Department of Computer Science

Submitted in part fulfilment for the degree of BEng

Predicting Video Reversibility Score with Mark Out Data

Thomas Auburn

06 May 2023

Supervisor: Kofi Appiah

ACKNOWLEDGEMENTS

I would like to first thank my supervisor Kofi for his thoughtful input and advice in our discussions about the project. I would also like to thank my girlfriend Rhiann for the continued support and motivation she has provided me not only over the course of this project, but over my degree as a whole.

Statement of Ethics

The only potential ethical/legal issue for this project is the use of short clips from football matches on YouTube to test the project's implementation. This use falls under the non-commercial research and private study copyright exception and qualifies as 'fair dealing' as it does not affect the market of the original work in any way and the amount of work used is small and reasonable for the task.

The wording from the government exceptions to copyright page is the following [1]:
"The purpose of this exception is to allow students and researchers to make limited copies of all types of copyright works for non-commercial research or private study. In assessing whether your use of the work is permitted or not you must assess if there is any financial impact on the copyright owner because of your use. Where the impact is not significant, the use may be acceptable."

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

Executive Summary

The reversibility of videos can be defined by the extent that the video appears to be playing forwards after it has been reversed. The motivation for studying video reversibility lies closely with the link between video reversibility and video memorability. Video memorability is a measure of how easy events in a video are to remember, and if memorability of videos can be analysed/predicted, this has huge potential to impact areas such as education [3].

The link between memorability of videos and whether they are reversed or not was explored in a paper [4] which tested the memory of monkeys on the order of events in test videos using temporal order judgement tests. It found that upon re-exposure to the videos, the monkeys performed significantly better at remembering the order of events in videos which were not reversed.

Familiarity with the contents of the video exacerbated this effect, which leads to the conclusion that the recollection of these memories was influenced and distorted by prior knowledge of the expected normal temporal order of events in reversed videos. The paper states that human memory is similarly affected by biases of prior knowledge and experience, and therefore it is a good motivation to explore how the reversibility of videos can be estimated, to provide data to further explore the link between the reversibility of videos and their memorability.

This project takes a dataset of short videos from football games and a copy of each of these videos which are reversed and attempts to classify each video as having been reversed or not, providing a score value where a higher score implies a higher confidence that the video was reversed.

The key aims of this project are to make accurate predictions of the classification of reversibility, and more specifically predict a level of reversibility of each video in the form of a score between 0 and 10, where a higher value indicates strong video reversibility, and a lower value indicates that it is most likely not reversible.

The methodology for determining a reversibility score involves passing each frame of a video through a pipeline of computer vision tasks to predict a value for the reversibility of each players movement in each frame. Using the YOLOv8 [9] object detection architecture, a model was trained on a custom dataset [30] to detect the important objects in each frame such as the players and the ball.

After the objects have been detected in the frame, the next step in the pipeline is to pass the detections to update the StrongSORT [15] multi-object tracking algorithm. This is used to associate detections of the same object over multiples frames together and can be used to estimate the velocity of these objects over time. To adjust for camera movement affecting the calculated velocity of the objects in the frame, sparse optical flow is used to calculate the camera movement, which can be subtracted from the object movement velocities to get more accurate readings of the direction and speed of the object's movement.

Next, keypoints of the poses of people in the frame are predicted with a multi-person YOLO pose estimation model. These sets of pose keypoints are matched to a movement track in the StrongSORT object tracker, and combining the calculated velocity of a track with the pose of the person the track is associated with can be used in combination with other factors in the environment such as the location and velocity of the ball, to estimate whether the person is moving in a direction that we would expect in a non-reversed video.

In this paper, multiple methods of determining the final reversibility scores are explored, with the most effective solution using a neural network to estimate the probability of individual player reversibility which is then totalled over all players in all frames. The results of this solution were very effective, achieving a very high classification accuracy of 97.6% on the test dataset, with reversed videos given a reversibility score on average of over 8/10 and non-reversed videos given an average score of under 2/10. Further developmental opportunities for the project are explored in this paper, with ideas such as giving higher weight to players who are less likely to move irregularly or backwards explored, such as when a player is in possession of the ball. Given the high accuracy and usefulness of the scores produced by the project, further investigation into how reversibility scores correlate with the memorability of the videos can also be explored with further research, which can help achieve the overall goals defined by the motivation of the project.

1 Introduction

Predicting reversibility scores for videos requires the use of several computer vision techniques. In this project, I will cover the use of several state-of-the-art techniques for computer vision tasks such as object detection, multi-object tracking and pose estimation to give a predicted reversibility score.

There are a few key aims for this project which will be used to judge the success of its implementation. Firstly, the project should be able to classify whether a video has been reversed or not with high accuracy. In examples where the classification is incorrect, the reasons for the misclassification will be analysed to try and understand the limitations of the implementation.

As well as correctly classifying the videos, the goal is to also provide a score between 0 and 10 where a value close to 10 means that the program is very confident that the video is reversed, and a score close to 0 means that it is very confident that it is not reversed. A score closer to 5 than either 0 or 10 should mean that there are factors in the video which make the video harder to classify either way, for example, people moving backwards in the frame where the video is not reversed. Analysing reversibility scores for individual videos should give a better insight than simply whether the video was classified as reversed.

The type of dataset which will be processed to determine reversibility will be a dataset of three-second videos from football matches, and the program should work equally as effectively regardless of which match is being analysed.

Finally, the program should also not take too long to run, for example, it should not take an unreasonable length of time to classify and predict a reversibility score of 100 short videos. As there are a few computer vision steps for the processing of each frame, processing speeds could become very slow so the usability of the program should be considered.

2 Motivation

A paper by E. Pöppel [2] states that the brain’s “mechanism of temporal integration binds successive events into perceptual units of 3s duration”. In this paper temporal integration is defined as comprising of “subjective phenomena such as simultaneity, successiveness, temporal order, subjective present, temporal continuity and subjective duration”.

What this means is that the way that the brain processes events is that it processes events in three second units, and that the brain processes the time within these three second units together.

In this project we will be analysing the temporal order of events in a dataset of three second videos, mimicking the size of a unit of processing in the brain. Using computer vision techniques, the temporal order of events in each video will be analysed to try to detect an ‘abnormal’ temporal order compared to what we would expect, or in other words, predicting whether a video has been reversed or not.

A related problem to estimating the reversibility of videos is to estimate the memorability of videos. A 2018 paper [3] studied the task of estimating the memorability of videos from a video dataset by using deep learning techniques. Participants in the study were shown videos and tested on them a few minutes later and then a few days later. This created a dataset of videos and memorability scores. Then a neural network was trained on the memorability scores and features extracted from each video using existing models.

The paper emphasises the potential usefulness of being able to measure the memorability of videos, citing that application could be used to improve “education and learning, content retrieval, search, filtering and summarizing, storytelling, etc.”. Certainly, if video content could be optimised to increase the chance that its viewers remember the content of the video, this could have a hugely important and influential impact. Another example not stated in this list is potential use in advertisements, where increasing the memorability of each advert is very important for its success.

A paper by L. Wang et al. [4] evaluated the link between the reversal of videos and the memorability of the videos in monkeys. The monkeys were shown videos both reversed and not reversed and their memory of the order of the events in the video was evaluated using a “temporal order judgement test”. It found that although the monkeys scored similarly on testing for both the reversed and not reversed videos on day 1, on day 2 and on a later test over 32 days later, the monkeys performed worse on the reversed videos.

The paper reaches two conclusions because of this, firstly it suggests that when recollecting the memory, the order remembered by the monkeys was heavily influenced ‘in accordance to their knowledge priors’. This means that there was a sense of what order the monkeys expected the events in the videos to occur based on their prior experiences, and this influenced the reconstruction of the memory of the video in the subsequent temporal judgement tests.

It also stated the effect of the memory distortion was “significantly accentuated by social relevance of the video content”, it explained that the effect of the memory distortion was larger for videos of other primates or other monkeys, compared to less familiar examples such as a sea lion moving backwards.

From this paper, we can infer that there is a connection in primates between the reversibility of videos and how easy they are to remember and how memories can become distorted due to preconceived expectations based on prior experiences and knowledge.

Given this paper also states that memory reconstruction is similarly based on past experiences in humans, to further the work of this project a possible investigation could be to look into whether the reversibility score of each video had a significant effect on the ability of people to remember the order of events in each video.

This would not only compare the memorability of reversed videos to non-reversed videos but also compare the level of memorability between non-reversed videos classified with differing levels of certainty and scores. Depending on the results of this investigation, this project could be furthered to help maximise the memorability of videos, which has numerous advantages as stated previously such as education, learning and improved memorisation.

An additional motivation for this project which is more directly related to the chosen dataset and the computer vision techniques used than for reversibility prediction is its use in football data analysis. Although predicting the reversibility of each video is of less specific use here, the fundamental steps of detecting each player and the ball in the frame, multi-object tracking and estimating velocities for each object, and the estimated pose for each player (and from this the direction the player is facing) can all be adapted for automated analysis of player behaviour.

3 Background and Literature Review

3.1 Object Detection:

Object detection is a key computer vision problem where given an image, the detector is tasked with identifying objects within the image, classifying them, and producing a bounding box which captures the location of the object in the image. This is important for determining reversibility of videos as it enables us to analyse the movement and behaviour of objects in each frame, and the detections can be analysed over time using object tracking which will be discussed later.

There are two main types of object detection methods. In single-stage object detectors, the image is passed through the detection model with one pass, which predicts bounding boxes for objects in the frame and classifies them in the same pass through the network. In two-stage object detectors, the first pass typically identifies regions of the image which it identifies as potential regions of objects, and the second pass classifies these identified regions and predicts a bounding box around the identified object.

Single-stage object detection models have the advantage that they are usually faster than two-stage detectors as each image only has a single processing step, which usually makes them better suited to real-time programs.

A popular single-stage object detector is YOLO (You Only Look Once) [5]. This object detection architecture achieved state-of-the-art speeds by becoming the first object detector to “frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities” where the “whole detection pipeline is a single network”. In other words, YOLO became the first single-stage object detector and inference only required being run once through a network to process an image. This gave record speeds of up to 155 frames per second while giving “double the mAP of other real-time detectors” such as R-CNN [6]. Here mAP (mean average precision) is a measure of the accuracy of the detector based on precision and recall results on a test dataset.

Since the original release in 2016, there have been multiple improvements in new iterations of YOLO, with versions v2 and v3 [7-8] being produced by the original YOLO creator Joseph Redmon. After this, there have been new improvements to YOLO produced by different creators with the latest version, YOLOv8 [9] released in January 2023.

Some of the latest improvements in YOLOv8 [10] include moving to an anchor-free architecture which increases non-maximal suppression (NMS) post-processing speeds and using a mosaic augmentation of images during model training which helps training accuracy.

Another single-stage object detector is the Single Shot Detector (SSD) [11]. This detector was released later in 2016, and produced similar speeds to the original YOLO model, with an accuracy just below the significantly slower but accurate Faster R-CNN [12] – a two-stage detector released in 2015 which was at the time state of the art for accuracy.

Both the SSD and YOLO architectures work in similar ways, splitting each image into a grid and sending it through a network of convolutional layers to extract features from the image which are used to create bounding boxes for each object in the frame.

One of the most popular two-stage object detectors is Faster R-CNN [12]. Although this architecture provides high levels of accuracy, this is significantly slower than the alternative single-stage options and therefore is not suited towards real-time applications. On the other hand, where accuracy is crucially important and the slower processing times are less of an issue, this can still be a good option to maximise the accuracy of your detections.

3.2 Multi-Object Tracking (MOT):

Robust object detection is crucially important for the second stage, multi-object tracking (MOT). Object tracking algorithms work over multiple frames and are used to re-identify the same detected object over multiple frames. Object tracking algorithms can help make detection more robust as in frames where an object is not detected, the object tracking algorithm can predict where an object is most likely to be based on previous movement. Conversely, the robustness of the object detector is also crucially important to the object tracking algorithm, as the tracking algorithm takes the object detections as input, and if the detector is regularly not detecting the necessary objects in a frame, it becomes very difficult to reliably track the object over multiple frames.

A common problem for object tracking algorithms is occlusion, which is where objects become partially/fully hidden behind other objects which results in the object detector failing to recognise the object. In a busy environment, how the chosen tracking algorithm handles occlusion will be vital to its success.

A key component of object tracking for predicting reversibility will be to estimate the velocity of an object, where we can compare whether it is moving in an expected direction based on the state of the object.

DeepSORT [13] is a multi-object tracking algorithm which builds on the SORT (Simple Online and Realtime) [14] tracking algorithm. SORT achieved state-of-the-art performance by using a combination of Kalman filtering and the Hungarian algorithm. Kalman filtering is an algorithm which predicts the state of an object based on the object's previous detections. The state of an object is modelled here as seven values: the x, y coordinates and height of the object, the velocity value for each of the x, y and height values, and a constant aspect ratio value. The Hungarian algorithm is used to match detections to existing targets while trying to minimise the intersection over union (IOU) distance value for each object tracked. This value is a measure of how much a new detection's bounding box overlaps with where the Kalman filter predicts the object should be.

DeepSORT built on SORT by adding a machine-learning element to the assignment process with an appearance descriptor for each tracked object. This is considered alongside trying to minimise the IOU distance and increases re-identification accuracy.

More recently, improving on DeepSORT, more advanced object tracking algorithms have been developed that use better re-identification networks and have more sophisticated representations of the appearance descriptor which helps increase accuracy along with other optimisations.

One current state-of-the-art performing object-tracking algorithm is StrongSORT [15]. This modernisation of the DeepSORT algorithm improves on the original significantly with more advanced feature representation, as well as two new features [16] called Gaussian-Smoothed Interpolation (GSI) and AFLink. GSI is a post-processing technique which improves the smoothness of predicted motion after detection misses and AFLink is an improvement of previous techniques for associating objects with previous and future frames.

An alternative top-performing object-tracking algorithm is ByteTrack [17]. This strays from the usual techniques of object tracking algorithms in that usually low confidence detections are simply discarded, however, the ByteTrack paper argues that this introduces reasonable risk that occluded objects could be missed unnecessarily. The solution ByteTrack introduces is a novel approach where these low-confidence detections are still considered.

It works by first matching the high-confidence detections with existing object tracks, based on a calculated IOU value and feature similarities, and then for each object track from previous frames checking each of the low-confidence detections to try and match the track’s predicted position and features to one of the detections. This allows the algorithm to detect occluded objects very well while still maintaining high accuracy.

3.3 Multi-Person Pose Estimation:

Multi-person pose estimation is a task where people are identified in a frame and the location of key points of the different parts of their body are calculated. For this project, it is important that the pose estimation algorithm can detect the pose of many people in a frame accurately as this data will be important to determine the reversibility of their movements.

