Temporal Localization of Group Activities in Football Videos

Master Thesis

Author
Jakob Thirup Fahl

Supervisors
Georgios Arvanitidis & Dimitrios Papadopoulos

March 15, 2024

DTU Compute
Department of Applied Mathematics and Computer Science

Abstract

Understanding the distinct phases of play in football games is crucial for deciphering opponent tactics, enabling managers and coaches to analyse the styles of play exhibited by rival teams. This knowledge is used for developing the best possible game plans designed to maximize team performance and exploit opponent weaknesses. However, the current manual process of labeling these phases in football games is laborious and time-consuming. In this thesis, we propose a novel machine learning approach for efficiently localizing and classifying group activities in football videos. Our solution comprises a two-stage methodology, featuring a video classification model and a transformer-based model. Remarkably, our video classification model surpasses human experts in classifying the phase of play, achieving an accuracy of 75% compared to 70.83% on a randomly selected subset consisting of 72 8-second clips of football games. Leveraging sequences of features from consecutive clips, extracted via the video classification model, our transformer-based model further enhances performance. Our two-stage model is able to classify a full 90-minute football game in less than six minutes, a fraction of the time required for manual labeling. In summary, we introduce a framework capable of accurately predicting the phase of play throughout a football game, significantly reducing the time required compared to current manual annotation methods.

Acknowledgements

I would like to extend a huge thanks to Dansk Boldspil-Union (DBU) for providing me with their data and inspiring the idea for this thesis. Without them, this thesis would not have been possible. I would also like to express my sincere gratitude to my supervisors, Associate Professors Georgios Arvanitidis and Dimitrios Papadopoulos, for their constant guidance and insightful feedback. Their expertise has been invaluable throughout this journey. I am incredibly grateful to everyone who took the time to complete the survey. Lastly, a big thank you to the DTU Computing Center for their High-Performance Computing services, which made the computational aspects of this thesis much smoother.

Preface

The present MSc thesis is a partial fulfillment of the requirements for the Master of Science degree in Human-Centered Artificial Intelligence at Technical University of Denmark.

The project accounts for 35 ECTS and is conducted between 15.09.2023 and 15.03.2024. The exact topic along with the objectives of the project are defined in collaboration between the author and the supervisors.

Kongens Lyngby, March 15, 2024

Jakob Thirup Fahl (s184419)

Contents

List of Figures

List of Tables

Abbreviations

Abbreviation	Description
VI	Danish for ”us” - used to denote the phases of play where the team attacking from left to right is in possession of the ball
DE	Danish for ”them” - used to denote the phases of play where the team attacking from right to left is in possession of the ball
CNN	Convolutional Neural Network
DBU	Dansk Boldspil-Union / Danish Football Association
DETR	Detection Transformer
FNN	Feedforward Neural Network
HPC	High Performance Computing
IOU	Intersection Over Union
MLP	Multi-Layer Perceptrons
NLP	Natural Language Processing
NN	Neural Network
R-CNN	Region-based Convolutional Neural Network
ReLU	Rectified Linear Unit
ResNet	Residual Network
TSN	Temporal Segment Network
ViT	Vision Transformer
YOLO	You Only Look Once - object detection model

CHAPTER 1

Introduction

1.1 Motivation

Football is the most watched sport globally[1], and the ever-increasing amount of followers drives teams to achieve the best possible results on the pitch. This, combined with the increased amount of money being pumped into football by TV-companies and sponsors[2], has provided teams with more incentive and resources for improvement. In recent years, a lot of these resources have been put into data science and machine learning approaches to analyse player and team performances and better understand the game.

One particular area where machine learning can be used to better understand a football game is detecting the phases of play throughout a game. Phases of play are used to describe which stage a game is in, based on the positions of the players on the pitch. For example, a phase could be when a team is attacking the opponent's goal, or when a team is trying to win the ball back high up the pitch. Analysing the distinct phases of play can provide valuable feedback for coaches about their own team, as well as crucial information about the style of play of rival teams.

1.2 Problem Statement

Currently, detecting phases of play is done manually, which is a timely process as it requires looking through over 90 minutes of footage for every single game and labeling the phases of play. In tournaments such as the FIFA World Cup[3], teams progressing to the knockout stages do not know their opponents until a few days before they have to play each other. Accelerating the phase detection process would furnish teams with a significant advantage, allowing for more comprehensive opponent analysis and strategic preparation within tight timelines.

1.3 Goals and Solutions

This thesis aims to address the aforementioned challenges. By leveraging machine learning techniques, we aim to streamline the process of phase detection, enabling faster and more accurate analysis of football games. As phases of play depend on the positions of all players on the pitch, they can be considered group activities. Hence, this thesis seeks to answer the research question:

"How can machine learning approaches be effectively applied to temporally localize and classify group activities in football videos?"

Our solution involves a two-stage approach consisting of a video classification model followed by a transformer-based model. The video classification model classifies video segments into distinct game phases, while the transformer-based model further refines these by looking at sequences of predictions.

1.4 Scope and Limitations

The available data provided by the Danish Football Association (DBU) comprises only games of professional men's senior national teams, thereby limiting the scope of this thesis. While this does not necessarily imply that our proposed solution will not generalize to other levels of play, such as women's or amateur games, it may lead to compromised performances due to differences in tactics, pace, etc. Additionally, all game footage used in this study was captured using a single camera setup, thus restricting us to games captured with this same setup.

1.5 Main Contributions

Our main contribution consists of a two-stage solution to the aforementioned research question, comprising a video classification model followed by a transformer-based model. In terms of detecting the phase of play in 8-second clips of football games, our video classification model surpasses human experts, achieving an accuracy of 75% compared to 70.83% on a randomly selected subset of these clips. Leveraging sequences of features from consecutive clips, extracted via the video classification model, our transformer-based model further enhances performance.

1.6 Thesis Outline

In Chapter 2, we delve into key machine learning concepts essential for this thesis, followed by an exploration of related works in Chapter 3. Chapter 4 provides an overview of the datasets utilized in this study, accompanied by initial descriptive statistics. Our proposed solution's design is outlined in Chapter 5, including the rationale behind our decisions. Chapter 6 details the experiments conducted and

presents the results obtained, while Chapter 7 provides a discussion of these results. We draw our conclusions in Chapter 8, before delving into potential avenues for future improvement in Chapter 9.

CHAPTER 2

Background and Theory

This chapter provides an overview of the theory and concepts employed in this thesis. It begins by covering general machine learning topics. Section 2.1 delves into deep neural networks, describing feed-forward neural networks (FNNs), convolutional neural networks (CNNs) and residual networks (ResNets) respectively, as these serve as fundamental building blocks for many advanced models. Further, Section 2.2 elaborates on transformer models' theory and advancements, including their adaptation to visual tasks. The chapter subsequently delves into advancements within relevant computer vision tasks, accompanied by key papers within the respective fields. Specifically, section 2.3 explores image classification, while section 2.4 elaborates on object detection theory. Moreover, section 2.5 delves into video classification, followed by section 2.6, which discusses temporal action localization.

2.1 Neural Networks (NNs)

Neural networks (NNs) are a class of machine learning models inspired by the structure and functioning of the human brain. They consist of interconnected neurons (or nodes) organized into layers, with each neuron performing a simple computation and passing its output to other neurons in the network. In this section, several types of neural network architectures are described.

2.1.1 Feed-Forward Neural Networks (FNNs)

Feed-forward neural networks, also known as multi-layer perceptrons (MLPs), represent the simplest form of neural networks. In FNNs, information flows from the input layer through one or more hidden layers to the output layer. Each neuron in one layer is connected to every neuron in the next layer, and each connection has an associated weight that is optimized during training through the process of backpropagation, aiming to minimize the difference between the predicted outputs and the true outputs for a given training set. A visualisation of a FNN can be

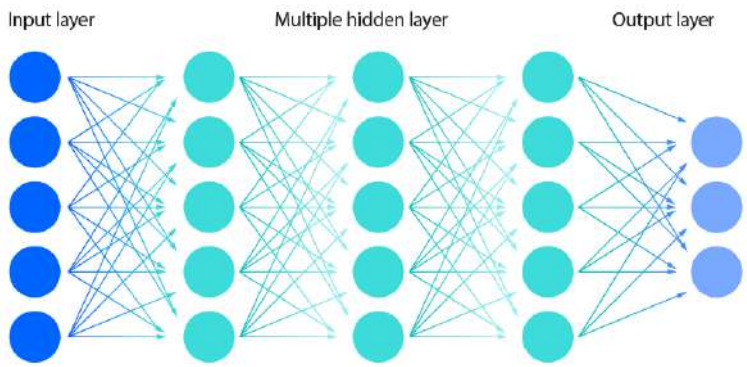


Figure 2.1: Feed-forward neural network (FNN) with 3 hidden layers. [4]

seen in Figure 2.1.

2.1.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs)[5] are a type of deep neural network primarily used for processing and analysing two-dimensional data, such as images. CNNs are mainly constructed from three types of layers: Convolutional layers, pooling layers and fully-connected layers.

Convolutional layers: These are the main building blocks of a CNN, and consist of kernels or filters which are applied to the input. In the case of a gray-scale image, a 3x3 kernel is applied by systematically traversing each pixel in the input image. At each pixel location, a convolution operation is performed by taking the sum of the element-wise multiplication of the 3x3 neighborhood surrounding the pixel with the corresponding elements of the kernel. This can be seen in Figure 2.2.

Pooling layers: These are typically employed following a convolutional layer, and they commonly utilize two primary types of pooling methods: Max-pooling and average-pooling. In the case of 2x2 max-pooling, the operation systematically scans the input data in 2x2 regions at a time and outputs the maximum pixel value found within each of these regions. This process effectively reduces the spatial dimensions of the data, aiding in dimensionality reduction and feature selection. This can also be seen in Figure 2.2.

Fully-connected layers: These are like the layers in feed-forward neural networks[4] in that all input neurons (pixels) are connected to all output neurons. Fully-connected layers are usually applied at the end of CNNs, and typically followed by a softmax activation function to classify inputs appropriately.

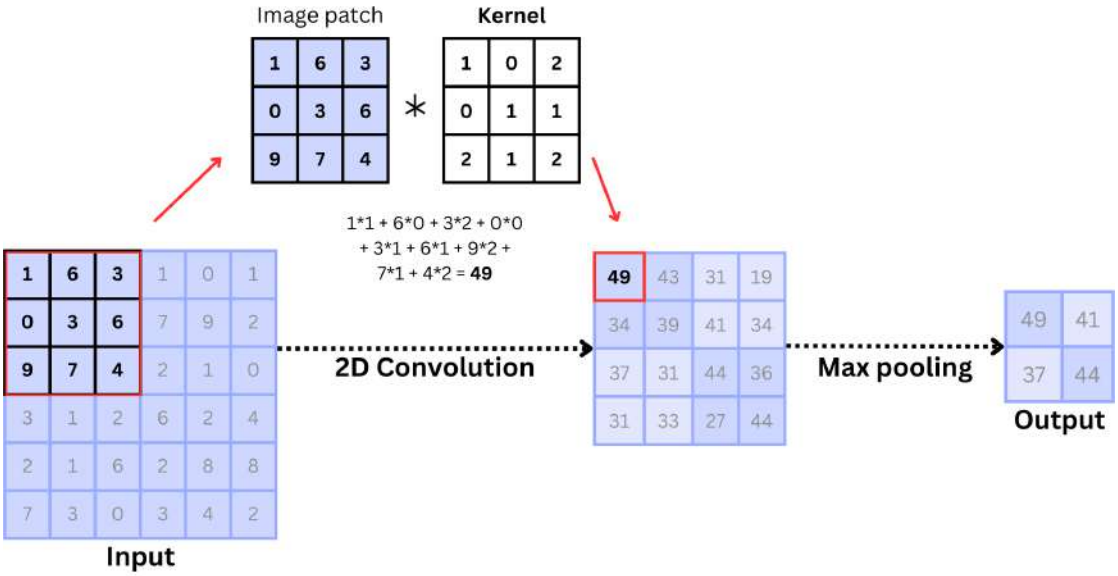


Figure 2.2: Convolution operation of a 3x3 kernel on a 6x6 image, followed by a 2x2 max pooling with a stride of 2.

2.1.3 Residual Networks (ResNets)

Residual Networks (ResNets), introduced by Kaiming He et al.[6], are deep neural network architectures designed to tackle the challenge of vanishing gradients encountered in training very deep networks. ResNets are comprised of multiple residual blocks, each consisting of convolutional layers followed by a Rectified Linear Unit (ReLU) activation function and a skip connection, also known as an identity shortcut. This skip connection allows the network to learn residual functions with respect to the layer inputs, making it easier to train very deep networks.

The architecture of a residual block is depicted in Figure 2.3. By integrating skip connections, ResNets ensure smooth gradient flow during backpropagation, effectively addressing the vanishing gradient problem. Consequently, ResNets enable the training of deeper neural networks with improved performance. Modern ResNets used as backbones for advanced models typically consist of up to 50, 101 or even 152 layers.

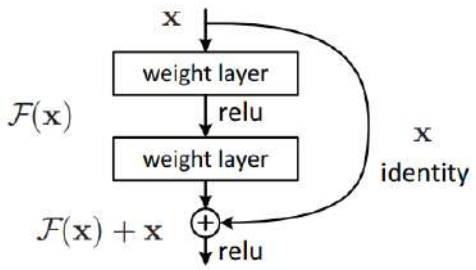


Figure 2.3: Residual block. [6]

2.2 Transformers

Transformers were initially proposed for Natural Language Processing (NLP) tasks in the 2017 paper "Attention Is All You Need" by A. Vaswani et al. [7], where they introduced a new model architecture based solely on attention mechanisms. Their proposed attention mechanism, Scaled Dot-Product Attention, takes as input queries, Q, and keys, K, of dimension d_k and values, V, of dimension d_v, where each of Q, K and V are vectors computed by multiplying the input by learned linear projections. The attention is then computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.2.1}$$

To enable the model to learn distinct positional representations within a sequence, the Multi-Head Attention mechanism is employed. This is achieved by creating multiple distinct sets of queries, keys and values through different learned linear projections. The Scaled Dot-Product Attention function (2.2.1) is then performed on these sets in parallel, and the resulting outputs are concatenated and projected to obtain the final representation

The complete architecture of the Transformer is visualized in Figure 2.4. The encoder is a stack of $N = 6$ identical layers consisting of a Multi-Head Attention layer and a feed-forward layer. Both layers have a residual connection and a normalisation layer. The input to the first encoder layer is the input embedding, and for the following layers the input is the output of the previous layer. The decoder (right) is also a stack of $N = 6$ identical layers. Each layer has the same two sub-layers as the encoder, but also a third sub-layer which performs multi-head attention over the output of the encoder. It is finished off with a linear layer and a softmax activation to get the output probabilities.

While transformers were initially designed for NLP tasks, they have become state-of-the-art in many other tasks such as computer vision tasks[8, 9]. For image classification, Vision Transformers (ViT)[10] reshape 2D images into a sequence of flattened 2D non-overlapping patches, which are then used as input for the transformer. For object detection, Detection Transformers (DETR) are composed of "a CNN backbone to extract a compact feature representation, an encoder-decoder transformer, and a simple feed forward network (FFN) that makes the final detection prediction." as stated in the paper "End-to-end object detection with transformers" by N. Carion et al.[11]. These DETR models are outperforming the CNN-based models and have become state-of-the-art models for object detection[9, 11, 12, 13]. Also when it comes to video classification are transformers currently the go-to models[14, 15, 16, 17].

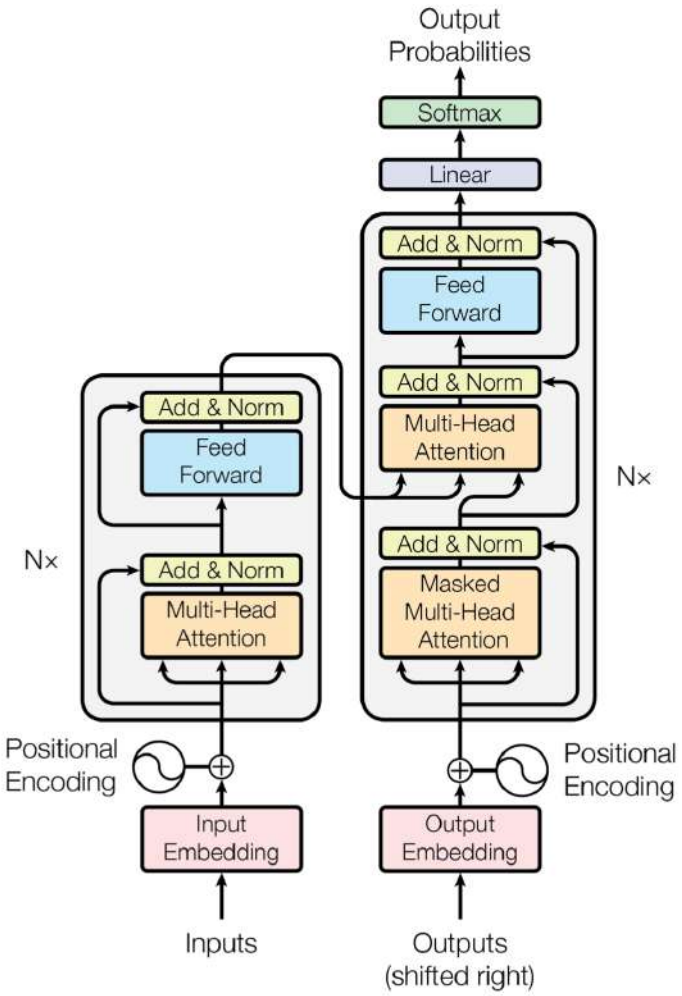


Figure 2.4: Transformer architecture from A. Vaswani et al.[7]. The inputs are embedded and fed through Nx encoder layers with multi-head attention and a feed-forward layer (left). The outputs are then fed through Nx decoder layers with multi-head attention, a feed-forward layer and a masked multi-head attention layer, followed by a linear layer and a softmax activation function (right).

2.3 Image Classification

Image classification is the task of assigning a label or class to an input image. The objective is to train a model to accurately predict the correct class for an input image from a predefined set of classes, such as [*player*, *goalkeeper*, *referee*, *ball*]. In 1998, Y. Lecun et al. introduced LeNet[18], a Convolutional Neural Network (CNN) model designed for image classification. LeNet, a pioneering model in this domain, continues to influence many state-of-the-art models.

The architecture of a CNN image classification model typically consists of multiple convolutional layers with possible pooling layers and activation functions in be-

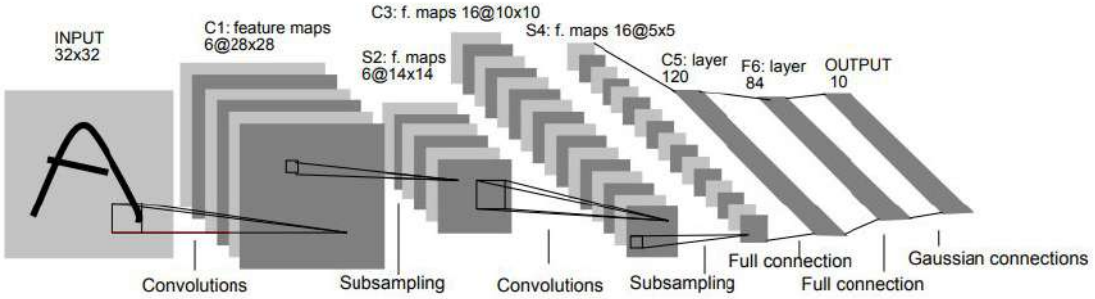


Figure 2.5: LeNet architecture from the original paper[18]. It processes a 32x32 image through two convolutional layers, each followed by average pooling. The output is flattened and passed through two fully-connected layers, and a final 10-neuron layer for digit classification using softmax activation.

tween. Subsequently, one or more fully-connected layers follow, leading to a final softmax layer for class probability predictions. The architecture of LeNet can be seen in Figure 2.5.

As the amount of computer power available increased, several authors built on top of LeNet and achieved even better results with larger CNNs. Models such as AlexNet[19] increased the size of the input image and layers, while other models such as VGG[20] and GoogLeNet[21] increased the number of layers. Another ground-breaking innovation came with ResNet[6] in 2015, where identity shortcuts were introduced to the outputs of each layer. This led to new state-of-the-art results, and today ResNet models are often used as backbones in object detection models, semantic segmentation models, video classification models and many more.

Furthermore, transformers, initially designed for Natural Language Processing (NLP) tasks, have become state-of-the-art models in image classification tasks as well. Vision Transformers (ViT)[10] reshape 2D images into a sequence of flattened 2D non-overlapping patches, which are then used as input for the transformer. This approach has shown remarkable performance in image classification tasks, rivaling traditional CNN-based models.

2.4 Object Detection and Tracking

Object detection is the computer vision task of locating objects within images or videos. It entails finding bounding boxes around the objects as well as classifying them. Before the deep learning era, object detection was done using hand-crafted features, such as HOG[22] and feature pyramids. With the advances of convolutional neural networks (CNNs) in image classification tasks, people started experimenting with these CNNs for object detection, which led to these models becoming state-of-the-art in this area. These CNN object detection models can be split into

two general categories: two-stage detectors and one-stage detectors.

Two-stage detectors: Also known as region-based methods, these were made popular in 2013 by R. Girshick et al.[23] when they introduced a Region-Based Convolutional Neural Network (R-CNN), an approach with two stages. The first stage finds regions which potentially contain an object by grouping together similar pixels since they most likely are of the same object. The second stage then feeds these regions into a CNN image classifier to classify what the object is. This original R-CNN model was quite slow, so to combat this Girshick and his team later introduced Fast R-CNN[24] and Faster R-CNN[25].

One-stage detectors: These combine the region proposal and classification into one single CNN. Since these methods only require one forward pass, they are generally faster than two-stage methods, and can even provide real-time object detection. However, the trade-off for this increase in speed is less accuracy compared to the slower two-stage detectors. In 2015, Redmon et al. introduced YOLOv1[26], which is a one-stage detector that uses dense predictions, that is it tries to predict classes and bounding boxes for all possible positions in the image. Several improvements have been made to the YOLO model throughout the years[27, 28, 29, 30, 31]. Particularly Ultralytic's YOLOv8[31] performs well, especially given its ability to run in real-time. The YOLOv8 architecture can be seen in Appendix A.1.

