

Xbox Kinect has been used frequently due to the combination of a camera and depth sensor. One use has been to detect gait of a person by using the skeletal model to calculate the joint angle of the knee[9]. Using angles between joints allows for a significantly lower input size for the ML model to handle compared to a whole video. This approach would also allow for much easier generalisation between movements as taking angles between joints and how they change can be modelled for any movement. This method cannot account biomechanics between people; people with differing bone structures may not be able to squat broadly consistent joint angles.

Recent advances

Recently development has been made using CNN's in 3D Pose estimation[10]. This includes Context-Aware networks and Occlusion-Aware networks. Occlusion-Aware networks aim to minimise the error from pose estimation using a single video source which will eventually find some body parts occluded[11]. Whilst this method boasts a higher accuracy than other models, it suffers from a lack of joints tracked other models make use of.

Summary

VNect is the strongest contender for pose estimation due to the number of joints tracked and information that can be inferred from this. Detailed information on how the back moves and how the body is rotating is crucial to be able to tell if a movement is being performed correctly.
Calculating joint angles would allow for a significant reduction of computation required throughout the ML system. This also provides variability in the use of the system and can allow for the machine learning system to be trained on how joint angles should move and change together rather than videos of movements.

1.2.3 Machine Learning

For this section of the literature review I will be investigating the use of two different types of possible input for the machine learning section of the system.

Machine Learning with video

There are multiple methods of performing classification tasks on a video input that have been investigated by researchers and it is a very densely researched field due to both wide-ranging applications (self-driving cars, biometrics, etc) and vast amount of video data that is available through the internet. CNN's have been proposed to classify video with notable successes[12], however, the large scales of data needed to train these networks make this infeasible. Deep Convolutional Graph Networks have been proposed for this same task with largely accurate results outperforming other methods tested[13].

Machine Learning with joint angle

Calculating joint angles through time would cast the problem into a time series classification problem[14] as classification would be on a graph of joint angles through time. Within time series classification several methods are well defined for this task. 1-Nearest-Neighbour Dynamic Time Warping is widely regarded as best for its accuracy[15] and the ability to cope with mismatched starting time of series. This technique is so successful that research has been conducted on improving it with deep learning[16][17]. LSTM networks with CNN's have been used achieving state of the art results[18] due to the ability of the network to recall previous events in the series.

1.2.4 Summary

VNect is ideal for the multiple joints tracked down the spinal column and the joint definition of the shoulders using a T structure rather than a single root of the arms. This allows for measuring rotation across the body as well as being able to accurately tell where the back may be bending, a crucial notion in some resistance training movements. Using VNect to track joint angles the system would allow recognition of when a joint is moving improperly. By using joint angles

rather than video for the ML model would transform the problem from video classification to time series classification, a much wider studied topic. This also introduces invariance in position and incurs lower computational costs due to the smaller dimension of the data. With 1NN-DTW being regarded as a benchmark algorithm for time series classification[19] it would make a good algorithm to use.

1.3 Design

1.3.1 Proposed Design

This system will comprise two main components supported by various tools to produce a complete chain from video input to classification, these additional tools ensure the correct format of any data passing through the system.

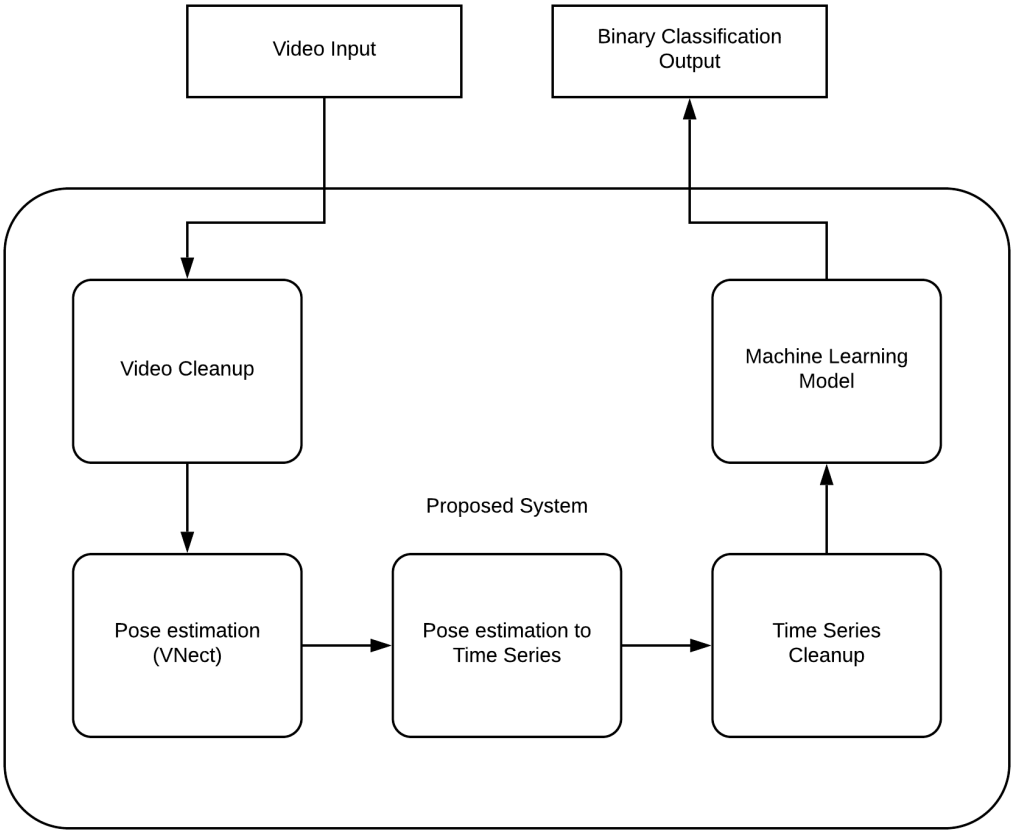

FIGURE 1.2: Initial proposed design of system

To facilitate easy training it is important to separate the system into two sections: the machine learning model, and the video to time series conversion. Doing this means there is a system to run training videos through to create training data of the correct format that will be used for training the machine learning model.

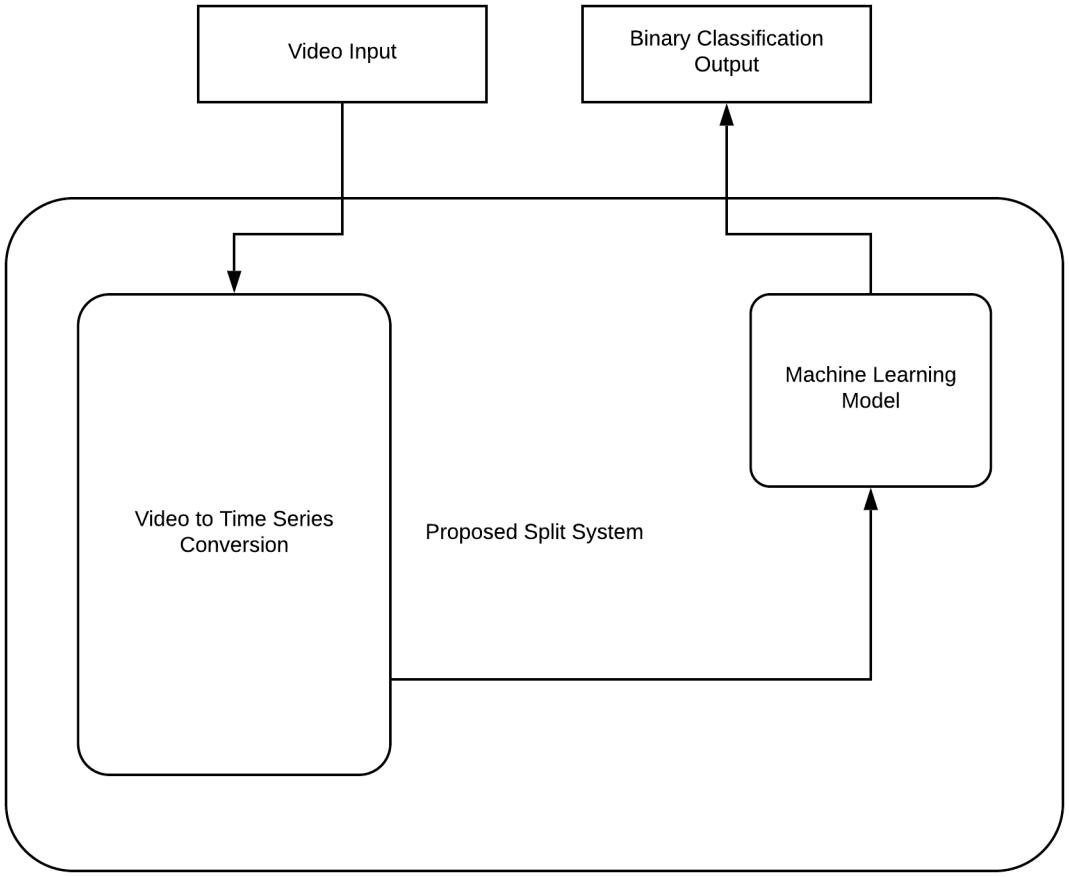

FIGURE 1.3: Proposed split design of system

Video to Time Series Conversion (VTSC)

This tool maps a video to a time series of joint angles. This involves all video preparation, pose estimation and extracting the joint angle time series necessary for the model.

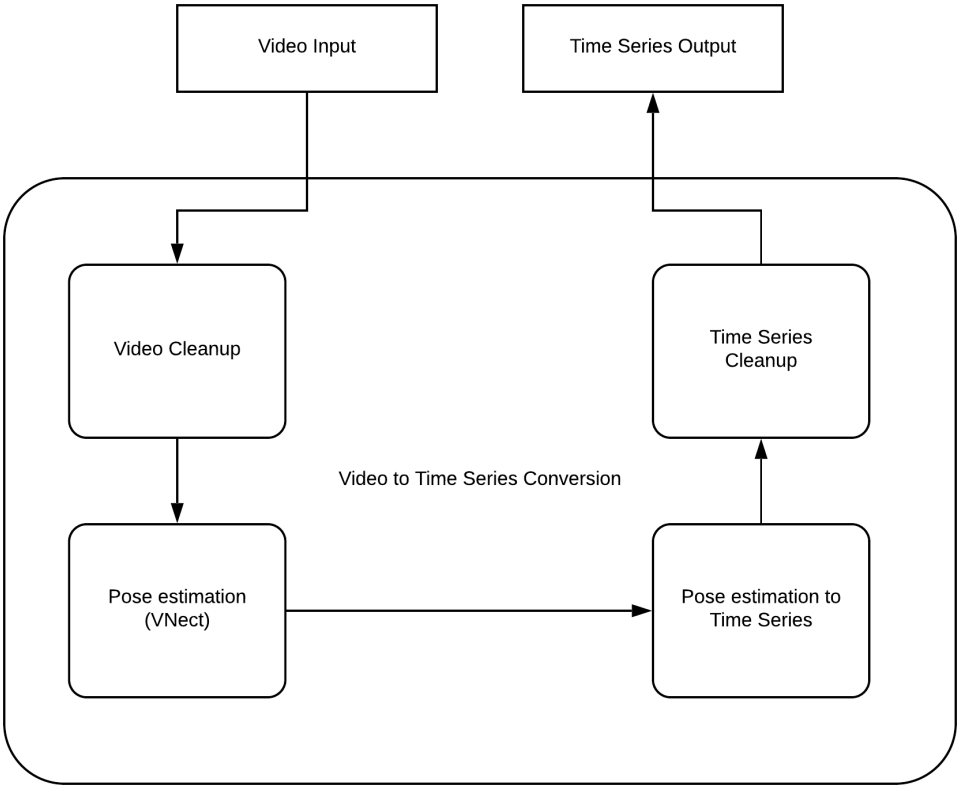

FIGURE 1.4: Design of Video to Time Series Conversion

Video Preparation

Input needs to be formatted on entry into pose estimation, this process involves splitting a video into a folder of its frames and storing metadata about the video such as file location and number of frames.

Pose Estimation

This is where the VNect pose estimation library will be used to convert the input video into pose estimation data.

Pose Estimation to Time Series

Extraction of joint angles isn't native to VNect, this process manually calculates the joint angles for every frame using the methods discussed in the literature review.

Storing time series in CSV format is ideal for this task due to the multivariate nature of analysing multiple joint angles. CSV would allow modelling of each joint with each record being a step through time.

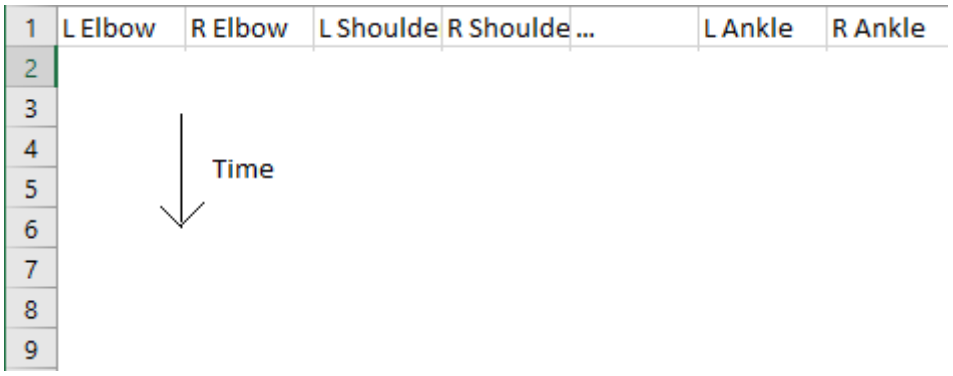

FIGURE 1.5: How time series data will be stored in the system

Machine Learning model

The model will consist of only input of a time series and outputting a choice result between classes. The modular design of this system allows for experimentation between different methods of classification, although development will start with 1NN-DTW.