YOLOv8, as previously mentioned, is a very fast and accurate state-of-the-art architecture, and has recently released pre-trained multi-person pose estimation models, with the largest model having a mAP score of 71.6 on the COCO pose dataset which is highly competitive with other state-of-the-art pose estimation models. As this model is trained on the COCO pose dataset, it outputs the COCO 17 body part keypoints, which is less detailed than other alternatives.

Other high performing pose estimation models include OpenPose [18] and MoveNet [19]. MoveNet, developed by Google, is a tensorflow-based model which is very suited to real-time applications due to its very high speeds of up to 30 frames per second, however a limitation to its use in this project is that it can only identify the pose of up to 6 people in a frame at once.

OpenPose is a model which can identify a far larger number of keypoints than the previous two models listed, with 135 different keypoints detected which includes 70 face keypoints, so this is well suited for problems where detailed facial analysis is needed. OpenPose remains one of the most popular pose estimation models due to its well-detailed Github documentation [20].

4 Methodology

For object detection, I have chosen YOLOv8. This option provides state-of-the-art speeds and accuracy (the largest pre-trained model having a mAP value of 53.9 on the COCO dataset), as well as comprehensive and clear documentation which will help me train a custom model and use it to predict on the chosen dataset. YOLOv8 is also a very popular option for object detection and there are multiple tools and websites which are designed specifically for YOLOv8, which helps with development ranging from dataset labelling to troubleshooting issues that other users have experienced. Processing speed is an important consideration for this project, as running object detection, multi-object tracking and pose estimation on each frame is computationally expensive and therefore a trade-off between accuracy and speed should be considered.

StrongSORT will be used as the object-tracking algorithm for this project, this modernisation of the DeepSORT algorithm ranks very highly at 4th in the MOT17 [21] dataset benchmark for multi-object tracking, and there is a pip-distributed packaged version of this algorithm adapted by Github user kadirnar [22] from the original StrongSORT repository [23]. An appropriate re-identification embedded network will be chosen from the torchreid model zoo [24] to find the network which gives the best accuracy for this problem. Tracking accuracy is especially important for this project for estimating object velocities, and therefore this is a great option for providing high-end accuracy while still being able to run at real-time speeds.

YOLOv8 is also used for the task of multi-person pose estimation. Similar to the reasons listed for the choice of YOLOv8 for object detection, YOLOv8-Pose is capable of running at high speeds with a good level of accuracy which makes it a good choice for this project. When considering this option compared to others, the level of detail in the high number of key points provided by OpenPose would likely overly complicate the task of trying to judge the direction which a person is facing and the more streamlined 17 keypoints by the YOLO pose estimation is easier to analyse. For this project, although MoveNet provides a good option for high-speed multi-person pose estimation, the limitation of only being able to estimate the pose of 6 people at a time makes YOLO a more appealing choice.

Other factors in consideration are again the high detail documentation available for YOLO which helps development and secondly, similarities in implementation between the pose models and the object detection models for YOLO make it an attractive option as understanding of implementation is transferable from one to the other.

The python module OpenCV [25] is a computer vision library which is used in this project to get each frame from a video for processing. To determine whether a video has been reversed or not, each frame of a video is passed through object detection, object tracking and pose estimation stages before reversibility calculations.

For the object detection stage, an object detection model will train on a custom dataset so that it can detect all the important objects in the frame with high accuracy. With a dataset of videos of football matches, this includes detecting all the players in the frame and the location of the ball. YOLOv8 has a useful python training API and CLI and community-made datasets on the website Roboflow which can be exported to the format needed for training YOLOv8 models.

Challenges which the object detection model could face include occlusion, where the view of either the ball or players is obscured, for example where multiple players overlap each other in the frame or are covering the ball. This increases the need for a dataset which contains enough examples of occluded objects to help with training.

Each frame is passed to the trained object detection model and a list of classes, bounding boxes and confidences are returned. When predicting the location of objects in each frame, it is important that a balance is found in the confidence threshold that a detection is discarded or not. If a confidence threshold is too high too many detections will be missed which will decrease the accuracy of our final reversibility prediction as there will be a smaller sample size to analyse, and if the confidence threshold is too low then we risk false positive detections which can decrease the accuracy of object tracking and our final reversibility predictions.

The detections from the object detection predictions are used as input to the StrongSORT object tracker. This assigns each object detection to a ‘track’ which aims to follow the motion and location of each object in the video over multiple frames. This uses both the IOU distance and an object appearance descriptor to accurately assign each detection to the correct track.

One important consideration is choosing the embedded re-identification network to maximise the tracking accuracy while also not being detrimental to performance.

From the Kalman filter class for each track, we can extract a property called the ‘mean’ of the track. From this, we can extract the bounding box of the track, as well as estimated velocities in the X and Y dimensions.

Finally, the frame is input into the YOLOv8 pose estimation model. This works in two parts, firstly it predicts bounding boxes around the detected people in the frame, and secondly, it predicts the keypoints on each person’s body. As found on a Github discussion forum [26] the different keypoints correspond to the following positions on the body:
"keypoints": ["nose", "left_eye", "right_eye", "left_ear", "right_ear", "left_shoulder", "right_shoulder", "left_elbow", "right_elbow", "left_wrist", "right_wrist", "left_hip", "right_hip", "left_knee", "right_knee", "left_ankle", "right_ankle"]

For each of these detected keypoints, there are x and y coordinates and a visibility score. Using these values, I can experiment with different methods to estimate the orientation of each detected pose in each frame in the x and y directions. Once the direction each player is facing has been estimated, we can use the estimated velocities from the object tracking to help determine reversibility. For example, if we estimate that a player is moving left and facing left, it is more likely that the video is not reversed.

Although most of the time in a frame there will be more people moving in the direction they are facing, sometimes players move backwards naturally. This could cause problems predicting reversibility as this would predict the player as moving more likely in reverse. Possible options to explore include considering whether the relative direction and distance from the ball for each player mean it is unrealistic that a player would be moving backwards or not.

These values can be combined either by analysing and finding a pattern manually or by taking the data for each player and using a neural network to calculate the probability of video reversibility for each player’s movement.

5 Implementation

5.1 Choice of datasets

Before starting the process of developing a solution first the type of video dataset which my solution can be tested on needs to be decided. This is important to decide now, as it is important that the dataset of images that the object detection model is trained on is reflective of the test videos.

For the videos chosen for testing, I have chosen to use full football match videos uploaded to the FA cup YouTube channel [27]. I have chosen to take these videos for testing as they are firstly easily accessible, are taken from camera angles which have a clear view of the players and the ball, and lastly because there are likely many training datasets which should be applicable for this choice. However, I will make an alteration to these videos - although the majority of camera shots are a view of the pitch, there are some camera shots which are close-ups, for example of the manager or the players not playing who are on the bench. I will be splitting the long video inputs into 3 second short clips to test for reversibility, and any clips which are not of the pitch will be removed from the dataset.

To train my YOLO object detection model, I then had to either find a well-annotated dataset which is similar to my test videos or create a dataset myself. Roboflow [28] is a website which has a large database of community-annotated datasets, as well as a good tutorial on how to train YOLO models on these custom datasets [29]. I found a good dataset for training which contains 1440 images from football games and a total of 12,643 annotations [30]. This includes 4 classes: player, goalkeeper, referee, and the ball.

5.2 Training and implementing the YOLO model

After exporting the dataset as per the Roboflow blog tutorial, I was able to train a YOLO model on this dataset. Initial accuracy results were poor, with the model only detecting the ball an average of around 25% of the time. This was due to the model training struggling when images were resized down to a size of 640 pixels wide. After adjusting the size that the image got resized to in training using the ‘imgsz’ parameter to 1280, the training accuracy increased dramatically.

In the evaluation section, I will explore the choice of model size, and finding the right balance in the trade-off between accuracy and consistency of detection, and speed of running inference on each frame. During implementation, the custom model used was trained on the yolov8n base model (nano size – the smallest and fastest version) to maximise speeds during development.

After training the yolov8n.pt pre-trained model for 150 epochs on the custom dataset, figures 5.1-5.4 show the plots produced which analyse the results of the training.

Figure 5.1 shows a confusion matrix for the model, this tells us several things about the model. Firstly, we can see that where there is a player in the frame, it is detected with a very high 98% accuracy. This is good as it means that it should very rarely miss any players in the frame and they should be therefore reliably tracked. There are, however, some limitations of this model. The ball is detected with an accuracy of 80%, which although high, means that the ball is missed in the frame 20% of the time which could result in difficulty tracking. There is also an issue with false positives, where players are detected instead of the background class with a high probability. To improve these results, either a bigger model or a higher quality, larger dataset could be used.

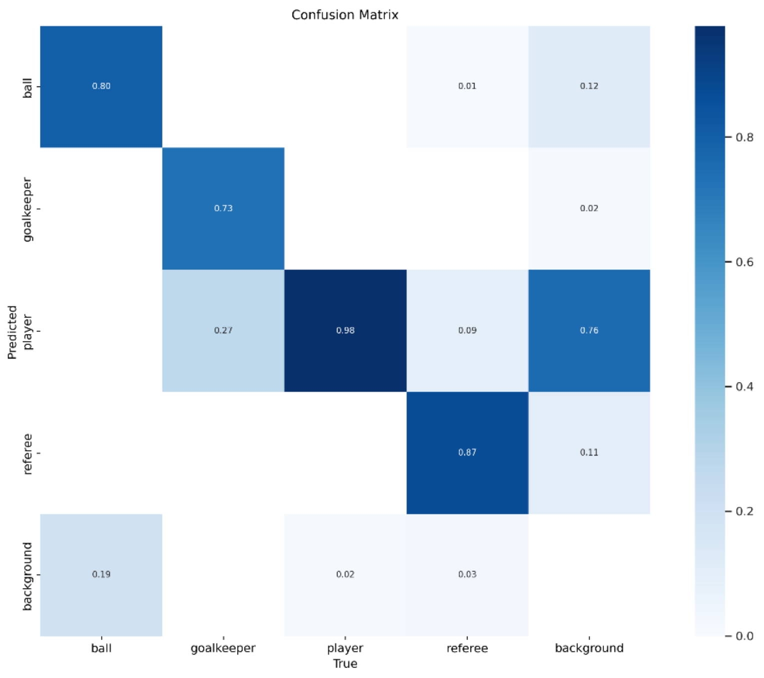


Figure 5.1: YOLO training confusion matrix

Figures 5.2 and 5.3 show two charts, the precision-confidence curve and the recall-confidence curve. Both precision and recall are both a measure of model accuracy [31], precision refers to the percentage of detections which are correct valid detections, and recall refers to the percentage of valid objects which are detected by the object detection.

Precision and recall are both affected by the confidence threshold of the model. A higher confidence threshold for detections results in higher quality detections which are more likely to be accurate, and therefore gives the model higher precision. On the other hand, a higher confidence threshold means a higher number of valid objects are not output as detections when the detections do not surpass the confidence threshold.

Therefore, it is important to strike a balance in the confidence threshold in ensuring that there are fewer false positives produced, while also maximising the percentage of objects in each frame that are detected.

Figure 5.4 shows the direct relationship between precision and recall, and shows that even with high recall values, the precision of the model remains high. From this, it was decided that a lower confidence threshold might produce better results, as precision remained high even at these lower confidence thresholds whereas recall suffered heavier at higher thresholds.

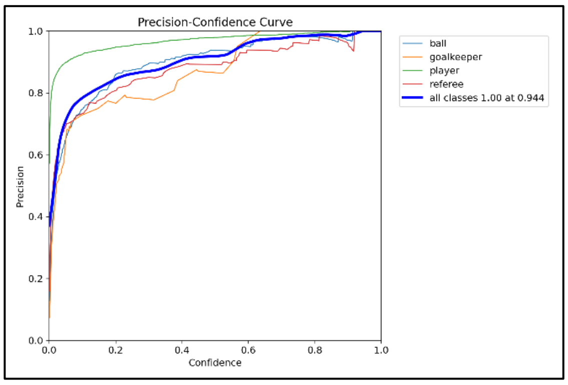


Figure 5.2: YOLO precision confidence curve

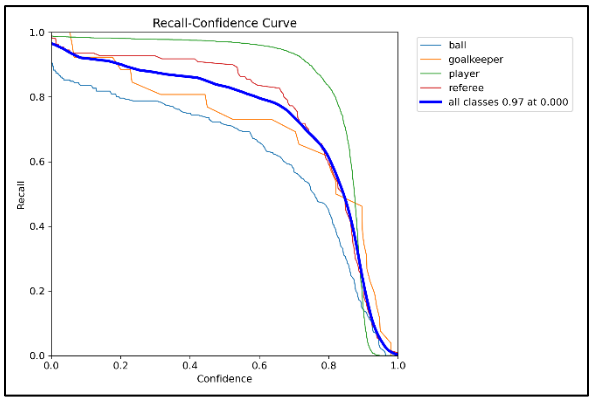


Figure 5.3: YOLO recall confidence curve

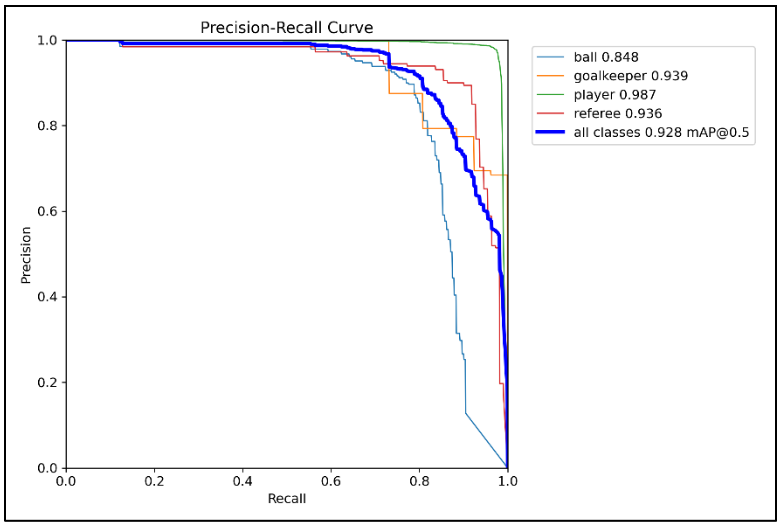


Figure 5.4: YOLO precision-recall curve

Figure 5.5 provides a good summary of the YOLOv8 architecture. There are three main stages: the backbone, neck, and head. The backbone is the main sequence of convolutional and fully connected layers which extracts feature maps from the input images. In YOLOv8, there is a sequence of two convolutional layers of kernel size 3, stride 2 and padding of 1, followed by a C2F (convolution layer to fully connected) layer. Next there is a sequence of convolution and fully connected layers before the output is sent to the 'neck' of the architecture.

