Automatic event detection in soccer videos

Olav Rongved

Thesis submitted for the degree of
Master in computer science
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2020

Automatic event detection in soccer videos

Olav Rongved

Automatic event detection in soccer videos

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Today, events in soccer are manually annotated by humans. Annotation is a tedious, time-consuming, and expensive task. Event annotations are useful for multiple different purposes, such as entertainment, analysis, and more. Automating the process can save much time, and provide access to event annotations in leagues where it may be financially infeasible to annotate manually. Furthermore, with a general approach, it would be possible to annotate events that were previously ignored. Our objectives are to research and find general solutions for automatic event detection that can be easily adapted to new events, implement and find a good approach through experimental prototyping, analysis of our approach, and comparison to the state-of-the-art. Action recognition and detection are active areas of research. There are many approaches to these problems, but in recent years, approaches have mostly been dominated by deep-learning-based methods. Based on current research in action recognition and detection, we experiment with four different architectures based on neural networks. For comparison, we use a large annotated dataset for soccer called SoccerNet [19], where the goal is to predict temporal anchors for the events '*Goal*', '*Substitution*' and '*Card*' events within some distance of the ground truth. Additionally, we download videos from Swedish Allsvenskan and Norwegian Eliteserien to test cross-dataset performance. To select our model, we experiment by using locally extracted video-frames in a classification task. We extensively test different configurations, such as the temporal extent of input, resolution, gray-scale input, and the use of pretrained models. We find that a pretrained ResNet 3D-18 model performs best with 128 frame inputs, resulting in 88.4% accuracy on the validation set for the classification task.

For the task of spotting in soccer videos, we use a sliding-window approach with a stride of 1 second. We report 51% Average-mean-Average-precision (Average-mAp) compared to the baseline 49.7% in Giancola et al. [19], we also find that at a relatively low tolerance of 5, our approach reports a mean-Average-precision of 0.38 compared to the baseline of less than 0.2. We further test the robustness of our approach on our dataset from Swedish Allsvenskan and Norwegian Eliteserien and find that the model generalizes well. To better understand our model, we analyze model behavior in a variety of settings such as densely sampled local predictions around events and through the use of Class Activation Maps [59]. We find that based on our results, our model performs well for the task of action recognition. Ultimately, the model suffers from false positives when used for event detection with a sliding-window approach. The analysis of our model grants us useful insight for further work, such as what the model reacts to, and what it may need in the future.

Acknowledgements

First and foremost I want to thank my supervisors Pål, Michael and Håkon for all the help. I also want to thank my family for the support, and all the others who helped, Hanna, Steven, Vajira, Philippe and Johan. Special thanks to all the guys from L3 as well.

The research presented in this thesis has benefited from the Experimental Infrastructure for Exploration of Exascale Computing (eX3), which is financially supported by the Research Council of Norway under contract 270053.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Scope and Limitations . . . 2
- 1.4 Research Method . . . 3
- 1.5 Main Contributions . . . 3
- 1.6 Outline . . . 4

2 Background **6**
- 2.1 Introduction . . . 6
- 2.2 Event definition . . . 6
- 2.3 Action Recognition . . . 6
- 2.4 Action Detection . . . 7
- 2.5 Machine learning . . . 7
  - 2.5.1 Supervised learning . . . 7
  - 2.5.2 Classification . . . 7
  - 2.5.3 Regression . . . 7
  - 2.5.4 Dataset . . . 7
  - 2.5.5 Neural Networks . . . 9
  - 2.5.6 2D Convolutional Neural Networks . . . 11
  - 2.5.7 Transfer learning . . . 12
- 2.6 Related works . . . 12
- 2.7 Summary . . . 16

3 Methodology **18**
- 3.1 Introduction . . . 18
- 3.2 Dataset description . . . 18
- 3.3 Event detection . . . 22
- 3.4 Data preprocessing . . . 22
- 3.5 Overview of models . . . 24
  - 3.5.1 Model architectures . . . 24
  - 3.5.2 Models . . . 26
  - 3.5.3 Implementation . . . 27
- 3.6 Evaluation of Models: Definition of metrics . . . 28
- 3.7 Model selection . . . 30
  - 3.7.1 Hyperparameters . . . 30
  - 3.7.2 Kinetics-400 pretrained models . . . 31
  - 3.7.3 Effect of different temporal inputs . . . 33

3.7.4 Reduced input . . . 37
3.7.5 SlowFast results . . . 38
3.7.6 Input resolution . . . 40
3.8 Summary . . . 42

4 Results **43**
4.1 Introduction . . . 43
4.2 Datasets . . . 43
4.3 Behaviour of model R3D . . . 44
4.3.1 Sliding window . . . 44
4.3.2 How many frames are needed? . . . 45
4.3.3 Class activation tubes . . . 50
4.4 Full video detection . . . 68
4.4.1 Processing output prediction . . . 68
4.4.2 Spotting results . . . 69
4.4.3 Comparison to baseline . . . 73
4.5 Generalization to other datasets . . . 73
4.5.1 Goals . . . 74
4.5.2 Goal attempts . . . 76
4.6 Discussion . . . 78
4.6.1 Corrupt samples . . . 79
4.6.2 Known bugs . . . 79
4.6.3 Just published work . . . 79
4.6.4 Scope of the work . . . 79
4.6.5 Further improvements . . . 79
4.7 Summary . . . 80

5 Conclusion **82**
5.1 Summary and contributions . . . 82
5.2 Future work . . . 83

A Appendix **89**

List of Figures

2.1 Illustration of three different functions (red line) used to fit the training set. The left image illustrates overfitting, the middle shows the underlying function, and the right image illustrates underfitting. 8
2.2 An example of gradient descent used to find the local minimum. On the left, we see an example of a linear regression line fit during each iteration; on the right, we see the loss for corresponding iterations of gradient descent. 9
2.3 Illustration of Neural Network 10
2.4 Illustration of convolution operation in CNN 11
2.5 Illustration of RGB channels in CNN 12
2.6 Number of clips available in recent action recognition datasets and their year of release. 16