Within the last couple of years, transformer models have also become state-of-the-art models in object detection. Detection Transformers (DETR) are one such example. They are composed of "a CNN backbone to extract a compact feature representation, an encoder-decoder transformer, and a simple feed forward network (FFN) that makes the final detection prediction." as stated in the paper "End-to-end object detection with transformers" by N. Carion et al.[11]. These DETR models outperform traditional CNN-based models and have become state-of-the-art models for object detection tasks[9, 11, 12, 13].

Multi-object tracking is the task of tracking one or more objects through consecutive frames. It builds on top of object detection in that it still needs to detect objects, but an additional requirement is now that it needs to keep track of the identities of detected objects throughout the frames. Multi-object tracking models are usually split up into two stages: detection and tracking. The detection is done using an object detection model, e.g. YOLOv8 [31]. The tracking algorithm then tries to link the objects in consecutive frames, e.g. by using the Kalman filter to predict the next position of each object based on its previous trajectory and velocity. Online multi-object trackers [32, 33, 34, 35] process data as it becomes available, i.e., they do not have access to future data. This is useful for analysing real-time videos, but can make it harder to track than for offline trackers [36, 37].

2.5 Video Classification

Video classification is the task of assigning a label or class to an input video. Similar to image classification, the goal of video classification is to train a model capable of accurately predicting the correct class for an input video from a predefined set of classes, such as [*walking*, *running*, *jumping*]. Compared to still images. videos can be very data heavy. Assuming a simple 8-second RGB video with a frame rate of 25 frames per second and a resolution of 640x640 pixels, the total number of pixels in the video clip (N) can be calculated as:

$$N = 8 \cdot 25 \cdot 640 \cdot 640 \cdot 3 = 245,760,000 \text{ pixels} \tag{2.5.1}$$

The substantial volume of data in just a short video clip underscores the computational challenges inherent in video classification. Consequently, models that take all consecutive frames of the clips as input are difficult to train. To address these challenges, recent advances in deep learning architectures have played a pivotal role. Karen Simonyan and Andrew Zisserman[38] proposed a two-stream method, where one stream focused on single frames and the other on optical flow. This also allowed the models to take advantage of the existing image classification models that have been pre-trained on large datasets. Several researches have built on top of this foundation, by using things such as 3D convolutions[39, 40] and sampling high resolution frames at low frame rates and low resolution frames at high frame rates[41]. Another approach is to select a few frames from the clips at various intervals, as L. Wang et al.[42] investigated with their temporal segment networks (TSN).

Vision Transformers (ViT) have proved to be powerful tools for not just image classification but also video classification tasks. ViTs operate by reshaping 2D images from each frame of a video into a sequence of flattened 2D non-overlapping patches. These patches serve as input for the transformer model. Leveraging the self-attention mechanism, ViTs excel at capturing global dependencies across the video frames effectively. Building on top of ViT[10], Arnab et al. [14] introduced ViViT, a vision transformer designed explicitly for video understanding tasks. Similarly, Bertasius et al. [15] proposed SpaceTime ViT, which extends ViTs to exploit spatiotemporal information in videos. Tong et al. [16] and Wang et al. [17] also showcased the effectiveness of ViTs in video classification by employing methods like VideoMAE.

2.6 Temporal Action Localization

Temporal action localization is the task of localizing events or actions within videos. Unlike video classification, where models predict the label of a video, temporal action localization models predict the start and end times, as well as the labels, of all actions occurring within a video. Methods for temporal action localization

can be roughly divided into two categories: anchor-based and anchor-free methods.

Anchor-based methods: First introduced by Wang et al. [42] in 2016, they work by generating predefined anchor boxes at different temporal scales, similar to object detection in images, and applying them at multiple timesteps within the videos using a sliding window approach. These anchor boxes are then scored based on their likelihood of containing an action, and high-scoring segments are then classified. Later works have improved the anchor-based methods by using different features to represent the videos, such as 3D convolutional features [43], graph convolutional features [44] and transformer features [45].

Anchor-free methods: These directly predict both the temporal boundaries and class of the actions without using anchors and were first proposed by Lin et al. [46] in 2018 to the task of temporal action proposal generation, and were later extended to the task of temporal action localization [47, 48, 49]. These methods typically employ regression-based approaches to directly estimate the temporal boundaries of actions without the need for predefined anchor boxes. Anchor-free methods are often more flexible and efficient compared to anchor-based methods since they do not require designing and tuning anchor boxes.

CHAPTER 3

Related Work

The previous chapter explained the theory behind some of the methods and models used in this thesis, as well as provided some of the pioneering papers within these topics. This chapter provides an overview of work more closely related to the thesis topic of *"Temporal Localization of Group Activities in Football Videos"*.

SoccerNet-v2: One dataset that has been a base for many similar projects is the SoccerNet-v2 dataset[50], which is a large-scale dataset for video understanding tasks in football. In addition to providing this impressive dataset they also create yearly challenges[51, 52] in topics such as player and ball tracking, action spotting and camera calibration.

Player and Ball Tracking: There has been several works in this topic, using multi-object tracking to detect the players and ball[53, 54, 55, 56]. The third-placed team[57] in the SoccerNet tracking challenge, utilized a separate ball-tracker in addition to the player tracker, as the ball is harder to track due to its small size and fast movement. Other works utilize several calibrated cameras for improved tracking results[58, 59].

Sport Field Registration: This is the task of mapping the sport field onto the images or video frames, by estimating the homography between the physical pitch and the frame space of the video. Previous works have implemented this to create a bird's-eye mapping for sports such as hockey[60, 61], basketball[62], and more relevantly, football[63].

Action Spotting: Action spotting in football is a critical task for understanding game dynamics. It involves identifying specific actions or events in a video sequence. The SoccerNet action spotting challenge provided a baseline and dataset for this task, which several works have expanded upon[51, 52, 64, 65, 66]. Due to the limited data available, some works have focused on active learning frameworks[67] and using tracking data[68].

Temporal Localization of Group Activities: This is the task of identifying and locating occurrences of various activities involving multiple individuals in a given video. Some works in this task include [69, 70]. In football, these group activities could be certain tactics or play phases. If available, tracking data can be used for this task[71].

Long-Form Video Understanding: This is the task of understanding videos that are longer than just a few seconds. It is a complex task as the amount of data in long videos is extremely large. This is an important task in football analysis, where understanding the progression and context over longer periods of time is essential. Some authors utilize a long-term feature bank spanning the entire duration of the video to aid short-term video understanding models[72]. Others have implemented transformer-based methods to tackle this problem [73, 74].

CHAPTER 4

Dataset Description

This chapter aims to provide an overview of the datasets used throughout this thesis. Additionally, we will describe the data preprocessing steps carried out. In section 4.1, we describe the data received from DBU, which consists of recordings of complete football games with phase annotations. Since this data does not contain any player or line markings, another dataset was used to train a player detection model. This dataset, SoccerNet-v2[50], is described in section 4.2.

4.1 DBU Data

This dataset contains recordings of 43 complete football games, captured with a single camera. The camera is located high up and to the side of the pitch, giving a view similar to watching a game on TV. With football games lasting 90 minutes, plus a few additional minutes, this dataset comprises over $43 \cdot 90 = 3870$ minutes of video, equating to 64.5 hours. The data includes annotations of the time periods for all phases of play, as well as events such as shots, goals, throw-ins, etc., for all games. While this thesis focuses solely on play phases, these additional events could be valuable for future projects.

4.1.1 The Play Phases

Play phases are used to define the stage of a game. In the scope of this thesis we will be working with nine unique phases: three attacking phases, three defending phases, two transition phases and a ”none” phase for when the game is in neither of the other phases. The phases are defined for the team attacking from left to right, and are designed such that when team A is in attacking phase 1, team B is in defending phase 1, and vice versa. Thus we only need to label the phases for the team attacking from left to right. For future reference, phases labeled as ”VI” followed by a number are when the team going from left to right is attacking and the other team defending, and vice versa for phases labeled ”DE” followed by a number. All of the phases are described below.

(a) VI 1

(b) VI 2

(c) VI 3

(d) DE 1

(e) DE 2

(f) DE 3

Figure 4.1: Examples of play phases. Figures a, b and c are examples of VI phases - that is when the team attacking from left to right has the ball. Figures d, e and f are DE phases which is when the other team is in possession of the ball.

Attacking / defending phase 1: In this phase, the attacking team has the ball in a non-dangerous position, usually in their own half, and the defending team is pressing the attacking team trying to win the ball back. This is usually the first phase after the attacking team has received the ball. Figure 4.1a shows an example of a game in phase "VI 1", that is the attacking phase 1 for the team attacking from left to right and defending phase 1 for the other team. Figure 4.1d shows an example of a game in phase "DE 1", where the team attacking from left to right is now defending in phase 1, and the other team is in attacking phase 1.

Attacking / defending phase 2: This is the second phase of an attack. In this phase the defending team has stopped pressing the attacking team high up and have dropped off to stand in a more compact defending shape. This means that the attacking team now usually has the ball higher up the pitch, but still not in a position that is threatening the defending team's goal. Figure 4.1b shows an example of a game in phase "VI 2", i.e. the attacking phase 2 for the team attacking left to right, and defending phase 2 for the other team. Figure 4.1e shows an example of a game in phase "DE 2".

Attacking / defending phase 3: This is the third and final attacking phase. This phase is when the attacking team finds a way past the defending team and gets into a dangerous position. This phase often ends up with the attacking team either having a shot, attempting a cross or having a chance to score in another way. Figure 4.1c shows an example of phase "VI 3", where the team attacking from left to right is in the third and final attacking phase, and the other team is in the third and final defending stage. Figure 4.1f shows an example of a game in phase "DE 3".

Transition phases: These phases are when one team loses the ball to the other team and they start a counter-attack. Phases labeled "VI-DE" represent situations where the team attacking from left to right loses the ball, while "DE-VI" phases represent situations where the team attacking from right to left loses the ball. Figure 4.2 shows a sequence of frames from a "VI-DE" phase. Note that these phases do not include when a team kicks the ball out of play and loses possession that way.

The "none" phase: This phase is when neither of the other phases are active. This is usually when the ball is out of play for e.g. a goal kick or a throw-in, or when there is a set piece such as a free-kick. It could also be in times when the game is chaotic and there is no real structure, for example after a goalkeeper has taken a goal-kick, and both teams are trying to win the ball. Examples of this phase are visualized in Figure 4.3.

Figure 4.2: Sequence of frames from a "VI-DE" transition phase. A player from the attacking team passes the ball (left) which goes to the opposition team (middle) who then start a counter attack (right).

(a) Corner (b) Free-kick (c) Goal-kick

Figure 4.3: Single-frame examples of the "none" phase. Figure (a) shows an example of a corner, Figure (b) shows an example of a free-kick and Figure (c) shows an example of a goal-kick.

4.1.2 Data Preprocessing

The data received for this project was given in the form of videos of full football games with corresponding xml files containing the timestamps of all play phases and events. To prepare this data for the video classification model, it needed to be split into short 8-second clips, which were obtained in two separate ways: Well-defined clips containing only one play phase and consecutive clips from full games. Both of these are defined below.

Well-defined clips: These clips were acquired by taking 8-second clips where the whole duration of the clip contains only a single play phase. This was achieved by going through all phases of a game that were at least 8 seconds long, and extracting as many non-overlapping 8-second sections as possible from each phase, as visualized in Figure 4.4. This was done to create well-defined clips from which the model is able to learn the unique characteristics of each play phase. These were the clips used to train the video classification model.

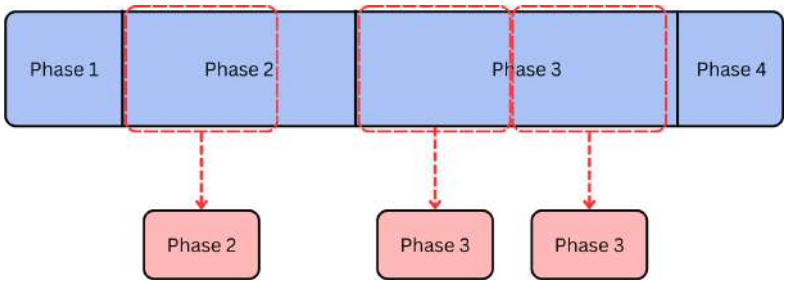


Figure 4.4: Extracting well-defined clips from a game. The large blue rectangle shows the timeline of the play phases of a section of a game, and the red rectangles show the extracted 8-second clips.

Consecutive clips: These clips were acquired by going through a full game and taking consecutive 8-second clips. This was achieved by extracting 8-second clips at 8-second intervals, starting from the beginning of the game, as visualized in Figure 4.5. Running the video classification model on these consecutive clips would then give the play phases of a full game. This was done in order to acquire the predicted phases of a full game, which could then be fed into the transformer-based model to get refined predictions. Some of these clips contained two or more play phases within them if there was a change of phase during the 8-second clip. In these cases the ground truth phase of the clip was set to the phase with the longest duration within the clip.

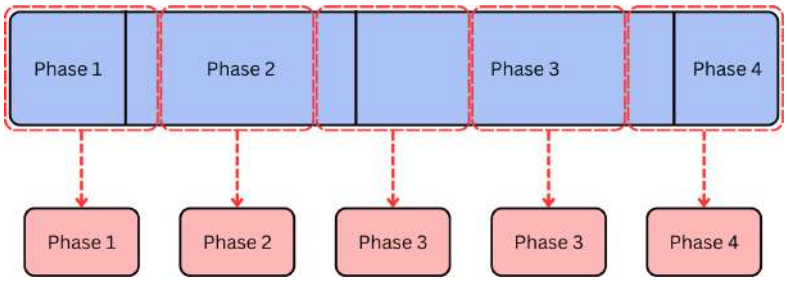


Figure 4.5: Extracting consecutive clips from a game. The large blue rectangle shows the timeline of the play phases of a section of a game, and the red rectangles show the extracted 8-second clips.

No further data preprocessing was required at this stage, since the video classification model itself further preprocessed the data in regards to the size, number of frames and sampling rate when sampling data.

4.1.3 Data Split

After preprocessing the data, it was split into three subsets: a training set, a validation set and a testing set. The data was divided in such a way that clips from any particular game would only appear in one of these sets, ensuring the

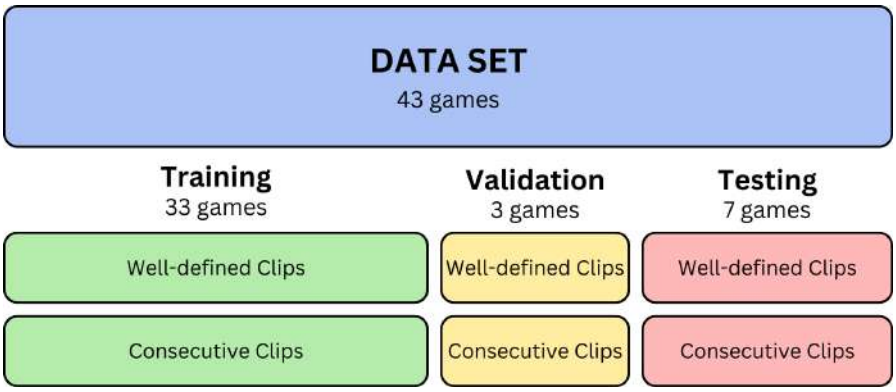


Figure 4.6: Data Split: Out of the 43 games, 33 are allocated for training, 3 for validation, and 7 for testing purposes. Each split comprises two sets: one consisting of well-defined clips and another consisting of consecutive clips.

model's performance evaluation on unseen game data. The training set was utilized for model training, the validation set for fine-tuning hyperparameters, and the testing set for evaluating model performance. Each subset comprised two sets of clips: one containing well-defined clips and another containing consecutive clips.

For the 43 games received from DBU, a split of 33-3-7 for training, validation, and testing respectively was chosen, as depicted in Figure 4.6. A larger testing set was chosen to enable testing on a diverse range of game scenarios. It is worth noting, that all games in the training and validation sets maintained a framerate between 25 and 30 fps, while some games in the testing set had a framerate of 50 fps. This variation allowed for the assessment of the model's performance across different framerate conditions.

4.1.4 Descriptive Statistics

In this section we will delve into some of the descriptive statistics of the dataset in order to try and better understand the data. The statistics in this section have been computed using a subset of the games consisting of the training and validation set games. We will be looking into the number of occurrences and average duration of each play phase, as well as the evolution of phases during a game.

Occurrences and Duration of Phases: Table 4.1 displays the total number of occurrences and average duration of each play phase. We can see that the average duration of the matching attacking and defending phases e.g. "VI 1" and "DE 1" are very similar for all four phases, which can be seen as an indicator that the phases are indeed the same. While there are a similar number of transition phases each way ("VI-DE" and "DE-VI") we can see that the "VI" team tends to have more attacking phases 2 and 3 than the "DE" team which has more attacking phase 1 phases. This could be due to the home advantage that is usually seen in football, where on average the home team has a higher tendency to win. Alterna-

Event	Occurrences	Average Duration (s)
VI 1	533	25.12
VI 2	895	32.32
VI 3	594	18.66
VI-DE	837	21.47
DE 1	797	25.40
DE 2	647	30.49
DE 3	435	16.65
DE-VI	819	20.79

Table 4.1: Occurrences and average duration of each phase of play.

tively, it could be due to human labeling inaccuracy, or simply a coincidence.

We can also see that phase 3 (”VI 3” and ”DE 3”), which is the final attacking phase, has the shortest average duration. This makes sense, since this phase occurs when the attacking team gets past the defending team, and thus these phases are likely to end quickly as the attacking team has an attempt at goal or gets tackled. The phase with the longest average duration is phase 2. This also makes sense, as this is a phase where the team in possession of the ball is neither about to finish their attack, nor being pressed hard by the opponent. Looking at the histograms of the duration of each phase as seen in Figure 4.7, we can see that there are several occurrences of phase 2 longer than 50 seconds.

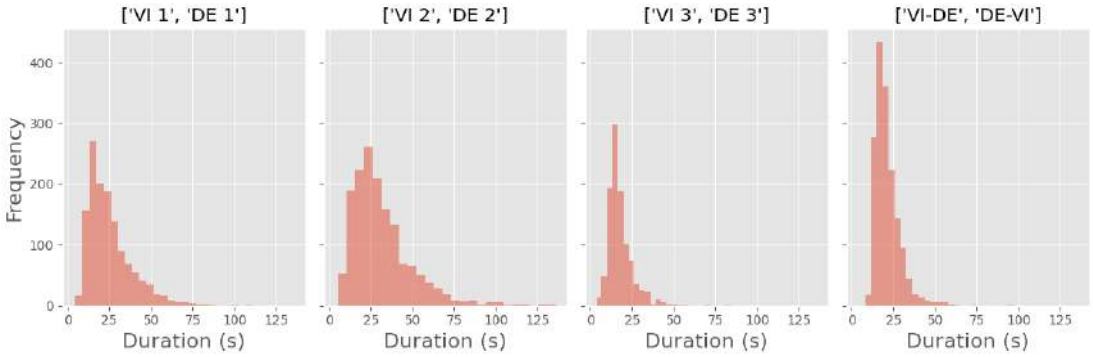


Figure 4.7: Histogram of duration of play phases. We see that the final attacking phases (VI 3 and DE 3), as well as the transition phases (VI-DE and DE-VI) are generally shorter than the other phases.

Phase Evolution: Another aspect to look at is the evolution of phases, i.e. which phases tend to follow each other. This can provide valuable information both for the analysis and understanding of the game, as well as the objective of this thesis - to create a model capable of predicting the phases of play throughout the duration of a football game. From Figure 4.8 it can be seen that a lot of the time after one of the phases it goes to a ”none” phase. This can be in cases where the ball goes

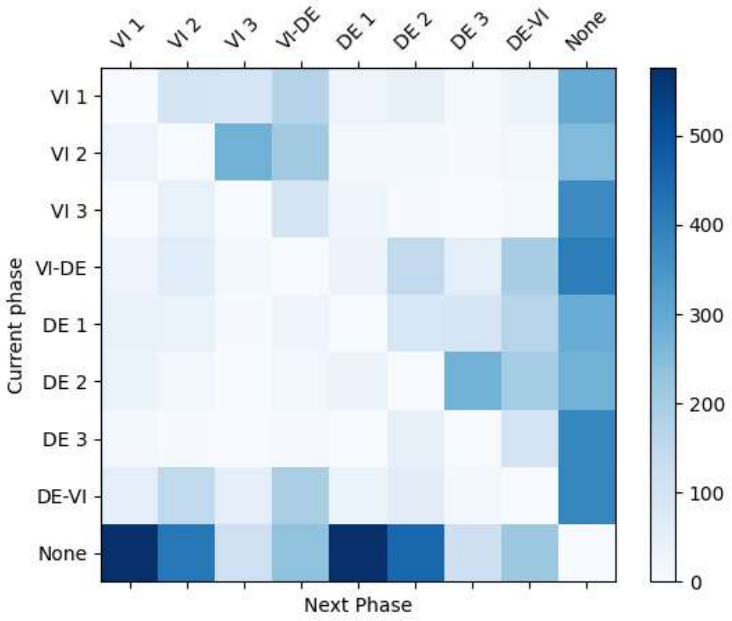


Figure 4.8: Evolution of play phases during football games. Most phases both follow and precede the ”none” phases, likely a cause of the ball going out of play. Furthermore, we also see a progression where ”VI 2” often follows ”VI 1” and ”VI 3” often follows ”VI 2”, mimicking an in-game progression of an attack. The same is seen for the ”DE” phases.

out for a throw-in, goal kick or corner, or when the other team commits a foul. We also see that the ”none” phase is most often followed by phase 1, and then phase 2. This is expected, as phase 1 is usually the start of an attack, while an attack could also start directly in phase 2 if the defending team isn't pressing high up the pitch. When looking at the ”VI” phases and ignoring when they go to the ”none” phase, we see that ”VI 1” often goes to either ”VI 2”, ”VI 3” or ”VI-DE” and that ”VI 2” usually goes to ”VI 3” or ”VI-DE” and ”VI 3” usually to ”VI-DE”. This seems to follow a logical progression that an attacking phase either leads to the next attacking phase, or the ball is lost. This is the same case for the ”DE” phases.