In the neck, there are further stages of processing that involve more convolutional and fully connected layers, upsampling, and concatenation of features before this output is sent to the 'head' of the architecture. Within the neck, the three layers P3, P4, and P5 form a 'feature pyramid' which allows the model to detect objects at different scales, with each of these layers outputting to their own detection function in the head.

In the head, the object detection functions provide a prediction for the location and identity of objects in the image. These predictions are then passed to a loss function which is minimised to train the network.

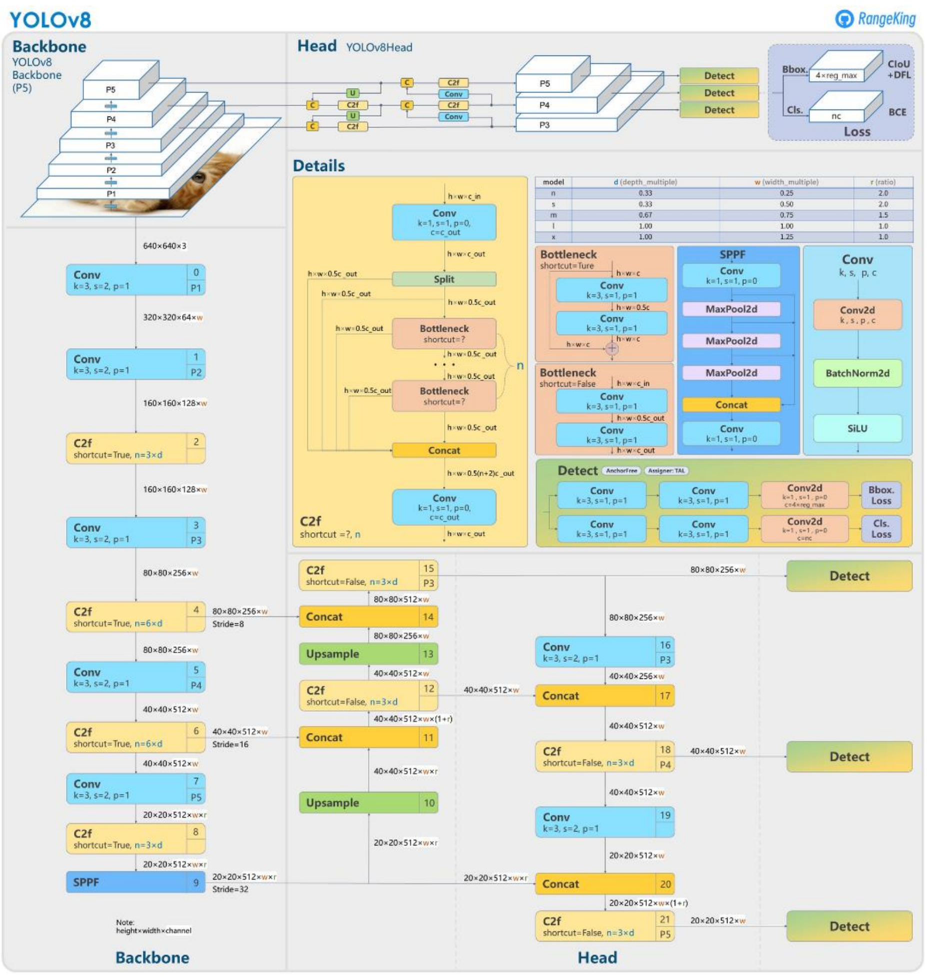


Figure 5.5: A detailed summary of the YOLOv8 architecture [32]

After passing a frame from the test video dataset through the newly trained YOLO model, Figure 5.6 shows the frame with the bounding boxes plotted onto the frame, alongside their class names and confidence scores. As shown in the confusion matrix in figure 5.1, the referee is sometimes misclassified as the player class, although for estimating forward movement, their purposes will be the same in this project so it shouldn't cause many issues. In the frame, the ball and all players are detected and classified with high accuracy, which should enable high accuracy multi-object tracking in the next stage.

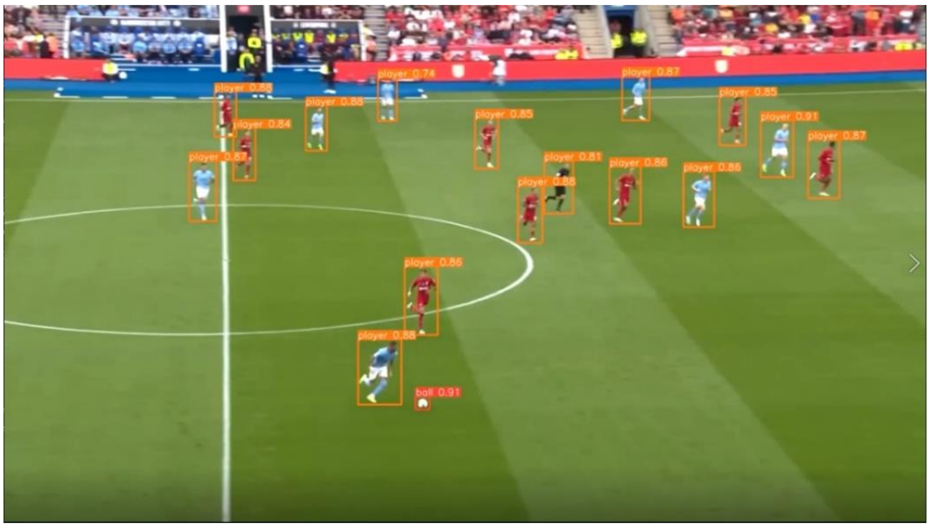


Figure 5.6 – Plotted Object Detections

5.3 Multi-object tracking using StrongSORT

Once a frame has been passed to the object detection model, the detections need to be passed to the object tracking algorithm. The two key parameters to consider are the 'max age' and 'max IOU distance' parameters. The max IOU parameter is a measure of how far away a detection is allowed to be from where the track expects the object to be for it to be considered. In this case, I have set this value to a high value of 0.9. A higher value increases the probability that detections will be erroneously associated with a class, however, I found with a lower value the ball was less likely to accurately be tracked especially when moving at high speeds where there would be a high IOU value.

The other parameter 'max age' is the number of frames in which a track is allowed to not have a detection associated with it before it is discarded. A lower maximum age value results in higher track id switches, where a moving object must be assigned a new track as the old track lost detection of it for too long, however, a too high maximum age value can result in false tracks where a track is following a predicted path for an object that is no longer there.

A balance needs to be found where a moving object is not immediately discarded when it is not matched with a detection, however, tracks won’t still carry on long after losing their associated object. Another potential issue with having too large a value for the maximum age is that this project relies heavily on the estimated velocities from each track, and too far an extrapolation of an object’s trajectory and velocity could lead to inaccurate data being evaluated.

Figures 5.7 – 5.8 show two frames a few seconds apart, in each frame each track’s bounding box and their track id values are plotted above the object. As shown in the figures, the track ids are maintained over the frames for each object, including the fast-moving ball as it passes the player.

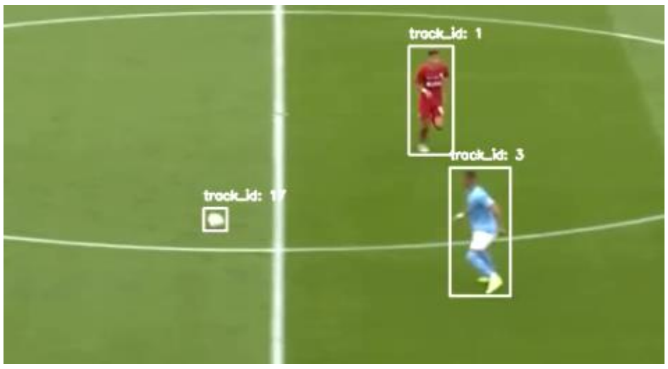


Figure 5.7: Object tracking in an earlier frame

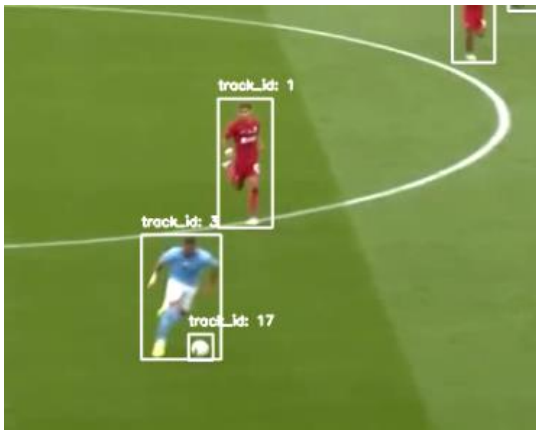


Figure 5.8: Object Tracking in a later frame

The next task is to extract the velocities of each object in the frame from the object tracker. This is easy to access through the ‘mean’ property of the track. The mean of a track is a list of 8 properties about the track. The first two values of the mean are the x and y coordinates of the centre of the bounding box of the track. The third value is the aspect ratio of the bounding box, i.e. the ratio between width and height of the box, and the fourth is the height of the bounding box.

The last four are based on the motion of the object and are the most useful for predicting reversibility. The fifth and sixth values are the velocities in the x and y dimensions, and the seventh and eighth are the accelerations in the x and y dimensions.

When plotting the velocities for each bounding box, however, I found a problem. In Figure 5.9, I have plotted the velocities along the x axis above each bounding box. As shown in Figure 5.9, at this point in the video, the velocities were all large velocities in the negative x direction, i.e., moving to the left. The cause of this issue was the panning camera, and even though the players were moving to the right, the camera movement meant that they were being moved to the left of the frame and hence the negative tracking velocities.

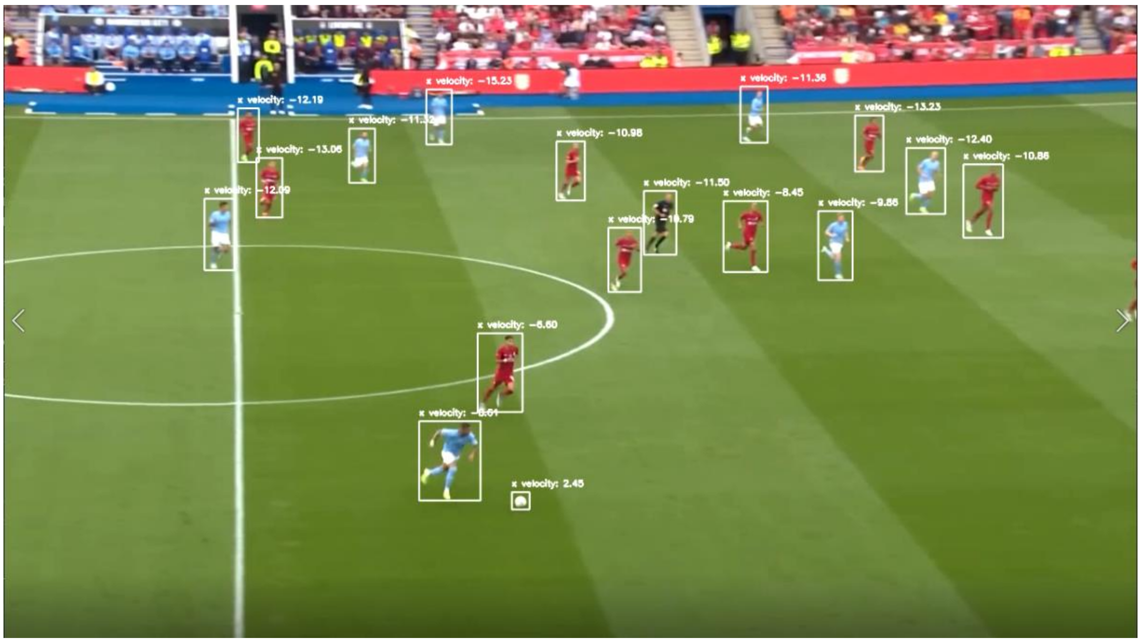


Figure 5.9: Detected velocities with camera pan

To adjust for the movement of the camera, I had to calculate the velocity at which the whole frame is moving, so that I could subtract the movement of the camera from the tracks' velocities, to give an accurate reading of the velocity of movement for each track.

5.4 Optical Flow Tracking

The best method to estimate camera panning speed is using optical flow tracking. According to the OpenCV documentation [33], optical flow is defined as "the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera". There are two main types of optical flow tracking, sparse optical flow, and dense optical flow. Sparse optical flow finds points which are easy to track in the frame and calculates the flow vector for each of those points. The second type, dense optical flow calculates the optical flow of every pixel in the frame. Dense optical flow, in general, is more accurate as it calculates optical flow using more pixels, however, this does make it more computationally expensive.

A problem with using dense optical flow to estimate camera movement, however, is that because it considers all pixels in the frame, this will include the pixels for our moving objects. This means that the estimation for camera movement will be offset by the movement of the players, which would decrease the accuracy of the camera movement velocity prediction.

Using sparse optical flow, if just the background pixels can be selected to track, then this does not have the same issue. A good solution to this requirement of just detecting the movement of the camera is to use a ‘mask’ on the frame when selecting the points to track. A mask is an array of 1s and 0s in the shape of the frame, where a 1 indicates that the pixel can be selected to track and a 0 indicates not to select this pixel to track.

Sparse optical flow can be implemented using the OpenCV module. Firstly, the features in the frame to track are chosen using a function [34] that looks for the ‘most prominent corners’ in an image. A corner can be defined as the meeting point of two edges, and these points are very important for optical flow tracking. This is because of the Aperture problem [35]. The Aperture problem is where when viewing an edge, “We can only tell the motion locally in the direction perpendicular to the edge”. This means that points along an edge are more difficult to track than a corner which intersects multiple edges as motion can be detected across all dimensions.

The Shi-Tomasi Corner Detector [36] is implemented in OpenCV in the ‘goodFeaturesToTrack’ function. This algorithm detects a list of corners by checking the relative change in intensities in the X and Y dimensions for each pixel and if the minimum of these values is greater than a set threshold, sets that pixel as a corner. Once the corner pixels have been decided, these can be tracked using the Lucas Kanade method [37]. This algorithm attempts to track pixels from one frame to the next by looking nearby to the pixel’s original location and trying to find a pixel which has a similar pixel intensity gradient to the pixels around them. This works under the assumption that both firstly the corner being tracked will only have moved a short distance from one frame to the next and secondly that the light intensity does not change much between frames. This algorithm is implemented in OpenCV in the ‘calcOpticalFlowPyrLK’ function and contains parameters such as the ‘window size’ for which to search for the corner to track in nearby pixels.

To create the mask for the corner detection function, first an array of ones the size of the frame's resolution is created. Then, for each bounding box output from the object detection predictions, the area covered by each bounding box is set to a set of zeros in the mask. This should ensure that the optical flow is just being measured for the background to measure the camera panning velocity.