3.1 A timeline of a sample soccer game half with annotated events . 20
3.2 Distribution of the distances between samples within the same class in seconds. **Events distances** shows distance distribution for distances between any events. Number of bins = 200. They are calculated from the training set. 21
3.3 Illustrates region in which a correct prediction would be considered as a true positive for $\delta = 15s$ 22
3.4 Sample frames visualized for a sample of 128 frames. The middle frame is at the annotated time. 23
3.5 llustrates 3D convolution where W = width, H = height, T = frame 24
3.6 Illustrates a residual 3D block 25
3.7 Figure illustrates the basic architectures of the models used [16, 47]. Here, $\bigoplus$ represents elementwise sum, and $\bigotimes$ represents concatenation. 26
3.8 Learning rate schedule used 31
3.9 Accuracy and loss for each model. 32
3.10 Accuracy and loss during training on training set 34
3.11 Accuracy and loss during training on validation set 34
3.12 Accuracy and loss for gray input frames for training and validation set. Models are trained on 16 frames 37
3.13 Figure shows accuracy and loss for SlowFast 2+1D 39
3.14 Learning rate schedule with warm restarts 39
3.15 Accuracy and loss for training and validation set when using warm restarts 39

3.16 Accuracy and cross entropy loss during training 41

4.1 Illustration of sliding-window approach 44
4.2 Softmax output for the event *Card* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction where 45
4.3 Softmax output for the event *Substitution* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction where 46
4.4 Softmax output for the event *Goal* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction for the 128 frames of the event. 46
4.5 Softmax output for the event *Background* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction for the 128 frames of the event. 47
4.6 Densely sampled output prediction over 138.4 seconds around event '*Card*'. Red dotted line represents the temporal anchor of the event. 48
4.7 Densely sampled output prediction over 138.4 seconds around event '*Substitution*'. Red dotted line represents the temporal anchor of the event. 49
4.8 Densely sampled output prediction over 138.4 seconds around event '*Goal*'. Red dotted line represents the temporal anchor of the event. 49
4.9 Densely sampled output prediction over 138.4 seconds around event '*Background*'. Red dotted line represents the temporal anchor of the event. 50
4.10 Illustration of feature volumes V_i 52
4.11 Illustration of calculation of class activation tubes 52
4.12 Confusion matrix for validation results using R3D with 128 frames 53
4.13 a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{card}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$ 54
4.14 Results from CAT's spatio-temporally and temporally for the event '*Card*'. a) Center input frame between interval [s*n,28+s*n] of 512 frame sample of event for the n'th sample. b) Corresponding center CAM $T_{card}(8, x, y)$ c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal for zero-padded 512 frame inputs at 64 frame stride. 55
4.15 a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{substitution}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$ 56

4.16 Results from CAT's spatio-temporally and temporally for event '*Substitution*'. a) Center input frame between interval [s*n,128+s*n] of 512 frame sample for the n'th sample. b) Corresponding center CAM $T_{substitution}(8, x, y)$ c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal for zero-padded 512 frame inputs at 64 frame stride.. 58
4.17 a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{goal}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$ 59
4.18 Results from CAT's spatio-temporally and temporally for the event '*Goal*'. a) Center input frame between interval [s*n,128+s*n] of 512 frame sample for the n'th sample. b) Corresponding center CAM $T_{goal}(8, x, y)$. c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal signal calculated with 512 frame inputs at 64 frame stride. 60
4.19 a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{background}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$ 61
4.20 Results from CAT's spatio-temporally and temporally for the event '*Background*. a) Center input frame between interval [s*n,128+s*n] of 512 frame sample of event '*Background*' for the n'th sample. b) Corresponding center class activation maps $T_{background}(8, x, y)$ c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal signal calculated with 512 frame inputs at 64 frame stride. 62
4.21 Model mistakes event '*Card*' as event '*Background*', figure shows samples from input clip alongside CAMs for event '*Card*'. 63
4.22 Model mistakes event '*Substitution*' as event '*Background*', figure shows samples from input clip alongside features for event '*Substitution*'. 65
4.23 Model mistakes event '*Goal*' as event '*Background*', figure shows samples from input clip alongside features for event '*Goal*'. . . . 65
4.24 Model mistakes event '*Background*' as event '*Card*', figure shows samples from input clip alongside features for event '*Card*'. . . . 66
4.25 Model mistakes event '*Background*' as event '*Substitution*', figure shows samples from input clip alongside features for event '*Substitution*'. 67
4.26 Model mistakes event '*Background*' as event '*Goal*', figure shows samples from input clip alongside features for event '*Goal*'. . . . 67
4.27 Illustration of moving average filter 68
4.28 Softmax confidence for each class over 45-minutes with ground truth. 70
4.29 Prediction for each second after mean filtering, after thresholding. Red dotted line shows ground truth for each class. 70
4.30 Recall, precision and F1-score of validation set over different thresholds using a tolerance $\delta = 2.5s$ 71
4.31 Recall, precision and F1-score of test set over different thresholds using a tolerance $\delta = 2.5s$ 71

4.32 mAp over different tolerances 73
4.33 Goal prediction for randomly sampled clips from Allsvenskan . . 74
4.34 Goal prediction for randomly sampled clips from Eliteserien . . . 74
4.35 Avg. Prediction from Norwegian Eliteserien clips and Swedish Allsvenskan 75
4.36 Results from Allsvenskan dataset containing goals for both full clip and limited interval 75
4.37 Results from Eliteserien dataset containing goals for both full clip and limited interval 76
4.38 Randomly sampled from Allsvenskan 'goal attempts' clips 77
4.39 Avg. prediction from 'goal attempts' clips 77
4.40 Results from Allsvenskan dataset containing goal attempts for both full clip and limited interval 78