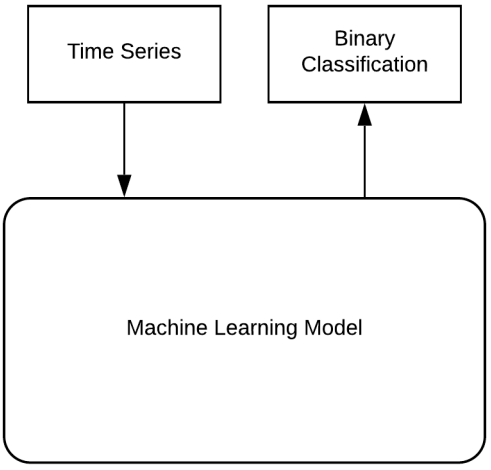

FIGURE 1.6: Design of the Machine Learning model

1NN-DTW can be inefficient due to the difficulty in indexing and storing time series data to build clusters. Due to these inefficiencies, it may be worthwhile attempting using Euclidean distances between time series or exact indexing techniques[20] to cluster on within KNN.

If major experimentation is to take place, development of the 1NN-DTW will follow methods presented in [21] due to the demonstration on increasing performance and how to generalise for multivariate uses. Otherwise to keep development time down library functions will be used to create 1NN-DTW classification.

1.3.2 Classes

There are three control flows within the system:

1. Converting a directory of labelled video data into training data.
2. Training a model using labelled training data.
3. Classifying a video using a trained model.

All data will be held on disk for ease of availability. All control flows can be operated from the Main class, but control flows 2 and 3 can be accessed standalone from the Time Series Analysis supposing correct data format is supplied.

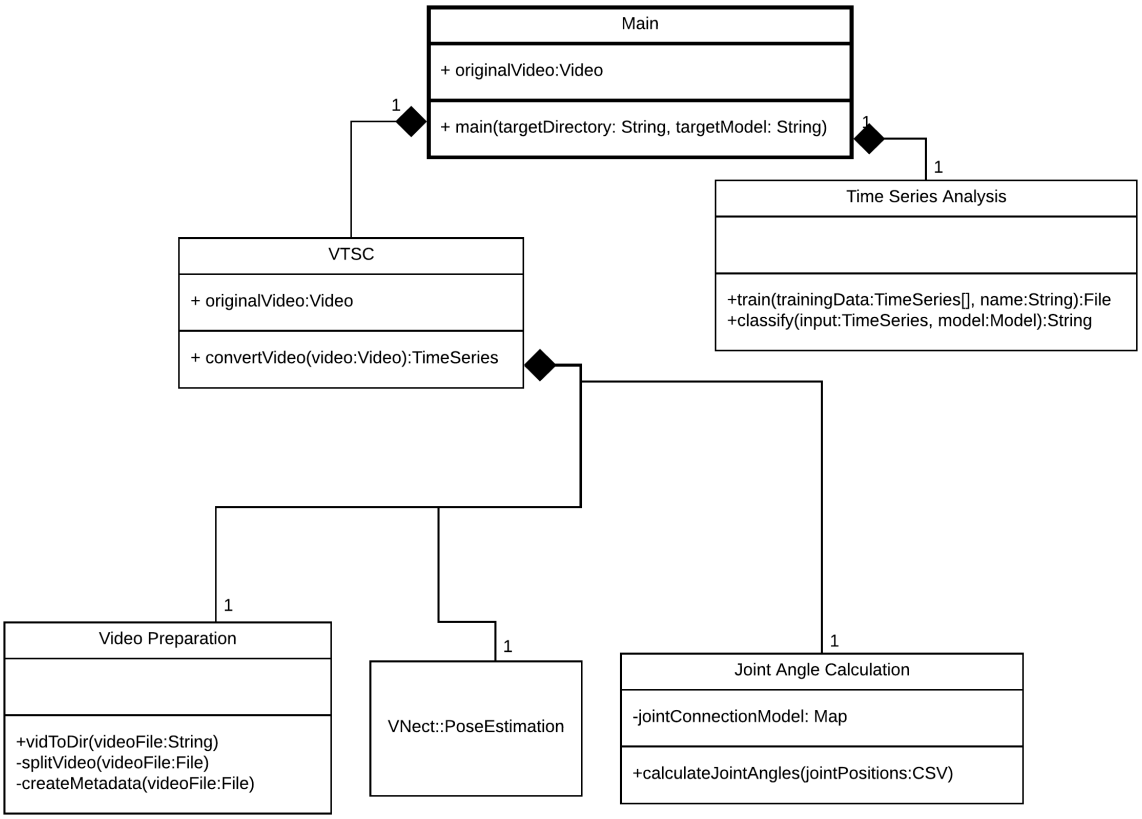

FIGURE 1.7: Class Diagram of the system

1.3.3 Justification of Design

This design allows for the modularity necessary to achieve the project goals, being able to classify many different types of movement given a trained model due to the split nature of this design. Data flow within the VTSC is designed to throughput video into generic time series data that may be manipulated by the model. Initially this will be designed to use 1NN-DTW due to how accurate the algorithm is within time series classification[21]. This model can be altered per experimentation to achieve better results on a movement, or to try and classify another movement within the same system framework. This is not limited to 1NN-DTW, other methods of time series classification can also be tried such as LSTM-CNN and attempting more novel approaches for this problem.

1.3.4 Tools and Technologies

VNect

Pose estimation will be driven by VNect for this project due to its ability to track many joints across the body accurately. The methodology behind VNect is similar to other systems where 2D heat maps are made for each joint using a CNN to create a set of 2D keypoints that correlate to the person in frame, these are then placed alongside 3D location maps to attempt to build a 3D pose and skeletal model of the human in a frame.
By using a 3D model this allows capturing much more data about the movements being performed and can pick up on discrepancies between performed and desired movements in ways 2D models could not.

Pandas

Pandas[22] is a well known data science library for Python that is compatible with many other large Python libraries, this makes it very useful for data manipulation. Pandas is able to model time series data with ease and has the functionality and widespread support to work well with TSLearn.

TSLearn

TSLearn[23] is a Python machine learning library designed around time series data. Such a focused library makes it incredibly useful as it already implements a DTW class which is key to the machine learning component of this system. Crucially this DTW class is able to be used as a metric for it's KNeighborsTimeSeriesClassifier allowing for much easier implementation for the classification algorithm.
The many algorithms built into TSLearn allows for lots of manipulation and experimentation on how the ML model will function for this project and can be optimized for this task without having to worry about personally implementing algorithms correctly.

1.3.5 Scope

Requirements

A table outlining requirements of the system to be produced has been outlined in Appendix 2.1.

Constraints

There must be several constraints on the system to ensure focused development to achieve the end goal of the project.

- This system will initially be designed to classify one type of movement, in this case the squat.
- The camera will be placed in a consistent location directly in front of the lifter throughout the dataset to minimise self-occlusion.
- There will only be one person in the video for any input.
- Any input of a movement must be trimmed to the start and end of the movement and be no longer than 10s.
- The system only has to handle and classify video of a single repetition.

Goals

Success of the system will be evaluated using the following criteria:

- Recognise a human in a video input in a gym setting.
- Convert video of movements into a skeletal model.
- Capture the joint angles of movements from pose estimation.
- Convert an input video into a time series of joint angles.
- Develop an ML model that can classify good or bad form given a video of a human performing a squat.

Video will be collected by myself to allow accurate labelling of training data. Standard machine learning testing methods can be used to investigate the accuracy of the system. When concluding system accuracy for success, the following criteria will be used.

Measured Accuracy	Success
$\leq 50\%$	Failure
>50%	Cautionary Success
>60%	Definite Success
>80%	Monumental Success

FIGURE 1.8: Table of measuring success

If these goals are met further goals that may be completed to further the project include:

- Demonstrate generality by developing another model that can classify a different movement within the system.
- Expand the system to classify multiple repetitions individually in a set of movements, either producing a score or a pass/fail for the entire set.
- Plan the system to use real-time video.

1.4 Implementation

This project was built primarily in C++ with some Python and compiled into a single executable file and Python script to be run from the command line. There are limitations on how this project can be used due to this duality as well as some of the dependencies of libraries within the project. This includes dll and config files being within the same folder as the executable and Python script when it is run & having data used be in the same folder or in equal or lower hierarchical folders from these files.

1.4.1 Video to Time Series Converter

Due to the VNect pose estimation library only supporting C++, this was the language Video to Time Series Converter was built with.

1.4.1.1 VideoToDir.cpp

This file is tasked with converting the raw video into a format that can be used with VNect, and is the first step of conversion into joint angle time series. For VNect to be able to perform pose estimation on a video, it needs the video to be split into a directory of it's frames and some metadata such as how many frames are in this directory and where this is located.

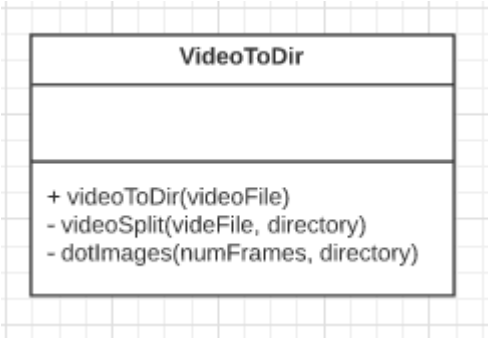

FIGURE 1.9: videoToDir class Diagram

VideoSplit will run through the video and save each frame as a separate jpg into a created folder within a directory for the video.

DotImages creates a supplemental file named imageCalibration.images that provides VNect information about this sequence of frames.
VideoToDir coordinates both of these actions and is the access point from VTSConverter into this class.

1.4.1.2 VTSConverter.cpp

VTSConverter is the main coordinator for creating the joint angle time series' from a video file. It is built to handle both converting a single file and receiving an entire directory and converting it in-place. This class is where VNect pose estimation takes place as well as the calls to vidToDir and TimeSeriesGenerator.

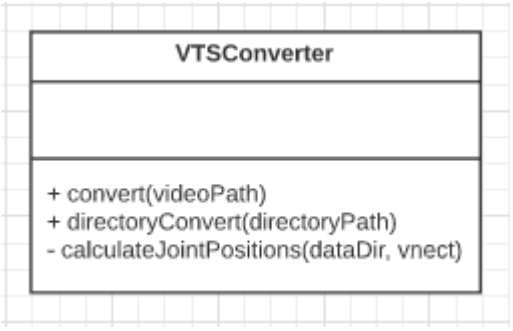

Figure 1.10: VTSConveter class diagram

Convert is used to convert a single video into the joint angle time series data. Within it are calls to both videoToDir and TimeSeriesConverter classes. This method is also utilised by the directoryConvert method.
DirectoryConvert is used to convert a directory of mp4 videos into their subsequent joint angle time series'. It performs this by pushing every file that has an mp4 extension through the convert method.
CalculateJointPositions performs the pose estimation on a given video using VNect, it is necessary to pass it the directory created by vidToDir for each video as this is what is utilised by VNect.

1.4.1.3 TimeSeriesGenerator.cpp

TimeSeriesGenerator is responsible for taking pose estimated time series of joint angles for a video and converting that into joint angles between all relevant body parts. This is done by building a model of connections between body parts and

using that to reference the file generated by the pose estimation for each 3d coordinate of every body part per frame.

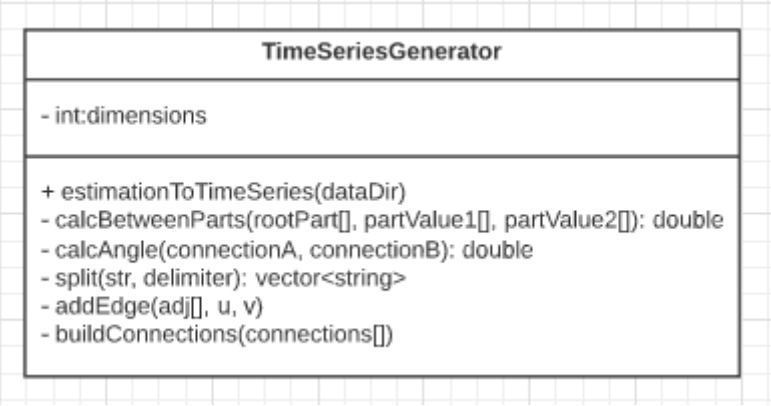

FIGURE 1.11: TimeSeriesGenerator class diagram

In talking about the calculation of joint angles, the examples I will give will be in the format of (root part, part 1, part 2).
EstimationToTimeSeries will convert data of joint positions into joint angles between body parts. Conversion from joint position to joint angle is done by sequentially reading through the joint positions csv and for every entry the angles around every joint are calculated.
A model is created to represent all joints tracked by VNect to easily reference connected parts and count the number of parts connected to any query body part.

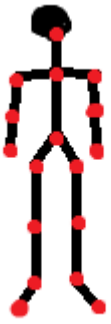

FIGURE 1.12: Drawing showing all tracked joints in red

Any joint can only be connected to a maximum of four others given this model, allowing for set behaviours to be established based on the number of joints connected to a root:

- One part: It is not a joint (ie head, foot, etc) and is ignored.

- Two parts: A simple connection such as through the elbow (elbow, wrist, shoulder) or knee (knee, hip, ankle), these will be passed to CalcBetween-Parts.
- Three parts: A unique position where the spine splits into left and right hip, in this case it makes most sense to separate this situation into two simple cases of spine into left hip (spine, neck, left hip) and spine into right hip (spine, neck, right hip).
- Four parts: Another unique position across the neck point, this is connected to both left and right shoulder, and head and spine. To most accurately model how the body will be moving across the neck it makes sense to split these four connections into a horizontal simple case (neck, left shoulder, right shoulder) and a vertical simple case (neck, head, spine) that will be recorded separately.

As the order of how parts will be queried is fixed by VNect it is known in what order certain joint angles will appear. This enables a hard-coded label at the top of the joint angles csv and for joint angles to be written to the output file at the end of processing through each part for an instant.

The following functions are responsible for calculation of joint angles between any three vectors representing location of body parts.
CalcBetweenParts is used to calculate the angle through a 'root' body part that will connect two body parts. An example would be calculating the angle of the elbow by passing a 3D vector of the elbow as the root part, then passing the 3D vectors for the shoulder and wrist as connecting parts.
The first stage of this is to subtract the root vector from connecting part vectors. This is an important step as otherwise all vectors are referenced from the origin so finding the angle between vectors would not be an accurate for angle through the joint.