4.2 SoccerNet-v2 Dataset

The SoccerNet-v2[50] tracking dataset consists of 12 complete football games from the main camera including: 200 clips of 30 seconds with tracking data, one complete 45-minute halftime annotated with tracking data, and the complete videos for the 12 games. In this thesis we will be using a subset of this data comprising: 57 30-second clips for the training set and 49 30-second clips for the validation set. The clips are 25 fps, and are stored in the form of 750 frames with annotated tracking data. The image resolution is 1080p (1920x1080 pixels).

4.2.1 Data Preprocessing

The SoccerNet-v2[50] data was used to train an Ultralytics YOLOv8 [31] player detection model. To align the data with the YOLOv8 model's requirements, certain modifications were necessary. The images themselves remained unchanged and were simply relocated to designated training and validation directories.

The SoccerNet-v2 labels were all given in one big file for each 30-second clip, and the boxes defined using the top-right corner and the width and height, given in pixels. The YOLOv8 model requires that each image is accompanied by a corresponding labels file containing bounding box coordinates. These coordinates must be expressed as the bounding box's middle point, width, and height relative to the dimensions of the full image, ranging from 0 to 1. Thus, adjustments were needed to get the labels files in the correct format.

Additionally, we wanted to experiment with things such as removing the labels for balls and having separate classes for each team, so several iterations of labels files were made. Finally, since a lot of the images were very similar, an option to only use every tenth image was explored.

4.2.2 Data Augmentations

In order to increase the amount of training data without having to acquire new images and tracking data, several data augmentations were made, and are listed below:

Horizontal flips: Since an image of a football game still looks like a football game image when mirrored, this was an obvious option for a way to augment the data.

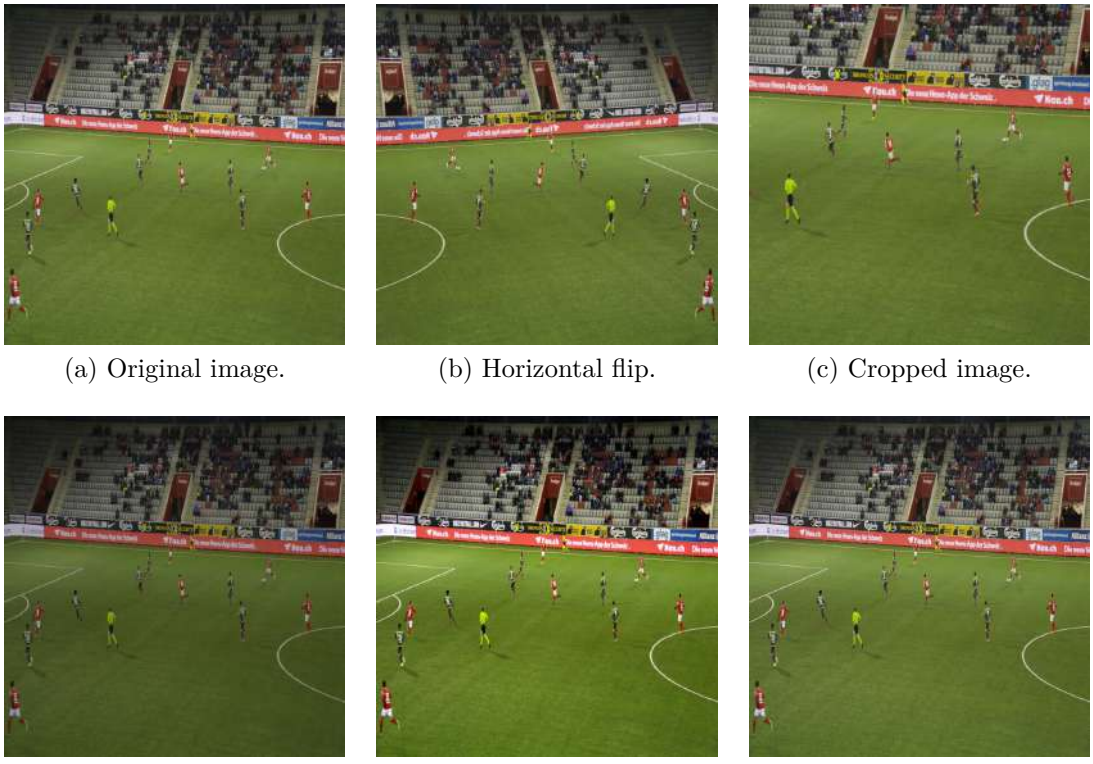
Brightness, contrast and sharpness: These were altered to augment the images while still having them look like games of football. Through visual inspection, maximum values of 0.25, 0.25, and 0.5 respectively were decided upon.

Cropping: Another way of adding augmentations was to crop the images. It was decided that a maximum of 25% should be cropped off any edge when applying cropping to an image, to not lose too much information.

Examples of each of these augmentations can be seen in Figure 4.9. With these augmentations, several iterations of the dataset were made, as described below. Note that these augmentations were only applied to the training set.

- **V0:** Dataset containing all the original SoccerNet-v2 training and validation images.

- **V1:** Dataset containing 1/10th of all images in V0.
- **V2:** Dataset containing 1/10th of all images in V0 where each image has a 50% change of being flipped horizontally.
- **V3:** Dataset containing 1/10th of all images in V0, where each image has a 50% chance of having each of the data augmentations applied to it.
- **V4:** Dataset containing 1/5th of all images in V0, where each image has a 50% chance of having each of the data augmentations applied to it.
- **V5:** Dataset containing 1/3rd of all images in V0, where each image has a 50% chance of having each of the data augmentations applied to it.



(a) Original image. (b) Horizontal flip. (c) Cropped image.

(d) Decreased brightness. (e) Increased contrast (f) Increased sharpness.

Figure 4.9: Augmentations performed on a SoccerNet-v2 image.

CHAPTER 5

Methods

The objective of this thesis is to develop a model capable of predicting the phase of play at every stage of a football game. This chapter outlines the methodologies employed to address this task, detailing the considerations and rationale behind the chosen approach. In section 5.1, we describe some of the methodological considerations we took into account when developing our solution. In section 5.2, our two-stage solution is presented. Finally, in section 5.3, we explain the approach of utilizing player and line detections.

5.1 Methodological Considerations

Length of games: Since a game of football lasts 90 minutes, it is not feasible to create a model that takes a full game as input at once, as this would require too much computer power. The number of pixels in a 90-minute RGB video at 25 fps and a resolution of 512x512 can be calculated:

$$N = 90 \times 60 \times 25 \times 512 \times 512 \times 3 = 106,168,320,000 \text{ pixels} \tag{5.1.1}$$

Since a video of a full game of football contains over 100 billion pixels, it would be infeasible to make a model with such a large input. Therefore, it was decided that the game should be split up into smaller clips.

Length of Clips: After analyzing the data, it was observed that the duration of most play phases falls within the range of 20 to 40 seconds. Considering the limitations of current video classification models, which are typically unable to process lengthy videos efficiently, a decision was made to segment the games into shorter 8-second clips. This decision was backed by human analysis. Having looked at numerous 8-second clips they were deemed long enough to determine the phase of play. Although the duration of the clips were determined based on these factors, it should be noted that further experimentation regarding the length of clips could be conducted with additional time and resources.

One phase at a time: It is known that at any point during a football game, exactly one phase of play will be active. With this in mind, it would be possible to create a video classification model to detect phases of play in short clips, and then apply this model to consecutive clips to get the phases of play throughout a full game.

Phase evolution: Another known fact that could provide a boost for the model is that some phases are more likely to follow each other. In section 4.1.4 we showed that this was indeed the case, that some phases tend to follow each other, such as ”VI 1” usually being followed by ”VI 2”, ”VI 3” or ”VI-DE”. This information could be learned by incorporating a transformer-based model on sequences of the original predictions.

Positional information: The phase of play obviously depends on the positions of the players and the ball on the pitch, as this is part of what defines the phases. We thus hypothesized that if it was possible to extract the positions of the players and the ball, this information could be used for the model.

5.2 The Model

The final model design consists of two stages. The first stage is a video classification model and the second stage is a transformer-based model that uses the output features of the video classification model as input. In this section we describe these models individually, as well as the pipeline of the combined model.

5.2.1 Stage 1: Video Classification Model

The video classification model takes as input 8-second clips and classifies them as one of the nine phases of play. For this model, mmaction2's implementation[75] of Temporal Segment Networks (TSN)[42] was used. The TSN model was trained on the well-defined clips acquired from the preprocessing of the DBU data, as described in section 4.1.2.

The TSN model operates by sampling clips from *num_clips* evenly spaced segments of the input video. These clips have variable lengths, determined by the *clip_len* parameter, which specifies the number of frames per clip. Additionally, the frame interval between consecutive frames within each clip is controlled by the *frame_interval* parameter. Feature representations of each of the clips are acquired through a ResNet backbone. These features are then combined and fed through a final linear layer and softmax activation function to get the output probabilities. This sampling strategy allows the model to capture temporal infor-

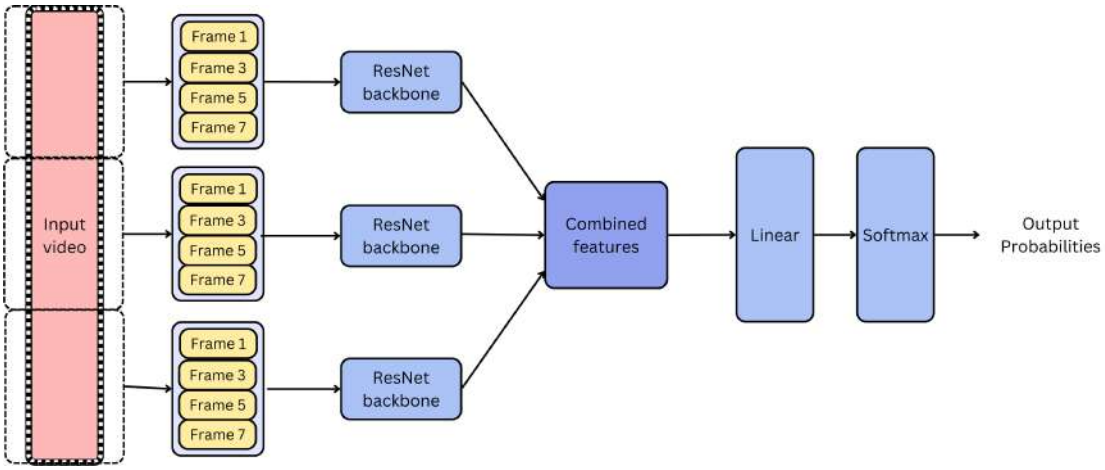


Figure 5.1: TSN pipeline. In the example shown here, the input clip is split into *num_clips* = 3 segments of length *clip_len* = 4, sampled at intervals of *frame_interval* = 2 frames. Each of these segments are then fed through a ResNet backbone, and their output features are combined. These combined features are then fed through a linear layer and a softmax activation function to get the output probabilities.

mation spanning the entire duration of the input video, facilitating effective action recognition. The pipeline of this model can be seen in Figure 5.1.

5.2.2 Stage 2: Transformer-based Model

The transformer-based model processes a sequence of features to predict class probabilities. This sequence of features is obtained by applying the video classification model to consecutive clips and extracting the final layer of features (shown as "Combined features" in Figure 5.1). Given a l_{seq} long sequence of features extracted from l_{seq} consecutive clips, the model's task is to predict the label of the middle clip. The sequence of features gives the model added information about the preceding and succeeding clips, which can help it in predicting the correct label.

The model architecture can be seen in Figure 5.2. It is inspired by that of the transformer[7], but with a few alterations, notably it contains no decoder as the output is not sequential. Our transformer-based model comprises:

- **Input Projection:** The input projection is a linear feed-forward layer that projects the 2048-dimensional features onto a d_{model}-dimensional layer, where d_{model} is the dimension of the encoder input. This transformation is applied to all the features in the sequence, converting the dimensions from $[l_{seq}, 2048]$ to $[l_{seq}, d_{model}]$. Essentially, it maps the input features to a lower-dimensional space that the model can work with more effectively.
- **Positional Encoding:** The positional encoding adds positional information to the l_{seq} sets of features. It ensures that the model can distinguish

between features from different positions in the sequence, helping capture temporal dependencies. It is implemented with a combination of sine and cosine functions with different frequencies

- **Transformer Encoder Layers:** The model has n_{layer} encoder layers, each of which contains multi-head attention, a feed-forward network and layer normalization.
  - **Multi-Head Attention:** Each encoder layer has n_{heads} attention heads with distinct queries, keys and values. The Scaled Dot-Product Attention function (Equation 2.2.1) is then performed on all of these in parallel and the results are concatenated.
  - **Feed-Forward Network:** Each encoder layer has a position-wise feed-forward network that consists of two linear transformations with a ReLU activation function in between. The input layer of dimension d_{model} is first mapped to a layer of dimension d_{ff} and then back to a layer of dimension d_{model}. Thus, the dimension of the model stays constant throughout the encoder layers.
  - **Layer Normalization:** Layer normalization is applied after both the multi-head attention mechanism and the feed-forward network. It helps stabilize the training process by normalizing the activations across the feature dimension.

- **Feed-Forward Layer:** This final layer takes the encoder output of only the middle clip. This is because that is the clip the model is trying to predict the label of, and information from the preceding and succeeding clips is being learnt in the encoder layers. The encoder output of dimension d_{model} is then mapped to a layer of dimension 9, followed by a softmax activation function for classification.

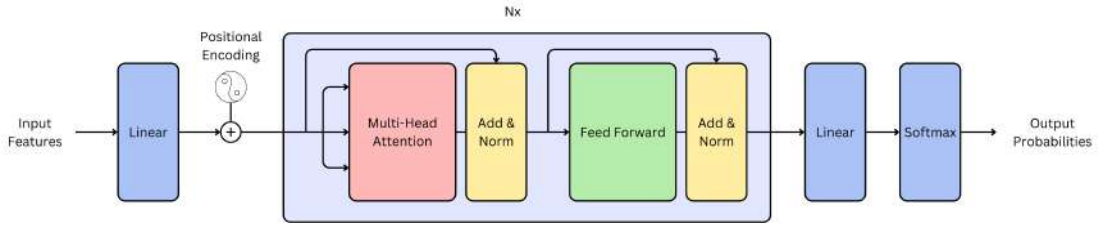


Figure 5.2: Pipeline of transformer-based model. The input sequence of features is fed through a feed-forward layer before a positional encoding is added. These are then passed through n_{layer} encoder layers, each of which contains multi-head attention, a feed-forward network and layer normalization. It is then passed through a final linear layer and softmax activation function, resulting in the output probabilities.

5.2.3 The Complete Model

Combining these two stages results in the complete model. The pipeline of this model is depicted in Figure 5.3, and can be expressed in these steps:

1. The full football game is split up into consecutive 8-second clips.
2. Each of these clips are classified through the video classification model described in section 5.2.1.
3. The features of these clips are extracted and combined into sequences.
4. These features are passed to the transformer-based model described in section 5.2.2 which outputs the predicted class.
5. Consecutive class predictions are concatenated to get the predicted phase of play throughout the whole duration of the game.

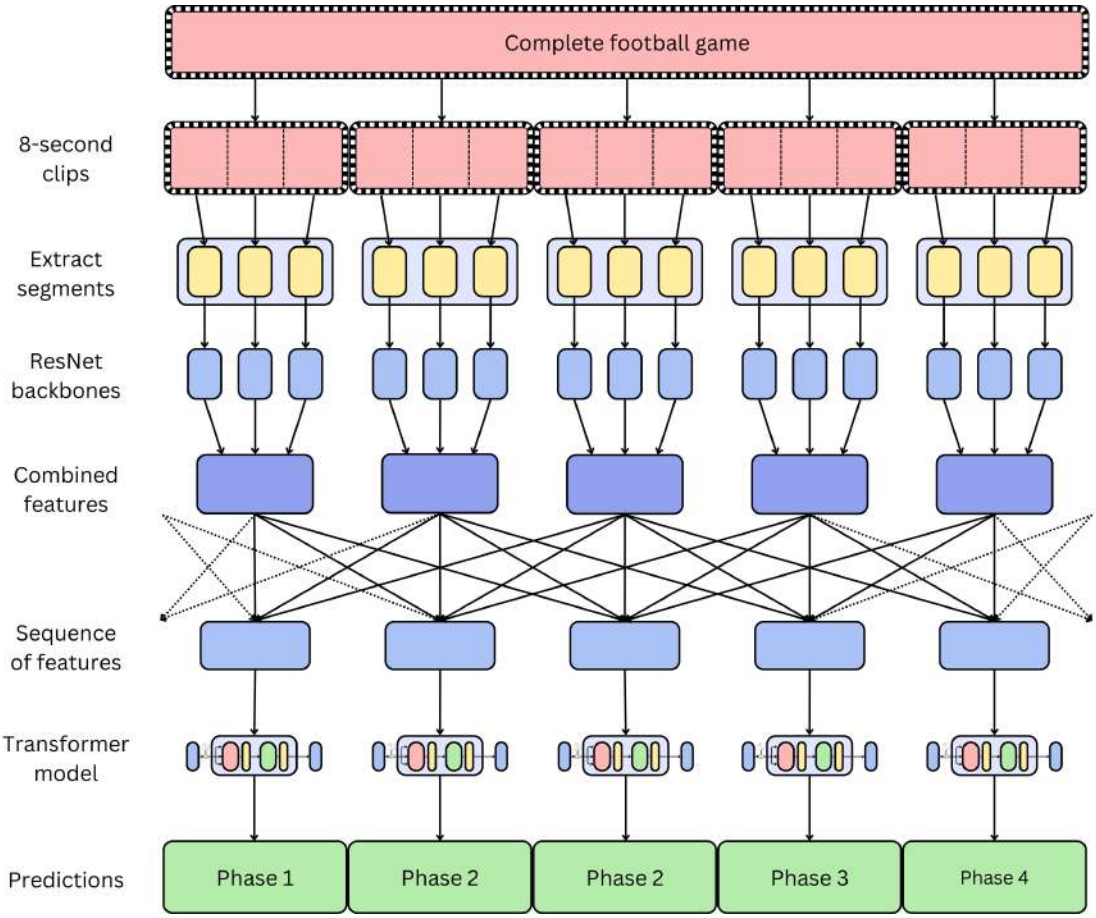


Figure 5.3: Pipeline of the complete model. A full game is split into 8-second clips from which segments are extracted and fed through a ResNet backbone to acquire a feature representation of these clips. Sequences of features from consecutive clips are then fed to the transformer-based model to get a phase prediction.

5.3 Player and Line Detection Input Clips

To investigate whether the positional information of players alone could determine the phase of play, we experimented with replacing the original video clips with simplified versions containing only pitch lines and players represented as colored dots. The rationale behind this approach was to retain positional information while simplifying the input data. Here's how we implemented this approach:

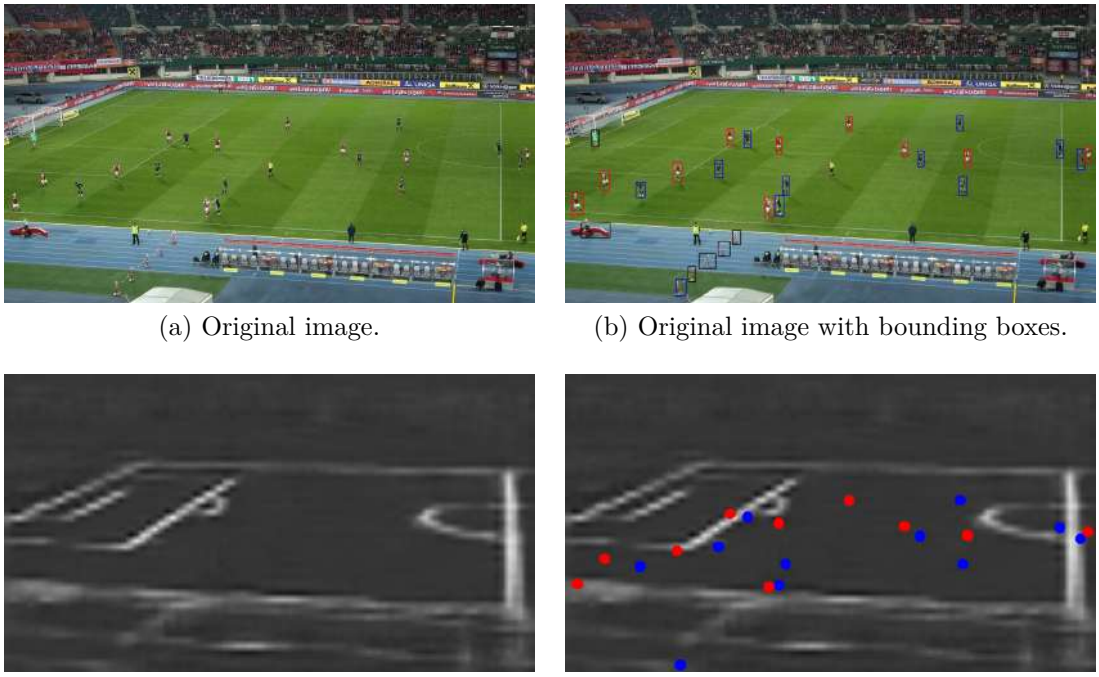
1. We detected the players using a YOLOv8 object detection model trained on the SoccerNet-v2 dataset. The model was trained to detect players, goalkeepers, referees and balls. For this approach we only cared about the players, so all other detections were ignored.

2. The detected players were then split into two teams using KMeans clustering. This was achieved by resizing all the detected player bounding boxes to the same dimension, and then performing KMeans clustering on these images. Figure 5.4b shows the clustered bounding boxes. Note that there are also a few black bounding boxes, which was due to the fact that we actually grouped the detected players into three clusters and ignored the smallest cluster. We did this because the player detection model would often have a few wrong detections (goalkeeper, substitute, item, etc.) and this proved to be an efficient way of removing those.

3. We then used a pretrained DeepLabV3 segmentation model to detect the pitch lines. This gave us a segmentation image as can be seen in Figure 5.4c.

4. The detected players were then mapped onto the pitch line image, as can be seen in Figure 5.4d. We used the center of the base of the bounding boxes as the player locations, as this is where their feet are.

5. This process was repeated for all frames in a clip and combined into a video.



(a) Original image.

(b) Original image with bounding boxes.

(c) Image of detected pitch lines.

(d) Image of detected lines with detected players mapped onto it.