Once the corners to track are found, for each frame the average of the velocities of all the corners will be calculated to get an accurate estimation of the camera panning speed. This can then be subtracted from the velocities calculated for each object's track to get a more accurate estimation of the velocity of each object in the frame. Once the corners to track leave the frame and are lost, the goodFeaturesToTrack function can be re-run to detect a new set of features for the current frame.

In Figure 5.10, having implemented sparse optical flow tracking, I have plotted the predicted velocities in the X dimension for each track after subtracting the calculated optical flow velocity in the X dimension. As shown in the figure, the velocities are more accurate with the players moving to the right having a positive velocity in the X dimension.

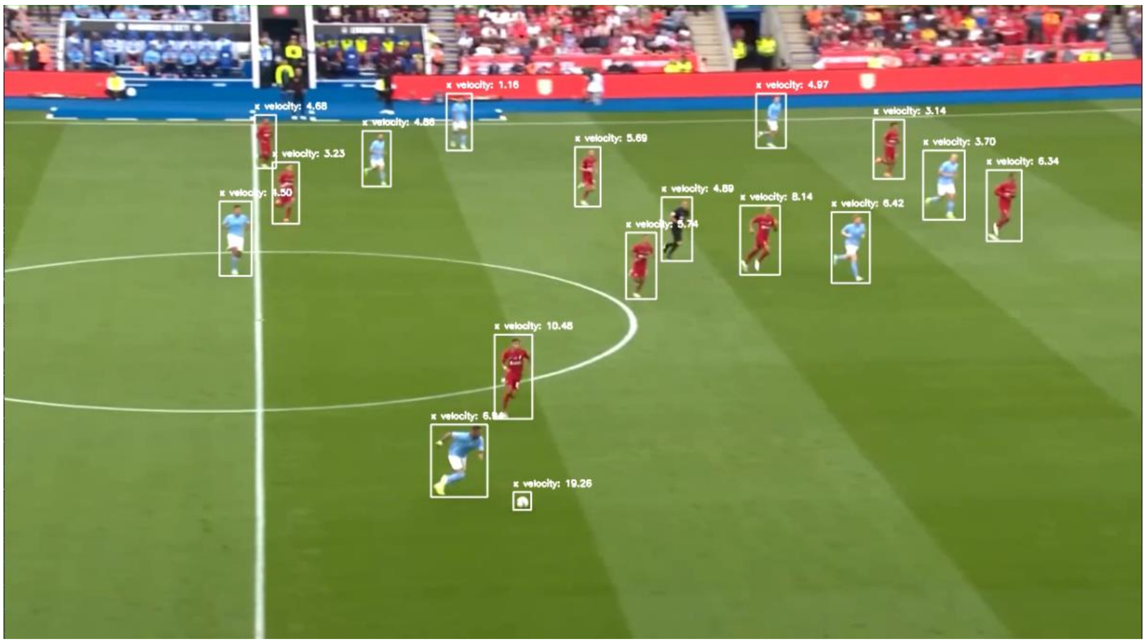


Figure 5.10: Velocities adjusted for camera pan

5.5 Multi-Person Pose Estimation

For the task of pose estimation, I decided against training the pre-trained YOLO pose models on a custom dataset. The main reason being that the purpose of the pose estimation task is to get the keypoints on the people in the frame, which the pre-trained models already achieve.

Therefore, all that is needed is to load the pose estimation model, and run the model on each frame to get a list of bounding boxes around each person and a keypoints array of x, y and visibility values for each keypoint for the person detected. As with the object detection YOLO model, the default processing image size of 640x640 was too small for the model to provide enough detections due to the scale of the people in the frame. I found that after changing the ‘imgsz’ parameter to 1920, although slowing the speed of the model, detected most people in the frame and their keypoints.

Next, each of the detections made by the pose estimation model must be matched to a track. The simple implementation for my solution to this problem is to iterate through the detected poses, and then for each pose iterate through the tracks from the object tracker and match the detected pose to the track which has the smallest difference in the coordinates of their bounding boxes.

After matching each pose to a track, Figure 5.11 shows the bounding boxes and the velocity for the associated track plotted onto the frame.

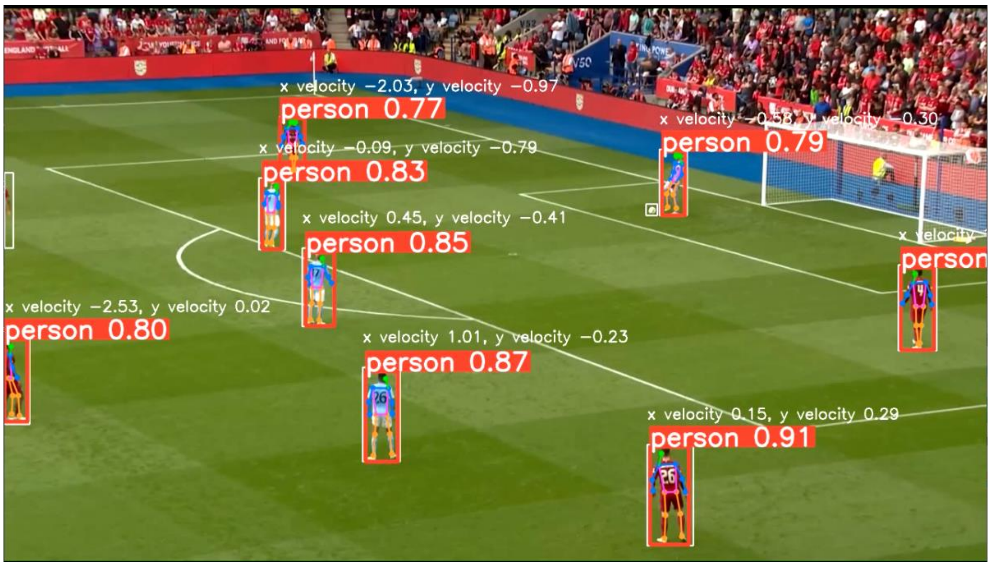

x velocity −2.03, y velocity −0.97
person 0.77
x velocity −0.09, y velocity −0.79
person 0.83
x velocity 0.45, y velocity −0.41
person 0.85
person 0.79
person
x velocity −2.53, y velocity 0.02
person 0.80
x velocity 1.01, y velocity −0.23
person 0.87
x velocity 0.15, y velocity 0.29
person 0.91

Figure 5.11: Matching poses to tracks

5.6 Determining a Video Reversibility Score

The primary influence in predicting the reversibility of each video will be determining whether the players are moving forwards or not. I have implemented two separate strategies which use the data gathered from the previous sections to make a prediction on video reversibility.

The first strategy used both the coordinates and the visibility values for each of the keypoints in a player’s pose to predict in the x and y dimensions whether the player was facing one way or another. This works by comparing the keypoints assigned to the left of the body to the right side of the body to estimate player orientation.

To determine whether the player was facing left or right, the visibility values for the keypoints are totalled, and if the right-sided keypoints have higher visibility values then it is judged that the player was facing to the right and vice-versa.

To estimate which way a player was facing in the Y dimension, instead of using the visibility values, the X coordinates of each side’s keypoints were totalled and if the right-sided keypoints’ X coordinates were larger, then it is more likely that the player is facing upwards in the frame and if the left-sided total was larger then it was more likely facing downwards.

There are two variables which are used to determine whether it is likely that a video is reversed, one variable is increased when a player is estimated to be moving backwards, and the other is incremented when a player is estimated to be moving forwards. The amount a variable is increased by is decided by the velocity of the player in that dimension, this gives higher weight to players who are moving faster as in general people move faster while facing the direction they are moving.

At the end of each video, whichever of these two variables has the larger value can be viewed as the prediction for whether the video is reversed, and for each video, an estimated reversibility score can be calculated by the estimated reversed variable divided by the sum of the two variables to get a value between 0 and 1, and then multiplied by 10 to get a score between 0 and 10, as aimed for in the introduction.

This approach is simple; however, it does have significant limitations. Firstly, the orientation of a player in each dimension is only considered as one direction or another, not an angle of orientation. This limited representation means that very slight orientations in one direction can influence the overall prediction.

The second strategy is more complex and involves using the player’s data as input into a neural network. My implementation of a neural network is implemented using the Pytorch module [38-39] and takes key data points as input for each player detected.

Each player has an input into the network, and each input is a flattened list of 57 values, below is a breakdown of where this value comes from:

- The first 51 values are a flattened list of the (17, 3) shaped keypoints for a player's detected pose in the frame. There are 17 keypoints, with each keypoint represented by three values: its x, y coordinates and visibility value.
- Two values for the x and y velocities of the player
- Two values for the x and y coordinates of the ball
- Two values for the x and y velocities of the ball

This option is significantly more computationally expensive than the first strategy although it has several advantages. One advantage includes a more holistic analysis of the data, where the neural network can develop a sophisticated understanding of the relationships between keypoints, the velocities of both the ball and the player and how the distance between the player and the ball can affect the confidence of the prediction.

Secondly, with the sigmoid activation function in the last layer, the model's prediction can be viewed as a measure of confidence in how sure the model is of its prediction. In comparison with the initial strategy (where borderline differences in keypoint values could have a significant effect), this measure of confidence can be used to provide a weighting in the final prediction where low confidence results are counted less compared to examples where the model is surer.

To train this neural network, first training data needed to be gathered. After running through a large dataset of videos forwards and backward to get equally split training data, the 57 input values were gathered in a list for each player's pose detected. This list eventually contained over 130,000 training examples to train the neural network. After several hundred epochs of training, the model was able to predict whether the player's movement had been reversed with approximately 79% accuracy for each player in each frame.

As with the previous method, the sum of the predictions of reversibility or non-reversibility made for each player's movement can be used to give a prediction for the whole video. The sigmoid output from the model can also be multiplied by the player's velocity (as with the previous method), to give greater weight to the players moving at higher speeds.

6 Results and Evaluation

To evaluate the success of my implementation, I created a dataset of 126 three second videos to evaluate and give a reversibility score for each. Half of these videos (63) are reversed.

The first half of these videos were created by taking three seconds of video at regular intervals from two football matches, the first match being Manchester City vs Liverpool in July 2022 [40] and the second match being Brighton vs Liverpool in January 2023 [41]. These two games are played at different stadiums with different camera angles to provide variety in the dataset.

For each video clip captured from the source videos, a new video was created of that video clip in reverse. This means that we can reliably see any difference in how effective the program is at classifying reversed videos compared to classifying non-reversed videos as the only difference between the two halves of the dataset is the order of the frames.

There were initially 100 videos each for both not reversed and reversed videos, however, a significant number of the videos captured were removed from the original dataset as they were not footage of the match itself but included other shots such as the crowd or the managers for example.

First, each of the methods to determine the final reversibility score will be tested and evaluated. For each video, a score between 0 and 10 will be given, where a higher score is a stronger prediction that the video is reversed, and a lower value is less likely that the video is reversed. Values lower than 5 will be classified as not reversed, and values over 5 will be classified as reversed. The score is calculated by first dividing the total value of the variable for predicting that the video is reversed by the sum of this variable and the other variable for predicting that the video is not reversed. This value will be between 0 and 1 but is multiplied by 10 to give a more human-readable score.

The first aim was to accurately classify the videos as reversed or not reversed, and the success of each method will primarily be based on the classification accuracy in what percentage of the videos it classifies correctly. However, it is also important to consider the reversibility score for each video to evaluate how confident it was in its prediction and to evaluate the cause of the situations where the program was less certain either way.

After each video is evaluated, the video path and its reversibility score will be placed into one of four lists. These lists are:

- ‘True Positives’ (TP): where the program correctly predicts the video is reversed
- ‘True Negatives’ (TN): where the program correctly predicts the video is not reversed
- ‘False Positives’ (FP): where the program incorrectly predicts that the video is reversed
- ‘False Negatives’ (FN): where the program incorrectly predicts that the video is not reversed.

Table 6.1: Reversibility classification methods results summary

Classification Method	Classification Accuracy	TP	TN	FP	FN
Basic pose-velocity analysis	80.2%	51	50	13	12
Neural network	94.4%	58	61	2	5
Neural network: (x Velocity)	96.8%	60	62	1	3

Table 6.1 shows the results comparing each of the classification methods. Although the non-neural network approach still produced a respectable accuracy of 80%, this is significantly less than the classification accuracies produced by using the neural network to classify each player.

The data also suggests that multiplying the neural network output by the players’ velocity has helped boost the overall accuracy of the classification, by providing higher weight in the classification decision to players who are moving at faster speeds.

Next, I wanted to test what the effect of changing the object detection model size from the ‘nano’ YOLO size to ‘small’. Interestingly, there was no increase in classification accuracy, and even more surprisingly it had misclassified an additional video as reversed when it was not reversed.

This was surprising as the confusion matrix for this model had shown a higher training accuracy than the nano model, however in practice it had produced worse results. An article by Justin Deschenaux [42] even suggested that despite the larger models achieving higher scores on the validation datasets, there was not always an increase in the robustness of the model, and sometimes a larger model even led to lower robustness and worse generalisation.

As increasing the model size did not lead to an increase in classification accuracy, using the smallest trained model is recommended for running this project, as the accuracy of classification was mostly unaffected by the change in model size and the smallest model will also run at a faster speed.

Table 6.2: Detection model size test results

Classification Method	Classification Accuracy	TP	TN	FP	FN
Neural network: velocity multiplier (nano model)	96.8%	60	62	1	3
Neural network: velocity multiplier (small model)	96.0%	59	62	1	4

To evaluate how the best-performing classification method can be improved we can look at the videos in which it failed in classifying correctly and try to understand why it failed in these instances.

One video which failed to classify correctly in any of the classification approaches was 'reversed_test_81.avi'. Upon inspection of the reversibility scores, each run for this video showed a score of 0.

However, once the video itself was viewed, the reason behind its misclassification was clearer. This clip showed an extra wide camera angle which made the players and the ball very small scale, which resulted in the pose estimation and detection models struggling to detect the objects in the frame.

Although this is largely a limitation in the scale which can be detected by the detection models used, for both object detection and pose estimation, setting a higher maximum age for the object tracker can help mitigate the sparse detections. The maximum age of a track was set to a low value of 2, and in the case of this video where the detections would have been sparse, the object tracker found it very difficult to match these sparse detections with existing tracks as the tracks were discarded quickly.

Although it can be mitigated with a higher maximum track age, the issue of sparse detections is not likely one that could be solved in this project without scaling the image size up further in the detection models, which would significantly slow the speed of the program, however, it does show future scope for potential improvement of the program although this would likely require greater computing power than I had available on my desktop PC to provide adequate performance.