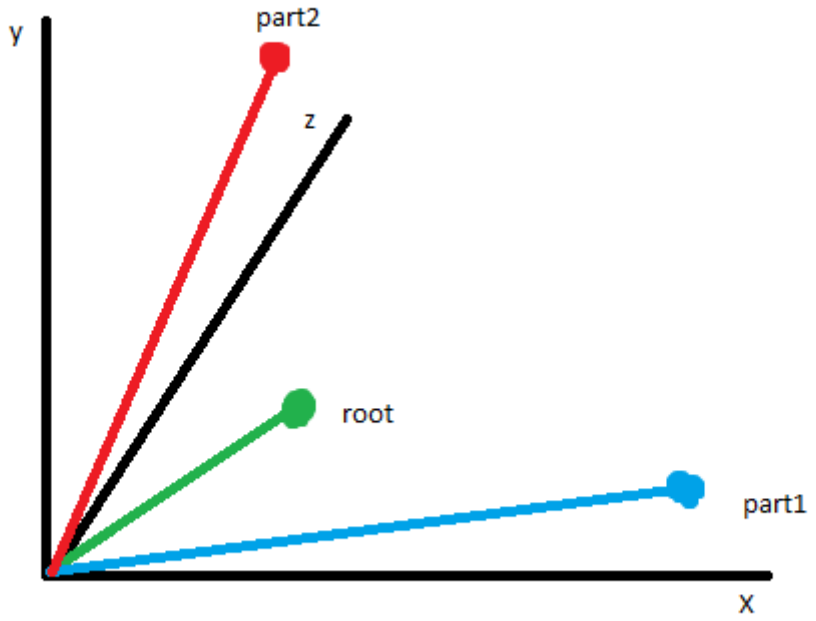

FIGURE 1.13: Showing vectors to the 3 parts involved in an angle

Subtracting the root's vector position from each of the connected part's vectors will create a vector from root to each of these connected parts. This enables CalcAngle to handle these root based vectors to calculate an angle connecting parts 1 and 2 through the root.

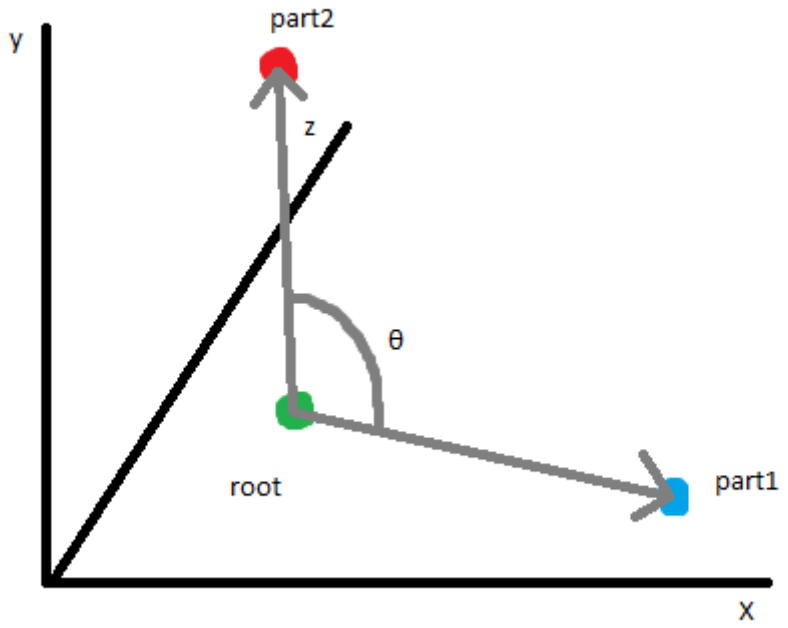

FIGURE 1.14: Transformation into an angle around the root

CalcAngle then computes the angle between two 3D vectors of part 1 and 2 using $\theta = \arccos(\frac{\vec{part1} \cdot \vec{part2}}{\|\vec{part1}\|\|\vec{part2}\|})$.

Split is a basic function used to take a line from the pose estimation output and return an array of body part positions from within that line by splitting the string on a set delimiter. This produces the necessary data to calculate joint angles for each joint from that frame of video.
AddEdge is used as a helper function of buildConnections and is responsible for adding an edge between two body parts u and v in the model adj[]. This is completed by simply adding a label to each of the part's lists of the other part.
BuildConnections creates a model of all body parts and how they connect together. This is used to find all body parts connected to a single part and accurately model the joints of the human body.

1.4.1.4 Implementation Notes

By converting from video format to a joint angle csv format, there is a vast reduction of the size of data being worked with. A lot of efficiencies are being gained

from unnecessary data being thrown away and important details being abstracted into sole features the models have to learn on. This has the effect of pointing the machine learning algorithms at exactly what features are to be used in making decisions, rather than leaving the system to try and abstract meaningful parts out of huge quantities of data. Doing so should therefore improve both accuracy of the models and also allow for a far greater level of customization.
Average file size reduces from a video of 2376KB to csv of 11.7KB, a reduction of 99.5%. Smaller file sizes improve the portability of the system greatly, as the k-means algorithm used for machine learning currently requires the original data to be saved as part of the model file. Reducing the size of this data allows for a smaller trained model file size as well as allowing greater quantities of data in order to improve accuracy without sacrificing this portability.

VNect is designed to be used in real time pose estimation. This resulted in issues creating a single instance of the VNect object without it treating each individual video to convert from a directory as part of a single incoming stream. This resulted in output files having as many frames as the VNect system had seen up to that point (ie sum of all frames of all videos up to current time) which produced inaccurate joint angle calculations.
In order to rectify this VNect had to be reinitialised for every video being passed through the system. While this was not a challenge to implement, it resulted in the underlying neural network being restarted for every video adding several seconds of execution time for each. This may be solvable by using a different library for pose estimation, however due to the needs of this project only producing 1-2 comparably small datasets it is acceptable to run this training conversion once for a long duration for each dataset.

1.4.2 Machine Learning Model

1.4.2.1 The Script

The Machine Learning Model was developed as a single Python script and is called by the main C++ project via command line.

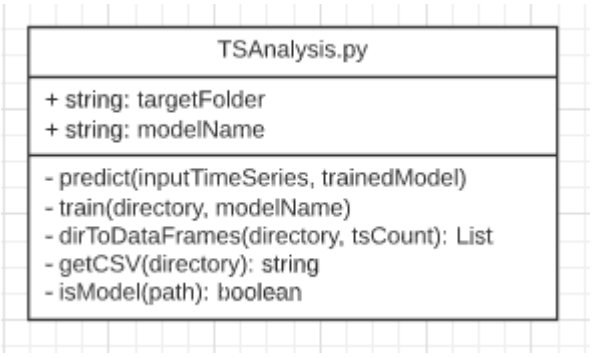

FIGURE 1.15: TSAnalysis class diagram

The only parameters entered into TSAnalysis are the target folder and model name. If the script is to classify a time series, target folder will be the folder containing this time series file and model name will be the pretrained .model file. If TSAnalysis is to train a new model, target folder will be the folder containing all data to train on and model name will be a string that the produced .model file will be named.

Predict performs classification of an unknown time series. It will input a time series and a model and will print the prediction if the movement was performed correctly or not. The time series needs to be converted from Python lists to TSLearn format, then it can be used with TSLearn functions to predict with the KNeighbors classification.
Train is used to train a new model based on input data. Data is pulled from a known folder structure that is created from the VTSC. Training data is labelled using filenames of the folder associated with each video clip, correct will include "_C_" and incorrect including "_I_", allowing a pairing to be created from time series to label. Training is performed using the same KNeighbors classification with identical settings used in Predict, once this model is trained it is saved as a pickle file using the provided model name as "[modelName].model".

The following functions are used as helper functions for the previous two main

functions of this script.
dirToDataFrames converts a directory full of training data into a list of dataframes to be used for training, this is the initial step in converting the directory into TSLearn format.
getCSV retrieves time series csv's from a directory of processed data, this is used in both testing if the first input argument is a single video and in converting a directory full of video files into dataframes as a part of dirToDataFrames.
isModel tests if the second input argument is a model file in the decision process of whether TSAnalysis is to perform prediction or classification.

TSLearn includes a KNeighborsTimeSeriesClassifier function that can be tuned by number of neighbours and, crucially, metric used to define distance. One of these metrics built into TSLearn is dynamic time warping, allowing for quick construction of a 1NN-DTW algorithm as defined in the literature as having excellent classification accuracy for time series data. KNeighborsTimeSeriesClassifier can be run on multivariate sequences, a key aspect when working with this type of data trying to classify sequences of 14 variables through time. This classifier also works on sequences of arbitrary length, a useful feature which means no stretching or compressing of video timings is necessary that could introduce inaccuracies.
TSLearn also features functions to save and load models from disk, allowing for a guarantee of the general purpose desires of the system. As these will produce pickle files, there is a known standard of format for these model files that the system may encounter. The system should therefore be able to easily detect if any data being input from these files is not as it should be for this application.

1.4.2.2 The Theory

The KNeighbors classifier performs classification by finding the k nearest entries to an unknown point, with the unknown point being given the classification of the most occurring label of these k points. To prevent unbreakable ties k must be an odd number. In Figure 1.16 where k=3 the new point would be classified as Class B, however with k=7 the new point would be classified as Class A.
As classification is on multivariate data, usually the procedure would be to extend KNeighbours to multiple dimensions where distance in each dimension is checked. However, Dynamic Time Warping is able to compute multivariate time sequences such as the data in this project into a single distance measure so this is not

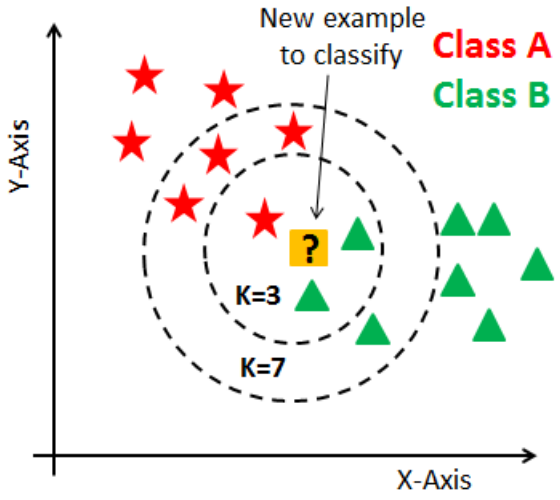

FIGURE 1.16: Diagram of KNN classification[1]

necessary.

To better fit use of time series data, it has been shown in literature that using Dynamic Time Warping as the distance metric between points within KNeighbors rather than Euclidean distance is far more accurate for classification purposes. Dynamic Time Warping works on the principle of best matching a point in a time series that may differ in frequency/speed. This is completed by attempting to find the best possible alignment between two time series, whereas Euclidean matching would simply match each point at t=0,1,2...n.

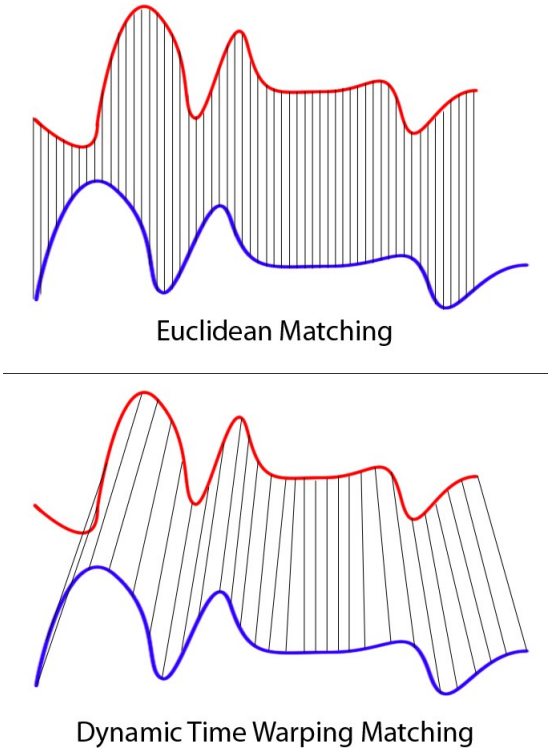

FIGURE 1.17: Euclidean Matching vs Dynamic Time Warping Matching[2]

This is performed by calculating a distance matrix between two time series' using a dynamic programming method of finding the path from base of the problem at 0 distance and building up to full Warped distance between two time series'.

Algorithm 1 Dynamic Time Warping Algorithm [24]

```
function DTW(s[0..n], t[0..m]))
    DTW ← array[0..n, 0..m]
    for i ← 1..n do
        for j ← 1..m do
            DTW[i, j] ← ∞
        end for
    end for
    DTW[0, 0] ← 0
    for i ← 1..n do
        for j ← 1..m do
            cost ← |s[i] − t[j]|
            DTW[i, j] ← cost+minimum(DTW[i−1, j], DTW[i, j−1], DTW[i−
1, j − 1])
        end for
    end for
return DTW[n, m]
end function
```

This algorithm has time complexity $O(n^2)$. Due to small time frames corresponding to n and m within this project (less than 150), this should not become a limiting process as more data is used to train the system so long as each video is kept short. This is even despite needing to calculate the distance between a new point to all trained data due to the non-associative property of DTW distance.

1.4.2.3 The Data

Several initial videos were performed to gather an understanding and visualisation of these joint angles throughout a rep of several movements, as well as a pair of videos of a squat being performed correctly and incorrectly to compare.

Figure 1.18 shows each joint angle colour matched between plots, plotted as their current detected angle at each frame of video. This will be all the data that KNeighbor will receive to perform classification of video clips. Whilst differences can be seen between the two plots, reducing the amount of joint angles plotted to only 4 key joints for this movement will show us a lot more relevant principles behind this data.