Figure 5.4: Example of how a player and line detection frame is made. From the original image (a), players are detected using our YOLOv8 player detection model trained on the SoccerNet-v2 dataset. These players are then clustered into two teams using KMeans clustering on the bounding boxes (b). A DeepLabV3 segmentation model is used to detect the pitch lines (c), and finally the detected and clustered players are mapped onto this image (d).

CHAPTER 6

Experiments and Results

This chapter presents the experiments conducted and the results obtained. In Section 6.1, we describe the training experiments conducted for the video classification model and compare its performance with human accuracy. Additionally, we analyze the model's learned features through visualization. Section 6.2 details the training experiments for the transformer-based model and showcases the improved performance achieved. We also provide an analysis of the features learned by this model. Finally, in Section 6.3 we discuss the experiments carried out to enhance the YOLOv8 player detection model and present the results obtained when using the player and line detection clips as input for the video classification model.

The hardware for all of these experiments were provided by DTU's High Performance Computing (HPC) services. All models were trained and evaluated on either Tesla V100-PCIE-16GB, Tesla V100-PCIE-32GB, NVIDIA A100-PCIE-40GB or NVIDIA A100-PCIE-80GB GPUs. When mentioning training or inference times, the specific GPU used will be stated.

6.1 TSN Video Classification

In this section, we delve into the experiments and results of the TSN video classification model. Section 6.1.1 details the experimental setup and evaluation metrics. Section 6.1.2 shows the experiments and steps taken to optimize the model. The results of our model are presented in section 6.1.3 and compared to human accuracy scores in section 6.1.4. In section 6.1.5, we conduct an analysis of the model's features, and finally in section 6.1.6 we provide further analysis of our model.

6.1.1 Experimental Setup and Evaluation Metrics

The experimental setup and evaluation metrics for the TSN video classification model are outlined below.

- **Dataset**
  - For these experiments, the DBU datasets consisting of the well-defined clips were utilized.
  - During training, the models were trained on the training set and validated on the validation set. The best model was then evaluated on the test set.
- **Evaluation Metrics**
  - **Top 1 Accuracy**: This metric represents the proportion of correctly predicted labels among the total predictions made by the model.
  - **Top 5 Accuracy**: This metric indicates the proportion of predictions where the correct label is among the top 5 highest probabilities predicted by the model. In other words, it measures how often the model correctly identifies the correct label among its top 5 predictions.
  - **Mean Top 1 Accuracy**: This metric calculates the average accuracy across all classes, where accuracy for each class is computed as the number of correct predictions of that class divided by the total occurrences of that class.
- **Training Procedure**
  - All models were trained for a maximum of 100 epochs, with a warm-up period of 10 epochs. Early stopping was employed if a training run exhibited clear signs of stagnation or decline in performance.
  - Stochastic gradient descent (SGD) was used for optimizing, with a learning rate of 0.01, momentum of 0.9 and weight decay of 0.0001.
  - The iteration with the highest top 1 accuracy on the validation set was considered the best-performing model for evaluation.
  - The hyperparameters we looked to optimize were the *clip_len*, *interval* and *num_clips* parameters. The *num_clips* parameter denotes the number of segments sampled from the input video, and *clip_len* and *interval* denote the length and frame sampling rate of these segments respectively.

6.1.2 Training Experiments

In our initial experiment, we delved into the impact of segment length on the model's performance. Specifically, we maintained a frame interval of 1 and a constant number of clips at 2, while systematically altering the *clip_len* parameter. The outcomes of these investigations are depicted in Figure 6.1. Based on these findings, we opted for segments with a clip length of 10, as this achieved the highest mean top 1 accuracy and second highest top 1 and top 5 accuracy.

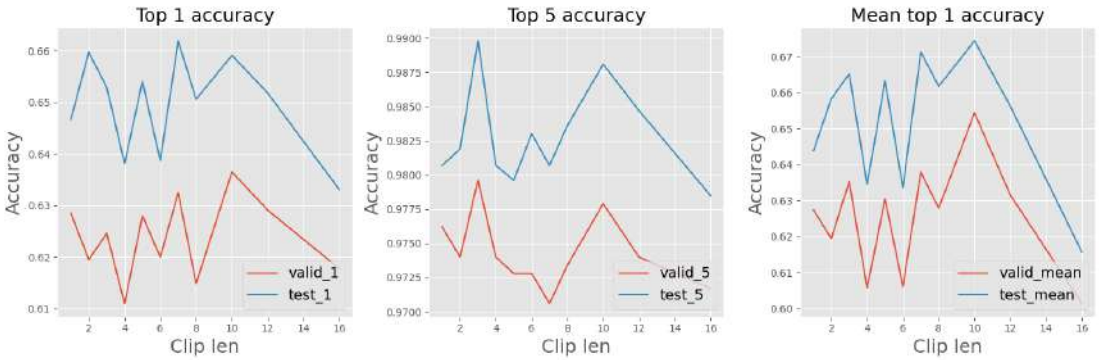


Figure 6.1: Performance evaluation of the TSN video classification model with varying *clip_len* parameters. We see that setting the *clip_len* = 10 results in the best performance on mean top 1 accuracy and second best performance on top 1 and top 5 accuracy.

In our next experiment we wanted to explore the impact of frame sampling interval on model performance. Here, we maintained the number of clips at 2 and kept the *clip_len* parameter consistent at the previously determined value of 10. We systematically adjusted the *interval* parameter in order to identify the optimal value. The results of these experiments are presented in Figure 6.2. From this experiment we found that keeping the frame sampling rate at *interval* = 1 was the optimal solution, as this yielded the best scores in both top 1 accuracy and mean top 1 accuracy while also achieving a decent score in top 5 accuracy.

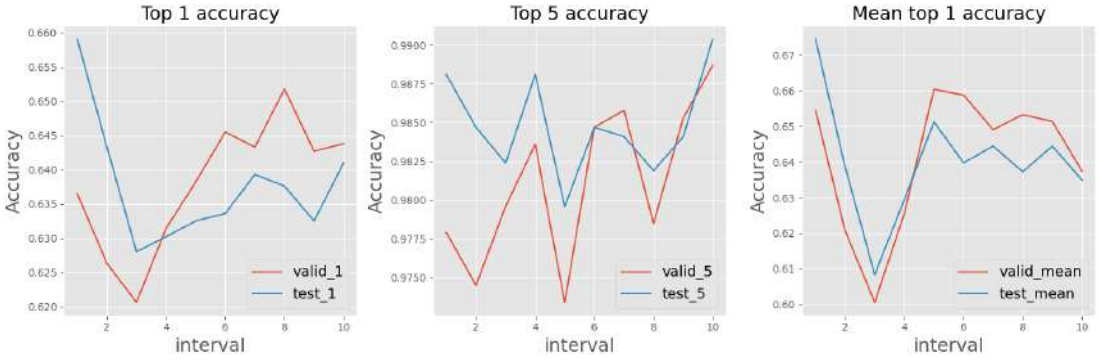


Figure 6.2: Performance evaluation of the TSN video classification model with varying *interval* parameters. We see that the original value of *interval* = 1 is optimal, as this yields both the best top 1 accuracy and mean top 1 accuracy scores.

In our final investigation, we examined the impact of the number of segments the input video was spilt into, where our initial choice of 2 proved to be optimal. Consequently, the ideal model configuration consisted of the input video being split into two segments, each comprising 10 frames, sampled at 1-frame intervals. This

configuration yielded a top 1 accuracy of 65.91% on the validation set of well-defined clips. A table of all experiments carried out can be found in Appendix A.2.

6.1.3 Results

On the test set consisting of well-defined clips, the model achieved a top 1 accuracy of 67.52%, which surprisingly is even better than on the validation set. To gain deeper insights into its strengths and weaknesses, we can analyze the confusion matrix presented in Figure 6.3.

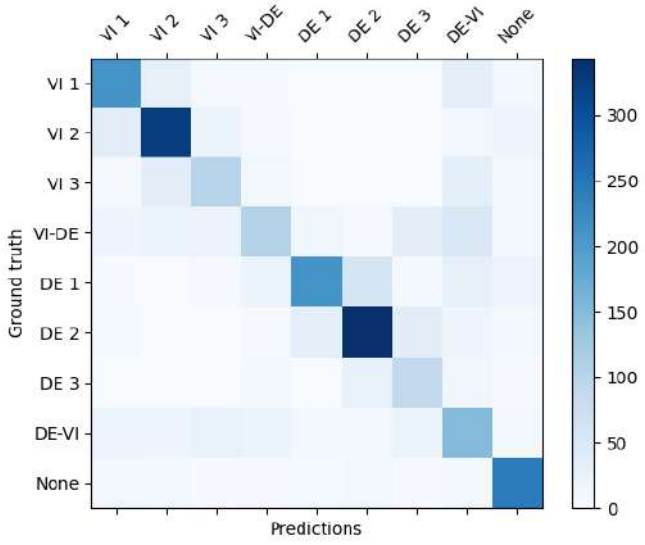


Figure 6.3: Confusion matrix of our model's predictions versus the ground truths, showing the number of occurrences for each predicted and true class. Generally our model's prediction match the ground truths. However, we see our model sometimes struggles to differentiate between "VI 1" and "VI 2" and "DE 1" and "DE 2". Furthermore, the transition phases (VI-DE and DE-VI) also cause some confusion.

Notably, the model encounters challenges distinguishing between "VI 1" and "VI 2", as well as "DE 1" and "DE 2", which is understandable given the similarities between these phases. In fact, one of the labelers at DBU even said that in some cases he is unsure if a stage of a game should be labeled as phase 1 or phase 2. Another area where the model struggles is the transitional phases "VI-DE" and "DE-VI". Here the model sometimes mixes up the two transitional phases, or predicts a transitional phase when it in fact was an attacking or defending phase, or vice versa. This behaviour could stem from the fact that transitional phases are labeled also while the team that won possession is in the counter attack. Thus, the exact moment of transition might not be captured during the 8-second clip.

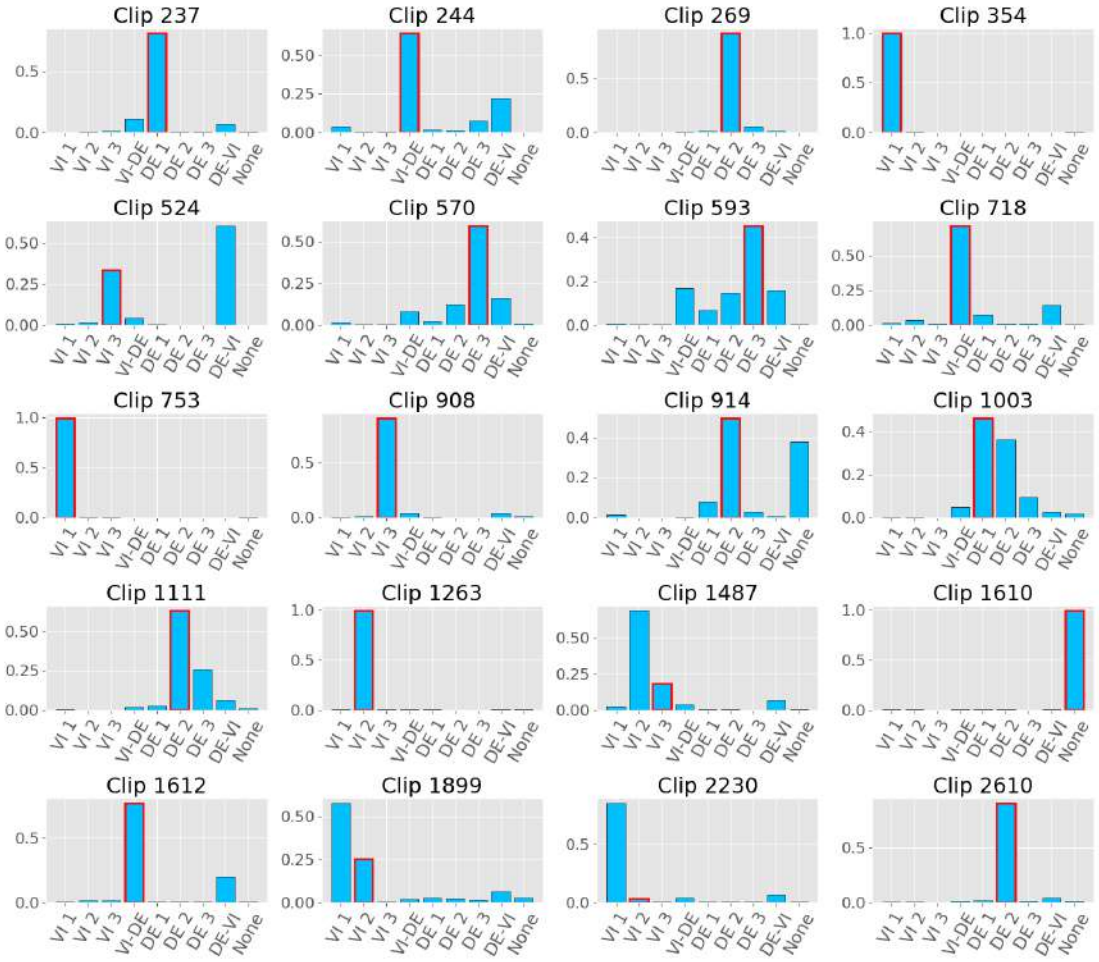


Figure 6.4: Bar plot of output probabilities of our model on a sample of clips from the test set. The blue bars show the output probabilities for each phase, and the bars marked with red denote the ground truth phase. Clip 524 shows an example of the model struggling with the transition phase, and clips 1003, 1487 and 1899 show the model struggling to differentiate between phases 1 and 2, and phases 2 and 3.

We can further study the behaviour of our model by looking at the output probabilities on a sample of the test set, as shown in Figure 6.4. Here we see that the model is often very sure in its predictions, and also correct, e.g. clips 269, 354, 753 and 908. In clip 524 we see a case where the model predicts a "DE-VI" transition phase, but the actual phase is "VI 3". We see the model struggles to differentiate between phase 1 and 2 in clips 1003 and 1899, and in clip 1487 we see it struggles to differentiate between phase 2 and 3. To try and understand which clips our model is good at and which clips it struggles with we will visualize some of the clips. In Figure 6.5 we show examples of clips where the model predicts the correct label, and in Figure 6.6 we show examples where it predicts the wrong label.

(a) Clip 237. Here our model correctly predicts "DE 1". We see the team attacking from right to left passes the ball around in their own half while the other team is pressing, so this is clearly a phase "DE 1".

(b) Clip 354. Here our model correctly predicts "VI 1". The goalkeeper from the team attacking from left to right has the ball in his own box, so this is also a clear case of phase "VI 1".

(c) Clip 593. Here our model has a bit more uncertainty, but still correctly predicts "DE 3". In this clip we see the team going from right to left gets in behind the defending team and puts a cross into the box.

(d) Clip 1610. Here our model correctly predicts "none", as the ball is clearly not in play.

(e) Clip 1612. Here our model correctly predicts "VI-DE". We see that the team attacking from left to right is in possession of the ball, but lose it due to a bad touch, and the other team starts a counter attack.

Figure 6.5: Examples of clips where our model predicts the correct label.

(a) Clip 524. Here our model wrongly predicts "DE-VI". The correct phase is "VI 3", which is our model's second prediction. We see the team attacking from left to right has the ball a bit past the halfway line and moves up the pitch and gets close to the other team's penalty box. Our model is likely confused here due to the fast pace of the attack looking like a counter attack that could occur right after having won possession, like in a "DE-VI" phase.

(b) Clip 1487. Here our model predicts "VI 2" while the correct label is "VI 3". Looking at the clip, however, it seems as though this is indeed a "VI 2", as the team attacking from left to right is in possession of the ball quite high up the pitch, but have not broken through the other team's defensive lines. This could be a labeling inaccuracy, and is likely because a "VI 3" phase is going to occur right after this clip, and when labeling they set the start time of that phase too early.

(c) Clip 1899. Here our model wrongly predicts "VI 1" while the correct label is "VI 2", which is our model's second prediction. Looking at the clip we can see the confusion. The team going from left to right has the ball in their own half, but the other team is not pressing at the start, which shows characteristics of both phase "VI 1" and "VI 2".

(d) Clip 2230. Here our model predicts "VI 1" but the correct label is "VI 2". Looking at this clip, however, it clearly looks like it is indeed a phase "VI 1", as the team attacking from left to right has the ball in their own half and the other team is pressing high.

Figure 6.6: Examples of clips where our model predicts the wrong label.

We see in Figure 6.6 that some cases where our model predicts the wrong phase, one could argue that the label is incorrect and that our model's guess was indeed correct, e.g. clips 1487 and 2230. In other cases, such as clips 524 and 1899, the confusion of our model is understandable as there are similarities between the

wrongly predicted phase and the correct phase. In most cases, however, we see that if our model does not predict the correct label, its second or third guess would usually be correct.

We want to investigate our model's accuracy when looking at both top 1, top 2, top 3, etc predictions. Figure 6.7 shows the top 1, 2, ... 9 accuracy of each of the 7 test games individually. We see that there is indeed a drastic increase in accuracy when we look at the top 2 accuracy, and also a good further increase when we go from top 2 to top 3. For most games, the top 3 accuracy is above 90%. Another thing we notice is that there is a large difference in performance for each of the test games, with one of the games having a top 1 accuracy of only 59.78%, and another a top 1 accuracy of 78.74%!

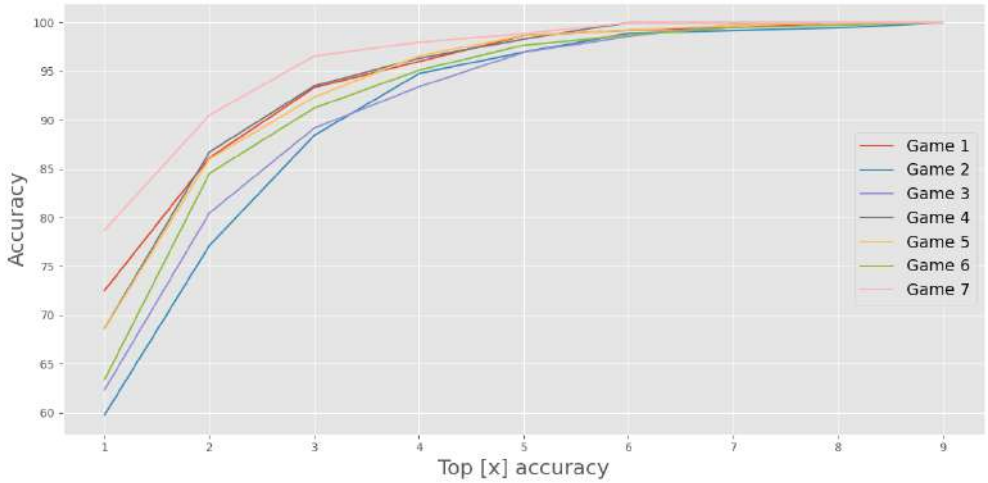


Figure 6.7: Top 1, 2, ... 9 accuracy of our TSN video classification model on each of the 7 test games. We see a great increase from top 1 to top 2 accuracy, and a further good increase to top 3 accuracy. We also see a large difference in performance between the individual test games.

There could be several reasons as to why the model performs much better on some of the test games compared to others. One reason could be due to the labeling of the data. Some games might not be labeled very precisely, so it could be that the model is in fact predicting the right phase, but it has been labeled incorrectly in the dataset, as we saw in some of the example clips in Figure 6.6. Another reason could be the nature of the games themselves. Camera angle, colors, tactics, etc. could all have an effect on the prediction abilities of the model. A manual inspection of the games did not show any inherent visual reasons why there is such a large variety in performances. Interestingly, the test games with a frame rate of 50 fps (game 1, game 4, game 5) are not among the lowest scoring games, despite the model being trained purely on games of frames rates between 25 and 30 fps.

A final observation we would like to make is the inference time of the model.

Running on a NVIDIA-A100-80GB-PCIe GPU resulted in an average inference time of 0.47 seconds per 8-second clip. This means that inference over a full 90-minute game, which contains $90 \cdot 60/8 = 675$ consecutive 8-second clips will take a total of $675 \cdot 0.47 = 317.25$ seconds, less than 5.5 minutes. This shows that our model is indeed efficient.

6.1.4 Human Comparison

In order to get a better understanding of how good these results are, we decided to test our model against humans. We created a subset consisting of 72 clips, eight clips of each of the nine unique phases, drawn at random from the validation set of well-defined clips, and asked humans with an understanding of football to predict the label of each clip. As can be seen in Figure 6.8, our model outperforms everyone, even a human expert within this field.

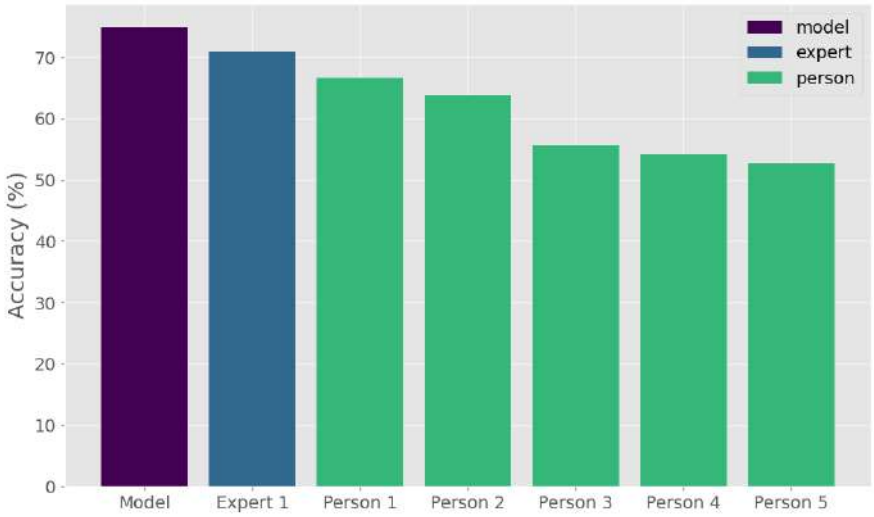


Figure 6.8: Top 1 accuracy of our model compared to humans. Our model (purple) outperforms normal humans with a knowledge of football (green) as well as human experts in this field (blue).

Our model's top 1 accuracy of 75% on this subset is much larger than on the complete validation set, which suggests that this might have been a slightly "easy" subset. However, it was the same set for the humans taking this, so we can still conclude that our model outperforms even human experts.