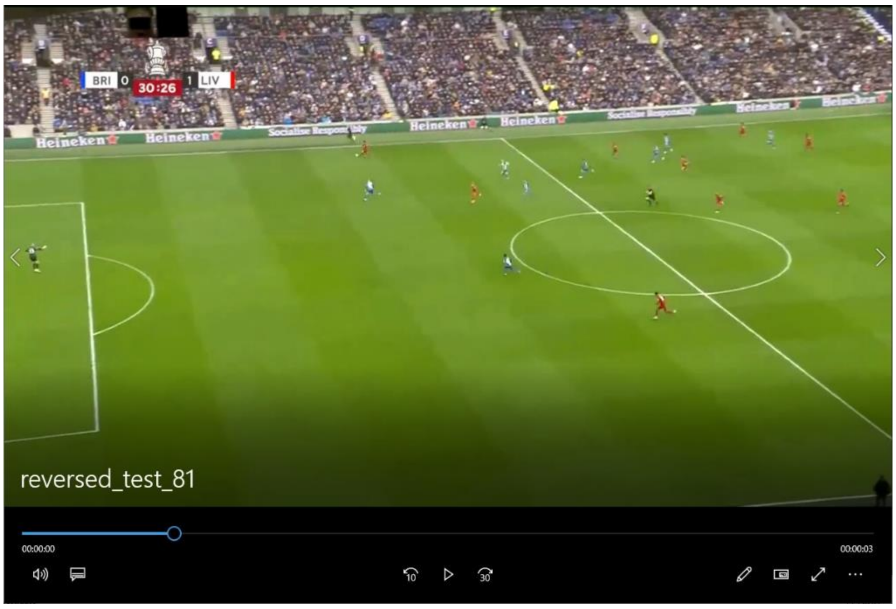


Figure 6.1: Small Scale Video problem

To try and find an optimal value for the maximum age of tracks for the object tracker, two different values for the maximum age, 6 and 15 were tested as shown in table 6.3. Table 6.3 shows that increasing the maximum age to 15 led to a decrease in the accuracy of classification. This could be due to lower accuracy track velocity estimation on tracks extended beyond where the object was last detected. A maximum age of 6 frames maintained the classification accuracy on the previously correctly classified videos and improved on the previous best by correctly classifying ‘reversed_test_81.avi’.

An additional test with a maximum age of 6 frames and larger tracking reidentification embedded network from the torchreid model zoo ('osnet_x0_5_imagenet.pt') misclassified this video again despite providing some detections. This shows that despite increasing the maximum track age allowing the program to make some predictions, without consistent detection on a video, performance will be erratic.

Table 6.3: Maximum tracking age results

Maximum Age	Classification Accuracy	TP	TN	FP	FN
Max Age: 2	96.8%	60	62	1	3
Max Age: 6	97.6%	61	62	1	2
Max Age: 15	93.7%	58	60	3	5
Max Age: 6 (0.5x embed)	96.8%	60	62	1	3

Another video which caused the program issues was ‘test_59.avi’ and ‘reversed_test_59.avi’. Both these videos were still misclassified and therefore this source video is useful to analyse.

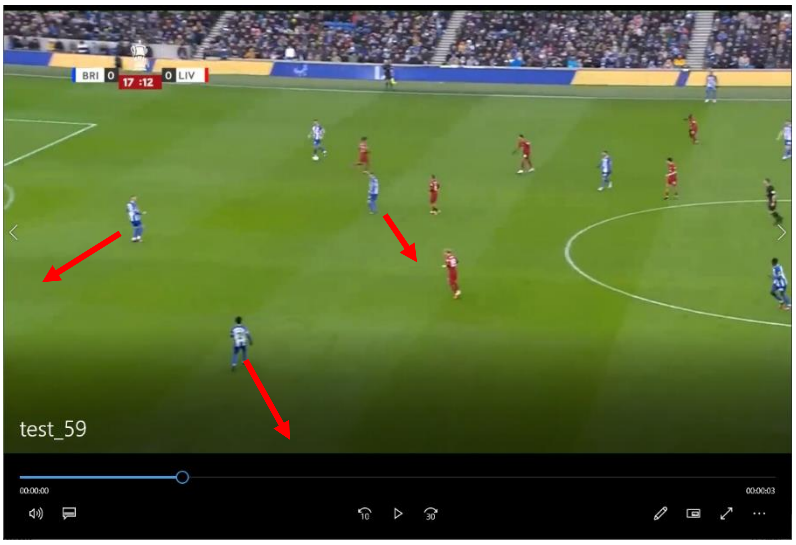


Figure 6.2: Regularly misclassified test video

This video is an interesting challenging example to the program as there are multiple players who are moving backwards in the original video, which fool the program into predicting that the video is backwards in the original video.

Although it is understandable in this specific case that it is a difficult video to correctly classify, there are lessons that can be learned from this video to improve the program with further development.

Firstly, identifying when a player is currently in possession of the ball and providing greater weighting to their movement could be an adjustment which results in fewer misclassifications such as this. In general, it is far more natural for a player to move backwards on the pitch when they don’t have the ball than when they are in possession of the ball. Therefore, placing a higher weight on the prediction of this player’s movement’s reversibility value would make sense when looking for unusual and unnatural movement.

Another potential improvement which could improve the classification of videos like this is to consider the effect of scale on the velocity predictions. In this program, the velocities in the frame are calculated as the pixels moved per frame.

Although this issue has seemingly not had a significant effect and arguably has a small impact, players further away from the camera on the other side of the pitch are smaller scale in the video and therefore the number of pixels they move in the frame is smaller compared to players closer to the camera.

As the effect on the overall reversibility score is affected by the velocity of the number of pixels moved by each player per frame, this means that players closer to the camera have a higher weighting in the overall prediction.

This could be argued to be a positive as the players close to the camera will be detected and their velocity estimated with higher accuracy than players further away, although, on the other hand, this can lead to a bias where a couple of players closer to the camera are more likely to skew the overall prediction where players further away from the camera are predicting the opposite.

An option which could be considered in further development would be to calculate this scale variance and use it to get a more accurate value of the velocity of the objects in the frame based on real-life velocities instead of pixel velocities. This could be a complicated processing step and would have to be high in accuracy to avoid adding additional bias to the problem.

Revisiting the original aims of the project, we can compare these results to the original success criteria to evaluate its success.

The first of the aims of this project is to produce a solution which classifies the input videos as either reversed or not reversed with high accuracy. This has undoubtedly been a success, with a very high classification accuracy, with the best accuracy achieved on our test dataset reaching 97.6%. Misclassified videos have been analysed, and clear reasons for their misclassification have been determined and, where issues were found, potential improvements and suggestions for further development have been discussed which could help lower the impact of the issues found.

The second aim was to provide a reversibility score for each video between 1 and 10, from which a degree of confidence in the classification prediction can be inferred. I would argue that this has been an overall success, the average reversibility score for a reversed video was over a score of 8, and the average score for a non-reversed video was below 2. In general, the results show that the score of misclassified videos usually had a score closer to 5 than these averages, which supports the idea that a more polarised score usually results in a classification which is more likely to be accurate.

The third aim was that the project should work just as well for multiple different matches as video sources equally effectively. This is hard to measure, especially as there was a very high overall accuracy. Although all videos were classified with high accuracy, it could be argued that the classification was more effective on the first match's videos, as there were fewer misclassifications on the videos from this match. However, it can easily also be argued that with such a small sample size of misclassified videos, these could just be outliers of the test dataset and the overall classification accuracy between the two matches is minimal.

When considering the speed of the program, on my desktop PC with a low-end GPU, it took on average 30-40 seconds per 3 seconds video to classify. This was manageable during testing, however, still required waiting up to 1.5 hours to run through the entire dataset. This project is heavily dependent on the use of the GPU, and using a faster GPU would increase the overall speed. This does decrease the accessibility of the program to those who have access to GPUs, however, to run the multi-stage computer vision tasks used in this project it would be a difficult task to manage this on just the CPU.

7 Conclusion

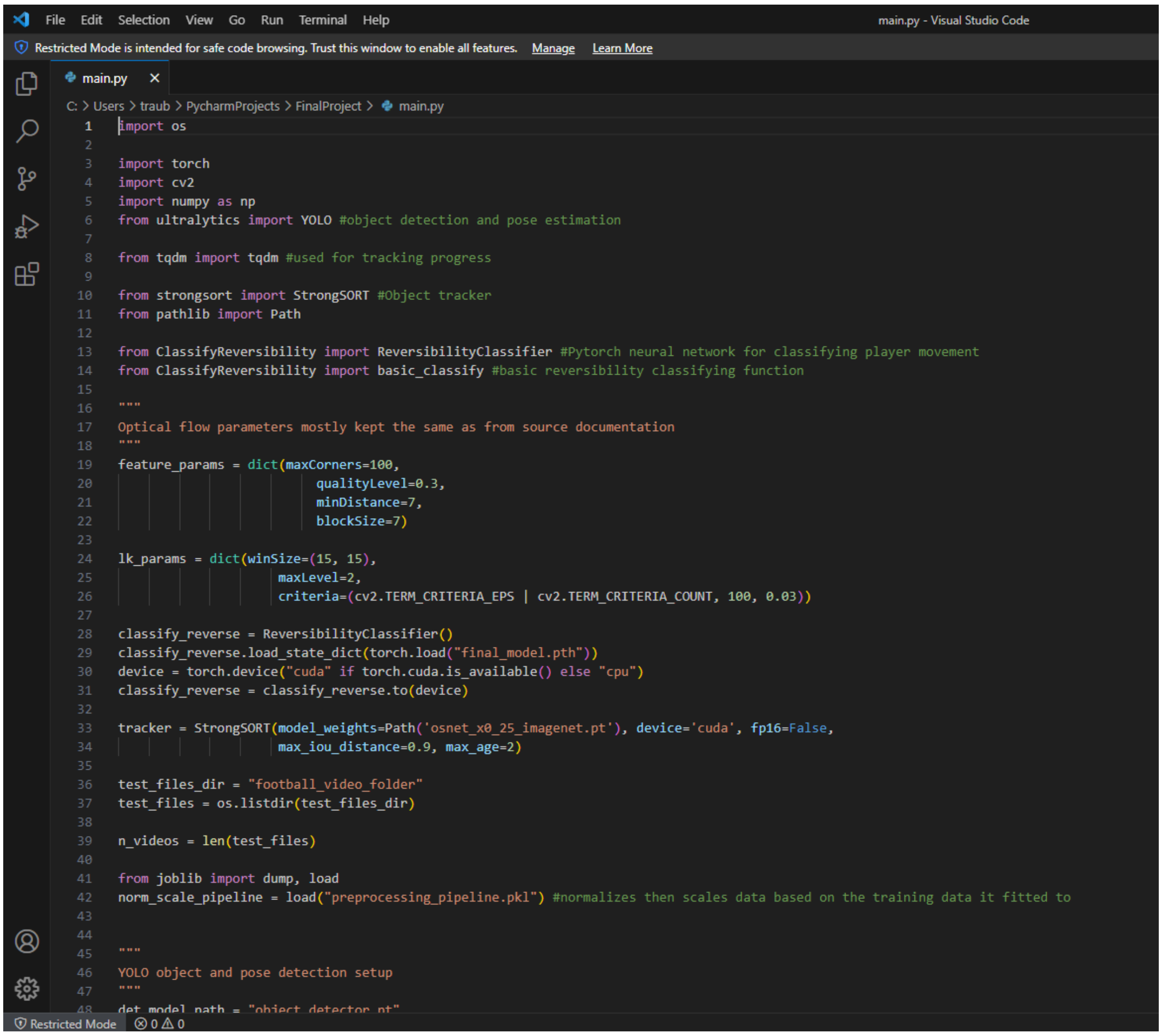
Overall, it has been shown that the approach outlined in the methodology has produced high levels of accuracy in predicting a reversibility score for this dataset of videos. This project has covered the challenges related to object detection, multi-object tracking, using optical flow measurement to help with velocity estimation with an unstable panning camera, and multi-person pose estimation. Different methods of calculating the final score predictions have been evaluated, compared, and analysed.

Based on the original aims, I would say that the project has been a success, and there are clear areas for further development which could be explored. Examples of further areas of development, as explored in the results and evaluation section, include considering which player is in possession of the ball, distinguishing between and detecting the differences between the players from both teams and exploring the effect of scale variance based on distance from the camera.

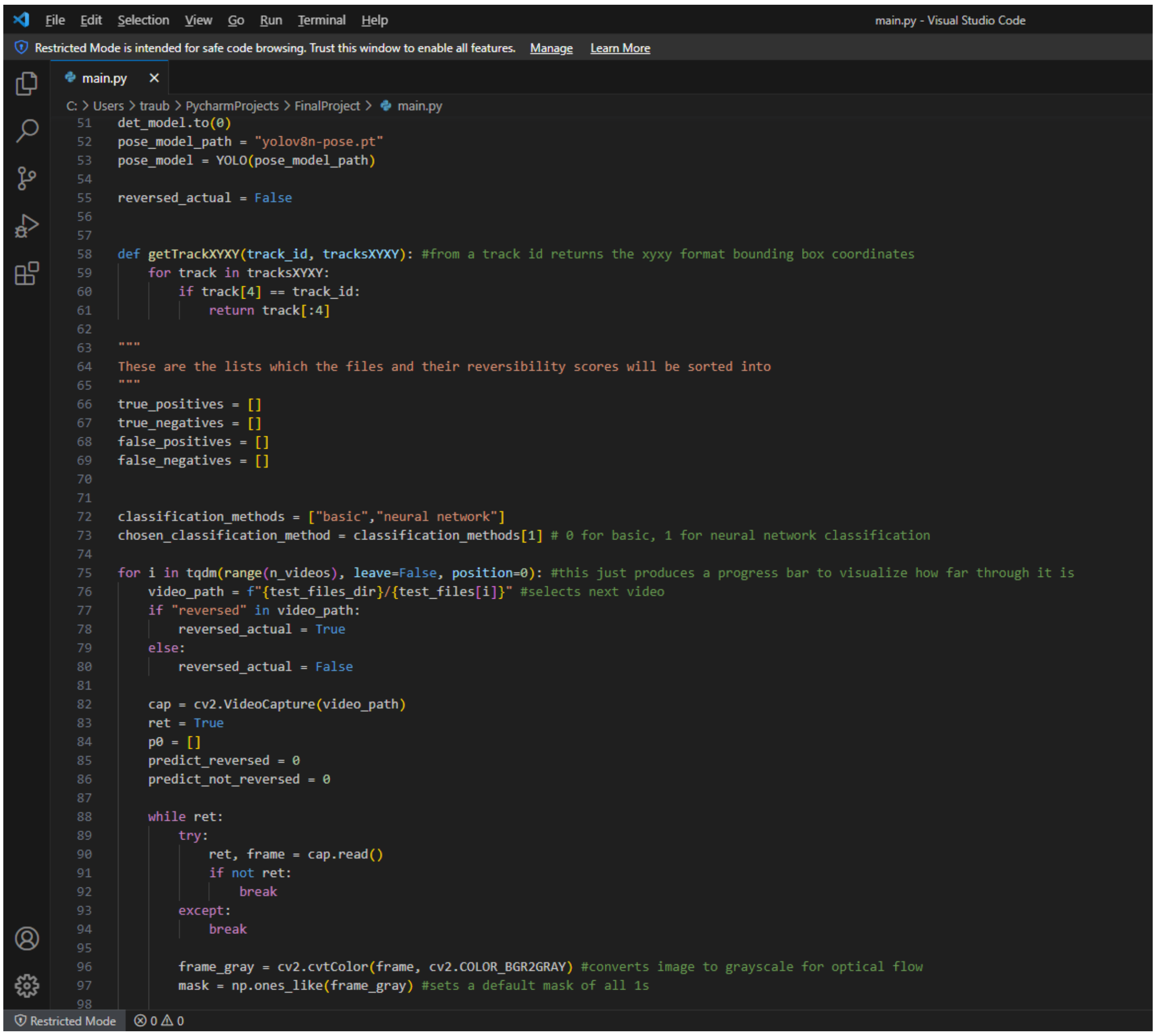
Relating back to the motivations for the project, the results of this project are high accuracy, and therefore I would be confident in using the scores produced by the program on a dataset of videos in further research. As suggested in the motivation section, the memorability of a dataset of videos could be studied and measured as in the study by R. Cohendet et al. [3], these memorability scores could be then compared to the reversibility scores output by the program, and then these two sets of values could be analysed to check if there is any statistical significance in a connection between the two sets of scores.