List of Tables

3.1 The number of full soccer games with respect to season and league. Reprinted from Giancola et al [19]. 19
3.2 General information about a compressed version offered by the authors. Adapted from Giancola et al [19]. 19
3.3 The number of samples for classes for each split. 20
3.4 Minimum, maximum, and mean distance in seconds between events for each class from the training set. 'All' contain distances regardless of class. Calculated from training set. 21
3.5 Percentage of samples for each class, which has a distance of less than a threshold T seconds between samples in its own class training set. 'All' denote distances between any events regardless of class. 22
3.6 N number of samples for classes. 23
3.7 Parameters for the different models 26
3.8 Per-class precision, recall and F1-score. Higher is better, best result per class in bold. Both models are trained using 16 frame inputs. 32
3.9 Weighted and unweighted precision, recall and F1-score. Higher is better, best result per metric in bold. Both models are trained using 16 frame inputs. 33
3.10 Accuracy, precision, recall and F1-score per class for R (2+1)D pretrained on Kinetics-400 with different number of consecutive frames as input 35
3.11 Accuracy, precision, recall and F1-score per class for R3D pretrained on Kinetics-400 with different number of consecutive frames as input 35
3.12 Accuracy, precision, recall and F1-score per class for MC3 pretrained on Kinetics-400 with different number of consecutive frames as input 36
3.13 Validation accuracy as well as weighted and unweighted average precision, recall and F1-score. Higher is better, best result per class in bold. 36
3.14 Results for models trained on 64 and 128 frames when tested with 128 and 64 frames respectively. 37
3.15 Results per class for models trained on gray-scale inputs 38
3.16 Results average over all for models trained on gray-scale inputs . 38
3.17 Per class metrics for SlowFast 2+1D trained using two different learning rate schedules 40

3.18 Average scores weighted and unweighted for SlowFast 2+1D trained with two different learning rate schedules. 40
3.19 Comparison of results for 112 vs 224 resolution 41
3.20 Average results weighted and unweighted for 112 vs 224 resolution 42

4.1 Dataset statistics for Allsvenskan and Eliteserien clips 44
4.4 Table with results on test set for mAp, AP, Precision, Recall, F1-score and number of predictions made at each threshold . . . 72
4.2 Average-precision and mAp for classes on validation set 72
4.3 Average-precision and mAp for classes on test set 72
4.5 Results for Allsvenskan and Eliteserien at different thresholds . . 78

Chapter 1

Introduction

1.1 Motivation

Video as a medium has been popular ever since the TV was introduced. With more today's access to laptops, smartphones, and high-bandwidth internet at all times, it is easier than ever to watch a video. This is further empathized with online providers of video content such as Netflix and Youtube. Every day, more than one billion hours of video are viewed on Youtube. Furthermore, more than 70% of this is viewed on a mobile device [27]. In this context, sports have been shown on TV since the 1936 Olympics, and in 2018, more than 3.5 billion people were estimated to have watched some part of the FIFA 2018 World [17]. Today, there are companies whose business model is sports analytics, and there are several sites and providers of game content across leagues.

Automatic event detection can increase the availability of events for user consumption. Furthermore, statistics from events can be used in broadcasts for entertainment value, betting companies, fans, or the teams themselves. Annotations for exciting events is useful for those reading text summaries of sporting events and can be used to create statistics, or as reference for later extraction of highlights. Data that contain information on video files has thus become valuable. As the world's video content keeps increasing, it is not feasible to manually analyze all videos. Therefore, an automatic method to extract annotations from the video can be valuable, especially to lower leagues or sports where funds are limited.

Previously, SoccerNet [19], for example, has tried using I3D [8],C3D [48], and ResNet [24] pretrained on ImageNet [11] as fixed feature extractors, followed by dimensionality reduction with PCA. These features are sampled every 0.5s throughout the soccer videos. Afterward, based on these features, the authors use convolutional layers to capture temporal information, followed by pooling layers and a fully connected layer. This process is done locally around an annotated event where the final trained model predicts events on full videos by a sliding-window approach and post-processing of the predictions. The authors test this approach by using temporal windows ranging from 5-60 seconds, where 5-second windows performed best given strict requirements for distance between predicted event and event, while 20-second windows performed best overall.

However, there are several other machine-learning-based approached suggested both earlier and after that potentially provide better results. Therefore, an open question is: 'How well do state-of-the-art deep-learning based models perform for automatic detection of events in soccer videos?'

1.2 Problem Statement

Today, sports events are manually annotated. This thesis aims to answer the following question:

'How can we automatically detect events in untrimmed soccer videos?'.

With recent advances in deep-learning-based action recognition and detection methods, it may be possible to create a general data-driven model that generalizes to multiple sports and events therein. We explore how well state-of-the-art deep-learning models perform in soccer event detection. Our goal is to automatically annotate soccer events in the video as close to the actual event in a scalable manner. To achieve this in a way where the solution can be scaled up if needed, we search for a solution that makes predictions based on directly meaningful information. One approach to evaluate automatic event detection is through the task of spotting. With spotting, evaluation is measured with strict rules for distances between actual events and predicted events. We use multiple datasets for evaluation and provide a baseline comparison from SoccerNet [19]. Our objectives are as follows:

Objective 1 Research and find a general approach to automatic event detection in soccer videos that can be easily adapted to new events.

Objective 2 Implement and find a good configuration for the given approaches through experimental prototyping.

Objective 3 Analyze weaknesses and strengths of the selected approach.

Objective 4 Compare experimental results to state-of-the-art.

1.3 Scope and Limitations

Modern event detection has multiple challenges, such as action recognition where short video clips are classified, action detection on untrimmed videos where the goal is to find a temporal location and event in time, and spatio-temporal action detection that aims to find location and event both spatially and temporally. There are limited publicly available datasets for soccer, our thesis focus on the SoccerNet dataset [19] where temporal anchors for three different events are annotated over 784 hours of video. Additionally, we use soccer videos from the Norwegian Eliteserien and Swedish Allsvenskan leagues. Given our datasets, we are limited to action recognition and detection. The datasets available contain the events '*Goal*', which is defined as the moment the ball crosses the goal line, '*Goal attempt*', defined as the moment a player attempts a goal, '*Card*', which is the moment the referee shows a yellow or red card to a player, and

'*Substitution*', the moment a new player enters the field. Thus, we are limited to these events. There are multiple different sources of information that can be used for event detection. Audio, video, and commentaries are some examples. Audio and commentaries may differ significantly from league to league and country to country. A short video clip of an event should be enough information for a human annotator to correctly describe the event regardless of language or league. Therefore, we limit our scope to use video and not audio or commentaries. There are different approaches to event detection through the use of traditional computer vision methods. However, we limit the scope to deep-learning-based methods.