In figure 6.9, we present the confusion matrix depicting the percentage of clips correctly predicted by both our model and humans, incorrectly predicted by both, or correctly predicted by one and incorrectly by the other. For human predictions we used the majority vote, resorting to the human expert's vote in the event of a tie. We see that 51.4% of the clips were correctly predicted by both our model and humans. Interestingly, only 6.9% of clips were wrongly predicted by both

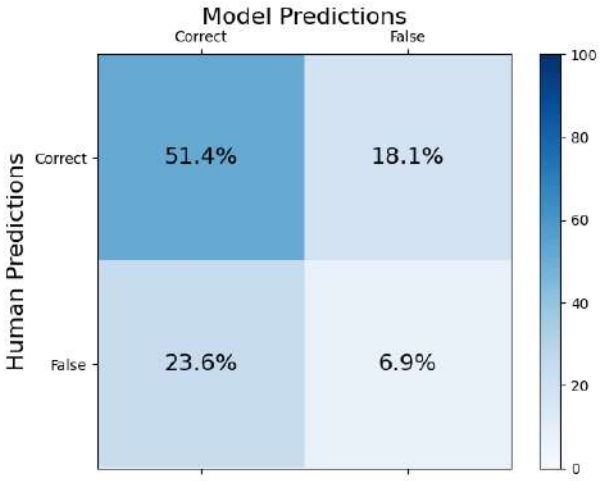


Figure 6.9: Confusion matrix of correct and wrong predictions of our model and humans. The majority of clips are accurately predicted by both our model and humans, with only 6.9% of clips being incorrectly predicted by both.

our model and humans. The relatively high number of occurrences where our model predicts the correct label and humans do not (23.6%) suggests that our model learns some trends in the data that are difficult for humans to pick up on. For example, our model is good at differentiating between a counter attack and a phase 3 attack, which humans struggle with when the change in possession does not occur during the clip. It also suggests that our model picks up on trends that are not consistent with the definitions of the phases. For example, at goal kicks our model might predict a phase 1, despite the ball being out of play. This is likely a result of the phases generally being labeled to begin too early.

6.1.5 Feature Analysis

To gain insights into our model's learning process, we conducted an analysis of its last layer of features. Initially, we extracted the feature representation of input clips post-model processing by removing the final linear layer during testing. We then employed T-distributed Stochastic Neighbor Embedding (t-SNE)[76] to reduce the 2048-dimensional features to 2 dimensions, facilitating visualization. This dimensionality reduction was carried out using sklearn's t-SNE implementation. Figure 6.10 illustrates the 2D feature representation of all input clips, with each phase of play depicted in distinct colors.

Upon examining the plot, a distinct segregation emerges between attacking phases (VI) and defending phases (DE). Within the cluster of attacking phases (blue), we observe a sequential arrangement where "VI 1" is adjacent to "VI 2", and "VI 2" is adjacent to "VI 3", illustrating a sequential progression of phases. This sequential trend can also be seen within the cluster of defending phases (orange). Transitional phases ("VI-DE" and "DE-VI") are positioned between the attacking

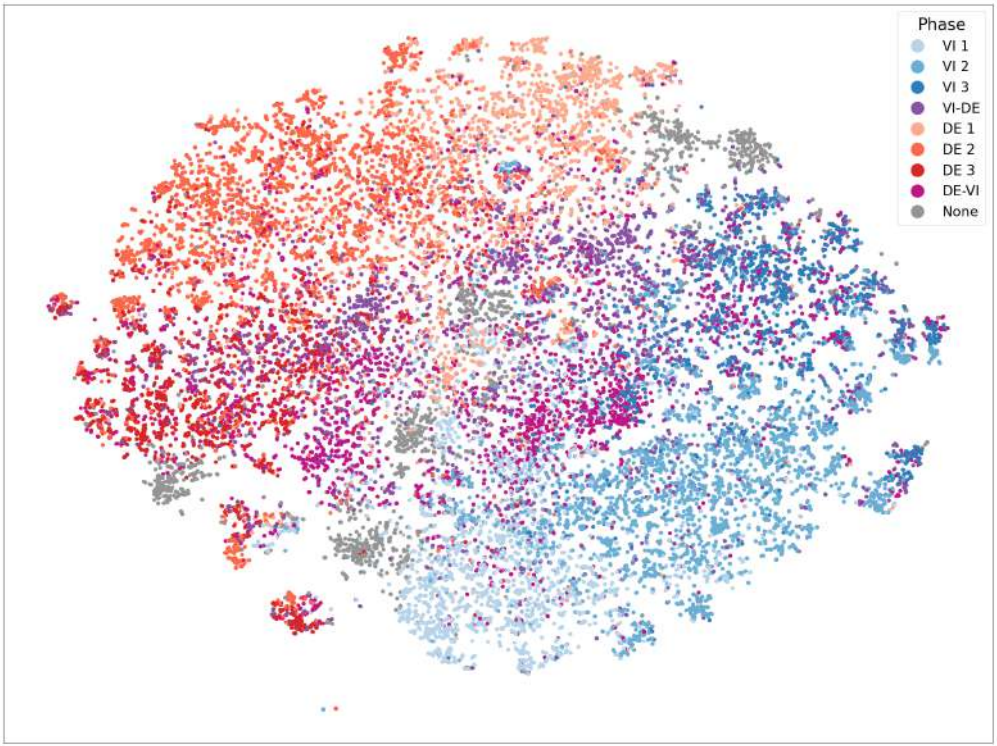
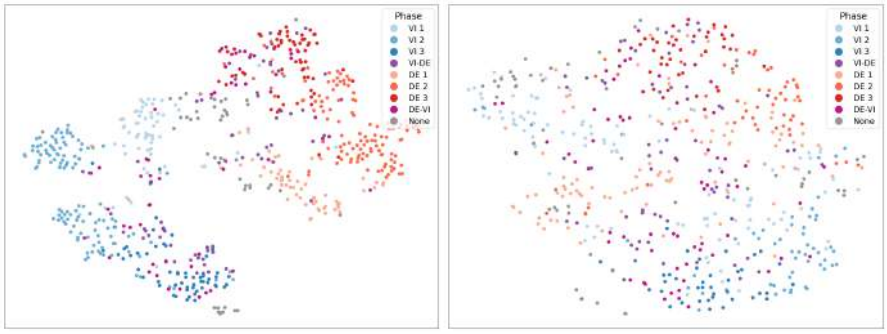


Figure 6.10: 2D representation of features from all clips. The features were reduced to two dimensions using t-SNE. We notice a separation of attacking phases (blue) and defending phases (orange). Additionally, ”VI 1” is next to ”VI 2”, and ”VI 2” is next to ”VI 3”, showcasing a similarity between phase 1 and 2 and phase 2 and 3. This similarity can also be seen with the defending phases.

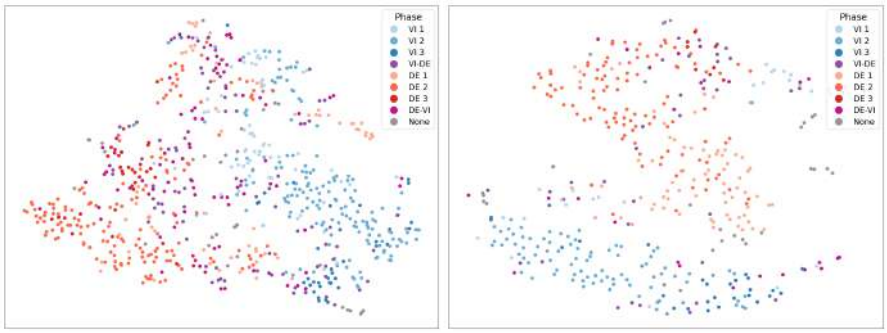
and defending clusters, aligning with the fact that these phases occur when ball possession changes between teams. Moreover, a cyclic pattern emerges as we traverse the plot counterclockwise, passing through distinct clusters of phases: ”VI 1”, ”VI 2”, ”VI 3”, ”VI-DE”, ”DE 1”, ”DE 2”, ”DE 3”, ”DE-VI”, and returning to ”VI 1”. This pattern could mimic an in-game situation where the VI team has the ball and progresses through the attacking phases (”VI 1”, ”VI 2”, ”VI 3”) before losing possession to the other team (”VI-DE”) and then has to defend as the other team progresses through the phases (”DE 1”, ”DE 2”, ”DE 3”).

We see a few odd occurrences within the graph where there seems to be a few small clusters containing all play phases scattered around. These smaller clusters seem to also follow the trends described before, so we hypothesized that each of these could be a complete game in itself. Figure 6.11 shows the 2D feature representation where each game has been colored distinctively. From this plot we see that generally, the 2D feature representation of the clips for all games seems to be very scattered with a few exceptions. For example, in the center near the top we see a large cluster of feature representations from the game with ID 36.

Figure 6.11: 2D representation of features from all clips with each game colored differently. We have highlighted some of the games where all the clips are clustered closely together.



(a) Game 3 (b) Game 25



(c) Game 36 (d) Game 52

Figure 6.12: 2D representation of features for individual games. We see that each of these follow the same patterns mentioned before.

The reason why certain games are clustered together could be due to certain characteristics of those games such as camera angle, color of the teams or tactics. When we perform t-SNE on these games individually to get a 2D feature representation, as shown in Figure 6.12, we see that these games still follow the overall patterns mentioned before.

6.1.6 Further Analysis

We carried out a few extra experiments to get an even better understanding of our model, as well as what is required for a model to perform well. Initially, we trained a model on single frames to see whether it would be possible to predict the phase of play from a still image. This resulted in an accuracy of 58.43%. While this is not completely terrible, it is still almost 10% less than our original model, which goes to show that it is important to include several frames from the input clip.

Another interesting result occurred when we tested our model on clips that had been reversed. When we did this, we found that the accuracy remained nearly identical to that of normal clips. A closer examination of our TSN video classification model revealed that it simply combines features by averaging the representation across all frames. This suggests that our model may not be fully utilizing the temporal dynamics of the input clips, as it disregards the order of frames.

6.2 Transformer-based Model

In this section we will present the experiments and results related to stage two of our solution - the transformer-based model. The experimental setup and evaluation metrics are described in section 6.2.1. In section 6.2.2 the experiments carried out to optimize the model are described, and the results are presented in section 6.2.3. Finally, in section 6.2.4 we will again analyse the feature representation of the model.

6.2.1 Experimental Setup and Evaluation Metrics

The experimental setup and evaluation metrics for the transformer-based models are outlined below

- **Dataset**
  - For these experiments we used the DBU datasets consisting of the consecutive clips.
  - We extracted the features of these clips using our TSN video classification model.
  - We then combined sequences of these, which was the input used for our transformer-based model.

  - During training, the models were trained on the training set and validated on the validation set. The final model was then evaluated on the test set.

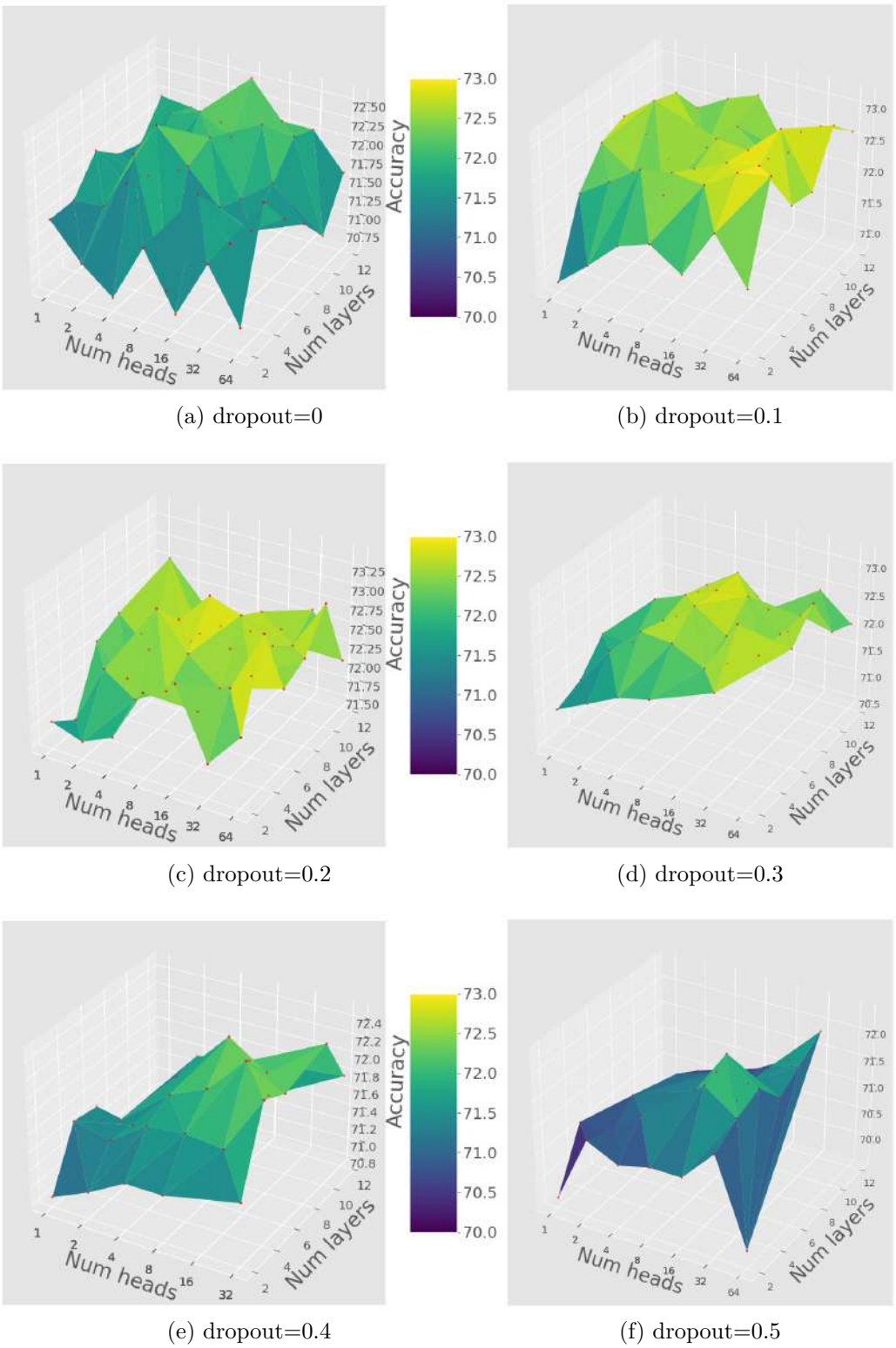
- **Evaluation Metrics**
  - **Top 1 Accuracy**: This metric represents the proportion of correctly predicted labels among the total predictions made by the model.

- **Training Procedure**
  - All models were trained for 100 epochs. During the first 10 epochs the data was sampled using a weighted random sampler, and for the remaining epochs a random sampler was used.
  - The model was trained using the Adam optimizer. We configured the optimizer with a learning rate of 0.0001, the beta parameters as (0.9, 0.98) and the epsilon value as 1e-9 to avoid division by zero.
  - The iteration with the highest top 1 accuracy on the validation set was considered the best-performing model for evaluation.
  - The hyperparameters we looked to optimize were the dropout rate, $dropout$, sequence length, l_{seq}, and the hyperparameters related to the size of the model: d_{model}, d_{ff}, n_{layers}, n_{heads}.

6.2.2 Training Experiments

Through some initial testing we found that setting the parameters $l_{seq} = 11$, $d_{model} = 64$ and $d_{ff} = 128$ proved to be good values. To limit the amount of computational resources and experiments required we decided to stick with these values. We investigated the effect of dropout rate, the number of layers and the number of heads by training models with varying $dropout$, n_{heads} and n_{layers} parameters in a grid-search way. The results are visualized in Figure 6.13.

Initially we see that setting the dropout to 0.1 or 0.2 leads to the best accuracy for all combinations of n_{heads} and n_{layers}. Furthermore, we see that the optimal number of layers is between 4 and 10. It is a bit harder to tell what the optimal number of heads should be, but generally it seems like the more the better. The configuration that resulted in the best top 1 accuracy was 64 heads and 4 layers with a dropout of 0.2, which yielded a top 1 accuracy of 73.37% on the validation set consisting of consecutive clips.



(a) dropout=0 (b) dropout=0.1

(c) dropout=0.2 (d) dropout=0.3

(e) dropout=0.4 (f) dropout=0.5

Figure 6.13: Top 1 accuracy of transformer-based model with varying hyperparameters. The parameters $l_{seq} = 11$, $d_{model} = 64$ and $d_{ff} = 128$ were kept constant while the number of heads and number of layers were varied. The experiment was performed with different dropout values. We see that the best dropout values are 0.1 and 0.2, and that optimal number of layers is between 4 and 10. The combination of hyperparameters that resulted in the best top 1 accuracy was 64 heads and 4 layers with a dropout of 0.2, which yielded a top 1 accuracy of 73.37%.

6.2.3 Results

We tested the effects of applying this transformer-based model to the features extracted from the TSN video classification model. For this we used the test set consisting of consecutive clips. We achieved a top 1 accuracy of 71.72% when applying the transformer-based model to the features extracted from the video classification model. This is much better than the top 1 accuracy when only using the video classification model, which was 64.08% on the same set. To get an idea of how this model performs better, we can look at the precision and recall scores for each phase of play, as visualized in Figure 6.14.

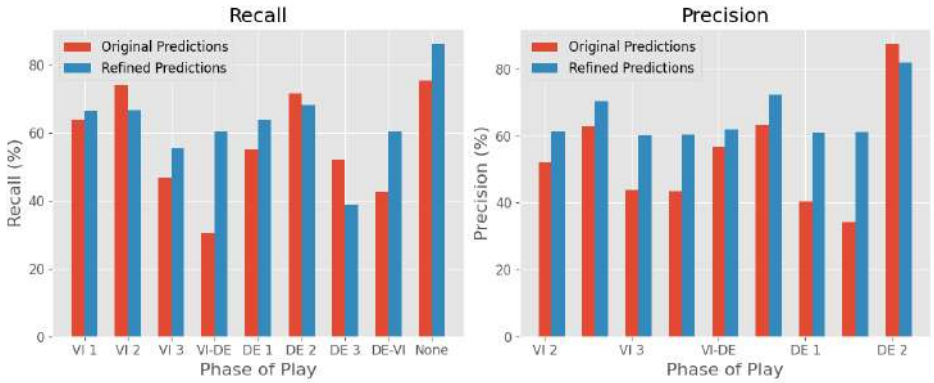


Figure 6.14: Precision and recall scores for all phases for the original TSN video classification predictions (red) and the refined predictions from the transformer-based model (blue). Some of the recall scores for the refined predictions are better than the original predictions, and some are worse. However, precision scores are drastically increased for all phases, except the ”none” phase.

From this we see the recall is increased for some of the phases, but decreased for others when we refine the results using our transformer-based model. The precision however, is increased for all phases except the ”none” phase. This suggests that this refined model only predicts a phase that is not ”none” when it is more sure in its decision. This is also what causes the recall to be slightly lower for some of the phases.

Another consequence of this is that the refined results are more likely to predict a ”none” phase - both when it is correct and when it is wrong. Comparing the confusion matrices of the original predictions of the video classification model and the refined predictions of the transformer-based model, as visualized in Figure 6.15, also shows this. We see that for most phases, if our model does not predict the correct phase, it will predict ”none”. We also see here that our refined model does get more correct predictions. Similar to the video classification model, this model also struggles with phases ”VI 1” and ”VI 2”, ”VI 2” and ”VI 3”, ”DE 1” and ”DE 2” and ”DE 2” and ”DE 3”. This could still be partly due to some labeling inaccuracies, as were discussed in section 6.1.3. It could also be due to the similarities of these phases.

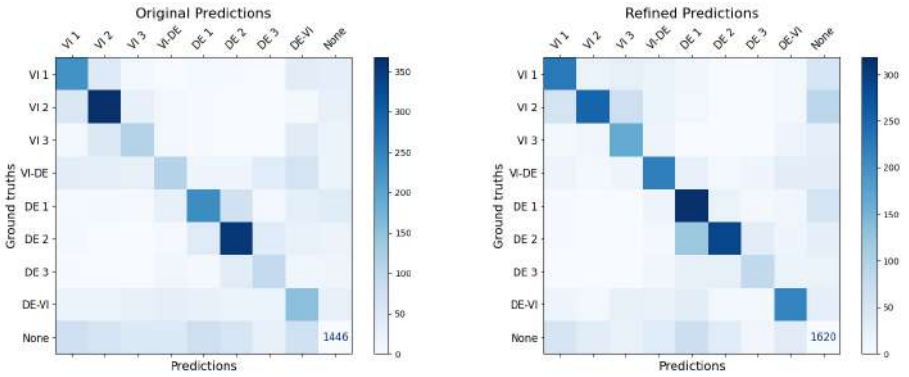


Figure 6.15: Confusion matrix of the predictions for our original TSN video classification model (left), and our refined predictions using the transformer-based model (right). They show the number of occurrences for each predicted and true class. Note the high number of occurrences where the models correctly predict "none", this is due to the large number of "none" phases in the test set consisting of consecutive clips.

We would like to see how our model performs on a consecutive section of a football game. In Figure 6.16 we show the predictions of the original TSN video classification model on a full 45-minute half of one of the test games, as well as the refined predictions achieved from applying the transformer-based model to the features of the original predictions. We see that the refined predictions fit the ground truths much better than the original predictions. Particularly, these refined predictions are less likely to jump from one phase to another and back, as can be seen for example from second 620 to 700 and second 1100 to 1150.

A final observation is that running inference with this transformer-based model is extremely fast. Running inference on all seven test games takes only a few seconds. Thus, the total time of stage one and stage two combined is still only 5.5 minutes for a complete 90-minute game.