8 Appendix

8.1 Main code in main.py pages 31-35



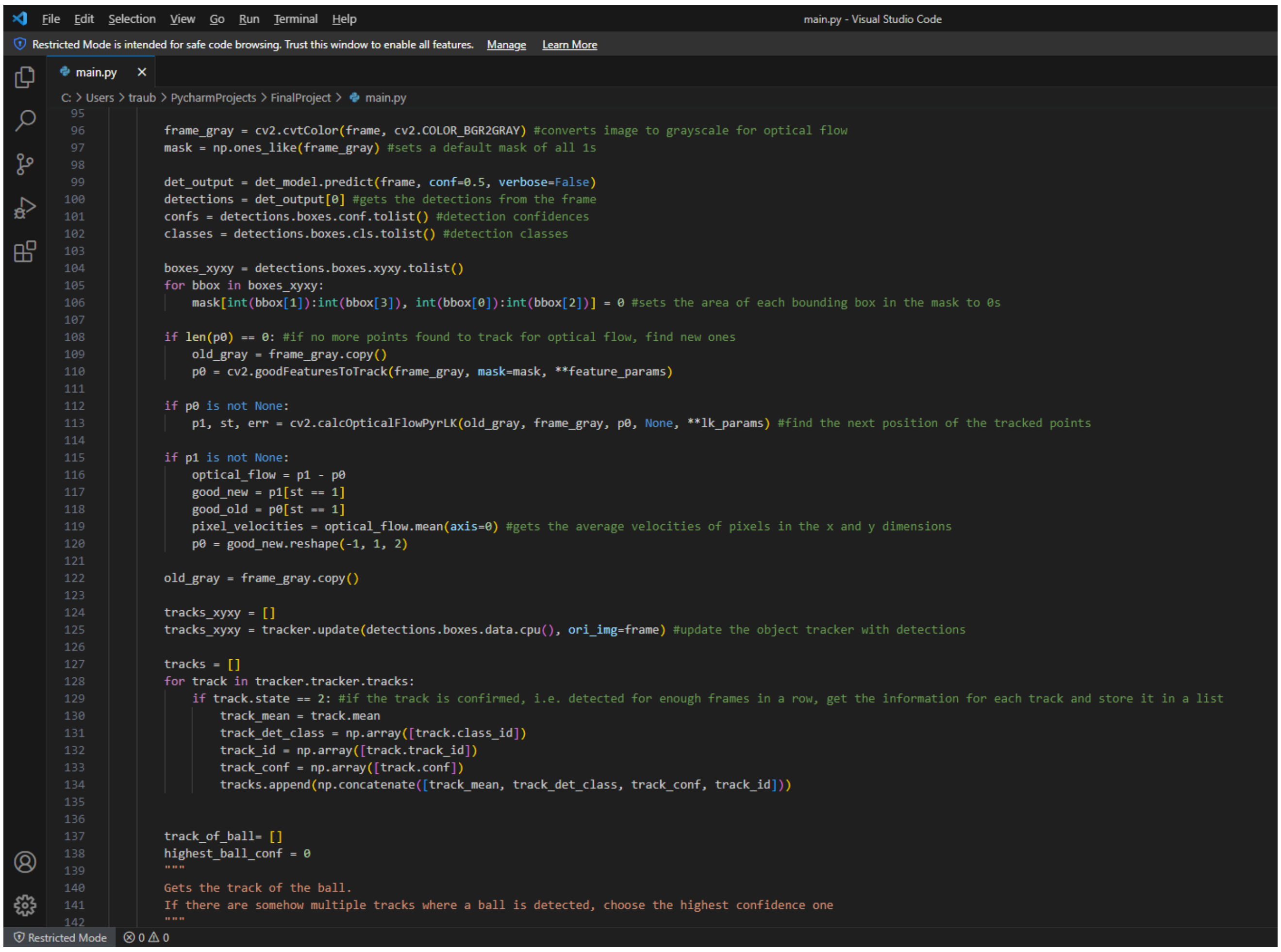
```python
import os

import torch
import cv2
import numpy as np
from ultralytics import YOLO #object detection and pose estimation

from tqdm import tqdm #used for tracking progress

from strongsort import StrongSORT #Object tracker
from pathlib import Path

from ClassifyReversibility import ReversibilityClassifier #Pytorch neural network for classifying player movement
from ClassifyReversibility import basic_classify #basic reversibility classifying function

"""
Optical flow parameters mostly kept the same as from source documentation
"""
feature_params = dict(maxCorners=100,
                          qualityLevel=0.3,
                          minDistance=7,
                          blockSize=7)

lk_params = dict(winSize=(15, 15),
                     maxLevel=2,
                     criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 100, 0.03))

classify_reverse = ReversibilityClassifier()
classify_reverse.load_state_dict(torch.load("final_model.pth"))
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
classify_reverse = classify_reverse.to(device)

tracker = StrongSORT(model_weights=Path('osnet_x0_25_imagenet.pt'), device='cuda', fp16=False,
                     max_iou_distance=0.9, max_age=2)

test_files_dir = "football_video_folder"
test_files = os.listdir(test_files_dir)

n_videos = len(test_files)

from joblib import dump, load
norm_scale_pipeline = load("preprocessing_pipeline.pkl") #normalizes then scales data based on the training data it fitted to


"""
YOLO object and pose detection setup
"""
det_model_path = "object_detector.pt"
```


File Edit Selection View Go Run Terminal Help
main.py - Visual Studio Code
Manage
Learn More
main.py
C: > Users > traub > PycharmProjects > FinalProject > main.py
Restricted Mode
0 0

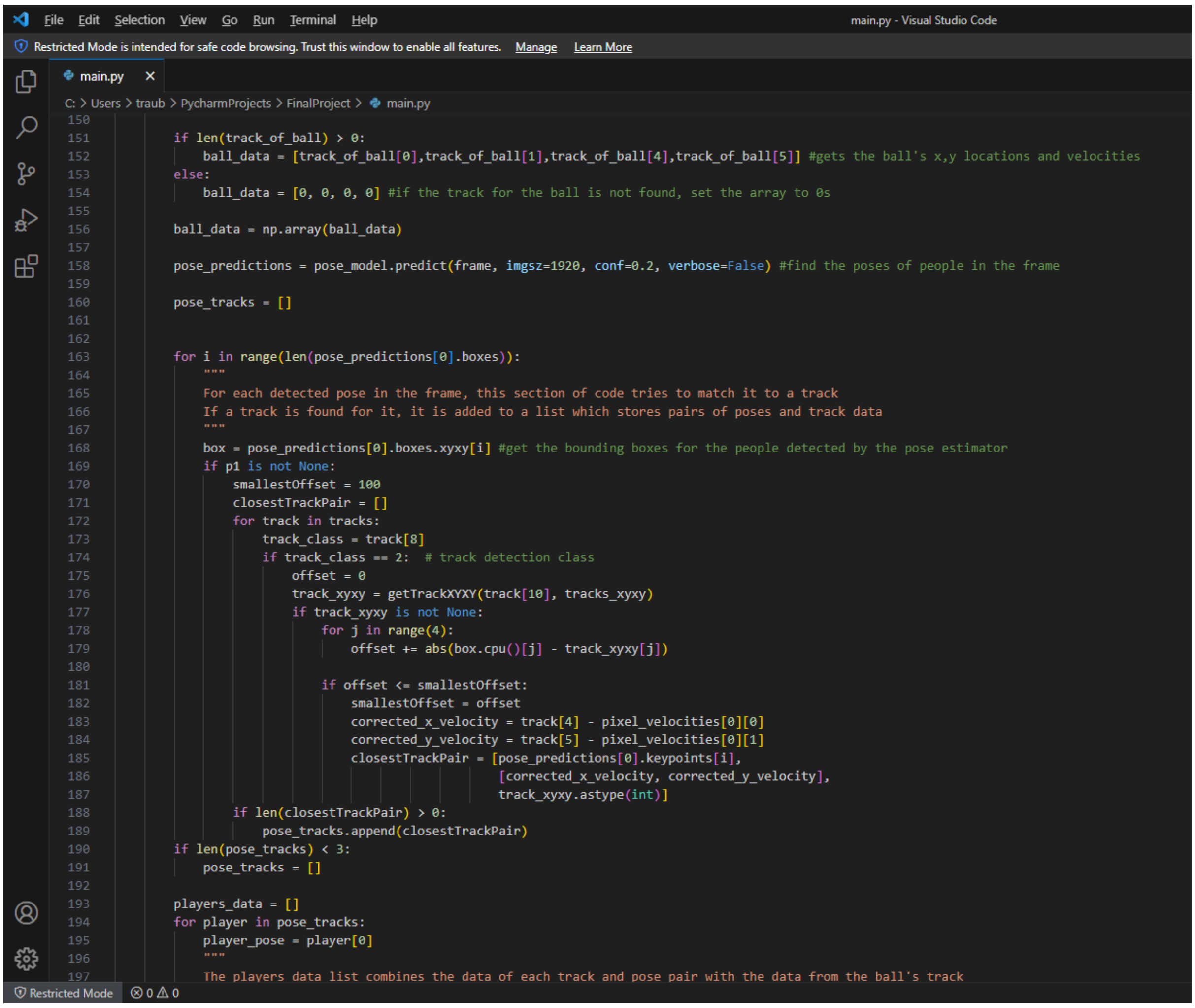
Restricted Mode is intended for safe code browsing. Trust this window to enable all features.

```python
det_model.to(0)
pose_model_path = "yolov8n-pose.pt"
pose_model = YOLO(pose_model_path)

reversed_actual = False


def getTrackXYXY(track_id, tracksXYXY): #from a track id returns the xyxy format bounding box coordinates
    for track in tracksXYXY:
        if track[4] == track_id:
            return track[:4]

"""
These are the lists which the files and their reversibility scores will be sorted into
"""
true_positives = []
true_negatives = []
false_positives = []
false_negatives = []


classification_methods = ["basic","neural network"]
chosen_classification_method = classification_methods[1] # 0 for basic, 1 for neural network classification

for i in tqdm(range(n_videos), leave=False, position=0): #this just produces a progress bar to visualize how far through it is
    video_path = f"{test_files_dir}/{test_files[i]}" #selects next video
    if "reversed" in video_path:
        reversed_actual = True
    else:
        reversed_actual = False

    cap = cv2.VideoCapture(video_path)
    ret = True
    p0 = []
    predict_reversed = 0
    predict_not_reversed = 0

    while ret:
        try:
            ret, frame = cap.read()
            if not ret:
                break
        except:
            break

        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #converts image to grayscale for optical flow
        mask = np.ones_like(frame_gray) #sets a default mask of all 1s