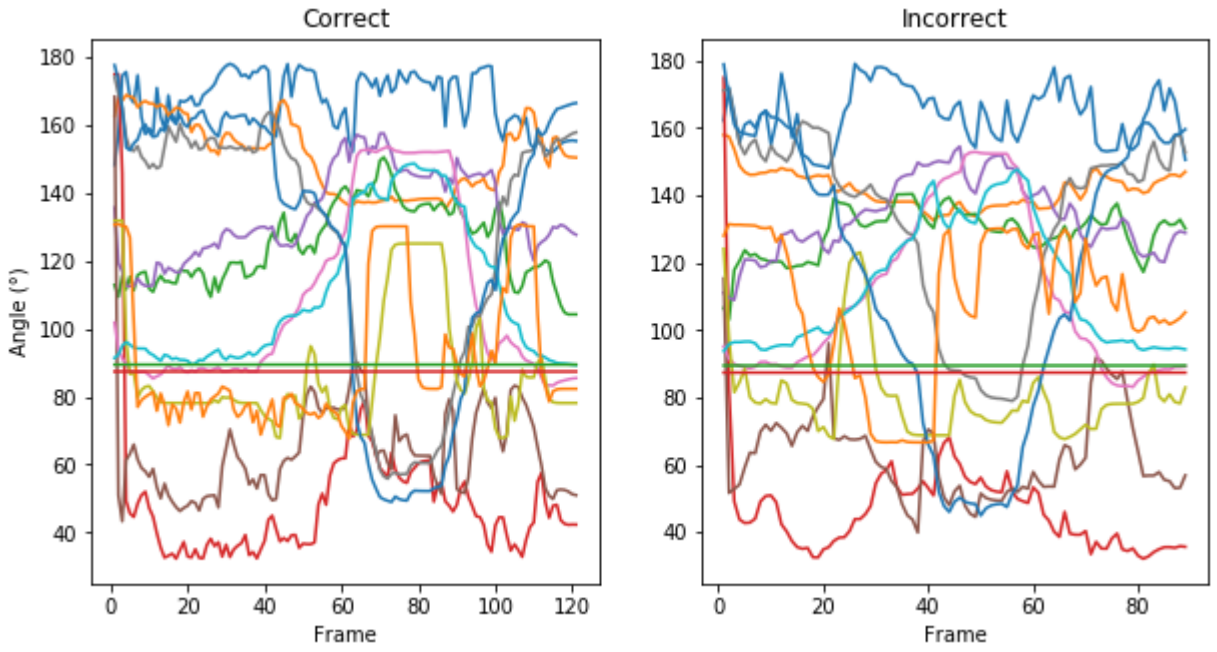

FIGURE 1.18: Comparison of joint angle through Correct and Incorrect performance of a squat

With the simplified view in Figure 1.19 it is clear to see differences between a correctly performed squat and an incorrectly performed squat, with this particular example showing a large leg imbalance throughout the movement. This becomes clear when looking at differences between the left and right knee, and is further shown by apparent compensation throughout the body as indicated by variations between left and right elbow.

In the movement it appears the left knee is performing to a larger depth indicating an imbalance to how weight is being distributed. The apparent opposite movement of both arms, which can be seen as the right arm extends closer to 80° while the left arm stays tighter to the body at closer to 40°, indicates compensation in balance to weight distribution through the legs. Imbalance appears to be arrested near the middle of movement at the bottom of the squat (where most stability can be gained) but reappears during the concentric phase as the body moves upwards. In the instance of correct form, it can be seen that both knees follow a movement pattern almost exactly in sync with each other. This is followed with a very similar movement pattern between the left and right elbow. However, as arms are not as strictly involved in the movement as the legs, variation between the two sides can be laxer due to a mainly stabilising role.

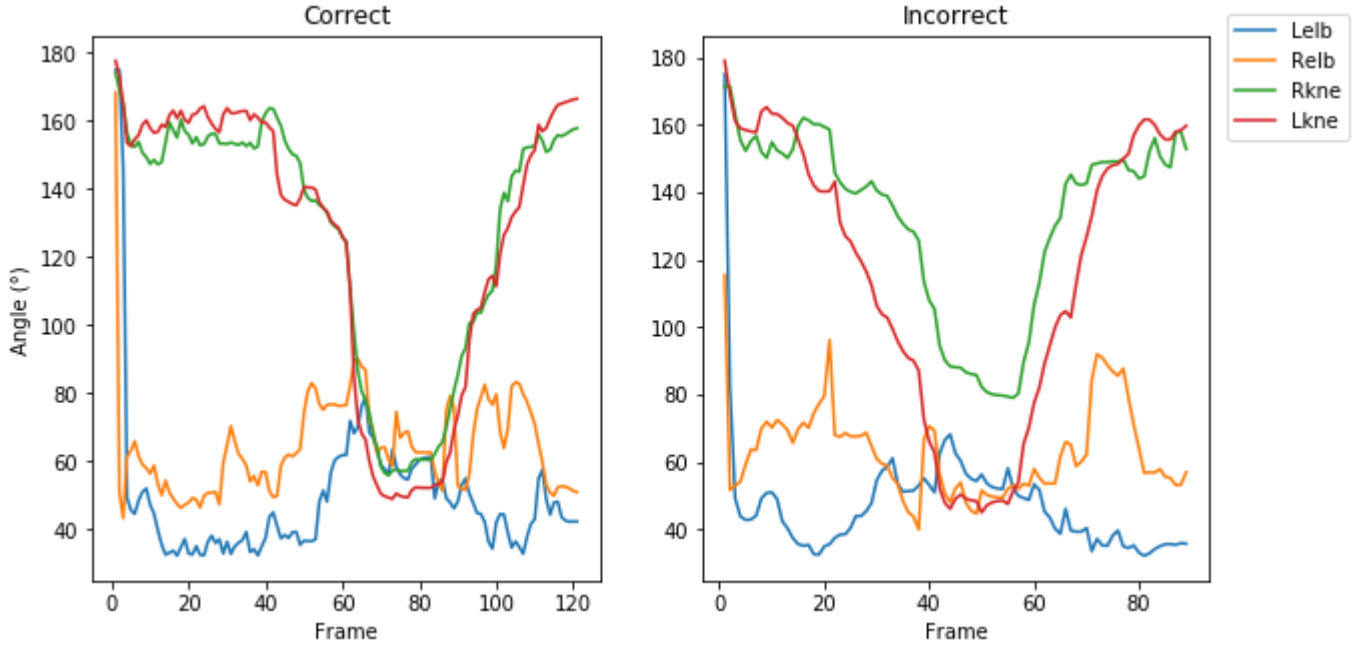

FIGURE 1.19: Simplified comparison of joint angle through Correct and Incorrect performance of a squat

These are some differences that will hopefully be learnt by the system when training on the squat movement.

This sort of analysis can also show how the squat varies with different forms of movements, and gains confidence in how unlikely it may be that any movement may create a false positive for any other movement. This likelihood only decreases as more dimensions are factored into models, these graphs only show 4 tracked variables whilst the system will use 14, hopefully almost entirely removing the possibility for false positives between different movements.

With generality of the system being a goal, it is important to note just how distinct the joint angle graphs are between all three movements. Squats and overhead press are the furthest similarity from each other whilst both are able to maintain strict forms as opposed to an inexact nature of fast paced movements like star jumps. Therefore, it makes sense to aim to train a second model on overhead press to demonstrate this generality.

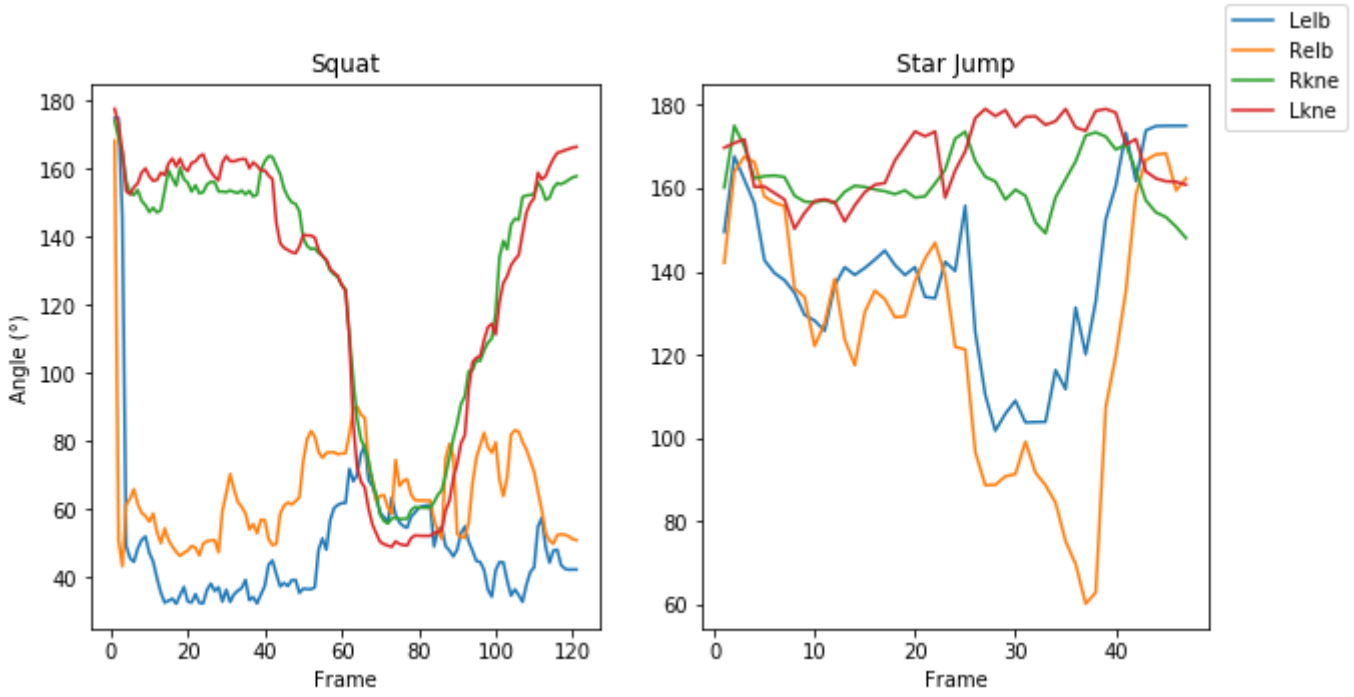

FIGURE 1.20: Simplified comparison of joint angle between Squat and Star Jumps

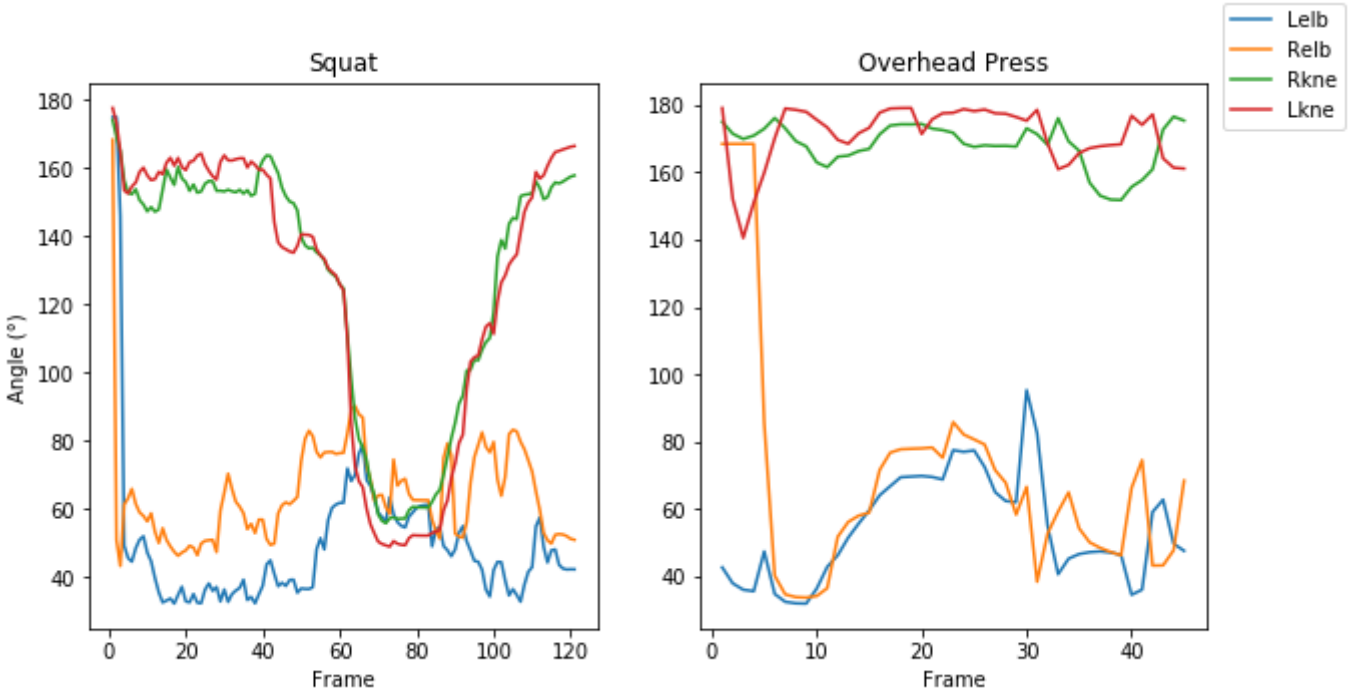

FIGURE 1.21: Simplified comparison of joint angle between Squat and Overhead Press

1.5 Testing

This testing strategy will cover unit, integration and acceptance testing. Each of the two modules of the system will be tested separately, covering each component of that module in unit testing before the entire module will be subjected to integration testing. Once this is completed the entire system will be subjected to acceptance testing.
Testing will be carried out in an order such that it is guaranteed if any function A is to be used by another function B, function A must appear before function B during testing. This is to ensure that any errors that may not have been tested yet in function A do not affect function B as well as protecting the validity of previous testing against any changes that may result from failed tests.

The testing plan for each module, including integration testing and the entire system can be found in Appendix 2.2.

1.5.1 Video to Time Series Converter

To perform unit testing on the Video to Time Series Converter module I will use Google Test, due to the ease of use in creating tests and as it is built into Visual Studio which I am using for C++ development.

1.5.1.1 VideoToDir.cpp

Unit tests which have failed have been mitigated as such:

1. dotImages_DirNotExist
   In the control flow videoToDir, the only place dotImages is called from, the directory that is passed to dotImages is guaranteed to always exist. This is as there is a check where, if this directory does not exist, it is created before being passed to dotImages. In turn this mitigates the failure of this test.
2. videoToDir_DirAlreadyPresent
   This problem presents as there is a check in videoToDir intended to speed up execution where if a video folder is detected it will be skipped, otherwise the folder is created and populated. This is fixed by only using the check for folder creation. On retesting with this fix this failure is mitigated.

3. videoToDir_dotImageExists
   This follows a similar error in logic as item 2 where a .image file will only be created for a video if no directory was present. This can be rectified by splitting out checks for the .image and image sequence outside of depending if no directory presently exists. On retesting this failure has been mitigated.