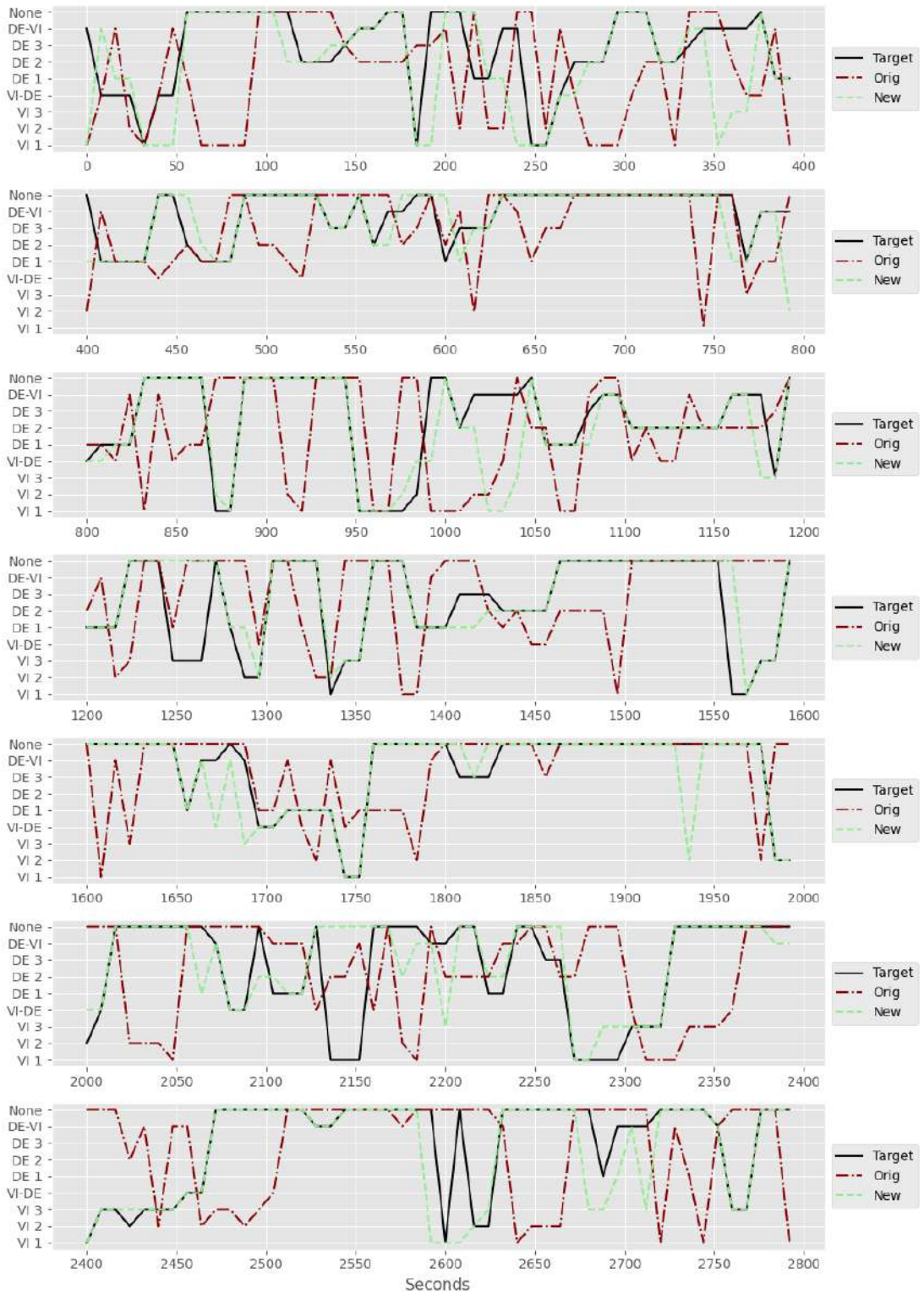


Figure 6.16: Original TSN video classification predictions (red) and refined predictions (green) over a complete half of one of the test games. The refined predictions fit the ground truths much better than the original predictions and appear more stable.

6.2.4 Feature Analysis

We will now investigate the feature representation of the clips after being passed through both the TSN video classification model and the transformer-based model. To do this we first extracted the feature representation of all well-defined clips from the TSN model. We subsequently fed sequences of each of these feature representations into our transformer-based model, and extracted the final 64-dimensional layer of features. We then employed T-distributed Stochastic Neighbor Embedding (t-SNE) to reduce the 64-dimensional features to two dimensions. This is visualized in Figure 6.17.

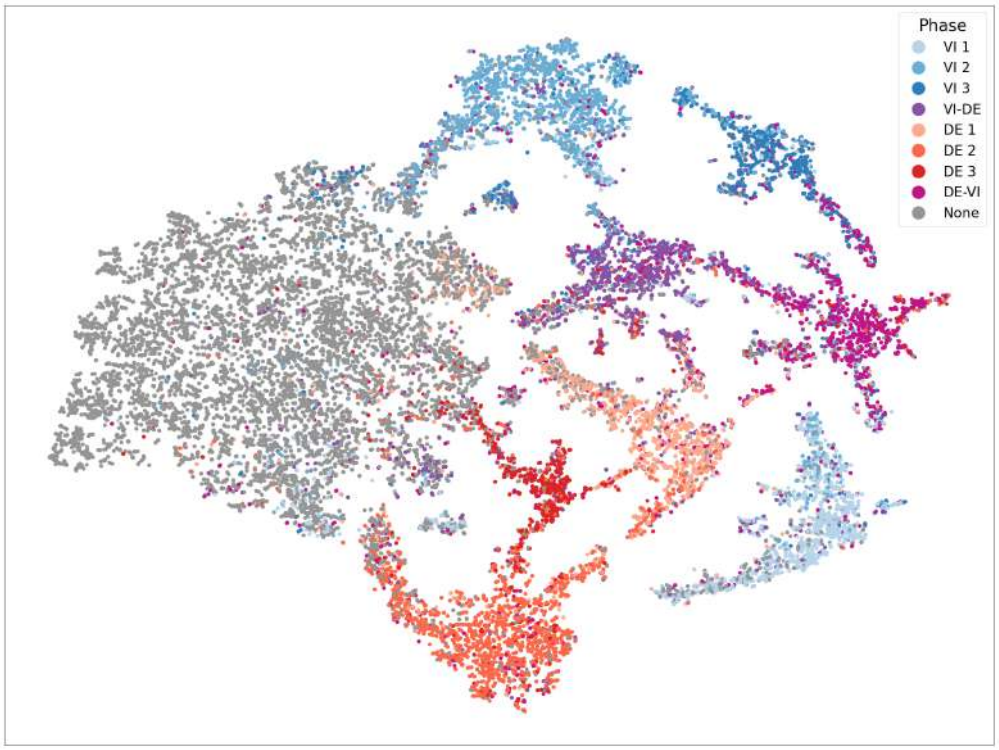


Figure 6.17: 2D representation of features from all clips after being passed through the transformer-based model. We notice a clear separation of every single phase. Unlike the feature representation of the TSN video classification model, there does not appear to be the same grouping of the attacking phases (blue) and defending phases (orange).

We notice that there are distinct clusters of play phase. Although not as apparent as with the feature representation of clips that had only been fed through the TSN video classification model, these clusters seem to also be grouped in attacking and defending phases. Certainly, phases ”DE 1”, ”DE 2” and ”DE 3” are near each other, and so are phases ”VI 2” and ”VI 3”. One thing that is not similar here is that these clusters do not follow the same cyclic pattern, instead each cluster seems to be more isolated.

6.3 Player and Line Detection Clips

In this section we will describe the experiments and results of the alternative approach mentioned in section 5.3, the player and line detection clips method. The main experiments for this involved improving the YOLOv8 player detection model, which is described in sections 6.3.1 and 6.3.2. The player and line detection clips were then used as input for the video classification model, and those results are presented in section 6.3.3.

6.3.1 Experimental Setup and Evaluation Metrics

The experimental setup and evaluation metrics for the YOLOv8 player detection models are outlined below.

- **Dataset**
  - All models were tested and validated on the SoccerNet-v2 validation set described in section 4.2. During training, validation was performed on a subset comprising 1/10th of the complete validation set to speed up the process.
  - The models were trained on the different augmented versions of the training sets, for comparability of these augmentations.
- **Evaluation Metrics**
  - **Precision**: This metric quantifies the model's ability to accurately localize and predict objects. It is computed as the ratio of true positive predictions to the total number of positive predictions.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.3.1}$$

  - **Recall**: This metric measures the proportion of actual objects detected by the model. It is calculated as the ratio of true positive predictions to the total number of actual positive instances.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.3.2}$$

  - **mAP50 (Mean Average Precision at IoU 50%)**: This metric computes the average precision of the model with an IoU (Intersection over Union) threshold set to 50%. I.e., this is the number of predicted bounding boxes with an IoU of at least 50% with a ground truth bounding box of the same class, divided by the total number of predictions.
  - **mAP50-95 (Mean Average Precision across IoU 50-95)**: This metric calculates the average precision of the model across IoU thresholds ranging from 50% to 95%.

- **Training Procedure**
  - The YOLOv8 models were trained for 300 epochs with a patience of 200 for early stopping.
  - The best-performing model, determined by performance on the validation set, was selected for evaluation on the complete validation set.

6.3.2 YOLOv8 Training Experiments and Results

For a baseline, the pretrained nano (*yolov8n*), small (*yolov8s*), medium (*yolov8m*), large (*yolov8l*) and extra-large (*yolov8x*) YOLOv8 models were run on the SoccerNet-v2 validation set. These baseline models have not been trained to be able to distinguish between referees, players and goalkeepers, so the labels of all these classes were changed to "person", and the balls are under the label "sports ball". The performances of these models can be seen in Figure 6.18.

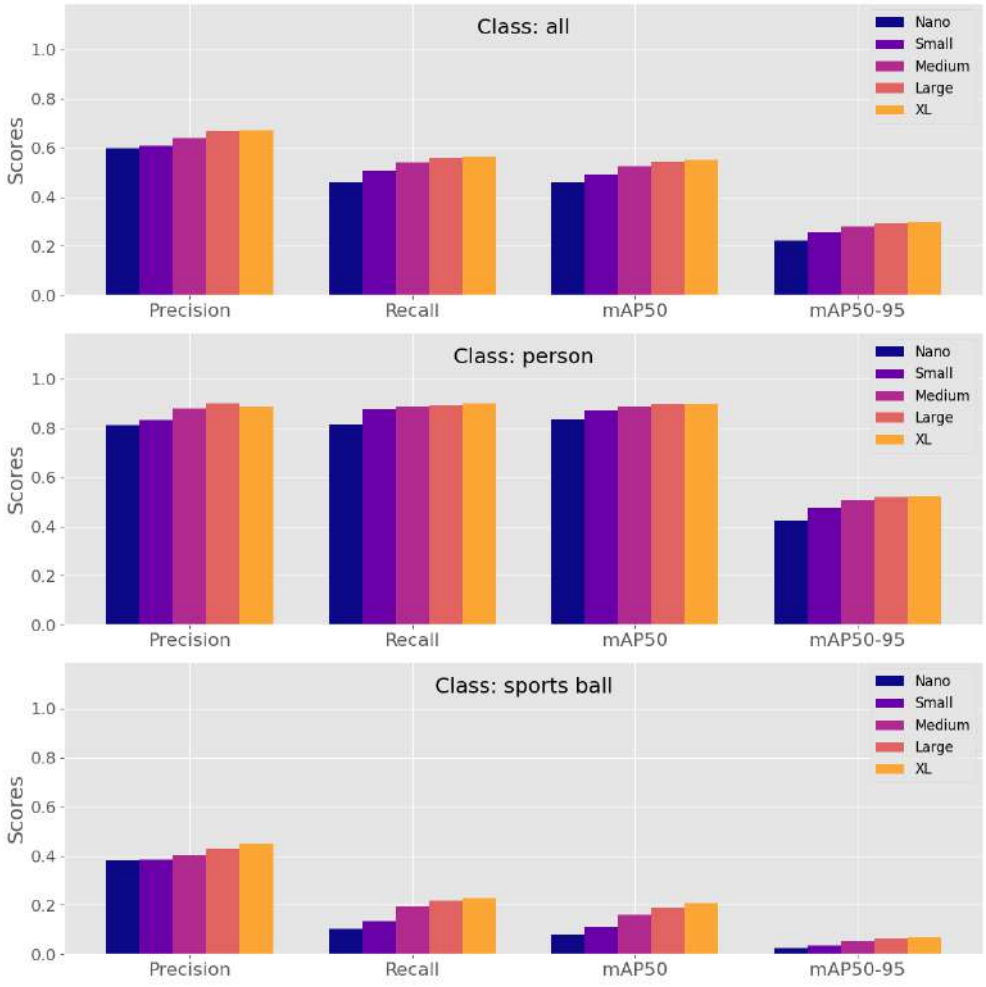


Figure 6.18: Performance of the YOLOv8 baseline models on the SoccerNet-v2 validation set. We see that the larger models perform better in all metrics. We also see that the models find it much harder to detect the balls than the persons.

Overall we see that the larger models perform better than their smaller counterparts in all metrics. It is only when we go from the large model to the extra-large model that it performs worse in one of the metrics - the precision score for the person class. Overall however, the extra-large model outperforms all other models. One thing that is clear is that all of the models struggle when it comes to detecting the sports balls correctly. This is due to the balls' small size and fast movement, as well as the fact that they are sometimes partly or fully hidden behind a player.

One thing to take into account with these baseline models is that their performance could be hindered due to the fact they have been trained to detect all persons in the frame. The SoccerNet-v2 dataset contains persons such a medics, substitutes and managers that are not labeled, and thus predicting these as a person would count as a false positive prediction. Another thing that is worth noticing is that all these models can be run in real-time, with the extra-large model running inference at 100 fps on a Tesla V100-PCIE-32GB. While this is not a necessity for this problem, it could be useful for other tasks.

In order to achieve better results and to be able to distinguish between players and other persons such as referees and managers, YOLOv8 models were trained on the augmented SoccerNet-v2 datasets described in section 4.2. The full analysis of the experiments carried out for this hyperparameter tuning can be found in Appendix A.3. It was found that in order to get the best results, the very large model (*yolov8x*) should be trained on the v4 dataset on images of size 1280x1280 pixels and a momentum value of 0.85. The performance of this model is presented in Figure 6.19.

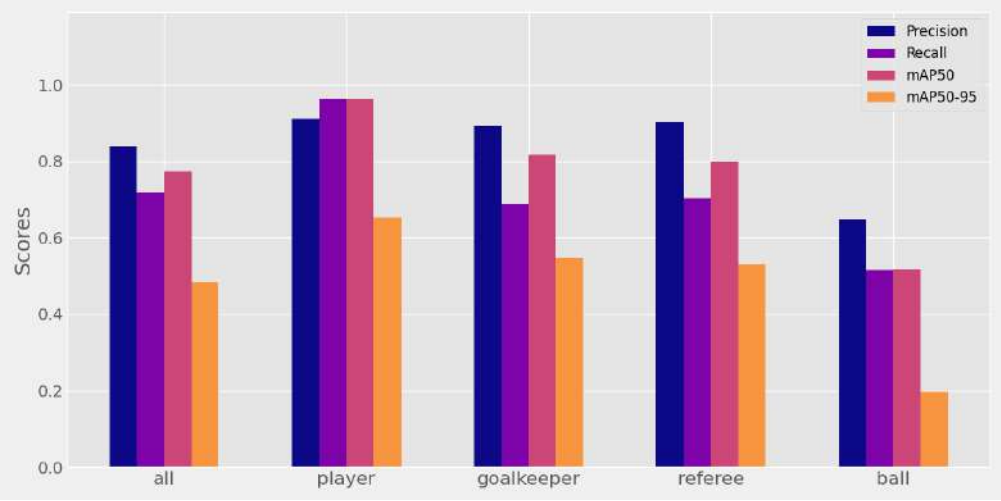


Figure 6.19: Performance of the final YOLOv8 model on the SoccerNet-v2 validation set. The *yolov8x* baseline was trained on the v4 dataset on images of size 1280x1280 pixels and with a momentum of 0.85. We see that it is particularly good at detecting the players, but still struggles when it comes to detecting the balls.

We see that our trained YOLOv8 model performs particularly well when it comes to detecting players, which is also the most import class for us. For the goalkeepers and referees we see that our model has a comparatively higher precision than recall. This suggests that our model learns to predict these classes only when it is sure in its prediction. This is likely an effect of the model having to ignore persons such as medics, substitutes and manager, as these might be similar in appearance to the goalkeepers.

We compare the performances of our player detection model and the best YOLOv8 baseline model in Figure 6.20. Note that we have excluded the goalkeeper and referee classes in this figure as we are only interested in the others. We see that our model outperforms the baseline model by a considerable margin for all classes. One thing to note when comparing the results of our model to the YOLOv8 baseline model is that ours has separate labels for players, referees and goalkeepers, which makes it more challenging to predict correctly.

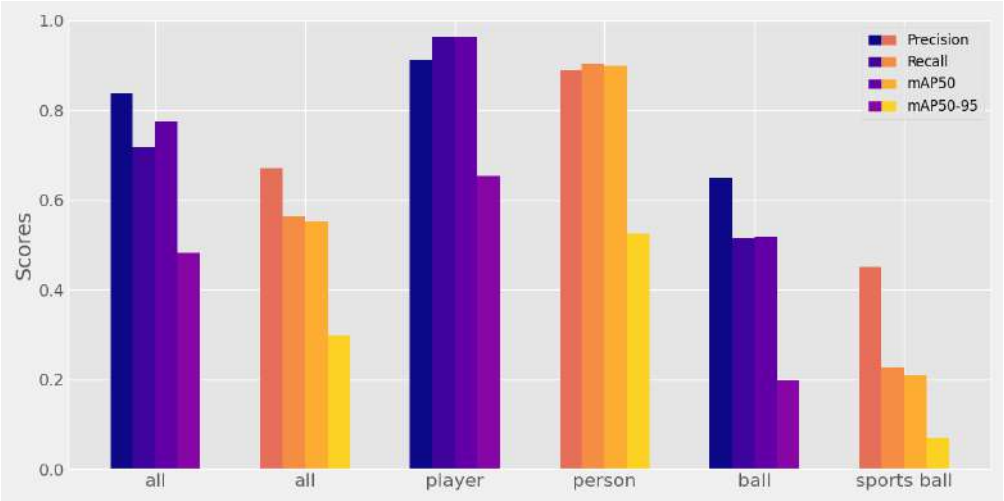


Figure 6.20: Comparison of the pretrained YOLOv8 extra-large model (orange/yellow) and our trained model (blue/purple). When comparing overall scores, as well as class-by-class scores, our model performs significantly better than the pretrained model.

6.3.3 Video Classification with Player and Line Detection Clips

This player detection model was then used in the pipeline described in section 5.3 to create player and line detection clips. These clips were created from both the full training set and the full validation set of well-defined clips. A TSN video classification model was then trained on these new clips, with parameters $clip\_len = 8$, $interval = 2$, $num\_clips = 3$.

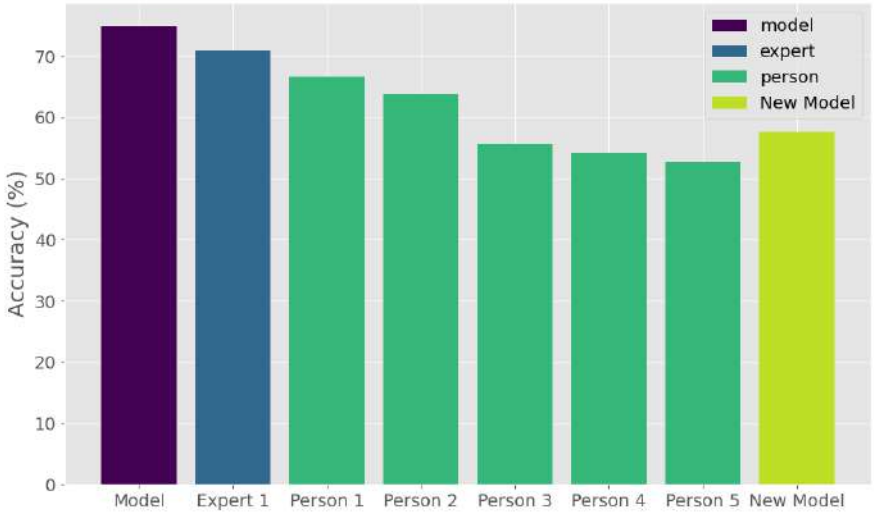


Figure 6.21: Top 1 accuracy of our new model (lime) compared to our original model (purple), normal humans with a knowledge of football (green) as well as human experts (blue).

This model achieved a top 1 accuracy of 57.65% on the validation set, which is considerably lower than the original model. However, it is still comparable to the results of non-expert humans, as can be seen in Figure 6.21. Note that the accuracy of this model has been computed over the full validation set. One positive this model has over the original model is that it is even faster due to the scale of the input images being smaller. Running inference on a Tesla V100-PCIE-16GB GPU resulted in a time of 0.2870 seconds per 8-second clip, compared to 0.47 seconds for the original model despite running on a slower GPU. Note however, that the time taken to create these clips is not taken into account here.

CHAPTER 7

Discussion

In this chapter we will discuss some of our findings and methods. We will touch on topics such as the dataset, the model and the results. We will also look into the alternative method we explored, using player and line detection clips as input for the video classification model, discussing its shortcomings and what we could do to improve it.

DATASET:
Initially, we decided to go with the nine phases described in section 4.1.1. Some of the games we received from DBU also contained annotations of an attacking phase 4 (VI 4 and DE 4). We opted to exclude this phase from our project due to its limited presence in the data, which would have resulted in insufficient samples for effective model learning. Moreover, phase 4 is essentially a subset of phase 3, occurring when the attacking team has an attempt at goal. Consequently, games lacking phase 4 annotations would label these occurrences as phase 3. This would cause the dataset to contain attempts at goal annotated as both phase 3 and phase 4, introducing inconsistency into the dataset.

We briefly mentioned the inclusion of action events such as shots, throw-ins and corners in the data we received. We decided to focus purely on the phase annotations, however, integrating these action events could potentially enhance our model's performance. For instance, training our model to also detect corners and throw-ins could provide insights into the phase of play, given that the ball is out of play during such events. Similarly, leveraging shot annotations could help differentiate between phase 3 and phase 4. By training a shot detection model and combining it with our existing model, we could infer that if a shot occurs during phase 3, the game would in fact be phase 4.

As shown when presenting the results of our model on a per-game basis, there seemed to be a large difference in performance from game to game. While we suggested a few possible causes of this, we are not sure what the actual reason is. We suspect there might be some inaccuracies in the labeling of the data, so this is

something we would have liked to look into if more time was available.