1.4 Research Method

The overall goal for this thesis is to build and test a system for automatic event detection in soccer videos. To do this, we have based our research method upon the Association for Computing Machinery (ACM)'s methodology. In 1989 ACM's Education Board released the report '*Computing as a Discipline*' [12]. In the report, three paradigms described.

The theory paradigm consist of four different steps as follows. Characterize objects of study (definition), hypothesize possible relationships among them (theorem), determine whether relationships are true (proof), interpret results. Rooted in the experimental scientific model is the paradigm of abstraction. Form a hypothesis, construct a model and make a prediction, design an experiment, analyze results. Related to engineering is the paradigm of design. Design consist of the following steps. State requirements, state specifications, design and implement the system, test the system. For all three paradigms there is an expectation of an iterative process. In theory there is an iterative process due to errors that arise, in abstraction, the prediction might not agree with the results. In the design paradigm, the system test might reveal lacking results with regards to requirements.

This thesis mostly applies the design paradigm for the general problem of event detection. We have a specific task that requires a certain level of accuracy and computational efficiency to be feasible. We further design, implement and test with experimental prototyping, and test the final system to determine the practical use for event detection in soccer videos.

1.5 Main Contributions

In this thesis, we address the question of event detection, as described in Section 1.2. For objective 1, we analyze state-of-the-art approaches to action recognition and detection and find deep-learning-based methods promising. Based on state-of-the-art results in action recognition we select the three architectures ResNet 3D-18, ResNet (2+1)D-18, Mixed Convolution-18 [47]. Additionally, we implement a model based on the SlowFast architecture [16], which has previously shown state-of-the-art performance in both action recognition and spatio-temporal action detection. In the context of objective 2, we test different configurations for the selected model in the task of action recognition of video-clips. We use the SoccerNet [19] dataset during training, which contains

about 784 hours of untrimmed soccer games at 25 frames per second, which is about 70560000 individual frames. The dataset contains 6637 annotated events: '*Card*', '*Substitution*' and '*Goal*'. Additionally, we download 617 clips from Norwegian Eliteserien and Swedish Allsvenskan with a length of 60-90 seconds each, where 533 of the clips contain goals, and 84 contain goal attempts. As an action recognition task on SoccerNet, we show that the use of pretrained models can significantly boost generalization, with the best performing pretrained model reporting 83% compared to its from-scratch counterpart 72.8% with the same configuration. We further test the effect of using reduced input in the form of gray-scale videos and show that color may not be an essential aspect of action recognition with a 0.6% drop in validation accuracy for the best performing model. We experimentally show that the use of a greater time-window as input can provide a boost in generalization for classification of soccer events in video with a 5-7% increase in validation accuracy. There is a high cost to system memory and computation with regards to video resolution. We, therefore, experiment with different frame sizes and show that resolution may not be critical to our approach. In the context of objective 3, we try to understand our model behavior in a practical case. We use a sliding-window approach locally around annotated events with zero-padded frames. This provides useful insight into how much of the actual event our model needs, thereby granting us a basis for the choice of stride for full videos. We repeated the experiment for larger temporal windows of 138.6 seconds to determine noise in model predictions, as well as reactions to replays. These replays are not annotated and therefore resulted in false positives. To understand what our model reacts in terms of spatio-temporal features, we calculated and analyzed class activation maps and class activation signals, which contains information on which parts of the input video-clip the model reacts to. This approach was applied for both correctly classified and misclassified samples. For objective 4, we test our sliding window approach on 200 full games, providing a comparison to the baseline of 49.7% Average-mAp in SoccerNet [19] with 51% Average-mAp in our approach on the test set. Additionally, we report a 0.38 mAp at a tolerance of 5, compared to a baseline that, based on visual inspection, had less than 0.2 mAp. With a threshold of 0.5, we still keep a recall of 0.88, while having reduced the number of predictions from 550000 to 5711 in the case of the event '*Goal*'. Based on the results, we believe that the model works well in the case of classification, and it mostly reacts to events as intended. However, there is a challenge with false positives, believed to be a result of similar scenes and replays.

1.6 Outline

Chapter 2 Background In Chapter 2, we describe essential machine learning concepts and terminology. We discuss current research into action recognition and detection and how deep-learning has impacted the field in the last decade.

Chapter 3 Methodology In Chapter 3, we describe the dataset and the associated task. We further discuss data preprocessing steps before training. A description of different candidate models and hyperparameters are given before we continue with experimental prototyping to decide the best model configuration for our case.

Chapter 4 Results In Chapter 4, we first describe the datasets used for the final evaluation. We describe the behavior of our model with multiple approaches that grant us insight into why and when our model may fail or succeed. The overall objective event detection is evaluated on multiple datasets, and the results are analyzed and discussed.

Chapter 5 Conclusion In chapter 5, we provide a summary of this thesis and its contributions. We discuss future work in relation to both our approach and in general.

Chapter 2

Background

2.1 Introduction

We want to find a system that can automatically detect events in soccer. Our focus is on the task of spotting, where the goal is to predict a class and a single point in time. This chapter is structured as followed. First, important terminology and methods for used in the thesis are defined. These include terms such as *deep learning*,*Convolutional Neural Network (CNN)*, *Action Recognition* and *Temporal Action Detection*.

Next, **related works** explores the state of the field with respect to current and past challenges, improvements in models, and datasets over the years. Lastly, specific challenges towards soccer-specific action detection and the most relevant datasets and models are discussed for the task at hand.