4. videoToDir_imageSequenceExists
   As this is the same logic error in 3 it can be mitigated in the same way by creating a specific test to ensure each directory contains an image sequence. On retesting this failure has been mitigated.

1.5.1.2 TimeSeriesGenerator.cpp

Unit tests which have failed have been mitigated as such:

1. addEdge_OutOfBounds
   This test fails as there is no check on the values of u and v at the start of the function to ensure they are within the bounds of the graph before any additions are made to the array of vectors. By adding a check on both items this test passes.

2. calcAngle_0
   This test fails as there is no check within calcAngle if there is about to be a division by 0, which occurs when the magnitude of either angle is 0. This has been rectified by introducing a check before the angle calculation.

3. calcBetweenParts_0
   This test fails as it calls on calcAngle, so as a fix is introduced this test also passes.

1.5.1.3 VTSConverter.cpp

No tests need to be mitigated.

1.5.1.4 Integration

These tests will mostly focus on intake of data from entry of the program to VTSConverter, beyond there the system has been tested thoroughly above.

Tests that have been failed have been mitigated as such:

1. FileClassification Notmp4
   This test failed as there is no check between program entry and passing in to VTSConverter if the file is in fact an mp4. By introducing a check here for file type the test passes.

1.5.2 Machine Learning Model

As there is only one unit in this module, unit and integration testing starts to blur together.
Most testing efforts is placed on the helper functions for train and predict as they are responsible for cleaning up inputs and validating correctness, if these helper functions are provably correct any data which is passed into train and predict should be valid and correct.

No tests need to be mitigated.

1.5.2.1 Integration

This tests the entire ML Module, so will capture behaviour of error handling code as input is being read into the script which is missed by unit testing of various functions within the script.

No tests need to be mitigated.

1.5.3 Entire System

No tests need to be mitigated.

1.6 Evaluation and Analysis

1.6.1 Methodology

1.6.1.1 Preparation of Testing Footage

Before any videos are input into the system, they are split into sequential reps to comply with the scope of this project. Each of these reps are reviewed to be classified manually into correct and incorrect form, fixing issues of inconsistent performance during a set. Such errors occur during performance of a repetition where the intention was to be performed improperly, but 'muscle memory' resulted in correct performance. The opposite can happen where a rep from within a correctly performed set was actually performed incorrectly.
This double checking of each repetition video ensures they will be correctly labelled.

1.6.1.2 Preparation of Testing Script

To speed up measuring the accuracy of the system, a script was created to run each testing video through the system and classify them against a model. This allows for measuring of accuracy and speed over many runs of individual videos. This script will run each compatible file in a specified directory individually through the system and will track the prediction and correctness for each video file and total time to complete all files, measuring an average time for each file.

1.6.1.3 Producing Scores

Confusion Matrices[25] are a method of representing many different aspects of scoring for binary classification problems, allowing for analysis of how a model performs within each class and produces scores able to summarise many aspects of a model.

	Predicted: Y	Predicted: N
Actual: Y	True Positive	False Negative
Actual: N	False Positive	True Negative

FIGURE 1.22: Example confusion matrix showing label for each part

A confusion matrix will be derived from test script results for each model, allowing for two scores to be produced with the intention of capturing both general accuracy of each model as well as a deeper understanding of how well it performs within each class.

1. Accuracy
   A general measure of correctly predicted over total tested.
   $Accuracy = \frac{TruePositive+TrueNegative}{TruePositive+TrueNegative+FalsePositive+FalseNegative}$
2. F_1 Score[26]
   The harmonic mean of precision and recall, allowing an insight to how well a model is able to distinctly identify key features and if it has learnt the differences between classes.
   Precision is a measure of how many of a model's predictions for correct are actually correct. $Precison = \frac{TruePositive}{TruePositive+FalsePositive}$
   Recall is a measure of how many of the true correct tests were positively identified by the model. $Recall = \frac{TruePositive}{TruePositive+FalseNegative}$
   $F_1 = 2\frac{Precision \cdot Recall}{Precision+Recall}$

1.6.2 Results of Testing Script

Movement	Test videos used	Training:Testing	Accuracy	F_1 score
Weighted Squat	14	3.28:1	64.29%	61.54%
Bodyweight Squat	40	1.675:1	87.5%	88.89%
Overhead Press	40	1.525:1	80%	77.78%

FIGURE 1.23: Results of Testing Script

The confusion matrix for each movement can be found in Appendix 2.3.

1.6.3 Analysis of Results

1.6.3.1 Squat movement

There are a number of differences between the weighted squat and the bodyweight squat datasets produced that may lead to the differences shown.
There were severe limitations on both quality and quantity of data that could be collected for the weighted squat which could have led to a disparity in scores between bodyweight and weighted squats. Quantity of data was affected by the covid-19 crisis.
In the gym used for collecting data, there existed a harsh lighting down the body that obscures the legs in shadow throughout the movement, making joint position and angles harder to distinguish.
This is contrasted to the bedroom in the bodyweight squat dataset which was well lit. This produces ideal lighting to see the entire body with no shadows obscuring any body part.

The change in camera angle seen in Figure 1.24 is mostly due to the inability to have any mounting system in the gym for simple setup. While this may have some effect on performance between each model, in both datasets only the feet are obscured. This offers VNect identical data so can possibly be disregarded as a factor in difference between model scores.

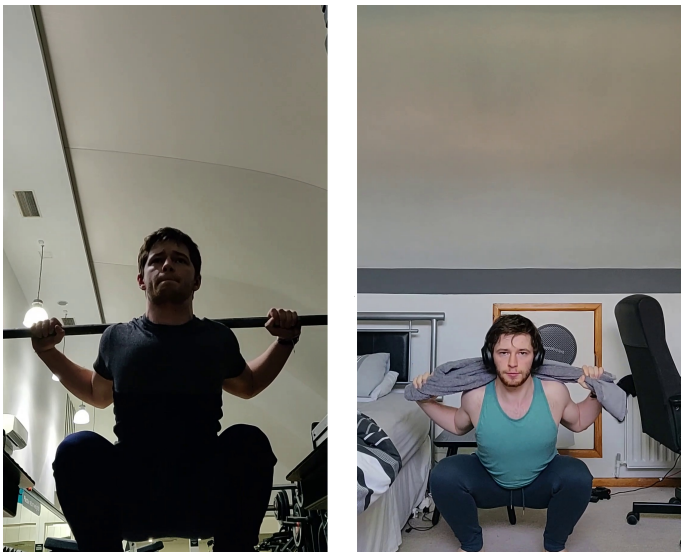

FIGURE 1.24: A sample frame from the weighted squat dataset and bodyweight squat dataset

Covid-19 crisis resulted in the gym used to collect data being shut, leading to a very small sample size of 60 repetitions to split into training and testing. A tiny test set of only 14 videos was created whilst the bodyweight squat dataset used 40 videos for the test set. It is unclear how much of an impact the amount of footage available has on the score.

Whilst there is a roughly 40% increase from weighted to bodyweight squat, the lower score being above 60% indicates that in the worst case it is feasible to train machine learning models for this task. With an upper end score close to 90% in this specific use case, this methodology can provide accurate representations of human movement where further information can be extrapolated from machine learning systems.

1.6.3.2 Overhead press movement

The overhead press dataset was filmed in the same circumstances as the bodyweight squat with ideal lighting and camera position.
With an accuracy of 80%, the model is able to accurately classify an overhead press movement. The confusion matrix reveals that this is a lot more accurate at predicting incorrect form and has a habit of incorrectly classifying correct form. This may be due to how similar the body is between correct and incorrect form, with only arms and centre of mass revealing a key difference. Any small variation between the arms during the performance of the movement will therefore lead to that repetition being classed as incorrect.

1.6.3.3 Overall

Under the correct circumstances this can be a powerful tool to aid in fitness, with the weighted squat results showing how if the circumstances are not ideal the tool is still relatively accurate on the correctness of a repetition.
This method of compacting entire movements into a time series of joint angles can accurately display and classify between types of movements, and more specifically different ways a movement can be performed. Converting movement into joint angles may prove to be an interesting method of performing other types of analysis on human movements due to the fidelity and completeness of it.

While not examined as part of the success criteria, time for execution is a significant factor in how well a solution performs. When including the Video to Time Series Converter this is not particularly quick, mostly due to the process of converting a video to image frames and pose estimation.

	Classification (s/video)	Training (s/60 videos)
Entire System	12.48	538.0
TSAnalysis	2.91	2.08

FIGURE 1.25: Measuring the time to complete operations

If a video has already been processed by pose estimation, it can simply be processed by TSAnalysis to significantly increase speed of execution. This speed increase can be seen clearly in Figure 1.25.

This shows a clear direction for improvement, by increasing the efficiency of Video to Time Series Converter would significantly improve portability of this system. This could be achieved through many methods such as changing which pose estimation library is used or by finding a way within VNect to keep a single network initialised rather than re-initialising every time.

The difference between timing for the entire system and TSAnalysis shows time to convert a single video (9.57s), much of this time is comprised of initialising the neural network of VNect. If this initialisation did not need to happen every time, time to convert a single file could drop significantly over large batches.

1.7 Conclusion

1.7.1 Successes of the System

The produced system completes all requirements set out in the design section, being able to input both a directory of training data or a single video file allows it to both train models and classify an input video based on a trained model.

In the preparation of joint angle time series for each video, four goals set out within the scope of the project are completed. During conversion of an input video into a time series of joint angles (Goal 4), the Video to Time Series Converter uses pose estimation tools to recognise a human in video input (Goal 1) and convert this into a skeletal model (Goal 2). Time Series Generator will then convert this skeletal model into a time series of joint angles (Goal 3) which completes this first stage.

The results prove not just the accuracy of the system, but ability to cope with many different movements while maintaining accuracy. With ideal data (lighting, camera angle, etc) the accuracy was tested to be between 80-87.5%, a monumental success as specified in the scope (Goal 5). Even with testing of datasets that provide sub-par conditions work with an accuracy measured at 64.29%, above random chance and within margins of a definite success. Accuracy testing also proves the system is general purpose enough to train and accurately classify multiple different movements (Goal 6). Three different scenarios and two different movement styles used in accuracy testing show clearly that accurate models can be produced for many different movements.

The system therefore completes all main goals set in the design scope, and expands to complete one additional goal.

The general purpose ability combined with demonstrated accuracy of this method shows promise for this type of problem, especially when applied to real world problems such as within fitness in places such as the gym as stated in the project introduction or other scenarios like track events or gymnastics.

1.7.2 Improvements

Whilst goals 7 and 8 were optional and designed to only further the projects main aims, they would have provided significant improvement in creating a tool for the problem statement in the introduction.

By being able to input an entire set of a movement (Goal 7), a much more fitness focused tool could be produced that would be more suited to in-depth analysis of movements. Joint angle time series analysis featured in 1.4.2.3 gives clues as to how this could be achieved. A single repetition would tend to follow a repetitive pattern so could be individually analysed by the machine learning component. This opens up other avenues for analysis such as rep counting and showing the percentage of a set performed correctly as well as being able to single out any poor reps from within a set.

Using real-time video (Goal 8) would allow this style of system to be used extensively in the fitness format that it was designed for, both by being built into gym equipment or used on portable equipment such as smartphones. This would require a fundamental reworking of how learning and classification is performed as currently the system compares a complete time series with the time series' that the model is trained with, which is not possible with a constant stream.

Adding a GUI would be a monumental step in improving user friendliness. Currently the system is operated by command line and involves two separate programs that must be within the same directory to function correctly. By combining all these parts into a single package with a GUI and either goals 7 or 8 would create a fully operational project that could conceivably see consumer use.

1.7.3 Further Work

Different approaches could be explored for this problem to improve accuracy. By changing the style of angles used from relative angle through a plane of a joint to absolute angle with each dimension in 3D space could increase the fidelity of body movements. Exploring different styles of learning on this type of data including deep learning or others mentioned in literature may be more suited to this type of problem. If large enough datasets were able to be gathered deep learning could be utilised to learn more useful and subtle patterns within a time series.

By breaking down this method of time series analysis could lead to exciting opportunities in multiple different fields.
Use of joint angles and time series to track human movement yields lots of areas for analysis, from professional sporting to general aid and mobility tools. By using finer fidelity tools to capture this data such as motion capture or depth camera hardware could propel the accuracy of these styles of joint analysis beyond the off the shelf tools this project was designed around.
This style of learning could be engineered into a training tool by using idealised joint angle time series' that an athlete could optimise their performance by replicating. This idealised joint angle time series could be derived from many places, such as record performances or custom generated in order to utilise as much performance from an athlete as possible. The learning style used within this system could be altered for this purpose to show a closeness to this ideal joint angle time series and give in depth analysis on a per joint basis.

1.8 Project Management

1.8.1 Changes throughout Project

1.8.1.1 Project Brief to Interim Report

As the complexity of the system grew, classifying multiple repetitions of a movement seemed extravagant compared to the main goals of the project of proving the ability for the system to be built in a generic framework. This is currently an extension of the current goals, analysing the period of joint movement may be used to break a set into multiple repetitions that could be individually classified.

1.8.1.2 Interim Report to Final Report

Some parts of the proposed design were not required for the functionality of Video to Time Series Converter. This includes the Video and Time Series Cleanup modules.
Video Cleanup module was not necessary due to the limited sizes of single repetition videos featured in this dataset. Extreme compression from video to time series afforded is not affected by how large the input video was at the start of the system, so scaling the resolution or crop of any video would simply result in lost data that provides no speed boost in the estimation stages of this system.
Time Series Cleanup did not need to be featured as it's own module and was instead incorporated directly into the Time Series Generator, this was so that data would not have to be modelled and passed through the system and the csv could instead be built linearly as the Estimation to Time Series module processed the pose estimation data per each frame.
It was realised that TensorFlow would not be as useful as initially thought, resulting in using a combination of pandas and TSLearn within the ML module. This change was implemented due to the native support for time series within these libraries and supporting functionality including dynamic time-warping.