We opted for 8-second clips as input for our video classification model. However, we believe that shorter clips could have potentially improved results. This belief stems from the fact that the feature representations were averaged across all frames in our video classification model, which suggests that our model was not fully utilizing the temporal dynamics of the input clips, as it disregarded the order of frames. Therefore, utilizing shorter clips would yield more fine-grained feature representations, enhancing the effectiveness of our transformer-based model.

THE MODEL:
Our initial choice of using mmaction2's implementation of Temporal Segment Networks (TSN) was originally just to get a video classification model implemented to set a baseline. However, we found it interesting to delve into the effects of different sampling methods, determined by the hyperparameters *num_clips*, *clip_len* and *interval*. Unsurprisingly we found that training a model that sampled single frames from the input clip was not as good as a model that sampled several frames. An interesting result came when we tested our video classification model on clips that had been reversed, as this led to an almost identical accuracy as when testing on normal clips. This results from our model averaging the feature representations across all frames, thus disregarding the order of frames. This leaves us with the belief that we could further optimize our model if we altered it to utilize the temporal dynamics of the input clips.

The second stage of our two-stage solution was the transformer-based model. One objective of this model was to smooth out predictions over consecutive clips, a goal successfully demonstrated in Figure 6.16. Another objective of this model was for it to learn priors concerning the sequential progression of phases. While we could not explicitly test if our model did learn this, its improved performance suggested that it did. When implementing this transformer-based model, we considered using the feature representation of the input clips acquired via the video classification model, as well as the output probabilities. We experimented with both, and using the features resulted in the best performances, likely due to the fact that more information is kept in these 2048-dimensional features than in the 9-dimensional output probabilities. When visualizing the feature representations of the clips (Figure 6.10) we saw that they exhibited clear patterns relating to the phase of play, suggesting that they did indeed contain a lot of information relating to the phase of play.

RESULTS:
When discussing the results and performance, it is important to consider the quality of the dataset and its annotations. The DBU dataset was not specifically tailored for machine learning purposes, but rather for faster analysis of football games. Consequently, the annotations prioritized rough timestamps for each phase of play over precise annotations. Furthermore, it is important to acknowledge that

assigning a specific phase label to a game section is not a purely objective task. Certain sections may exhibit characteristics of multiple unique phases of play, requiring subjective labeling decisions. As a result, our dataset contains some inaccuracies and inconsistencies, making it impossible to create a perfect model.

Another factor to consider when interpreting the results is the balance of the datasets. The datasets consisting of the well-defined clips were relatively well-balanced, with each of the nine phases occurring somewhere between 135 and 440 times in the test set. For the datasets consisting of the consecutive clips, however, there was an imbalance where the "none" phase occurred more frequently than the others. In the test set of consecutive clips, 40.04% of the clips were the "none" phase. A model that blindly predicted "none" every time would thus achieve an accuracy of 40.04%.

With these considerations in mind, our two-stage approach comprising a video classification model followed by a transformer-based model proved to be a successful model design. While our video classification model proved to be very accurate even on its own, surpassing human expert scores (Section 6.1.4) on 8-second clips, we saw in Figure 6.16 that it had a tendency to jump between predictions for consecutive clips. Our transformer-based model made the predictions more smooth, as can also be seen in Figure 6.16, and increased the accuracy from 64.08% to 71.72% on the test set of consecutive clips. Additionally, our model's impressive running time is noteworthy, with inference on a complete 90-minute football game for both models combined taking only 5.5 minutes. This means that our model would be able to annotate almost 100 games overnight, which could prove very valuable in tournaments where coaches only have a few days to analyse their opponents.

PLAYER AND LINE DETECTION CLIPS:
Although the results obtained from using player and line detection clips were suboptimal, we remain optimistic about the potential success of this method. The primary reason for the lackluster performance of this method lies in the need for optimization across several aspects. While we succeeded in developing a proficient player detection model, both our line segmentation and clustering models left room for improvement. With enhancements to these models, we are confident that this method could yield promising results.

CHAPTER 8

Conclusion

This chapter concludes our work, where we addressed the challenge of automating the detection of phases of play in football games, a process currently done manually. With our two-stage solution comprising a video classification model followed by a transformer-based model, we managed to streamline the process of phase detection. Our solution showcased impressive efficiency, capable of classifying a full 90-minute football game in less than six minutes — significantly faster than manual labeling.

Our video classification model, utilizing Temporal Segment Networks (TSN), demonstrated remarkable accuracy in classifying phases of play in 8-second clips, surpassing human experts 75% to 70.83% on a subset of 72 clips. T-distributed Stochastic Neighbor Embedding (t-SNE) analysis revealed that our model learns insightful patterns, including distinct clusters for attacking and defending phases, as well as sequential and cyclic progressions resembling in-game scenarios. The model, however, did not fully utilize the temporal information of the input clips as it averaged feature representations across all frames, an aspect we look to address in future iterations.

Our transformer-based model utilized feature representations of consecutive clips, processed through our video classification model, to generate refined predictions. Its main objectives were to smooth out inconsistencies in the original predictions, which we successfully demonstrated, and to incorporate prior knowledge regarding the evolution of phases. Although explicit evidence of the model learning these priors was absent, its significant accuracy improvement of 7.64% over the original predictions strongly suggests its success in doing so.

While we managed to create a good player detection model, our approach of using player and line detection clips failed to reach performance levels near those of our original model. This outcome was primarily due to sub-optimal line detection and clustering models. Nonetheless, despite its relatively modest performance, this approach demonstrated the potential of leveraging positional information of players

to determine phases of play.

In conclusion, our solution provides an accurate and efficient framework for detecting phases of play in football games, successfully answering the research question: *"How can machine learning approaches be effectively applied to temporally localize and classify group activities in football videos?"*. This research paves the way for more automated and precise analysis of football games.

CHAPTER 9

Future Work

After concluding our study, which demonstrated the effectiveness of our solution, we will discuss avenues for future investigation and enhancement. Furthermore, we will delve into alternative approaches that hold promise for future implementation.

One thing that can always be optimized is the hyperparameters, which is one path for future improvements. There are also several model modifications that could be investigated, such as using shorter input clips for our video classification model, longer sequences for our transformer-based model or a combination of features and probabilities for our transformer-based model. While our player detection model was good, it struggled with the task of detecting the balls, thus extending this model to also be able to accurately detect the balls, or alternatively creating a specialized ball detection model, is another model modification to look into. Additionally, our line detection and clustering models had plenty of room for improvement, which could provide the base for future work.

Football is a very dynamic sport where a lot can happen even within short time frames. We therefore believe that the temporal information of the input videos is crucial for making a good model. We saw that our TSN video classification model disregards the order of frames as it averages feature representations across all frames. This underscores the need for improvement in future iterations. One approach we propose involves sampling shorter input clips to preserve temporal dynamics. An extreme case of this would be sampling input clips of single frames, however this would likely slow down our model drastically. Alternatively, exploring different video classification models that utilize frame order presents another avenue for enhancement.

Another aspect we would like to improve is our dataset. As we discussed in Chapter 7, we suspected that there were some inaccuracies within the labeling of the dataset which could be studied further. Furthermore, we would like to increase the size of the dataset. This could be done in a semi-supervised fashion, where we use predictions from our current model to label new data, and for cases where the

model's uncertainty is above a certain threshold we will manually label those clips. This would speed up the labeling process compared to the current fully manual labeling process.

An initial idea of ours that we didn't get around to implementing was replacing the line segmentation model with a model that can map the image of the pitch to a bird's-eye view. Achieving a bird's-eye view mapping of the pitch, combined with an accurate player detection model, would give us the exact coordinates of the players on the pitch. From this we could create bird's-eye view clips of player positions with a static pitch as background. This would remove the noise of inaccurate line detections, and would simplify the data while still keeping all the positional information of the players.

Extracting the exact player coordinates would open up many possibilities when it comes to detecting the phase of play. One such possibility is a temporal graphical model, where the players are connected to each other based on their distances between each other. This would vastly reduce the dimensions of the data as each frame would go from a 448x448 (or larger) image to a graph of 22 nodes.

In conclusion, our solution provides an accurate and efficient solution to the task of detecting phases of play in football games. However, there are several possible areas of improvement as explored in this chapter. Particularly, we believe that incorporating more temporal information into our model would lead to increased performance.

References

[1] Soccer is a sport in the world. URL: https://www.bartleby.com/essay/Soccer-Is-A-Sport-In-The-World-FJV5YW2U49V.

[2] Deloitte. Annual review of football finance europe. https://www2.deloitte.com/uk/en/pages/sports-business-group/articles/annual-review-of-football-finance-europe.html, 2023. Accessed: 05/03/2024.

[3] Fifa.com. https://www.fifa.com/fifaplus/en/tournaments/mens/worldcup/canadamexicousa2026, 2024. Accessed: 27/02/2024.

[4] I. C. Education. What are neural networks? URL: https://www.ibm.com/cloud/learn/neural-networks.

[5] I. C. Education. What are convolutional neural networks? URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. arXiv: 1512.03385 [cs.CV].

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017. arXiv: 1706.03762 [cs.CL].

[8] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: a survey. *ACM Computing Surveys*, 54(10s):1–41, Jan. 2022. ISSN: 1557-7341. DOI: 10.1145/3505244. URL: http://dx.doi.org/10.1145/3505244.

[9] T. Shehzadi, K. A. Hashmi, D. Stricker, and M. Z. Afzal. Object detection with transformers: a review, 2023. arXiv: 2306.04670 [cs.CV].

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: transformers for image recognition at scale, 2021. arXiv: 2010.11929 [cs.CV].

[11] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers, 2020. arXiv: 2005.12872 [cs.CV].

[12] Z. Zong, G. Song, and Y. Liu. Detrs with collaborative hybrid assignments training, 2023. arXiv: 2211.12860 [cs.CV].

[13] Q. Chen, X. Chen, J. Wang, S. Zhang, K. Yao, H. Feng, J. Han, E. Ding, G. Zeng, and J. Wang. Group detr: fast detr training with group-wise one-to-many assignment, 2023. arXiv: 2207.13085 [cs.CV].

[14] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid. Vivit: a video vision transformer, 2021. arXiv: 2103.15691 [cs.CV].

[15] G. Bertasius, H. Wang, and L. Torresani. Is space-time attention all you need for video understanding?, 2021. arXiv: 2102.05095 [cs.CV].

[16] Z. Tong, Y. Song, J. Wang, and L. Wang. Videomae: masked autoencoders are data-efficient learners for self-supervised video pre-training, 2022. arXiv: 2203.12602 [cs.CV].

[17] L. Wang, B. Huang, Z. Zhao, Z. Tong, Y. He, Y. Wang, Y. Wang, and Y. Qiao. Videomae v2: scaling video masked autoencoders with dual masking, 2023. arXiv: 2303.16727 [cs.CV].

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. DOI: 10.1109/5.726791.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. arXiv: 1409.1556 [cs.CV].

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions, 2014. arXiv: 1409.4842 [cs.CV].

[22] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, 886–893 vol. 1, 2005. DOI: 10.1109/CVPR.2005.177.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. arXiv: 1311.2524 [cs.CV].

[24] R. Girshick. Fast r-cnn, 2015. arXiv: 1504.08083 [cs.CV].

[25] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks, 2016. arXiv: 1506.01497 [cs.CV].

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: unified, real-time object detection, 2016. arXiv: 1506.02640 [cs.CV].

[27] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger, 2016. arXiv: 1612.08242 [cs.CV].

[28] J. Redmon and A. Farhadi. Yolov3: an incremental improvement, 2018. arXiv: 1804.02767 [cs.CV].

[29] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. Yolov4: optimal speed and accuracy of object detection, 2020. arXiv: 2004.10934 [cs.CV].

[30] Ultralytics. YOLOv5: A state-of-the-art real-time object detection system. https://docs.ultralytics.com, 2021.

[31] G. Jocher, A. Chaurasia, and J. Qiu. YOLO by Ultralytics, version 8.0.0, Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[32] 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)*. IEEE, Sept. 2016. DOI: 10.1109/icip.2016.7533003. URL: http://dx.doi.org/10.1109/ICIP.2016.7533003.

[33] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric, 2017. arXiv: 1703.07402 [cs.CV].

[34] J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M.-H. Yang. Online multi-object tracking with dual matching attention networks, 2019. arXiv: 1902.00749 [cs.CV].

[35] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang. Bytetrack: multi-object tracking by associating every detection box, 2022. arXiv: 2110.06864 [cs.CV].

[36] J. Li, X. Gao, and T. Jiang. Graph networks for multiple object tracking. In *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 708–717, 2020. DOI: 10.1109/WACV45572.2020.9093347.

[37] X. Liu and H. Caesar. Offline tracking with object permanence, 2023. arXiv: 2310.01288 [cs.CV].

[38] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos, 2014. arXiv: 1406.2199 [cs.CV].

[39] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks, 2015. arXiv: 1412.0767 [cs.CV].

[40] J. Carreira and A. Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset, 2018. arXiv: 1705.07750 [cs.CV].

[41] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition, 2019. arXiv: 1812.03982 [cs.CV].

[42] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. V. Gool. Temporal segment networks: towards good practices for deep action recognition, 2016. arXiv: 1608.00859 [cs.CV].

[43] H. Xu, A. Das, and K. Saenko. R-c3d: region convolutional 3d network for temporal activity detection, 2017. arXiv: 1703.07814 [cs.CV].

[44] R. Zeng, W. Huang, M. Tan, Y. Rong, P. Zhao, J. Huang, and C. Gan. Graph convolutional networks for temporal action localization, 2019. arXiv: 1909.03252 [cs.CV].

[45] J. Tan, J. Tang, L. Wang, and G. Wu. Relaxed transformer decoders for direct action proposal generation, 2021. arXiv: 2102.01894 [cs.CV].

[46] T. Lin, X. Zhao, H. Su, C. Wang, and M. Yang. Bsn: boundary sensitive network for temporal action proposal generation, 2018. arXiv: 1806.02964 [cs.CV].

[47] C. Lin, C. Xu, D. Luo, Y. Wang, Y. Tai, C. Wang, J. Li, F. Huang, and Y. Fu. Learning salient boundary feature for anchor-free temporal action localization, 2021. arXiv: 2103.13137 [cs.CV].

[48] T.-K. Kang, G.-H. Lee, and S.-W. Lee. Htnet: anchor-free temporal action localization with hierarchical transformers, 2022. arXiv: 2207.09662 [cs.CV].

[49] M. Xu, J.-M. Perez-Rua, X. Zhu, B. Ghanem, and B. Martinez. Low-fidelity end-to-end video encoder pre-training for temporal action localization, 2021. arXiv: 2103.15233 [cs.CV].

[50] A. Deliège, A. Cioppa, S. Giancola, M. J. Seikavandi, J. V. Dueholm, K. Nasrollahi, B. Ghanem, T. B. Moeslund, and M. V. Droogenbroeck. Soccernet-v2: a dataset and benchmarks for holistic understanding of broadcast soccer videos, 2021. arXiv: 2011.13367 [cs.CV].

[51] S. Giancola, A. Cioppa, A. Deliège, F. Magera, V. Somers, L. Kang, X. Zhou, O. Barnich, C. De Vleeschouwer, A. Alahi, B. Ghanem, M. Van Droogenbroeck, A. Darwish, A. Maglo, A. Clapés, A. Luyts, A. Boiarov, A. Xarles, A. Orcesi, A. Shah, B. Fan, B. Comandur, C. Chen, C. Zhang, C. Zhao, C. Lin, C.-Y. Chan, C. C. Hui, D. Li, F. Yang, F. Liang, F. Da, F. Yan, F. Yu, G. Wang, H. A. Chan, H. Zhu, H. Kan, J. Chu, J. Hu, J. Gu, J. Chen, J. V. B. Soares, J. Theiner, J. De Corte, J. H. Brito, J. Zhang, J. Li, J. Liang, L. Shen, L. Ma, L. Chen, M. Santos Marques, M. Azatov, N. Kasatkin, N. Wang, Q. Jia, Q. C. Pham, R. Ewerth, R. Song, R. Li, R. Gade, R. Debien, R. Zhang, S. Lee, S. Escalera, S. Jiang, S. Odashima, S. Chen, S. Masui, S. Ding, S.-w. Chan, S. Chen, T. El-Shabrawy, T. He, T. B. Moeslund, W.-C. Siu, W. Zhang, W. Li, X. Wang, X. Tan, X. Li, X. Wei, X. Ye, X. Liu, X. Wang, Y. Guo, Y. Zhao, Y. Yu, Y. Li, Y. He, Y. Zhong, Z. Guo, and Z. Li. Soccernet 2022 challenges results. In *Proceedings of the 5th International ACM Workshop on Multimedia Content Analysis in Sports*, MM '22. ACM, Oct. 2022. DOI: 10.1145/3552437.3558545. URL: http://dx.doi.org/10.1145/3552437.3558545.

[52] A. Cioppa, S. Giancola, V. Somers, F. Magera, X. Zhou, H. Mkhallati, A. Deliège, J. Held, C. Hinojosa, A. M. Mansourian, P. Miralles, O. Barnich, C. D. Vleeschouwer, A. Alahi, B. Ghanem, M. V. Droogenbroeck, A. Kamal, A. Maglo, A. Clapés, A. Abdelaziz, A. Xarles, A. Orcesi, A. Scott, B. Liu, B. Lim, C. Chen, F. Deuser, F. Yan, F. Yu, G. Shitrit, G. Wang, G. Choi,

H. Kim, H. Guo, H. Fahrudin, H. Koguchi, H. Ardö, I. Salah, I. Yerushalmy, I. Muhammad, I. Uchida, I. Be'ery, J. Rabarisoa, J. Lee, J. Fu, J. Yin, J. Xu, J. Nang, J. Denize, J. Li, J. Zhang, J. Kim, K. Synowiec, K. Kobayashi, K. Zhang, K. Habel, K. Nakajima, L. Jiao, L. Ma, L. Wang, L. Wang, M. Li, M. Zhou, M. Nasr, M. Abdelwahed, M. Liashuha, N. Falaleev, N. Oswald, Q. Jia, Q.-C. Pham, R. Song, R. Hérault, R. Peng, R. Chen, R. Liu, R. Baikulov, R. Fukushima, S. Escalera, S. Lee, S. Chen, S. Ding, T. Someya, T. B. Moeslund, T. Li, W. Shen, W. Zhang, W. Li, W. Dai, W. Luo, W. Zhao, W. Zhang, X. Yang, Y. Ma, Y. Joo, Y. Zeng, Y. Gan, Y. Zhu, Y. Zhong, Z. Ruan, Z. Li, Z. Huang, and Z. Meng. Soccernet 2023 challenges results, 2023. arXiv: 2309.06006 [cs.CV].

[53] G. Jin. Player target tracking and detection in football game video using edge computing and deep learning. *The Journal of Supercomputing*, 78(7):9475–9491, 2022. ISSN: 1573-0484. DOI: 10.1007/s11227-021-04274-6.

[54] P. Mavrogiannis and I. Maglogiannis. Amateur football analytics using computer vision. *Neural Computing and Applications*, 34(22):19639–19654, 2022. ISSN: 1433-3058. DOI: 10.1007/s00521-022-07692-6.

[55] W. Kim, S.-W. Moon, J. Lee, D.-W. Nam, and C. Jung. Multiple player tracking in soccer videos: an adaptive multiscale sampling approach. *Multimedia Systems*, 24(6):611–623, 2018. ISSN: 1432-1882. DOI: 10.1007/s00530-018-0586-9.

[56] A. Maglo, A. Orcesi, and Q.-C. Pham. Efficient tracking of team sport players with few game-specific annotations, 2022. arXiv: 2204.04049 [cs.CV].

[57] G. Shitrit, I. Be'ery, and I. Yerhushalmy. Soccernet 2023 tracking challenge – 3rd place mot4mot team technical report, 2023. arXiv: 2308.16651 [cs.CV].

[58] J. Komorowski and G. Kurzejamski. Graph-based multi-camera soccer player tracker. In *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2022. DOI: 10.1109/ijcnn55064.2022.9892562. URL: http://dx.doi.org/10.1109/IJCNN55064.2022.9892562.

[59] X. Fu, W. Huang, Y. Sun, X. Zhu, J. Evans, X. Song, T. Geng, and S. He. A novel dataset for multi-view multi-player tracking in soccer scenarios. *Applied Sciences*, 13(9), 2023. ISSN: 2076-3417. DOI: 10.3390/app13095361. URL: https://www.mdpi.com/2076-3417/13/9/5361.

[60] K. Okuma, J. J. Little, and D. G. Lowe. Automatic rectification of long image sequences. In *Asian Conference on Computer Vision*, volume 9, 2004.

[61] A. Gupta, J. J. Little, and R. J. Woodham. Using line and ellipse features for rectification of broadcast hockey video. In *2011 Canadian conference on computer and robot vision*, pages 32–39. IEEE, 2011.

[62] P.-C. Wen, W.-C. Cheng, Y.-S. Wang, H.-K. Chu, N. C. Tang, and H.-Y. M. Liao. Court reconstruction for camera calibration in broadcast basketball videos. *IEEE transactions on visualization and computer graphics*, 22(5):1517–1526, 2015.

[63] A. Maglo, A. Orcesi, J. Denize, and Q. C. Pham. Individual locating of soccer players from a single moving view. *Sensors*, 23(18), 2023. ISSN: 1424-8220. DOI: 10.3390/s23187938. URL: https://www.mdpi.com/1424-8220/23/18/7938.

[64] J. V. B. Soares, A. Shah, and T. Biswas. Temporally precise action spotting in soccer videos using dense detection anchors. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 2796–2800, 2022. DOI: 10.1109/ICIP46576.2022.9897256.

[65] A. Xarles, S. Escalera, T. B. Moeslund, and A. Clapés. Astra: an action spotting transformer for soccer videos. In *Proceedings of the 6th International Workshop on Multimedia Content Analysis in Sports*, MMSports '23, pages 93–102, New York, NY, USA. Association for Computing Machinery, 2023. DOI: 10.1145/3606038.3616153. URL: https://doi.org/10.1145/3606038.3616153.

[66] J. Denize, M. Liashuha, J. Rabarisoa, A. Orcesi, and R. Hérault. Comedian: self-supervised learning and knowledge distillation for action spotting using transformers. *arXiv preprint arXiv:2309.01270*, 2023.