```

Appendix

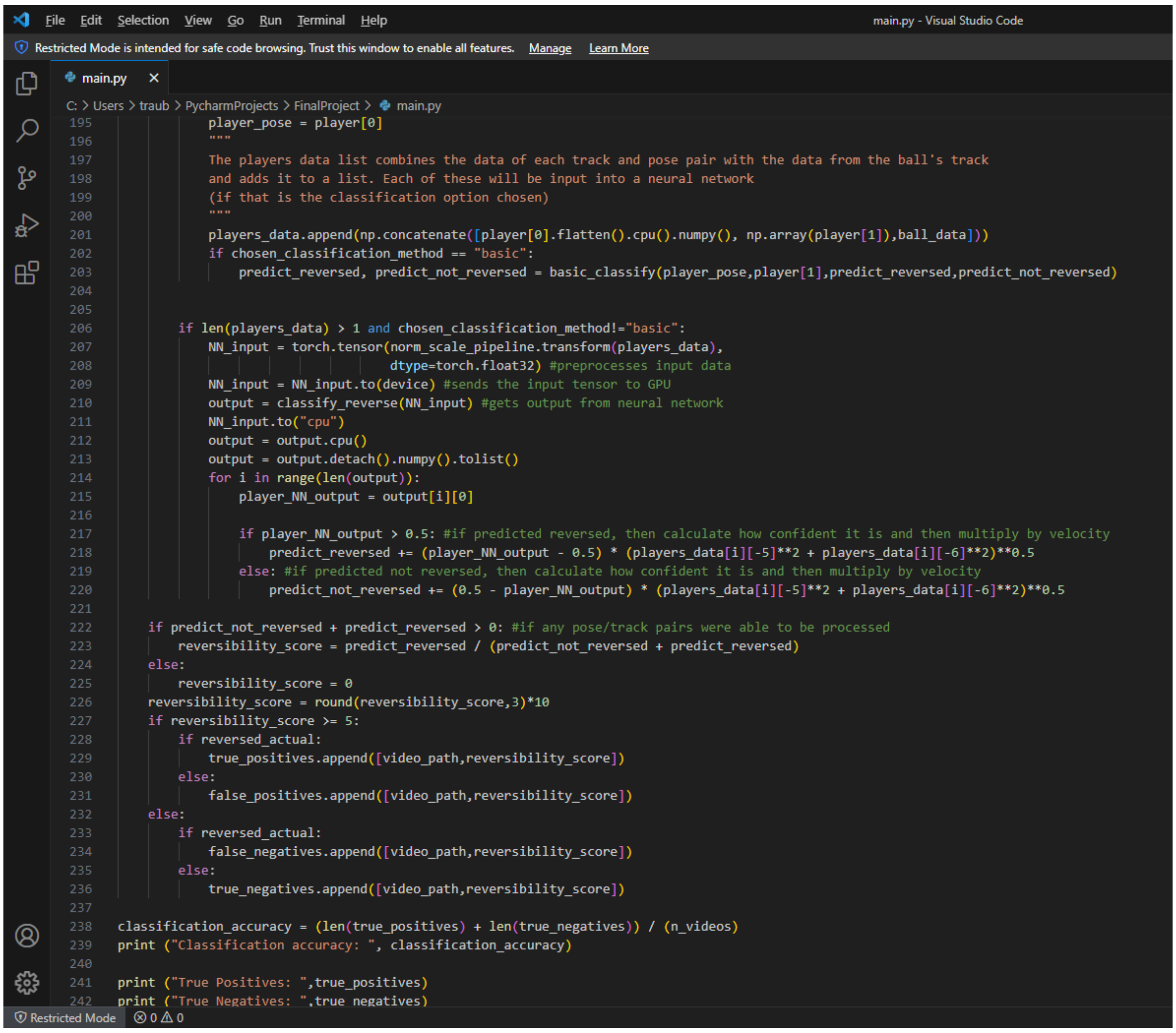

File Edit Selection View Go Run Terminal Help
main.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
main.py
C: > Users > traub > PycharmProjects > FinalProject > main.py
Restricted Mode
0 0

```python

        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #converts image to grayscale for optical flow
        mask = np.ones_like(frame_gray) #sets a default mask of all 1s

        det_output = det_model.predict(frame, conf=0.5, verbose=False)
        detections = det_output[0] #gets the detections from the frame
        confs = detections.boxes.conf.tolist() #detection confidences
        classes = detections.boxes.cls.tolist() #detection classes

        boxes_xyxy = detections.boxes.xyxy.tolist()
        for bbox in boxes_xyxy:
            mask[int(bbox[1]):int(bbox[3]), int(bbox[0]):int(bbox[2])] = 0 #sets the area of each bounding box in the mask to 0s

        if len(p0) == 0: #if no more points found to track for optical flow, find new ones
            old_gray = frame_gray.copy()
            p0 = cv2.goodFeaturesToTrack(frame_gray, mask=mask, **feature_params)

        if p0 is not None:
            p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None, **lk_params) #find the next position of the tracked points

        if p1 is not None:
            optical_flow = p1 - p0
            good_new = p1[st == 1]
            good_old = p0[st == 1]
            pixel_velocities = optical_flow.mean(axis=0) #gets the average velocities of pixels in the x and y dimensions
            p0 = good_new.reshape(-1, 1, 2)

        old_gray = frame_gray.copy()

        tracks_xyxy = []
        tracks_xyxy = tracker.update(detections.boxes.data.cpu(), ori_img=frame) #update the object tracker with detections

        tracks = []
        for track in tracker.tracker.tracks:
            if track.state == 2: #if the track is confirmed, i.e. detected for enough frames in a row, get the information for each track and store it in a list
                track_mean = track.mean
                track_det_class = np.array([track.class_id])
                track_id = np.array([track.track_id])
                track_conf = np.array([track.conf])
                tracks.append(np.concatenate([track_mean, track_det_class, track_conf, track_id]))


        track_of_ball= []
        highest_ball_conf = 0
        """
        Gets the track of the ball.
        If there are somehow multiple tracks where a ball is detected, choose the highest confidence one
        """
```

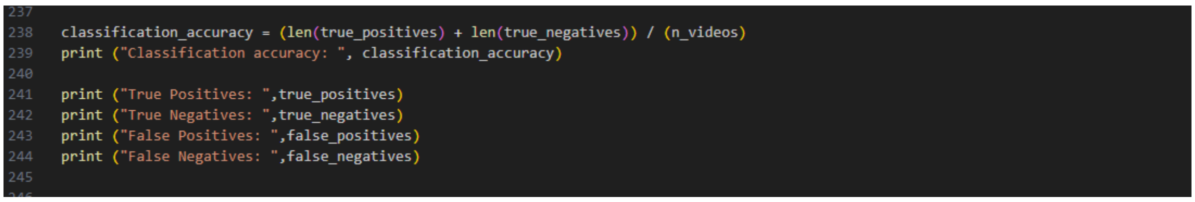

File Edit Selection View Go Run Terminal Help
main.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
main.py
C: > Users > traub > PycharmProjects > FinalProject > main.py

```python

        if len(track_of_ball) > 0:
            ball_data = [track_of_ball[0],track_of_ball[1],track_of_ball[4],track_of_ball[5]] #gets the ball's x,y locations and velocities
        else:
            ball_data = [0, 0, 0, 0] #if the track for the ball is not found, set the array to 0s

        ball_data = np.array(ball_data)

        pose_predictions = pose_model.predict(frame, imgsz=1920, conf=0.2, verbose=False) #find the poses of people in the frame

        pose_tracks = []


        for i in range(len(pose_predictions[0].boxes)):
            """
            For each detected pose in the frame, this section of code tries to match it to a track
            If a track is found for it, it is added to a list which stores pairs of poses and track data
            """
            box = pose_predictions[0].boxes.xyxy[i] #get the bounding boxes for the people detected by the pose estimator
            if p1 is not None:
                smallestOffset = 100
                closestTrackPair = []
                for track in tracks:
                    track_class = track[8]
                    if track_class == 2:  # track detection class
                        offset = 0
                        track_xyxy = getTrackXYXY(track[10], tracks_xyxy)
                        if track_xyxy is not None:
                            for j in range(4):
                                offset += abs(box.cpu()[j] - track_xyxy[j])

                            if offset <= smallestOffset:
                                smallestOffset = offset
                                corrected_x_velocity = track[4] - pixel_velocities[0][0]
                                corrected_y_velocity = track[5] - pixel_velocities[0][1]
                                closestTrackPair = [pose_predictions[0].keypoints[i],
                                                    [corrected_x_velocity, corrected_y_velocity],
                                                    track_xyxy.astype(int)]
                if len(closestTrackPair) > 0:
                    pose_tracks.append(closestTrackPair)
        if len(pose_tracks) < 3:
            pose_tracks = []

        players_data = []
        for player in pose_tracks:
            player_pose = player[0]
            """
            The players data list combines the data of each track and pose pair with the data from the ball's track
```

Appendix


File Edit Selection View Go Run Terminal Help
main.py - Visual Studio Code
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
main.py
C: > Users > traub > PycharmProjects > FinalProject > main.py
Restricted Mode
0 0

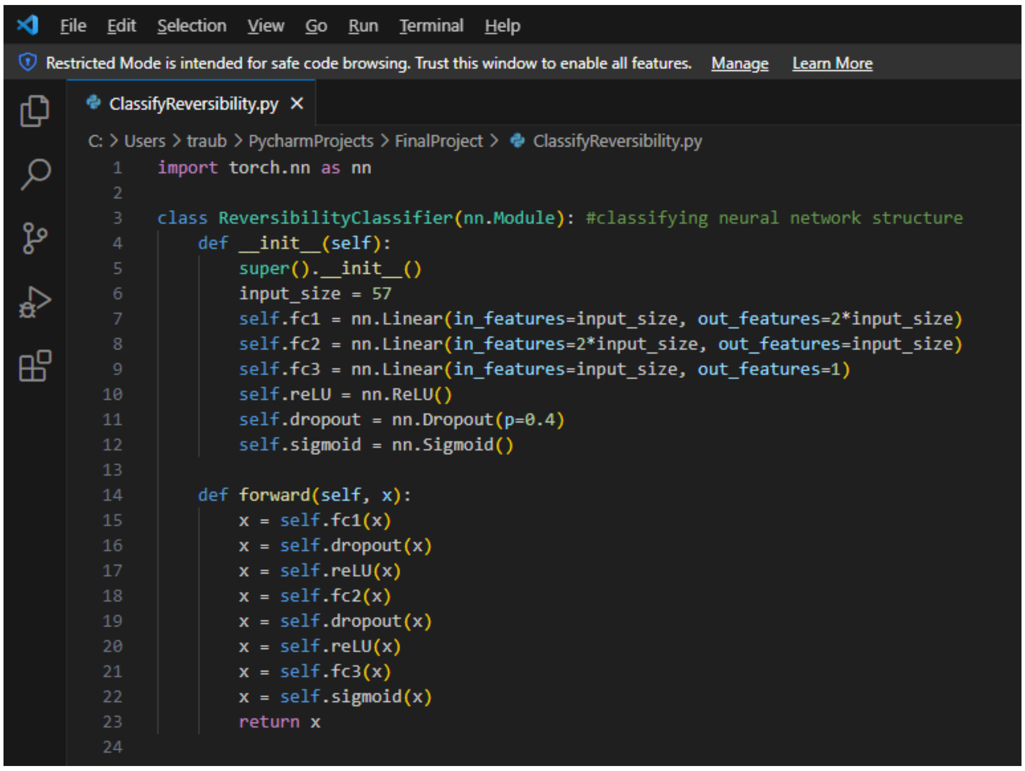
```python
            player_pose = player[0]
            """
            The players data list combines the data of each track and pose pair with the data from the ball's track
            and adds it to a list. Each of these will be input into a neural network
            (if that is the classification option chosen)
            """
            players_data.append(np.concatenate([player[0].flatten().cpu().numpy(), np.array(player[1]),ball_data]))
            if chosen_classification_method == "basic":
                predict_reversed, predict_not_reversed = basic_classify(player_pose,player[1],predict_reversed,predict_not_reversed)


        if len(players_data) > 1 and chosen_classification_method!="basic":
            NN_input = torch.tensor(norm_scale_pipeline.transform(players_data),
                                    dtype=torch.float32) #preprocesses input data
            NN_input = NN_input.to(device) #sends the input tensor to GPU
            output = classify_reverse(NN_input) #gets output from neural network
            NN_input.to("cpu")
            output = output.cpu()
            output = output.detach().numpy().tolist()
            for i in range(len(output)):
                player_NN_output = output[i][0]

                if player_NN_output > 0.5: #if predicted reversed, then calculate how confident it is and then multiply by velocity
                    predict_reversed += (player_NN_output - 0.5) * (players_data[i][-5]**2 + players_data[i][-6]**2)**0.5
                else: #if predicted not reversed, then calculate how confident it is and then multiply by velocity
                    predict_not_reversed += (0.5 - player_NN_output) * (players_data[i][-5]**2 + players_data[i][-6]**2)**0.5

    if predict_not_reversed + predict_reversed > 0: #if any pose/track pairs were able to be processed
        reversibility_score = predict_reversed / (predict_not_reversed + predict_reversed)
    else:
        reversibility_score = 0
    reversibility_score = round(reversibility_score,3)*10
    if reversibility_score >= 5:
        if reversed_actual:
            true_positives.append([video_path,reversibility_score])
        else:
            false_positives.append([video_path,reversibility_score])
    else:
        if reversed_actual:
            false_negatives.append([video_path,reversibility_score])
        else:
            true_negatives.append([video_path,reversibility_score])

classification_accuracy = (len(true_positives) + len(true_negatives)) / (n_videos)
print ("Classification accuracy: ", classification_accuracy)

print ("True Positives: ",true_positives)
print ("True Negatives: ",true_negatives)
```



```python
classification_accuracy = (len(true_positives) + len(true_negatives)) / (n_videos)
print ("Classification accuracy: ", classification_accuracy)

print ("True Positives: ",true_positives)
print ("True Negatives: ",true_negatives)
```

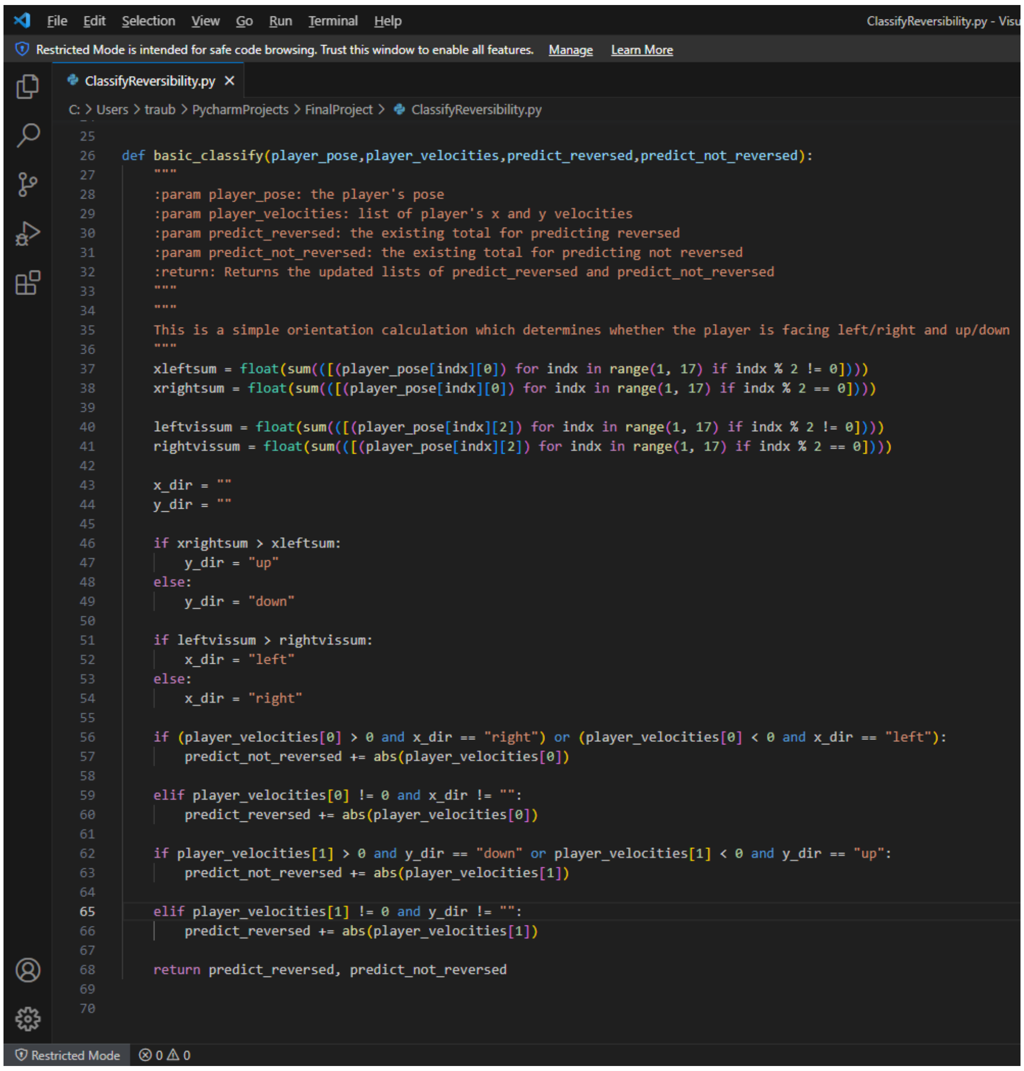
```python
print ("False Positives: ",false_positives)
print ("False Negatives: ",false_negatives)
```