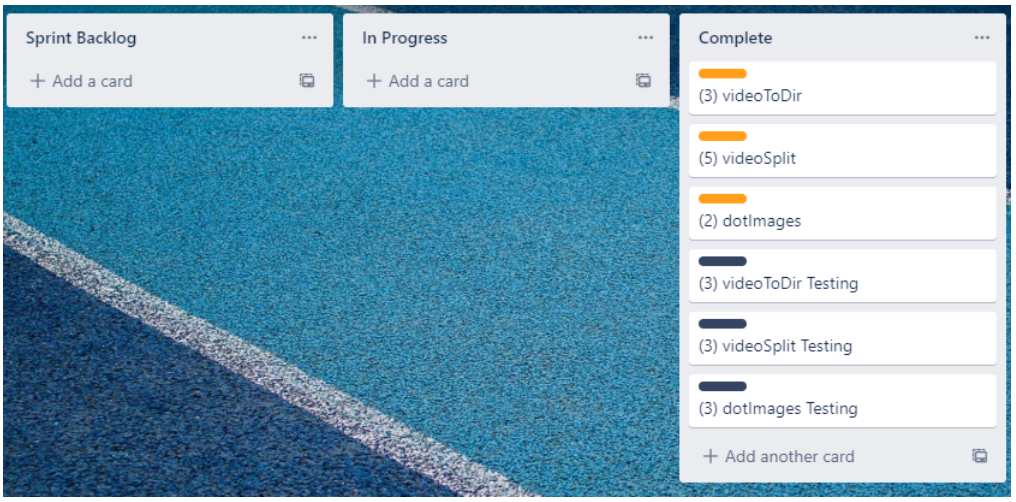

FIGURE 1.26: Trello board of VideoToDir sprint

1.8.2 Development Process

1.8.2.1 AGILE processes

The modular design of the project naturally gave rise to using a scrum methodology for development. By breaking every file into the focus of a separate sprint allowed for a short sprint duration of a week to focus on the development and testing of each file within the system.

Each file necessary within the system gets its own sprint and trello board, this is used to track each story covering both the development and testing of this module. To estimate effort of stories the fibonacci sequence (1,2,3,5,8...) was used to accurately map the many levels of lower efforts while having high efforts available if necessary.

1.8.3 Time Management

Unconventionally the entire project was also run as a single sprint, development of each file were ranked in difficulty as well as other tasks in such as data collection and report writing. This allows for all parallel running parts of the project placed as in progress, modelling the work structure more accurately. Tasks that involve this type of management include report writing and collection of datasets.

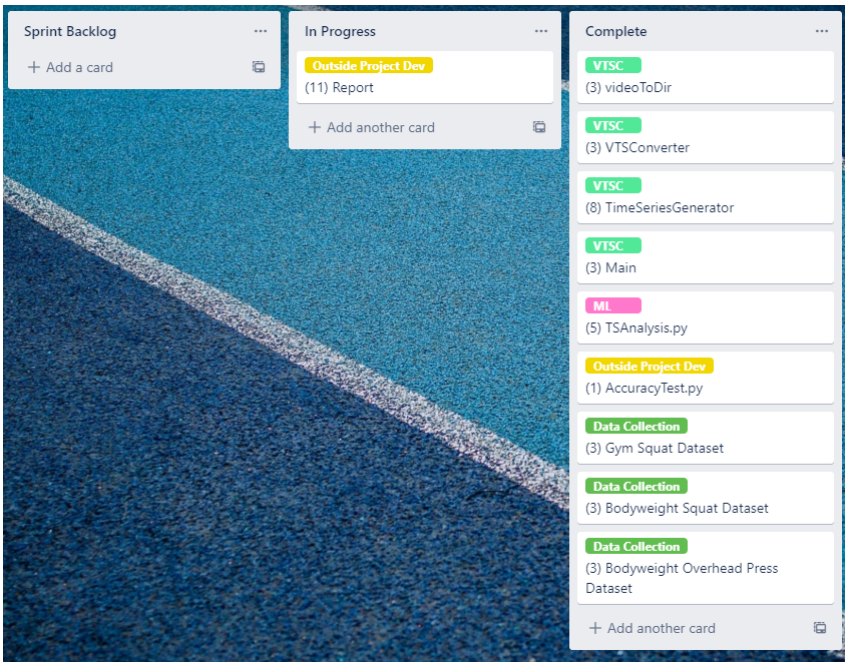

FIGURE 1.27: Trello board of project

1.8.3.1 Gantt Charts

Gantt Charts were used to keep the project on a fluid schedule and to monitor the projects progress against expected progress. This analysis allows for any delays in the project to be accounted for well in advance of any deadlines.

The original gantt chart (Figure 1.28) used is from the interim report, much of the planning that went into this chart was roughly based on perceived effort of each piece and was conducted before much of the finalised designs were in place.

As the project gained a much more rigid design, it was clear the gantt chart that accompanied this original design needed to be altered in order to accommodate both altering timing of components within the project as well as a change in planned units of work. A new Gantt Chat was created (Figure 1.29) with many of these changes implemented. The development phase of this plan includes the scrum methodology for each component of the design, along with testing included in these scrum weeks. It was also recognised that more time needed to be given to creation of the final report. Prioritisation of the project was altered to lessen the priority of a second model over report writing. In case of any delays with the project there were several slack points allocated to mitigate any risk of lateness, one slack point was dedicated to development of the system and another was dedicated to any other delays that may occur. As background and reading phases

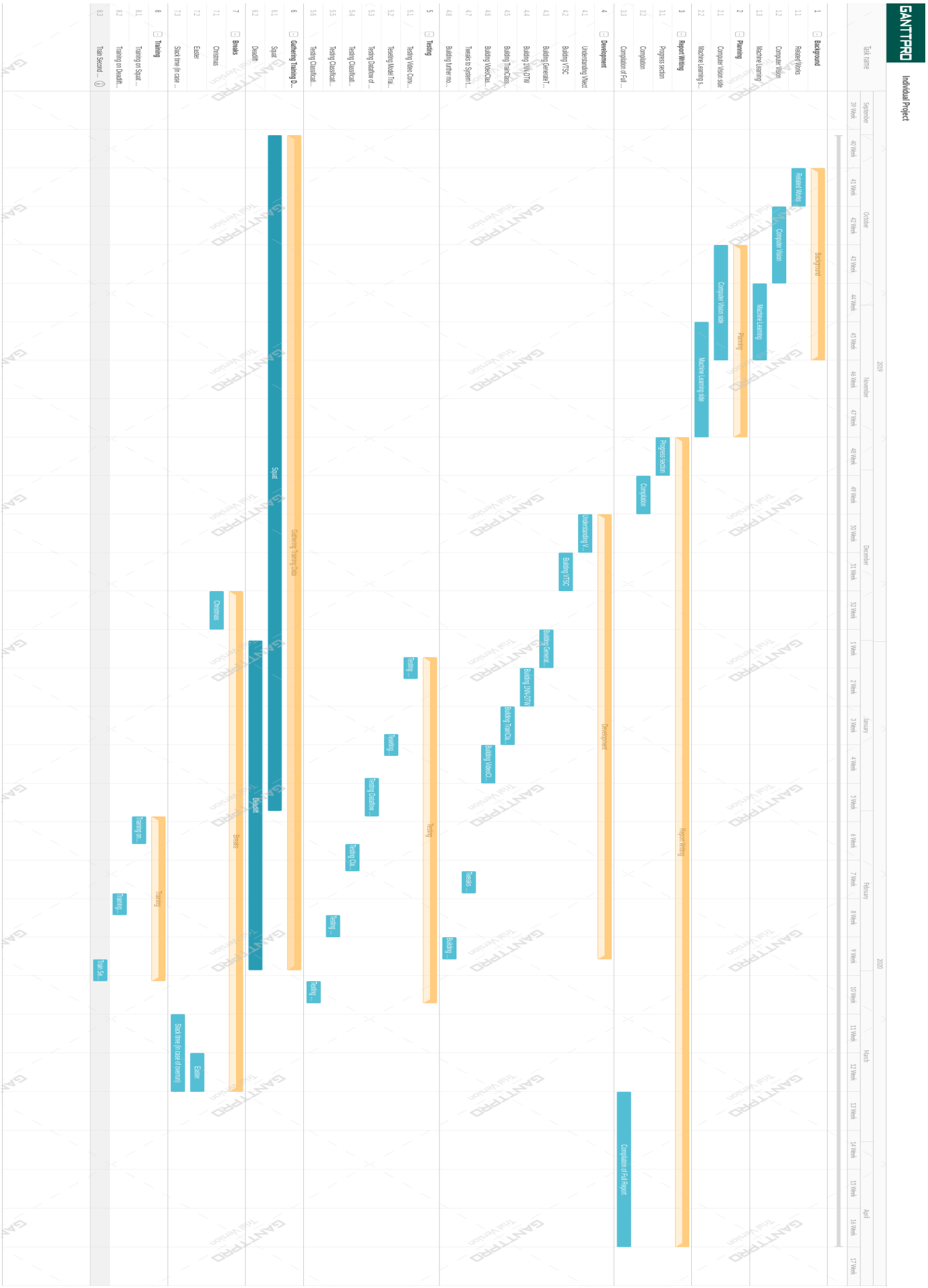

FIGURE 1.28: Original Gantt Chart

of the project had been completed on construction of the gantt chart these sections are minimised.

The actual timetable (Figure 1.30) of this project ended up very different to the original plan set. Much of the project initially was on track with the plan, with all software development targets hit on schedule. The only major setback in initial phases of the project was starting of collecting footage for the weighted squat dataset. Collection of weighted squat data was delayed due to the unavailability of suitable squat racks in the gym as well as taking an initial set time to find an optimal recording setup to collect this data. In the gym used there was only one squat rack that offered the distance and lighting to allow a subject to wholly appear in frame and moderately well lit.
However, this project was majorly affected by the coronavirus pandemic hitting the UK due to the necessity of gym access to gather data. It was necessary to take time and pause the project in order consider plans on how to continue with this critical lack of data, as about half the squat dataset had been collected and none of the deadlift dataset had been collected.

1.8.4 Risk Management

Using $Criticality = Severity \cdot Probability$, several risks have been outlined alongside mitigation for each scenario. This table is located in Appendix 2.4.

Risks 1 and 2 were encountered during the project, risk 2 was mitigated by extensive testing shown in the report to ensure that all logic is tested and any bugs can be fixed.

Risk 1 was more complicated to deal with as the mitigation method was unable to occur.
Unforeseen in risk management of this project was the coronavirus pandemic. Closure of public spaces and restrictions on movement has significantly affected the ability to collect datasets from the gym, with less than half of the weighted squat dataset collected and none of the deadlift dataset collected.

In an effort to still have quality data for testing, but also seizing an opportunity to test the ability of the system to cope with many different types of movement, two full size (100 videos) datasets were collected from home.

One of these datasets was the bodyweight squat, this was chosen to mimic the movement pattern involved with weighted squats so should represent the weighted