[67] S. Giancola, A. Cioppa, J. Georgieva, J. Billingham, A. Serner, K. Peek, B. Ghanem, and M. V. Droogenbroeck. Towards active learning for action spotting in association football videos, 2023. arXiv: 2304.04220 [cs.CV].

[68] F. Vidal-Codina, N. Evans, B. El Fakir, and J. Billingham. Automatic event detection in football using tracking data. *Sports Engineering*, 25(1):18, 2022. ISSN: 1460-2687. DOI: 10.1007/s12283-022-00381-6. URL: https://doi.org/10.1007/s12283-022-00381-6.

[69] C. Direkoğlu and N. E. O'Connor. Temporal segmentation and recognition of team activities in sports. *Machine Vision and Applications*, 29(5):891–913, 2018. ISSN: 1432-1769. DOI: 10.1007/s00138-018-0944-9. URL: https://doi.org/10.1007/s00138-018-0944-9.

[70] S. Li, Q. Cao, L. Liu, K. Yang, S. Liu, J. Hou, and S. Yi. Groupformer: group activity recognition with clustered spatial-temporal transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13668–13677, 2021.

[71] M. Ötting and D. Karlis. Football tracking data: a copula-based hidden markov model for classification of tactics in football. *Annals of Operations Research*, 325(1):167–183, 2023. ISSN: 1572-9338. DOI: 10.1007/s10479-022-04660-0. URL: https://doi.org/10.1007/s10479-022-04660-0.

[72] C.-Y. Wu, C. Feichtenhofer, H. Fan, K. He, P. Krähenbühl, and R. Girshick. Long-term feature banks for detailed video understanding, 2019. arXiv: 1812.05038 [cs.CV].

[73] C.-Y. Wu and P. Krähenbühl. Towards long-form video understanding, 2021. arXiv: 2106.11310 [cs.CV].

[74] F. Sener, D. Singhania, and A. Yao. Temporal aggregate representations for long-range video understanding, 2020. arXiv: 2006.00830 [cs.CV].

[75] Open MMLab. MMAction2. https://github.com/open-mmlab/mmaction2/blob/main/configs/recognition/tsn/README.md, 2024.

[76] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL: http://jmlr.org/papers/v9/vandermaaten08a.html.

[77] Ultralytics. Yolov8. https://docs.ultralytics.com, 2021.

Appendix

A.1 YOLOv8 Architecture

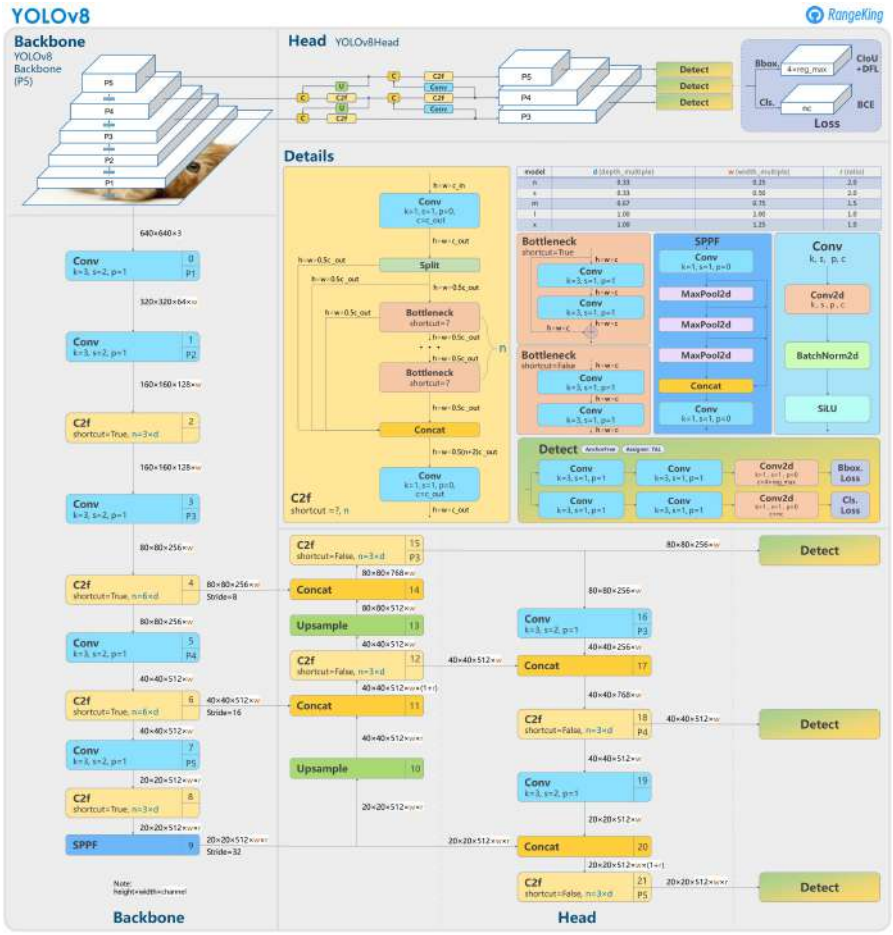


Figure A.1: Graph of YOLOv8 architecture by GitHub user RangeKing, taken from https://github.com/ultralytics/ultralytics/issues/189

A.2 TSN Experiments

scale	clip_len	interval	num_clips	epoch	top 1	top 5	mean top 1
448	1	1	1	56	0.6359	0.9728	0.6241
448	1	1	2	44	0.6467	0.9807	0.6439
448	2	1	2	41	0.6597	0.9819	0.6583
448	3	1	2	18	0.6529	0.9898	0.6653
448	4	1	2	21	0.6382	0.9807	0.6345
448	5	1	2	41	0.6540	0.9796	0.6632
448	6	1	2	24	0.6387	0.9830	0.6337
448	6	1	3	41	0.6625	0.9796	0.6567
448	6	2	3	25	0.6484	0.9881	0.6594
448	6	3	3	24	0.6387	0.9864	0.6371
448	6	4	3	15	0.6229	0.9813	0.6453
448	6	6	3	15	0.6382	0.9853	0.6351
448	6	8	3	13	0.6308	0.9892	0.6448
448	7	1	2	41	0.6619	0.9807	0.6713
448	8	1	2	36	0.6506	0.9836	0.6618
448	10	1	1	31	0.6251	0.9813	0.6244
448	**10**	**1**	**2**	**28**	**0.6591**	**0.9881**	**0.6744**
448	10	2	2	41	0.6433	0.9847	0.6386
448	10	3	2	21	0.6280	0.9824	0.6084
448	10	4	2	15	0.6302	0.9881	0.6294
448	10	5	2	28	0.6325	0.9796	0.6512
448	10	6	2	26	0.6336	0.9847	0.6397
448	10	7	2	21	0.6393	0.9841	0.6444
448	10	8	2	16	0.6376	0.9819	0.6372
448	10	9	2	10	0.6325	0.9841	0.6444
448	10	10	2	14	0.6410	0.9904	0.6349
448	12	1	2	41	0.6518	0.9847	0.6563
448	12	2	2	17	0.6472	0.9870	0.6604
448	12	3	2	25	0.6438	0.9836	0.6489
448	12	4	2	28	0.6478	0.9858	0.6625
448	12	5	2	21	0.6399	0.9841	0.6329
448	16	1	2	43	0.6331	0.9785	0.6157

Table A.1: Table of all TSN video classification model experiments carried out. It shows the parameters (scale, clip len, interval and num) for each given training run, the epoch where the best model was reached, as well as the performances on the validation set. Our final model is highlighted in bold.

A.3 YOLOv8 Experiments

Here, the YOLOv8 experiment results are shown in-depth. The graphs show the four evaluation metrics described in section 6.3.1, both for each class individually and the average among all classes, for the various experiments performed. Since the ”ball” class had comparably much smaller scores in all of the metrics, and also is not as important for this thesis, these values have been excluded from the graphs for better readability. Unless otherwise stated, all hyperparameters such as momentum, batch size, learning rte, etc. were set to the default values which can be found here [77].

Figure A.2 shows the performance of training the different sizes of the yolov8 models: nano (yolov8n), small (yolov8s), medium (yolov8m), large (yolov8l), and very large (yolov8x) on v1 of the dataset. Here it is clear to see the trend that the larger the model the better the performance. It is also worth noting that even the largest model can run inference in real time, as it had an inference of 10.2 ms / 98 fps.

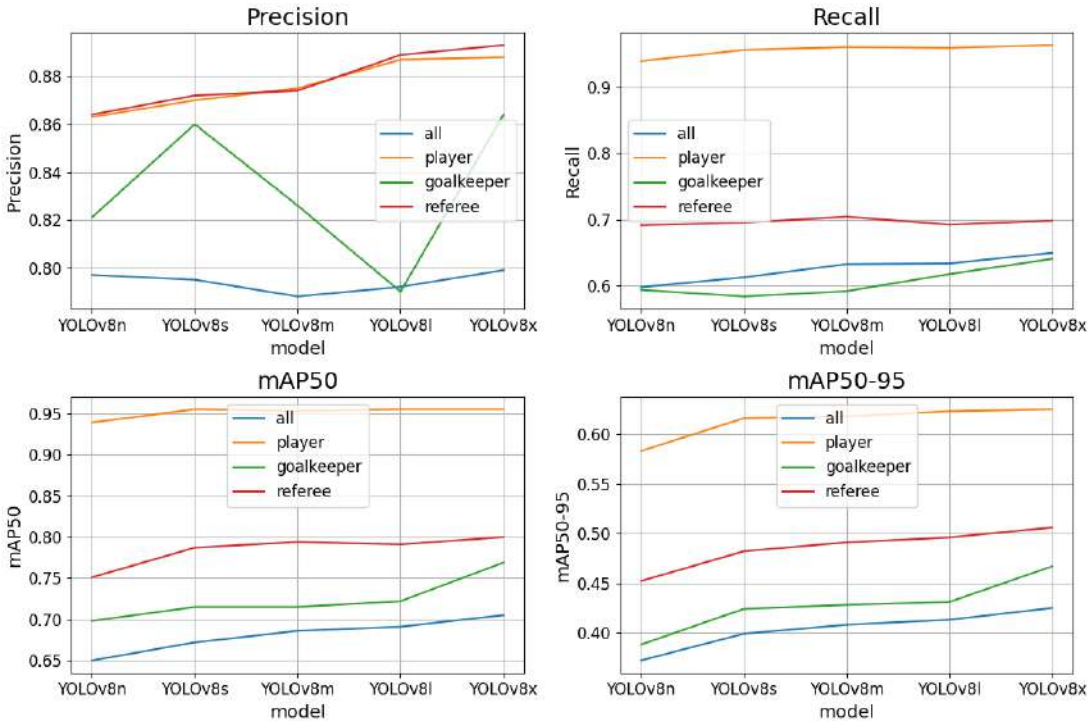


Figure A.2: YOLOv8 runs with different baseline models, trained on v1 of the dataset.

Figure A.3 shows the performances when training the large model (yolov8l) on the various versions of the dataset. Here the results vary a bit more, but generally runs trained on v3, v4 and v5 seem to have the best results.

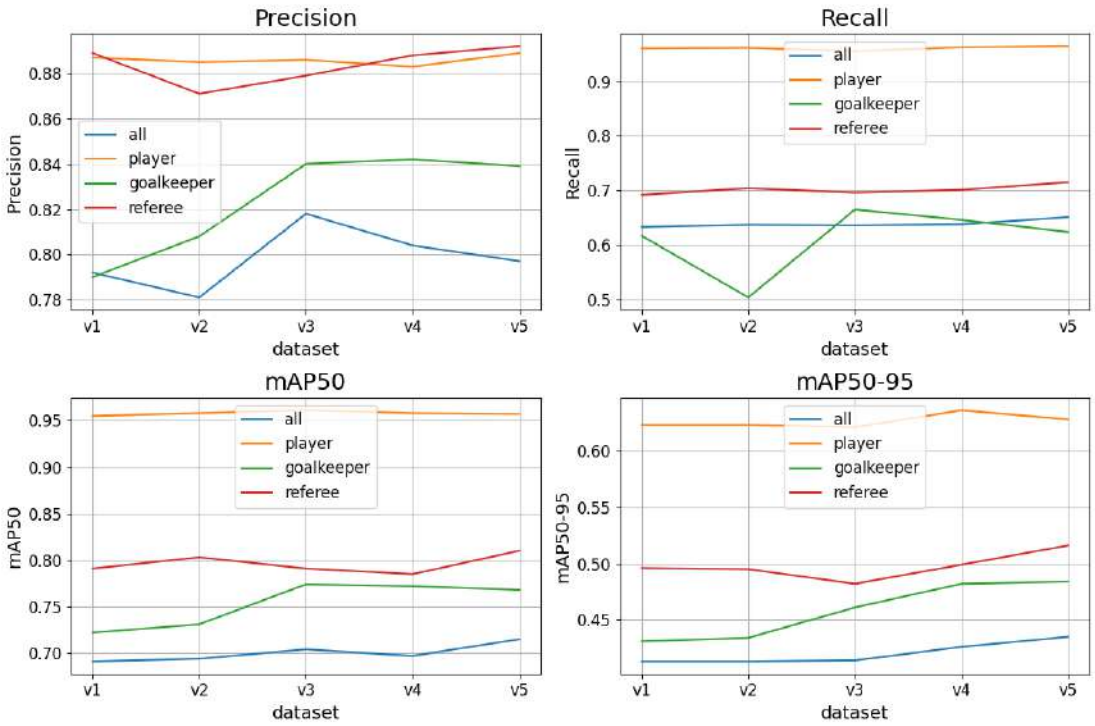


Figure A.3: YOLOv8l (large model) runs trained on different datasets.

Figure A.4 shows the performances when training the large model (yolov8l) on the v3 dataset with varying image sizes. Here there is also a clear trend that training on larger images leads to better results. However going from an image size of 1280 to 1920 pixels doesn't seem to improve the results much more.

Figure A.5 shows the performances when training the large model (yolov8l) on the v3 dataset with different momentum values and an image size of 1280. The results here are a bit harder to draw a conclusion from, but generally it seems that a momentum value in the range of 0.85 to 0.9 is optimal.

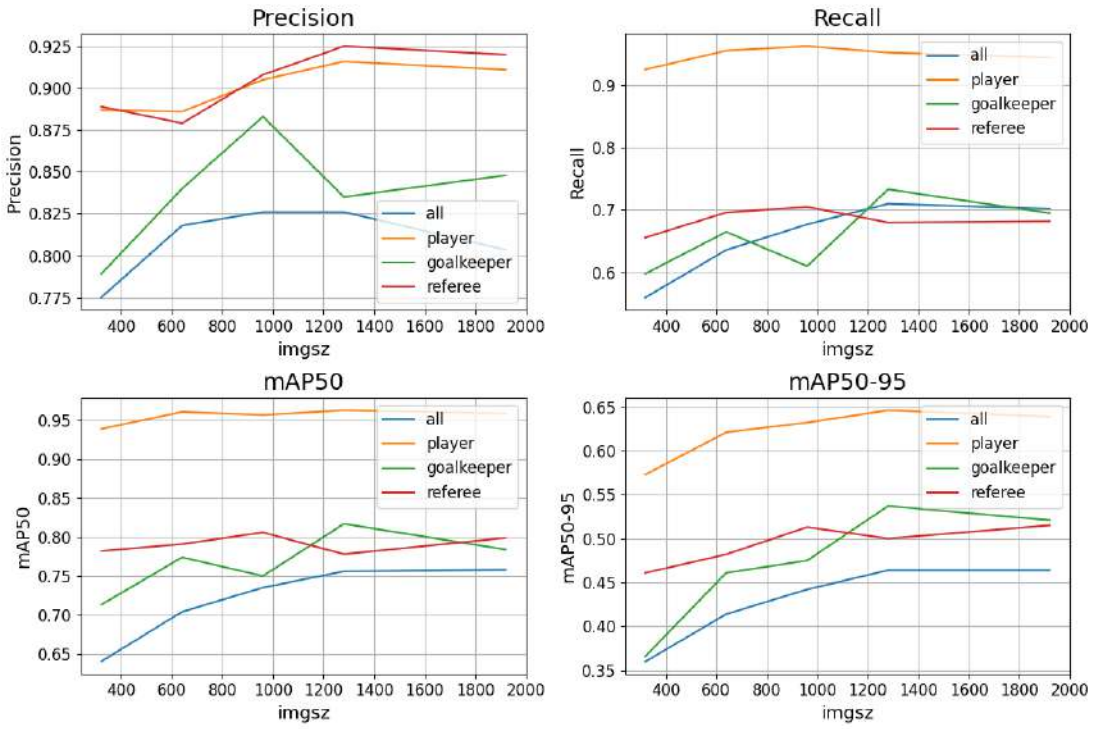


Figure A.4: YOLOv8l (large model) trained on the v3 dataset with different image sizes.

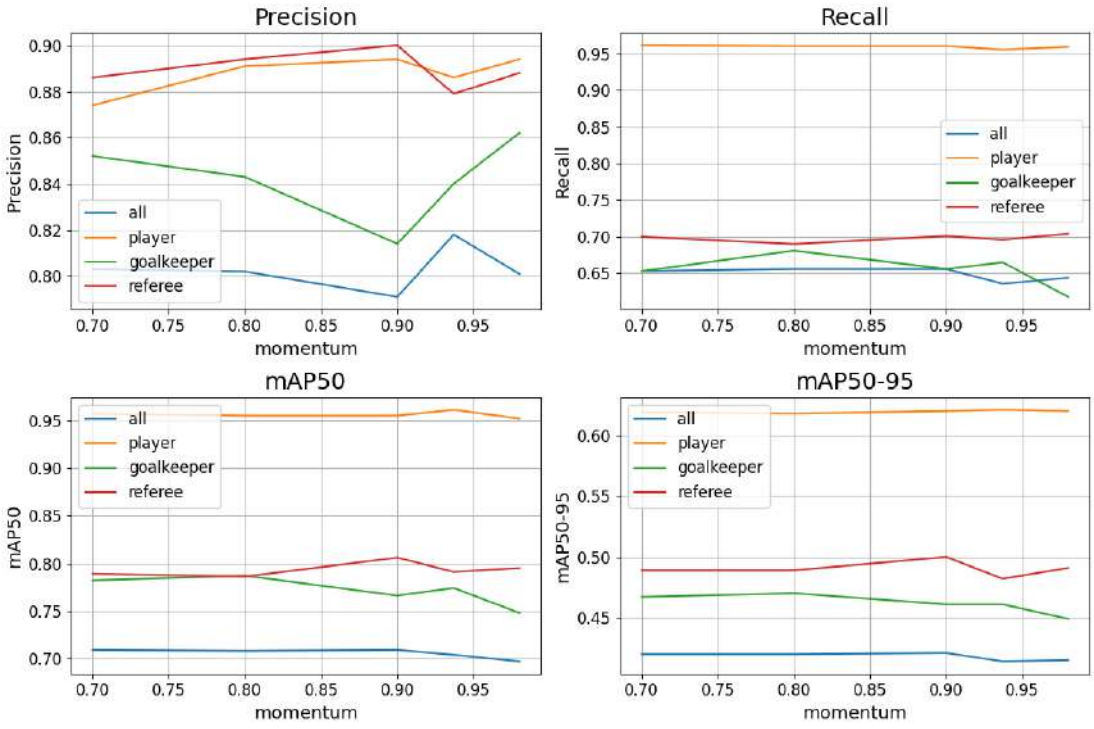


Figure A.5: YOLOv8l (large model) trained on the v3 dataset with different momentum values. Input images are 1280 pixels.

Based on the previous experiments, a final experiment was carried out by training the very large model (yolov8x) on the v3 and v4 datasets with different momentum settings and an image size of 1280 pixels. The results can be seen in figure A.6 and show that the best results are achieved by training the very large model (yolov8x) on the v4 dataset with a momentum of 0.85.

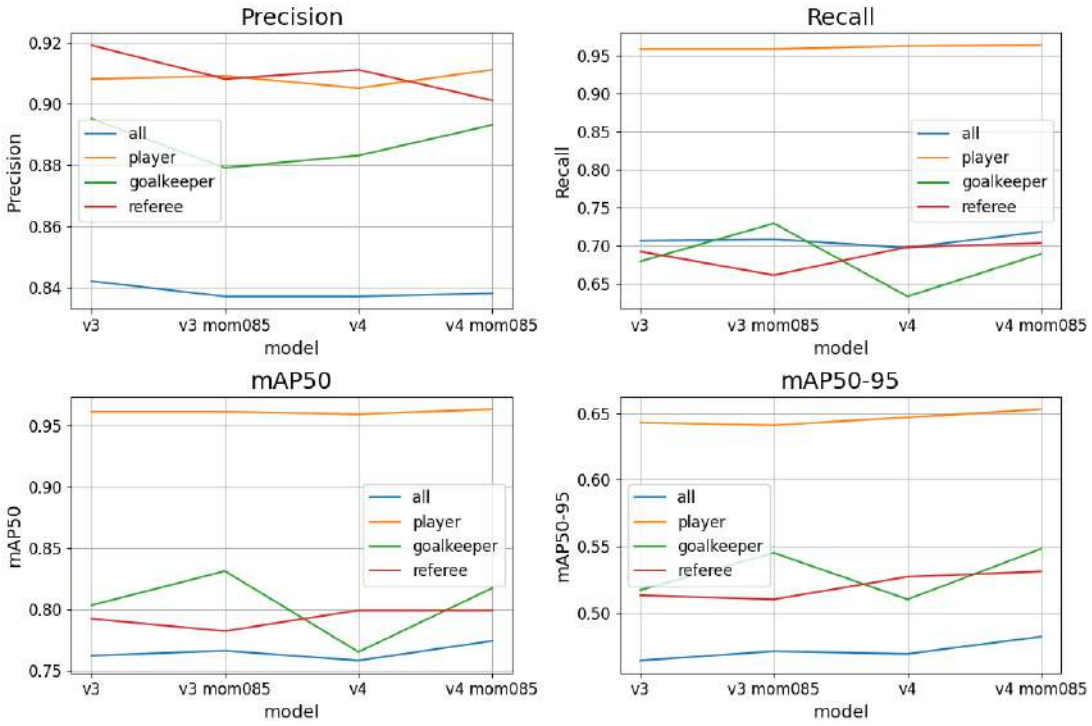


Figure A.6: YOLOv8x (very large model) trained on the v3 and v4 datasets with different momentum values. Input images are 1280 pixels.