8.2 ClassifyReversibility functions pages 36-37


File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
ClassifyReversibility.py
C: > Users > traub > PycharmProjects > FinalProject > ClassifyReversibility.py

```python
import torch.nn as nn

class ReversibilityClassifier(nn.Module): #classifying neural network structure
    def __init__(self):
        super().__init__()
        input_size = 57
        self.fc1 = nn.Linear(in_features=input_size, out_features=2*input_size)
        self.fc2 = nn.Linear(in_features=2*input_size, out_features=input_size)
        self.fc3 = nn.Linear(in_features=input_size, out_features=1)
        self.reLU = nn.ReLU()
        self.dropout = nn.Dropout(p=0.4)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.dropout(x)
        x = self.reLU(x)
        x = self.fc2(x)
        x = self.dropout(x)
        x = self.reLU(x)
        x = self.fc3(x)
        x = self.sigmoid(x)
        return x

```

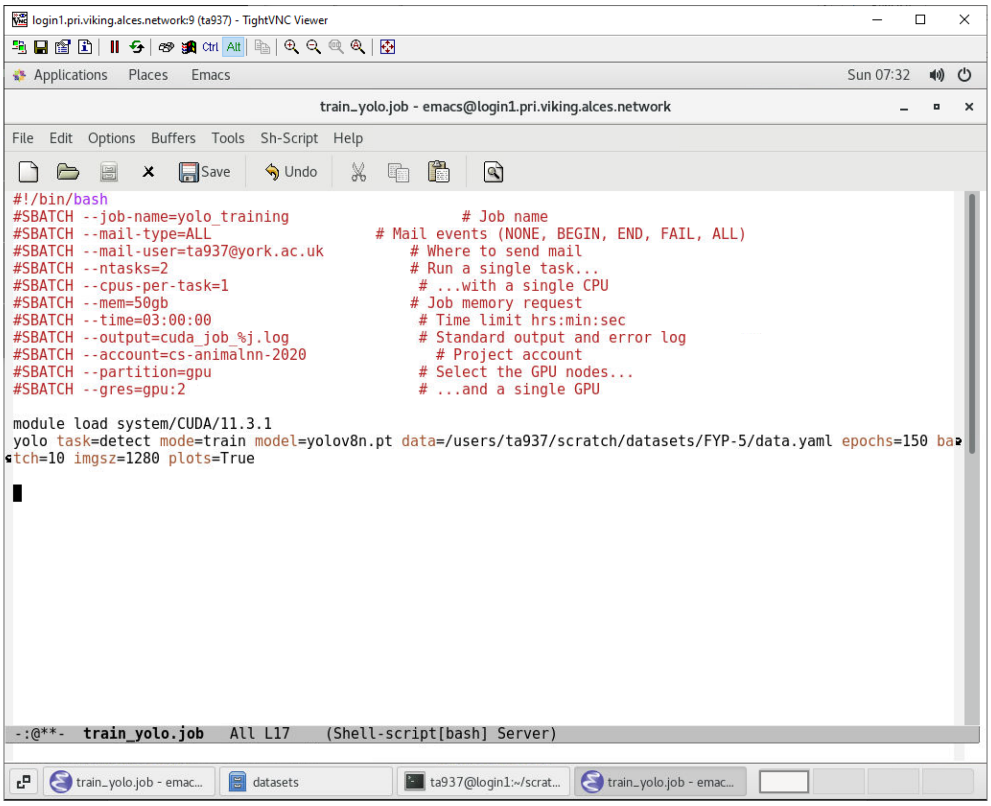
Appendix


File Edit Selection View Go Run Terminal Help
ClassifyReversibility.py - Visu
Manage
Learn More
ClassifyReversibility.py
C: > Users > traub > PycharmProjects > FinalProject > ClassifyReversibility.py
Restricted Mode
0 0

Restricted Mode is intended for safe code browsing. Trust this window to enable all features.

```python

def basic_classify(player_pose,player_velocities,predict_reversed,predict_not_reversed):
    """
    :param player_pose: the player's pose
    :param player_velocities: list of player's x and y velocities
    :param predict_reversed: the existing total for predicting reversed
    :param predict_not_reversed: the existing total for predicting not reversed
    :return: Returns the updated lists of predict_reversed and predict_not_reversed
    """
    """
    This is a simple orientation calculation which determines whether the player is facing left/right and up/down
    """
    xleftsum = float(sum(([(player_pose[indx][0]) for indx in range(1, 17) if indx % 2 != 0])))
    xrightsum = float(sum(([(player_pose[indx][0]) for indx in range(1, 17) if indx % 2 == 0])))

    leftvissum = float(sum(([(player_pose[indx][2]) for indx in range(1, 17) if indx % 2 != 0])))
    rightvissum = float(sum(([(player_pose[indx][2]) for indx in range(1, 17) if indx % 2 == 0])))

    x_dir = ""
    y_dir = ""

    if xrightsum > xleftsum:
        y_dir = "up"
    else:
        y_dir = "down"

    if leftvissum > rightvissum:
        x_dir = "left"
    else:
        x_dir = "right"

    if (player_velocities[0] > 0 and x_dir == "right") or (player_velocities[0] < 0 and x_dir == "left"):
        predict_not_reversed += abs(player_velocities[0])

    elif player_velocities[0] != 0 and x_dir != "":
        predict_reversed += abs(player_velocities[0])

    if player_velocities[1] > 0 and y_dir == "down" or player_velocities[1] < 0 and y_dir == "up":
        predict_not_reversed += abs(player_velocities[1])

    elif player_velocities[1] != 0 and y_dir != "":
        predict_reversed += abs(player_velocities[1])

    return predict_reversed, predict_not_reversed


```

8.3 Training YOLO model on Viking GPU cluster



```
#!/bin/bash
#SBATCH --job-name=yolo_training                          # Job name
#SBATCH --mail-type=ALL                          # Mail events (NONE, BEGIN, END, FAIL, ALL)
#SBATCH --mail-user=ta937@york.ac.uk                 # Where to send mail
#SBATCH --ntasks=2                                   # Run a single task...
#SBATCH --cpus-per-task=1                             # ...with a single CPU
#SBATCH --mem=50gb                                   # Job memory request
#SBATCH --time=03:00:00                               # Time limit hrs:min:sec
#SBATCH --output=cuda_job_%j.log                      # Standard output and error log
#SBATCH --account=cs-animalnn-2020                      # Project account
#SBATCH --partition=gpu                               # Select the GPU nodes...
#SBATCH --gres=gpu:2                                  # ...and a single GPU

module load system/CUDA/11.3.1
yolo task=detect mode=train model=yolov8n.pt data=/users/ta937/scratch/datasets/FYP-5/data.yaml epochs=150 ba
tch=10 imgsz=1280 plots=True
```

```
-:@**-  train_yolo.job    All L17    (Shell-script[bash] Server)
```

9 Bibliography

[1] “Exceptions to copyright,” GOV.UK. https://www.gov.uk/guidance/exceptions-to-copyright (accessed May 06, 2023).

[2] E. Pöppel, “A hierarchical model of temporal perception,” Trends Cogn. Sci., vol. 1, no. 2, pp. 56–61, May 1997.

[3] R. Cohendet, C.-H. Demarty, N. Q. K. Duong, and M. Engilberge, “VideoMem: Constructing, analyzing, predicting short-term and long-term video memorability,” arXiv [cs.CV], pp. 2531–2540, Dec. 05, 2018. Accessed: May 05, 2023. [Online]. Available: http://openaccess.thecvf.com/content_ICCV_2019/html/Cohendet_VideoMem_Constructing_Analyzing_Predicting_Short-Term_and_Long-Term_Video_Memorability_ICCV_2019_paper.html

[4] L. Wang et al., “Fallacious reversal of event-order during recall reveals memory reconstruction in rhesus monkeys,” Behav. Brain Res., vol. 394, p. 112830, Sep. 2020.

[5] J. Zobel, Writing for computer science, Springer, 2015. [1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 779–788.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580–587.

[7] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” arXiv [cs.CV], pp. 7263–7271, Dec. 25, 2016. Accessed: Apr. 25, 2023. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2017/html/Redmon_YOLO9000_Better_Faster_CVPR_2017_paper.html

[8] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” arXiv [cs.CV], Apr. 08, 2018. [Online]. Available: http://arxiv.org/abs/1804.02767

[9] G. Jocher, A. Chaurasia, and J. Qiu, “YOLO by Ultralytics. 2023,” URL: https://github. com/ultralytics/ultralytics, 2023.

[10] J. Terven and D. Cordova-Esparza, “A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond,” arXiv [cs.CV], Apr. 02, 2023. [Online]. Available: http://arxiv.org/abs/2304.00501

[11] W. Liu et al., “SSD: Single Shot MultiBox Detector,” in Computer Vision – ECCV 2016, 2016, pp. 21–37.

[12] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” arXiv [cs.CV], Jun. 04, 2015. Accessed: Apr. 25, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html

[13] N. Wojke, A. Bewley, and D. Paulus, “Simple online and realtime tracking with a deep association metric,” in 2017 IEEE International Conference on Image Processing (ICIP), Sep. 2017, pp. 3645–3649.

[14] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” in 2016 IEEE International Conference on Image Processing (ICIP), Sep. 2016, pp. 3464–3468.

[15] Y. Du et al., “StrongSORT: Make DeepSORT Great Again,” IEEE Trans. Multimedia, pp. 1–14, 2023.

[16] “StrongSORT: DeepSORT is back stronger! Upgraded tracking model!,” AI-SCHOLAR | AI: (Artificial Intelligence) Articles and technical information media, Dec. 07, 2022. https://ai-scholar.tech/en/articles/object-tracking/strongsort (accessed Apr. 27, 2023).

[17] Y. Zhang et al., “ByteTrack: Multi-object Tracking by Associating Every Detection Box,” in Computer Vision – ECCV 2022, 2022, pp. 1–21.

[18] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, “OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 1, pp. 172–186, Jan. 2021.

[19] A. Kendall, M. Grimes, and R. Cipolla, “PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization,” arXiv [cs.CV], pp. 2938–2946, May 27, 2015. Accessed: Apr. 27, 2023. [Online]. Available: http://openaccess.thecvf.com/content_iccv_2015/html/Kendall_PoseNet_A_Convolutional_ICCV_2015_paper.html

[20] G. Boesch, "The Complete Guide to OpenPose in 2023," viso.ai, Jan. 01, 2023. https://viso.ai/deep-learning/openpose/ (accessed Apr. 27, 2023).

[21] "Papers with code - MOT17 benchmark (multi-object tracking)." https://paperswithcode.com/sota/multi-object-tracking-on-mot17 (accessed Apr. 27, 2023).

[22] K. Nar, strongsort-pip: StrongSort-Pip: Packaged version of StrongSort. Github. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/kadirnar/strongsort-pip

[23] StrongSORT: StrongSORT: Make DeepSORT Great Again. Github. Accessed: Apr. 27, 2023. [Online]. Available: https://github.com/dyhBUPT/StrongSORT

[24] "Model Zoo — torchreid 1.4.0 documentation." https://kaiyangzhou.github.io/deep-person-reid/MODEL_ZOO (accessed Apr. 27, 2023).

[25] G. Bradski, "The openCV library," Dr. Dobb's Journal: Software Tools for the Professional Programmer, vol. 25, no. 11, pp. 120–123, 2000.

[26] Detectron. Github. Accessed: Apr. 30, 2023. [Online]. Available: https://github.com/facebookresearch/Detectron/issues/640

[27] The Emirates FA Cup. Accessed: Apr. 30, 2023. [Online Video]. Available: https://www.youtube.com/channel/UCChcWqwYXCEs657MQ00qVWA

[28] Dwyer, B., Nelson, J. (2022), Solawetz, J., et. al. Roboflow (Version 1.0) [Software]. Available from https://roboflow.com. computer vision.

[29] P. Skalski, "Train YOLOv8 on a custom dataset," Roboflow Blog, Jan. 10, 2023. https://blog.roboflow.com/how-to-train-yolov8-on-a-custom-dataset/ (accessed Apr. 30, 2023).

[30] FYP Dataset . Roboflow , 2023. [Online]. Available: %20https://universe.roboflow.com/fyp-v3pnw/fyp-amjew%20

[31] S. Saxena, “Precision vs recall,” *Medium*, May 11, 2018. https://medium.com/@shrutisaxena0617/precision-vs-recall-386cf9f89488 (accessed May 06, 2023).

[32] ultralytics. Github. Accessed: Apr. 30, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/189

[33] “OpenCV: Feature Detection.” https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html (accessed Apr. 30, 2023).

[34] “OpenCV: Optical Flow.” https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html (accessed Apr. 30, 2023).

[35] D. Rs, “#014 Calculating Sparse Optical flow using Lucas Kanade method,” Master Data Science, May 31, 2021. https://datahacker.rs/calculating-sparse-optical-flow-using-lucas-kanade-method/ (accessed Apr. 30, 2023).

[36] J. Shi and Tomasi, “Good features to track,” in 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, Jun. 1994, pp. 593–600.

[37] D. Biswas, “Lucas–Kanade method for optical flow,” Medium, Jun. 27, 2021. https://dibyendu-biswas.medium.com/lucas-kanade-method-for-optical-flow-87ea48dd3e69 (accessed Apr. 30, 2023).

[38] “Neural Networks — PyTorch Tutorials 2.0.0+cu117 documentation.” https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html (accessed May 02, 2023).

[39] A. Paszke et al., “PyTorch: An imperative style, high-performance deep learning library,” arXiv [cs.LG], Dec. 03, 2019. Accessed: May 02, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[40] The Emirates FA Cup, “FULL MATCH | Liverpool 3-1 Manchester City | FA Community Shield 2022-23,” Oct. 16, 2022. https://www.youtube.com/watch?v=cTqY53zWypk (accessed May 07, 2023).

[41] The Emirates FA Cup, “FULL MATCH | Brighton & Hove Albion 2-1 Liverpool | Fourth Round | Emirates FA Cup 2022-23,” Jan. 30, 2023. https://www.youtube.com/watch?v=T1yhBv1ytzw (accessed May 07, 2023).

[42] J. Deschenaux, “[Updated for YOLOv8] How robust are pre-trained object detection ML models like YOLO or DETR?,” Lakera – Protecting computer vision teams that disrupt the world, Jan. 26, 2023. https://www.lakera.ai/insights/pre-trained-poor-generalization (accessed May 06, 2023).