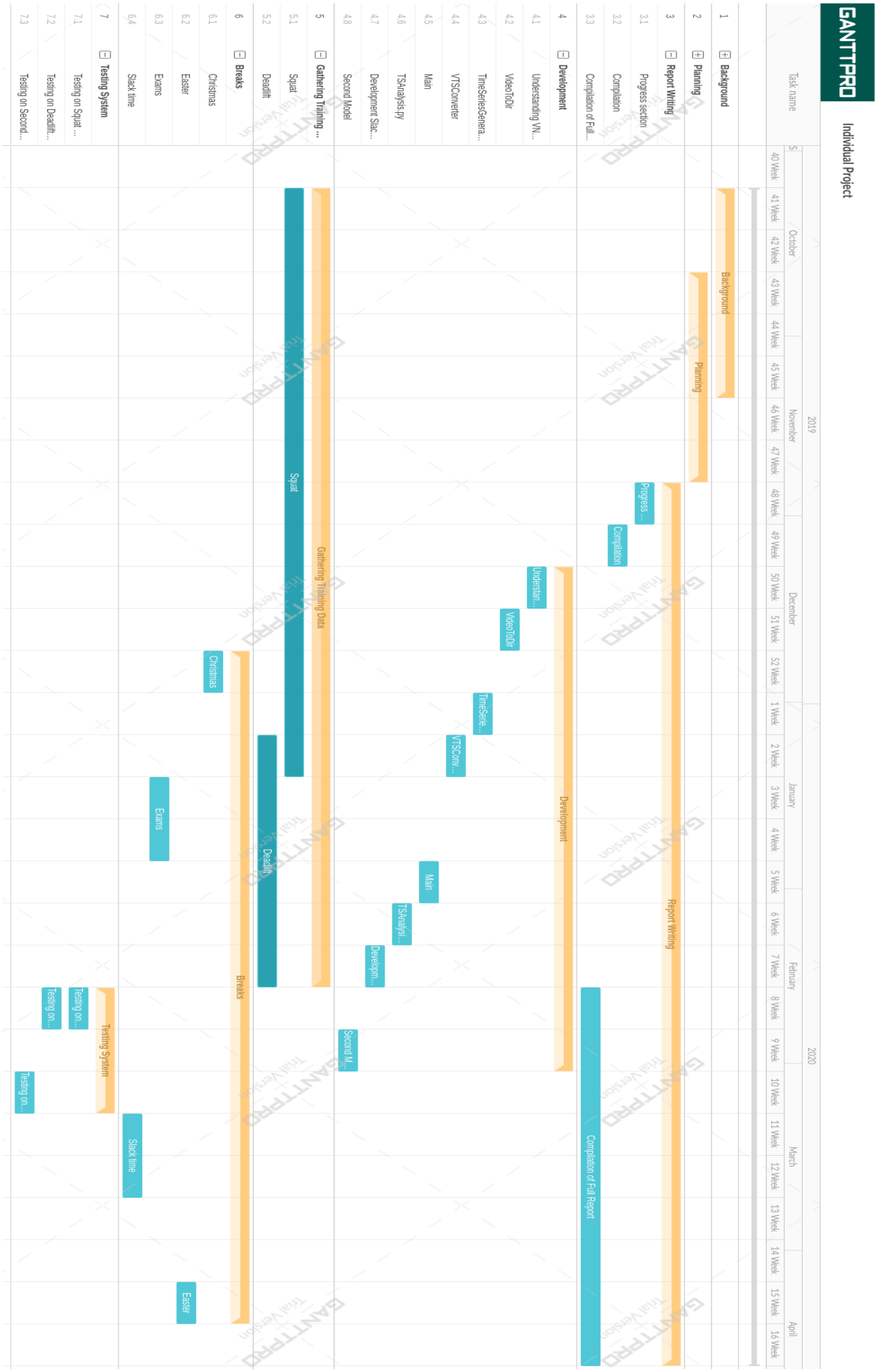

FIGURE 1.29: Modified Gantt Chart

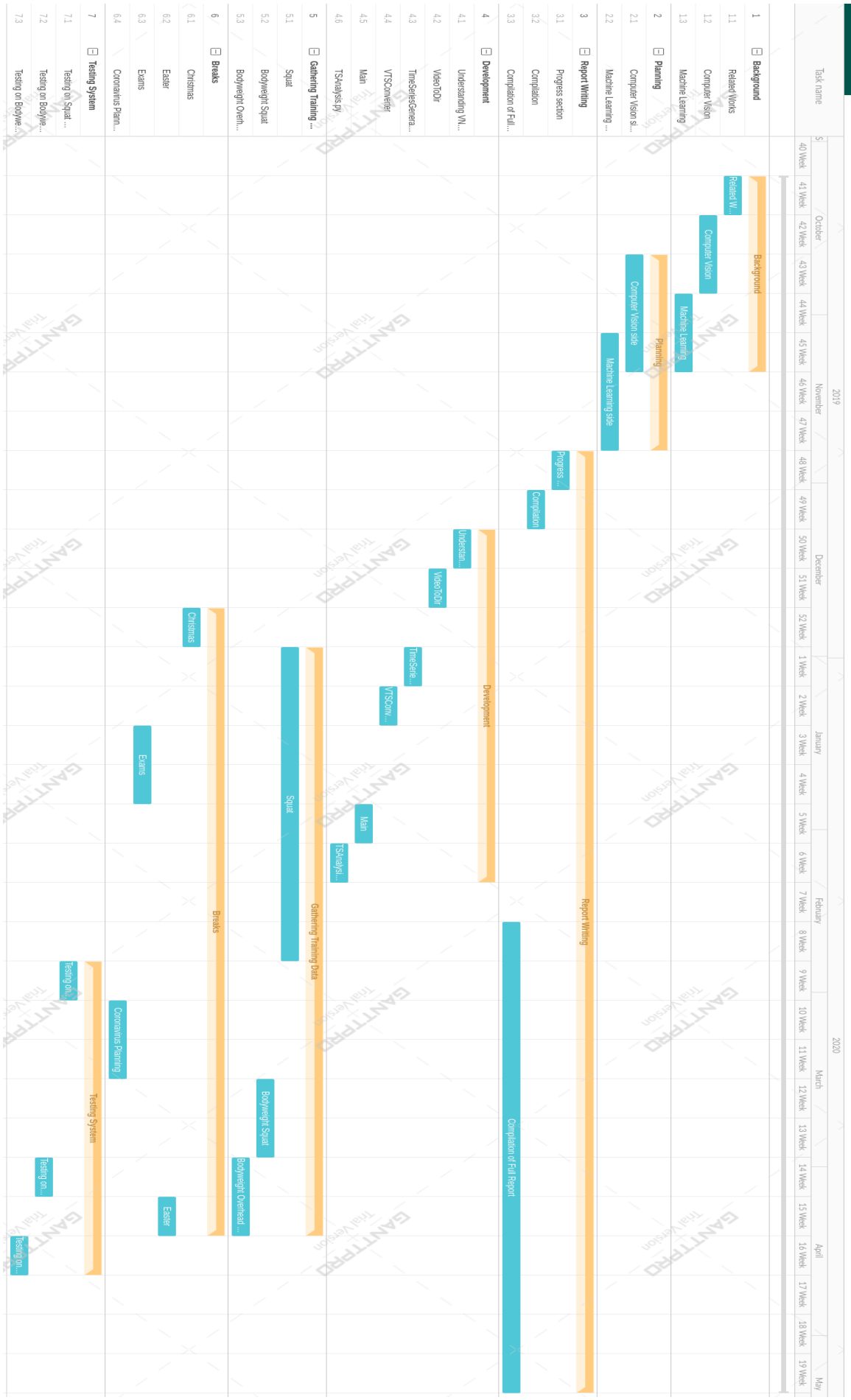

FIGURE 1.30: Actual project timeline

squat dataset if it could have been fully collected. Instead of performing these squats as a typical bodyweight squat these squats were performed with a simulated barbell in order to replicate the movement pattern of a weighted squat.

The other dataset was an overhead press using a makeshift barbell. The weight able to be loaded on such a makeshift bar could not replicate the weight necessary to feel strain on a deadlift, so overhead press was chosen instead as it involves lighter weights. Using a small weight with overhead press on a makeshift barbell would lead to a higher proportion of a maximum effort lift compared to a deadlift, resulting in higher likelihood of bad form. The overhead press also allows an opportunity to test a much finer threshold of correctness as well as focusing the system on the upper body rather than the primarily lower body focus in the squat. These are both due to how little action there is within the rest of the body during movement, the system needs to recognise that minor deviations within the legs for balance shifting do not affect form as much as action of the arms during movement.

Bibliography

[1] Avinash Navlani. Knn classification using scikit-learn, 2018.

[2] Jeremy Zhang. Dynamic time warping - explanation and code implementation, 2020.

[3] Ognjen Arandjelović. Computer-aided parameter selection for resistance exercise using machine vision-based capability profile estimation. *Augmented Human Research*, 2(1):4, Jul 2017.

[4] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A. Efros. Everybody dance now. *CoRR*, abs/1808.07371, 2018.

[5] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018.

[6] Yuliang Xiu, Jiefeng Li, Haoyu Wang, Yinghong Fang, and Cewu Lu. Pose flow: Efficient online pose tracking. *CoRR*, abs/1802.00977, 2018.

[7] Dushyant Mehta, Srinath Sridhar, Oleksandr Sotnychenko, Helge Rhodin, Mohammad Shafiei, Hans-Peter Seidel, Weipeng Xu, Dan Casas, and Christian Theobalt. Vnect: Real-time 3d human pose estimation with a single RGB camera. *CoRR*, abs/1705.01583, 2017.

[8] Federica Bogo, Angjoo Kanazawa, Christoph Lassner, Peter V. Gehler, Javier Romero, and Michael J. Black. Keep it SMPL: automatic estimation of 3d human pose and shape from a single image. *CoRR*, abs/1607.08128, 2016.

[9] Kohei Arai and Rosa Andrie Asmara. 3d skeleton model derived from kinect depth sensor camera and its application to walking style quality evaluations. *International Journal of Advanced Research in Artificial Intelligence*, 2(7), 2013.

[10] B. Yin, D. Zhang, S. Li, A. Hao, and H. Qin. Context-aware network for 3d human pose estimation from monocular rgb image. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2019.

[11] Yu Cheng, Bo Yang, Bo Wang, Wending Yan, and Robby T. Tan. Occlusion-aware networks for 3d human pose estimation in video. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[12] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[13] Feng Mao, Xiang Wu, Hui Xue, and Rong Zhang. Hierarchical video frame sequence representation with deep convolutional graph network. In *The European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[14] John Cristian Borges Gamboa. Deep learning for time-series analysis. *CoRR*, abs/1701.01887, 2017.

[15] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. Fast time series classification using numerosity reduction. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 1033–1040, New York, NY, USA, 2006. ACM.

[16] Yaniv Shulman. Dynamic time warp convolutional networks, 2019.

[17] Michael Moor, Max Horn, Bastian Rieck, Damian Roqueiro, and Karsten Borgwardt. Early recognition of sepsis with gaussian process temporal convolutional networks and dynamic time warping. In Finale Doshi-Velez, Jim Fackler, Ken Jung, David Kale, Rajesh Ranganath, Byron Wallace, and Jenna Wiens, editors, *Proceedings of the 4th Machine Learning for Healthcare Conference*, volume 106 of *Proceedings of Machine Learning Research*, pages 2–26, Ann Arbor, Michigan, 09–10 Aug 2019. PMLR.

[18] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. LSTM fully convolutional networks for time series classification. *CoRR*, abs/1709.05206, 2017.

[19] Anthony J. Bagnall, Aaron Bostrom, James Large, and Jason Lines. The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version. *CoRR*, abs/1602.01711, 2016.

[20] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, Mar 2005.

[21] Abdullah Mueen and Eamonn Keogh. Extracting optimal performance from dynamic time warping. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2129–2130, New York, NY, USA, 2016. ACM.

[22] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[23] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, and Eli Woods. tslearn: A machine learning toolkit dedicated to time-series data, 2017. https://github.com/tslearn-team/tslearn.

[24] Pavel Senin. Dynamic time warping algorithm review. 01 2009.

[25] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2018.

[26] Yutaka Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 01 2007.

2. Appendix

2.1 Requirements Table

Requirement	Description
Provide functionality for converting video into usable training data.	Enable a user to convert a library of labelled training videos into a directory suitable for training a model on.
Provide functionality for training new models.	Enable a user to train a model of a given type (defaulting to 1NN-DTW) if provided with a directory of suitable Time Series training data.
Allow classification from different trained models.	Allow the classification function to run from a provided model by the user, this model must be able to be changed between executions.
Classify input video as adhering to a movement or not.	Allow a user to input a video and a model and receive a classification if the provided video adheres to the movement pattern the model is trained for.

2.2 Testing

2.2.1 VideoToDir.cpp

Test	Purpose	Expected	Result	Action

videoSplit _Correct	Ensures splitting of video into frame images works with correct arguments	imageSequence folder produced	PASS	N/A
videoSplit _DirConflict	Ensures creation of frame images if given directory is already present	imageSequence folder produced	PASS	N/A
videoSplit _OutputFileExists	Ensures creation of imageSequence directory is already present	imageSequence folder produced	PASS	N/A
videoSplit _VideoNonexist	Checks error is thrown if video does not exist	Error thrown and execution abandoned	PASS	N/A
dotImages _Correct	Ensures creation of accurate config file with correct arguments	image config file produced	PASS	N/A
dotImages _DirNotExist	Checks .images file created if given directory does not exist	image config file produced	FAIL	See 1
dotImages _OutputFileExists	Checks .images file created if given directory already contains .images file	image config file produced	PASS	N/A
videoToDir _Correct	Ensures creation of valid directory of frames and config correct arguments	Full video directory produced	PASS	N/A
videoToDir _DirAlreadyPresent	Ensures video directory is produced if a folder of the same name is pesent	Video directory produced	FAIL	See 2
videoToDir _VideoNonexist	Ensures an error is thrown if there is no video to create a directory from	Error message thrown and execution aborted	PASS	N/A
videoToDir _dotImageExists	Ensures a folder will always populate the dotImage in case that only the image sequence is present	dotImage file produced for directory	FAIL	See 3

videoToDir _imageSequenceExists	Ensures a folder will always populate the image sequence in case that only the dotImage is present	image sequence produced for directory	FAIL	See 4

2.2.2 TimeSeriesGenerator.cpp

Test	Purpose	Expected	Result	Action
addEdge _Correct	Ensures adding edge to body model performs correctly	Each part has other listed in connections	PASS	N/A
addEdge _Empty	Checks adding edge works with larger arrays	Two parts connected and one empty	PASS	N/A
addEdge _OutOfBounds	Shows trying to add connection to invalid part produces error	Error displayed	FAIL	See 1
buildConnections	Builds body connection model as described in implementation	Connections built models body	PASS	N/A
split _csv	Checks split works on a simple string with comma delimiter	String split and returned as array	PASS	N/A
split _diffDelimiter	Checks split functions as expected with different delimiters	String split and returned as array	PASS	N/A

split _noSplits	Ensures a string containing no delimiters will not be split	String returned is same as passed in	PASS	N/A
split _multipleSplits	Checks split works on a string with multiple splits to be made	String split and returned as array	PASS	N/A
calcAngle _0	Checks calculating a 0 angle returns 0 instead of NaN	0 returned	FAIL	See 2
calcAngle _90	Checks calculating a 90 angle returns 90	90 returned	PASS	N/A
calcAngle _180	Checks calculating a 180 angle returns 180	180 returned	PASS	N/A
calcAngle _misc	Checks calculating a misc angle returns that angle	Misc angle (66.77) returned returned	PASS	N/A
calcBetweenParts _0	Checks calculating a 0 angle returns 0 instead of NaN	0 returned	FAIL	See 3
calcBetweenParts _90	Checks calculating a 90 angle returns 90	90 returned	PASS	N/A
calcBetweenParts _180	Checks calculating a 180 angle returns 180	180 returned	PASS	N/A
calcBetweenParts _misc	Checks calculating a misc angle returns that angle	Misc angle (66.77) returned	PASS	N/A
calcBetweenParts _rootTranslation	Ensures both vectors are translated by the root vector correctly, misc angle calculated correctly	Misc angle (66.77) returned	PASS	N/A
parsePartValues _simple	Checks parsing for partValue data works correctly on simple model	Each part has correct vectors for input string	PASS	N/A

parsePartValues _realData	Checks parsing for partValue data works correctly on real data	Each part has correct vectors for input string	PASS	N/A
parsePartValues _overfill	If too many values supplied for parts and dimensions exception should be thrown	Exception thrown	PASS	N/A
parsePartValues _underfill	If too little values supplied for parts and dimensions exception should be thrown	Exception thrown	PASS	N/A
anglesAround ConnectedParts _1connections	Ensures connections with only one part ignored	No value returned	PASS	N/A
anglesAround ConnectedParts _2connections	Ensures connections with two parts return single value	One value returned	PASS	N/A
anglesAround ConnectedParts _3connections	Ensures connections with three parts return two values (analagous of spine joint)	Two values returned	PASS	N/A
anglesAround ConnectedParts _4connections	Ensures connections with four parts return two values (horizontal and vertical component)	Two values returned	PASS	N/A
anglesAround ConnectedParts _real	Checks real values output correct results	Correct values returned compared to calculation	PASS	N/A
estimationTo TimeSeries _real	Checks jointAngle file produced and is accurate for real data	jointAngle csv produced and identical to calculated values	PASS	N/A

estimationTo TimeSeries _inputNonexist	Ensures error thrown if input file doesn't exist	Error thrown	PASS	N/A
estimationTo TimeSeries _outfileExists	Ensures calculation occurs if output file already exists	Output file reproduced	PASS	N/A