2.2 Event definition

What exactly is an event[1]? The oxford dictionary defines an event as:'*a thing that happens, especially something important*'. For humans, casually describing when and where a given event occurs comes naturally. Quantifying the exact start, ending, or length of an event, however, can be difficult. Sigurdsson et al. [42] explore the ambiguity in the temporal extent of human activities. In an experiment where humans were tasked with annotation of the start and end of events, they found that most of the time, the center was agreed upon, while the starting point and ending varied. For this thesis, we define events to be instantaneous. This is consistent with the spotting.

2.3 Action Recognition

Action recognition is a classification task in which the goal is to determine what action is present given some clip. This is also sometimes referred to as Activity Recognition. The input is commonly in the form of images or video. Typically,

[1]Events and actions are closely related in literature, and in the context of our problem, most events can be seen as a product of human actions. Therefore, we interchangeably use the terms events and actions throughout this thesis.

the video input will be *temporally trimmed video*, meaning that a small subset (often 3-10 seconds) of the original video. The task is, therefore, to classify a video, with usually only the target action in mind. Some datasets, however, feature multiple classes in the same temporally trimmed video. Classifying actions come with many challenges, e.g., camera motion, different view-points, occlusions, different quality of video in general. Another challenge comes with high computational and storage costs.

2.4 Action Detection

Action detection or event detection is the more general task of finding both when and where an event occurs. This means that any approach needs to find a temporal interval in addition to the classification task in action recognition.

2.5 Machine learning

Machine Learning (ML) is a broad term for data-driven learning algorithms [21]. ML is often seperated into the subcategories **Supervised Learning** and **Unsupervised Learning**.

2.5.1 Supervised learning

Supervised learning is a class of methods that is identified by the components used in the learning process. The components are an input dataset $X = \{x_1, x_2, ..., x_n\}$, a corresponding set of outputs $Y = \{y_1, y_2, ..., y_n\}$. The goal of a typical supervised learning algorithm is to associate the input X and Y such that, given an input x, it correctly produces the corresponding value y.

2.5.2 Classification

Classification is the process of assigning a class to a given input. This is often done in a supervised setting. An example of this can be, given an image, is there a cat present - yes or no. This is a binary classifier, since only two classes are present, where the most appropriate output would be either 0 or 1. Where 0 would refer to 'Not cat' and 1 for 'Cat'. For more classes, the output would typically be a vector $y = [c_1, c_2, ..., c_n]$ where the c is 1 if the class is detected, and 0 otherwise.

2.5.3 Regression

Regression is a similar problem, also associated with supervised learning. The difference here mostly lies in the output, and instead of predicting a class, the output is typically a real number. An example of this would be to predict the current salary of a worker based on the number of years of experience.

2.5.4 Dataset

In the context of supervised learning, a given task consist of a set of inputs X, and corresponding outputs Y. In a classification setting, Y is obtained

through an annotation process. In many deep-learning problems, this process is done manually by a human annotator, which for example, looks at an image, and labels the image as some class c (e.g., a cat or dog). These datasets are ultimately split into separate pieces, which serve different purposes, as discussed below. Datasets are ultimately split up into three independent sets, namely the *training set*, *validation set* and *test set*.

Overfitting Overfitting is a critical problem to be aware of in ML. This refers to a model that performs well on the *Training set* but ends up generalizing poorly to new data. Indeed, the goal is to create a model that works well on new data - and not one that performs well on already labeled data. This is illustrated in Figure 2.1, where we see three different solutions to the same data.

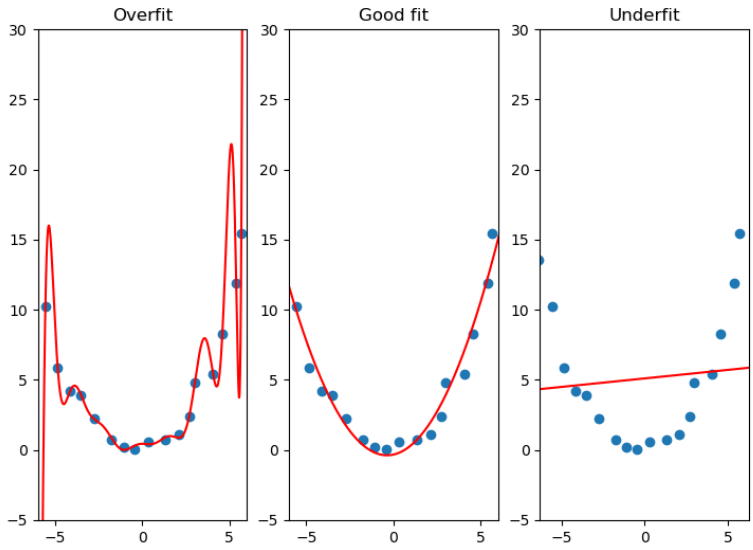


Figure 2.1: Illustration of three different functions (red line) used to fit the training set. The left image illustrates overfitting, the middle shows the underlying function, and the right image illustrates underfitting.

Training set is a subset of X and the corresponding Y. This is most often the majority of the total dataset, and serves the purpose of *training* the model.

Validation set is another independent subset that is used in training. This is usually used to tune hyperparameters of a model or to evaluate generalization and overfitting during training.

Test set is the most crucial set when evaluating the model. The test set should not be used in any way during training and serves the purpose of evaluating the model on unseen data.

Gradient descent Gradient descent is a popular algorithm used to optimize a model by finding the correct weights through an iterative process. Suppose we have model $F(X; w)$, where w are the learnable weights of the function. To improve our model, we need to find the correct weights w. To do this, we use a *cost function* $J(w; X)$.

Formally, gradient descent can be described as follows:

$$w_{t+1} = w_t - \mu \nabla_w J(w_t) \tag{2.1}$$

where μ is our learning rate, gradient descent finds a local minimum of a given function by the iterative process of 'moving' in the direction of a minimum based on the gradient. The learning rate is a critical hyperparameter to choose. If the learning rate is too low, it will update slowly, too large, and we will not converge to the minimum. Figure 2.2 shows a simple example of how we can fit a line by using gradient descent.