2.2.3 VTSConverter.cpp

Test	Purpose	Expected	Result	Action
convert _Correct	Ensures conversion works when single video given	jointAngle csv file produced	PASS	N/A
convert _VidNonexist	Ensures execution is halted if video cannot be found	Error thrown	PASS	N/A
convert _alreadyConverted	Makes sure steps are skipped if the given file is already converted	No change to directory, execution skipped	PASS	N/A
convert _alreadyFrames	Ensures frames are compiled if not present in video directory	Video to directory execution	PASS	N/A
convert _alreadyJoint Positions	Ensures pose estimation occurs if no csv is found	Pose estimation execution	PASS	N/A
directoryConvert _Correct	Ensures conversion works when directory given	jointAngle csv file produced for every file in directory	PASS	N/A

directoryConvert _noMP4	No conversion should be attempted if no matching files are found	No conversion attempted	PASS	N/A
directoryConvert _EmptyDir	No conversion should be attempted if no matching files are found	No conversion attempted	PASS	N/A
directoryConvert _DirNonexist	Error should be thrown if directory does not exist	Error thrown	PASS	N/A

2.2.4 Video to Time Series Converter Integration

Test	Purpose	Expected	Result	Action
FileClassification Correct	Ensures conversion works when single video given	jointAngle csv file produced	PASS	N/A
FileClassification VidNonexist	Ensures execution is halted if video cannot be found	Error thrown	PASS	N/A
FileClassification TooManyArgs	Ensures execution is halted if incorrect arguments given	Execution halted	PASS	N/A
FileClassification Notmp4	Ensures execution is halted if input file is incorrect	Execution halted	FAIL	See 1
DirectoryTraining Correct	Ensures conversion works when directory given	jointAngle csv file produced for every file in directory	PASS	N/A
DirectoryTraining dirNonexist	Execution should be halted if directory cannot be found	Execution halted	PASS	N/A
DirectoryTraining TooManyArgs	Ensures execution is halted if incorrect arguments given	Execution halted	PASS	N/A

2.2.5 TSAnalysis.py

Test	Purpose	Expected	Result	Action
train _Correct	Ensures training works when correct arguments given	.model file produced	PASS	N/A
predict _Correct	Ensures classification works when correct arguments given	Prediction feedback received	PASS	N/A
dirToDataFrames _Correct	Ensures dirToDataFrames works when correct arguments given, including invalid files within given directory	List of dataframes and labels returned	PASS	N/A
dirToDataFrames _NonExist	Ensures dirToDataFrames returns empty list when nonexistant directory given	Empty list returned	PASS	N/A
dirToDataFrames _DirContainsNoFiles	Ensures dirToDataFrames returns empty list when no files in directory are valid	Empty list returned	PASS	N/A
getCSV _exists	Ensures getCSV returns joint angles csv when valid directory given	CSV file location returned	PASS	N/A
getCSV _nonexists	Ensures getCSV returns None when invalid directory/argument given	None returned	PASS	N/A
isModel _ResultTrue	Ensures isModel returns True when model file given	True returned	PASS	N/A
isModel _DirectoryFalse	Ensures isModel returns False when directory given	False returned	PASS	N/A
isModel _EndsWithFalse	Ensures isModel returns False when wrong file type given	False returned	PASS	N/A
isModel _FileNonexist	Ensures isModel returns False when file does not exist	False returned	PASS	N/A

2.2.6 Machine Learning Module Integration

Test	Purpose	Expected	Result	Action
Training - correct arguments	Ensures training works when correct arguments given	.model file produced	PASS	N/A
Training - first argument single video dir	Ensures execution blocked when invalid folder structure given	Error thrown	PASS	N/A
Training - second argument model file	Ensures execution blocked when second argument exists	Error thrown	PASS	N/A
Classification - correct arguments	Ensures training works when correct arguments given	Prediction feedback received	PASS	N/A
Classification - first argument not video dir	Ensures execution blocked when invalid folder structure given	Error thrown	PASS	N/A
Classification - second argument not model file	Ensures execution blocked when model files given	Error thrown	PASS	N/A
Neither argument exists	Ensures execution blocked when no viable path	Error thrown	PASS	N/A
<2 arguments	Ensures training works when correct arguments given	Error thrown	PASS	N/A
>2 arguments	Ensures training works when correct arguments given	Error thrown	PASS	N/A

2.2.7 Entire System

Test	Purpose	Expected	Result	Action
FileClassification Correct	Ensures conversion works when single video given	jointAngle csv file produced	PASS	N/A
FileClassification VidNonexist	Ensures execution is halted if video cannot be found	Error thrown	PASS	N/A

FileClassification ModelNonExist	Ensures execution is halted if model cannot be found	Error thrown	PASS	N/A
FileClassification TooManyArgs	Ensures execution is halted if incorrect arguments given	Execution halted	PASS	N/A
FileClassification Notmp4	Ensures execution is halted if input file is incorrect	Execution halted	FAIL	See 1
DirectoryTraining Correct	Ensures conversion works when directory given	jointAngle csv file produced for every file in directory	PASS	N/A
DirectoryTraining dirNonexist	Execution should be halted if directory cannot be found	Execution halted	PASS	N/A
DirectoryTraining ModelFilePassed	Ensures execution is halted if incorrect arguments given	Execution halted	PASS	N/A
DirectoryTraining TooManyArgs	Ensures execution is halted if incorrect arguments given	Execution halted	PASS	N/A
<2 arguments	Ensures execution halted when incorrect arguments given	Error thrown	PASS	N/A
>2 arguments	Ensures execution halted when incorrect arguments given	Error thrown	PASS	N/A

2.3 Raw Results

2.3.1 Weighted Squat

```
FILE                          | SYS PREDICTION        | SYS CORRECT
TestingVids/TEST_C_001.mp4    | CORRECT               | CORRECT
TestingVids/TEST_C_002.mp4    | CORRECT               | CORRECT
TestingVids/TEST_C_003.mp4    | INCORRECT             | INCORRECT
TestingVids/TEST_C_004.mp4    | INCORRECT             | INCORRECT
TestingVids/TEST_C_005.mp4    | INCORRECT             | INCORRECT
TestingVids/TEST_C_006.mp4    | CORRECT               | CORRECT
TestingVids/TEST_C_007.mp4    | CORRECT               | CORRECT
TestingVids/TEST_I_001.mp4    | INCORRECT             | CORRECT
TestingVids/TEST_I_002.mp4    | CORRECT               | INCORRECT
```

TestingVids/TEST_I_003.mp4	INCORRECT	CORRECT
TestingVids/TEST_I_004.mp4	INCORRECT	CORRECT
TestingVids/TEST_I_005.mp4	CORRECT	INCORRECT
TestingVids/TEST_I_006.mp4	INCORRECT	CORRECT
TestingVids/TEST_I_007.mp4	INCORRECT	CORRECT

Correct: 9 Wrong: 5 Total: 14 Accuracy: 64.29% Time per classify: 22.66

2.3.2 Bodyweight Squat

FILE	SYS PREDICTION	SYS CORRECT
BWTestingVids/SQT_C_004_001.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_002.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_003.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_004.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_005.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_006.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_007.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_008.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_009.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_010.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_011.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_012.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_013.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_014.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_015.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_016.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_017.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_018.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_019.mp4	CORRECT	CORRECT
BWTestingVids/SQT_C_004_020.mp4	CORRECT	CORRECT
BWTestingVids/SQT_I_001_001_LEFTLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_001_002_RIGHTLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_001_003_FORWARDLEAN.mp4	CORRECT	INCORRECT
BWTestingVids/SQT_I_001_004_BACKLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_001_005_BALLFOOT.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_002_001_LEFTLEAN.mp4	CORRECT	INCORRECT
BWTestingVids/SQT_I_002_002_RIGHTLEAN.mp4	CORRECT	INCORRECT
BWTestingVids/SQT_I_002_003_FORWARDLEAN.mp4	CORRECT	INCORRECT
BWTestingVids/SQT_I_002_004_BACKLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_002_005_BALLFOOT.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_002_006_HEELFOOT.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_001_BARLEFT.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_002_BARRIGHT.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_003_LEFTLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_004_RIGHTLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_005_FORWARDLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_006_BACKLEANmp4.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_007_LEFTLEAN.mp4	INCORRECT	CORRECT
BWTestingVids/SQT_I_006_008_RIGHTLEAN.mp4	CORRECT	INCORRECT
BWTestingVids/SQT_I_006_009_BALLFOOT.mp4	INCORRECT	CORRECT

Correct: 35 Wrong: 5 Total: 40 Accuracy: 87.5% Time per classify: 12.48

2.3.3 Overhead Press

FILE	SYS PREDICTION	SYS CORRECT
OHPVids/TestingVids/OHP_C_003_001.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_002.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_003.mp4	INCORRECT	INCORRECT
OHPVids/TestingVids/OHP_C_003_004.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_005.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_006.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_007.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_008.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_009.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_010.mp4	INCORRECT	INCORRECT
OHPVids/TestingVids/OHP_C_003_011.mp4	INCORRECT	INCORRECT
OHPVids/TestingVids/OHP_C_003_012.mp4	INCORRECT	INCORRECT
OHPVids/TestingVids/OHP_C_003_013.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_014.mp4	INCORRECT	INCORRECT
OHPVids/TestingVids/OHP_C_003_015.mp4	INCORRECT	INCORRECT
OHPVids/TestingVids/OHP_C_003_016.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_017.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_018.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_019.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_C_003_020.mp4	CORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_001_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_002_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_005_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_006_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_009_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_010_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_013_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_002_014_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_001_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_002_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_003_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_004_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_005_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_006_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_007_LEFT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_005_008_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_IMBALANCE_004_LEFT.mp4	CORRECT	INCORRECT
OHPVids/TestingVids/OHP_I_IMBALANCE_005_LEFT.mp4	CORRECT	INCORRECT
OHPVids/TestingVids/OHP_I_IMBALANCE_009_RIGHT.mp4	INCORRECT	CORRECT
OHPVids/TestingVids/OHP_I_IMBALANCE_010_RIGHT.mp4	INCORRECT	CORRECT

Correct: 32 Wrong: 8 Total: 40 Accuracy: 80.0% Time per classify: 5.91s

2.4 Confusion Matrices

These are calculated from the data in Appendix 2.3.

2.4.1 Weighted Squat

n = 14	Predicted: CORRECT	Predicted: INCORRECT
Actual: CORRECT	4	3
Actual: INCORRECT	2	5

FIGURE 2.1: Weighted Squat Confusion Matrix

2.4.2 Bodyweight Squat

n = 40	Predicted: CORRECT	Predicted: INCORRECT
Actual: CORRECT	20	0
Actual: INCORRECT	5	15

FIGURE 2.2: Bodyweight Squat Confusion Matrix

2.4.3 Overhead Press

n = 40	Predicted: CORRECT	Predicted: INCORRECT
Actual: CORRECT	14	6
Actual: INCORRECT	2	18

FIGURE 2.3: Overhead Press Confusion Matrix

2.5 Risk Assessment

#	Risk	Severity	Probability	Criticality	Mitigation
1	Unable to gather large enough dataset	3	1	3	Dedicate a day to gym recording movements for dataset.
2	Bugs found in code of system	1	4	4	Thorough testing and dedicating test and bug fixing time.
3	Model unable to classify movement adequately	5	1	5	Use other model types or methods of classification discussed.
4	Falling behind in work plan	2	3	6	Plan laxly to allow for unforeseen circumstances and plan in overtime.
5	Unable to use VNect	4	2	8	Use another library researched (eg AlphaPose).

2.6 Original Project Brief

Title:
Tracking form to prevent injuries in resistance training using Artificial Intelligence.

Problem:
It is incredibly easy for someone to injure themselves during specific movements of resistance training (such as squats or deadlifts). This is primarily due to incorrect proper form during the movement. It is also reasonably difficult to try to check and correct form as this usually involves getting someone else to watch (personal trainer, friend, other gym-goers) or recording themselves performing the movement and reviewing using their own judgement which may be incorrect. By developing a system that could detect issues with form through a computer or mobile app the system may be able to prevent possibly very damaging injuries to occur. If this is the case it would allow people to be more confident with themselves during

training, encourage a safer sport and hopefully encourage more people to take up the sport.

Goals:

- Design a system able to pick out a human subject from a video in a gym setting
- Design a system able to recognise a single repetition of a movement
- Be able to track the human movement during the repetition
- Having the system be able to break down the human movement of the repetition into different body parts and how they are moving
- Develop a Machine Learning model that is able to recognise good or bad form of a given rep
- Be able to classify a tracked rep as good form or bad form
- (If successful) develop the system to use real-time video input rather than prerecorded video
- (If successful) expand the system to be able to recognise and accurately classify the form for more than one movement

I am hoping that if the system is successful in this task it will show an ability for many other similar problems with correcting form to be solved through the same methods, this could be expanded to other sports such as yoga or pilates where the form of a movement is key to the sport.

Scope:
As a proof of concept for the system I will be focusing on a single compound powerlifting movement due to how open to injury they are and the generality of the movements compared to isolation exercises. First I will be aiming to produce a system that can recognise good and bad form on either squats or deadlifts. Rather than immediately trying to solve the problem for real time video and providing immediate prediction I will be aiming to solve the problem for a prerecorded video that has been unseen by the system, from there I can use the same data gathered to train the system to provide predictions on real-time video.

2.7 Archive Contents

2.7.1 Compile

Executable files VTSC.exe and TSAnalysis.py
VTSC.exe will call on TSAnalysis.py after necessary videos have been converted.

Also included are a number of supplementary files, these are necessary for the pose estimation within VTSC.exe to work and must be in the same level as the executable.

2.7.2 Source

The C++ source files that compile to create VTSC.exe.

2.7.3 Data

Sample data to both train and classify with the system, partitioned into a train and classify folder.

These folders contain select data that is necessary to run these types of tasks, within train is a directory of video files as well as two more to classify after training. Within classify is a pre-trained model and two video files to classify.

2.7.4 Root

A readme is supplied with this archive explaining the contents of the archive and how to run the system.
AccuracyTesting.py was the testing script developed to aid testing, this may have variable results on other systems as it was hardcoded to the environment the project was developed with.

2.8 Word Count

According to Foxit Reader, the word count for the body of this report is 9,982.