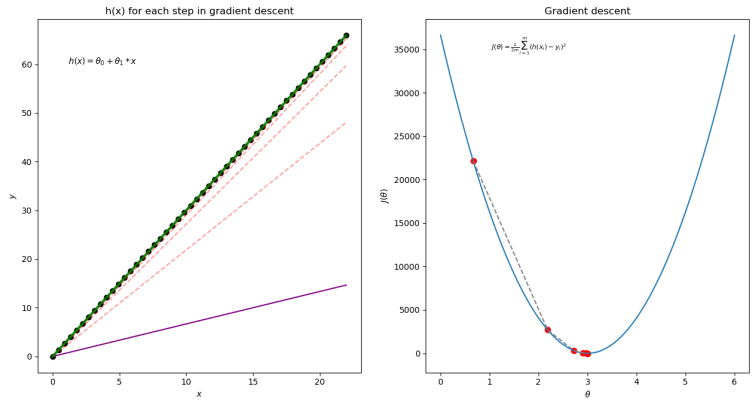


Figure 2.2: An example of gradient descent used to find the local minimum. On the left, we see an example of a linear regression line fit during each iteration; on the right, we see the loss for corresponding iterations of gradient descent.

2.5.5 Neural Networks

Neural Networks (NN) can be described a mapping from input space, that can be in the form of images, audio, text or other data, and maps it to output space, that can be in the form of continuous or discrete values depending on the task. We can consider NN as a function $F(X) = Y$, where X is our input, Y is our output.

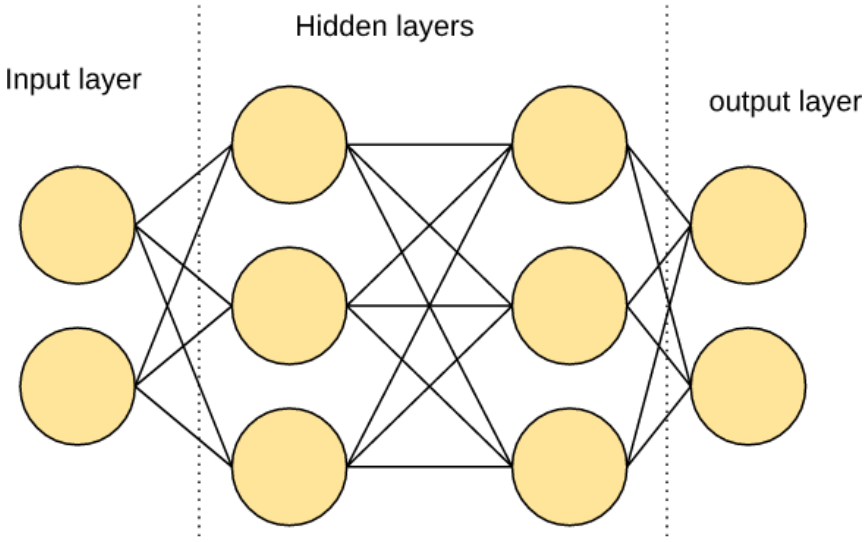


Figure 2.3: Illustration of Neural Network

Figure 2.3 illustrates the structure of a neural network. The lines illustrate the learnable weights, which produces a weighted sum. A function is added for the purpose of non-linearity at each node. We call this an activation function. The nodes illustrate the activation value at that point. Formally, this can be described as follows.

$$z_k^l = \sum_{j=1}^{n^{l-1}} w_{jk}^l a_j^{l-1} + b_k^l \tag{2.2}$$

Where z_k^l is the weighted sum at the k'th node in the l'th layer, prior to activation. w are the weights connected to the given node, here b is the bias, another learnable weight.

The activation value at a given point can be formally defined as:

$$a_k^l = g(z_k^l) \tag{2.3}$$

Where g is some non-linear function. A common activation function, which is later used in this thesis is called ReLu.

$$ReLu(x) = max(0, x) \tag{2.4}$$

ReLu serves the purpose of adding non-linearity. The removal of the activation function will reduce the output to a linear function. While there are many ways to adjust and change neural networks, architecturally, the number of layers, and the number of activation nodes per layer is one of the most important parameters to choose from. During the training of a neural network, we need 1) a loss function 2) optimization method. The loss function used in neural networks varies depending on the task. We can consider a loss function $L(\hat{y}, y)$ where $\hat{y}$ is our output prediction and y is our target. We perform a forward-pass, meaning that we use training data to produce our output prediction $\hat{y}$, and

calculate our loss based on that. For optimization, a common method is gradient descent, as described earlier. The chain rule of calculus is used to calculate the gradient of the loss function with respect to the weights, after which the weights are updated accordingly. This process is called back-propagation.

2.5.6 2D Convolutional Neural Networks

Convolutional Neural Network (CNN) is one of the most popular models in machine learning today. One of the earliest uses of CNNs were by Lecun et al. [35], and was later popularized by AlexNet [33]. CNNs use parameter sharing.

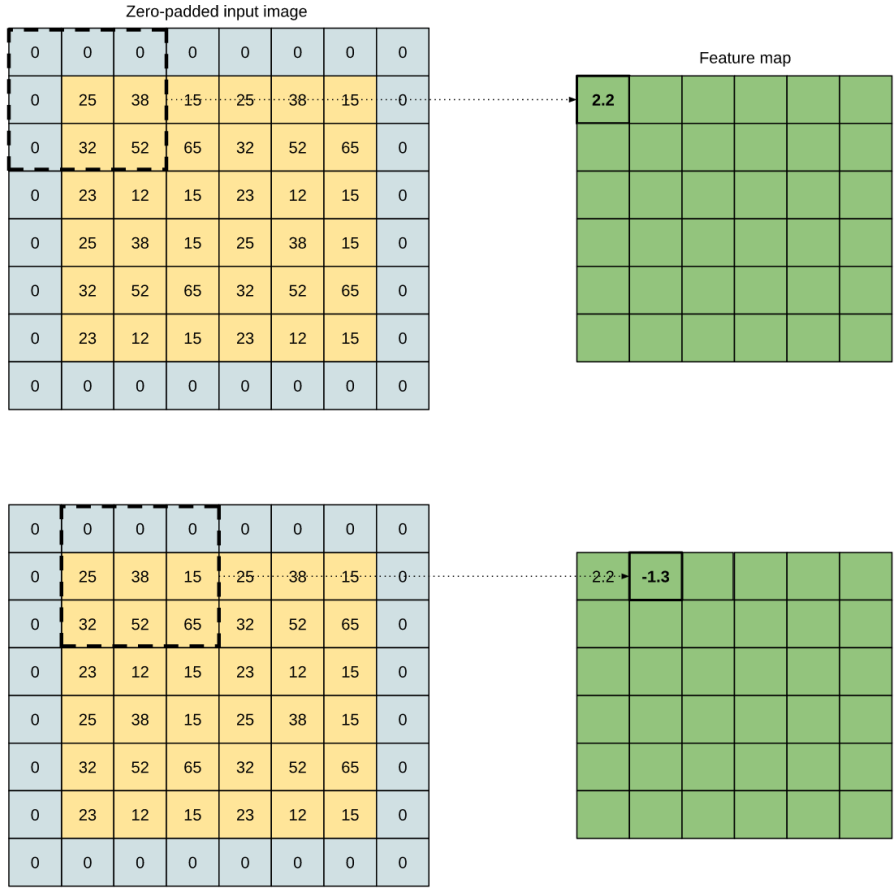


Figure 2.4: Illustration of convolution operation in CNN

Figure 2.4 illustrates how a 3x3 filter is used during convolution over a zero-padded image. Each point in the feature map is a weighted sum of a local region in the input image. This means that the input share weights, effectively finding important information such as edges within the image. Zero-padding is often used as a way to have better control over the feature map dimensions. In practice, this process is repeated in each layer with multiple filters, producing several feature maps.

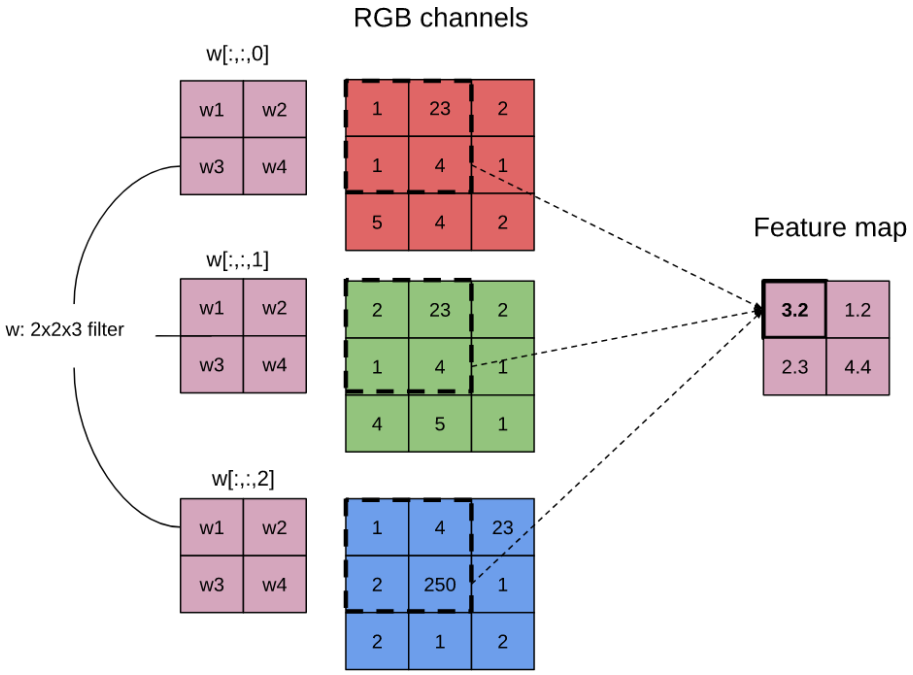


Figure 2.5: Illustration of RGB channels in CNN

Figure 2.5 illustrates how CNNs compute feature maps from RGB color images. For RGB images, we have three channels. After a convolutional layer with N filters, we say that there are N feature maps, meaning that for the next layer, we need appropriate filters.

2.5.7 Transfer learning

In the context of neural networks is the process of re-using weights for a different purpose. It has been shown that using a pretrained CNN on a different dataset, can be re-purposed by using it as a feature extractor, followed by some model to classify [1]. One approach to do this is to train a CNN on one dataset, then *freeze* (meaning that the parameters in these layers are not updated during training) n consecutive layers and retrain the last layers with the new appropriate output. Thereby, effectively using part of the original model as a feature extractor as input to new layers to be trained. Another way is to *finetune* a new model, which uses the same process but without freezing layers during training, thus optimizing all parameters according to the new dataset. One study showed that finetuning a model that had been pretrained on a similar dataset boosted generalization performence [58]. Pre-training has been successful also in action recognition [8, 43, 47, 4]

2.6 Related works

Action recognition as a general task has gotten more attention in the last decade. Datasets such as UCF101 and HMDB-51 [34, 45] have been an important

addition to the field. Handcrafted features such as Histogram of Gradients (HOG), Histogram of Flow (HOF), Motion Boundary Histogram (MBH) [10], dense trajectories [51, 50] showed promising results.

In 2014, Karpathy et al. [30] explored the use of CNNs. In order to cope with the computational cost, the authors implemented two 'streams'. The fovea stream takes cropped part in the center as input, while the context stream used a downsampled version of the original input. The authors further showed that the features were general and benefitted from transfer learning.

Simonyan et al. [43] introduced a new architecture Two-Stream CNNs. The authors mention that this architecture is related to the *Two-Stream hypothesis* [20], which is an idea of humans possessing two distinct visual systems. The ventral stream, which is responsible for spatial details, object recognition — the dorsal stream, responsible for temporal details, motion. The authors propose the use of two separate CNNs, the spatial CNN [33], pretrained on ImageNet, which takes RGB input by sampling frames from video. Additionally, the temporal stream, which uses optical flow fields as input. The Two-Stream structure is further built upon in Carreira et al. [8], which adds 3D convolution, Feichtenhofer et al. [15] explores the fusion process with 3D convolution and 3D pooling, the authors also note that datasets are too small at the time (2016) [3], which adds a neural network called MotionNet, which estimates the optical flow input for a more efficient pipeline. The Two-Stream architecture is also extended with ST-ResNet [14], which adds residual connections [24] in both streams and from the motion stream to the spatial stream. Research into a combination of hand-crafted features and deep-learned features has also shown promising results [52].

Wang et al. [41, 53] proposed Temporal Segment Networks (TSN) in 2016. Arguing that existing models mostly focused on short-term motion, rather than long-range temporal structures. The authors noted that consecutive frames may be redundant and that it might be unnecessary to sample frames densely. TSN takes a video input, separates it into multiple snippets, and for each snippet, make a prediction using Two-stream networks. The final prediction is then consensus-based on the entire video. C3D [48] explored 3D convolution. By using 3D convolution, spatio-temporal features are learned. When comparing existing 2D convolution solutions, it indeed seems that 3D convolution help in that it adds temporal information such as motion.

When the popular kinetics-400 [31] was released, an exciting baseline model was also introduced. This model is the Two-Stream Inflated 3D ConvNet (I3D) [8], using 3D convolution, along with the popular inception architecture [46]. I3D works much like the two-stream networks, using two networks, one for RGB stream input, and one for optical flow. The reason the model was called inflated had to do with a distinctive pre-training trick that they used. For the spatial net, they use the inception architecture, and first pre-trains a 2D CNN on ImageNet, next the *inflate* the filters such that they can now be used for 3D convolution. They show that this approach achieves better results on the Kinetics-400 dataset when compared to training only on Kinetics-400. In other words, it is a smart way of using transfer learning to initialize the 3D filters smartly.

While many modern action recognition techniques rely on either RGB or optical flow as input, PoTion, introduces an interesting idea. First, they use a pose detection algorithm [7], which is a separate challenge in the field. A pose

detection algorithm will find spatial locations frame by frame for joints and key parts of the human body, e.g., head, elbow, hand, etc. Essentially, finding a skeleton representation of any humans in the image. Next, they use this as features, along with RGB input. The authors note that they could use a much smaller CNN this way. In the paper, they also report that often, failure of the model arises when poses are not considered valuable information, e.g., a Point of view video of someone cooking. This is an excellent idea intuitively since it extracts semantically meaningful information. A downside of this, however, is the reliance on good results from the pose detection model.

In Tran et al. [47] Res (2+1)D was introduced, achieving state-of-the-art results on UCF101 and Kinetics-400. The model is based on (2+1)D convolutions, which are used to separate 3D convolution into two steps, first a 3D convolution with a 1 x S x S kernel where S is the spatial size of the kernel, followed by a T x 1 x 1 kernel, where T denotes the temporal extent. The idea is that is may be easier for the network to learn spatial and temporal features separately.

Feichtenhofer et al. [16] introduced the SlowFast architecture that showed state-of-the-art performance in both action recognition and spatio-temporal action detection. The model is based on the use of two different framerates as input, where the idea is to have a high-capacity '*slow*' pathway that subsamples the input heavily, and a '*fast*' pathway that has less capacity but significantly higher framerate.

The THUMOS[28] challenge saw multiple approaches, with a focus on a sliding-window approach with iDT features and SVM classifiers. In Shou et al. [41], the authors used a object-detection inspired approach from R-CNN [40]. The authors devised a multi-stage approach based on CNNs where first, a proposal network generated candidate spots through a sliding window approach, next a classification network classified the event, finally a localization network was used to find the location within the given frames. Xiong et al. [55] used temporal segment networks for proposals and classification. Xu et al. [56] created an end-to-end proposal classification architecture R-C3D, which significantly improved detection speed. Another approach is through the use of Recurrent Neural Networks (RNN). Here, the approach is typically to generate features through a CNN per frame, and feeding the features into the RNN [57, 37].

In the context of soccer, Ekin et al. [13], multiple algorithms are used. First, dominant field color, which is used for field segmentation, next is shot boundary detection, which is done by comparing two frames and looking at the dominant field color statistics. After which, shot classification, that attempts to classify views as long, medium, and short distance views. Using the information from the steps above, a rule-based system with rules such as a recent view of the referee and an out-of-field shot, red/yellow card events are classified. Goal detection had 27 correct classifications, 32 false, and three miss. 16/19 red/yellow card events were identified as 3/3 penalties and 5/8 freekicks.

In Assfalg et al. [6], a Finite State Machine (FSM) was used on a small dataset with highlight clips of 15-90 seconds duration with multiple events, including a penalty, corner, and freekick. The authors used a rule-based loop with a series of 'if conditions'. By having multiple different states, the FSM continuously tests to see if the current events qualify to move on to the next state, ultimately resulting in a positive prediction. The results showed 17/17 penalties found, 18/19 corners, and 17/18 free kicks.

In Tsagkatakis et al. [49], they gathered 400 2-3 long clips from Youtube

with the classes 'Goal' and 'No goal'. The authors subsampled the clips to 20 frames, then used a pretrained VGG-16 [44] as a fixed feature extractor, with the optical flow in a two-stream [43] architecture, after which they trained an SVM classifier. The authors reported a 98% accuracy.

Giancola et al. [19] used C3D, I3D, and ResNet as fixed feature extractors in SoccerNet with a sliding-window approach at 0.5s stride. The authors reduced the dimensionality with PCA and trained neural networks with a variety of pooling layers for a 60-second multi-label classification task. They apply the best performing model in a sliding-window fashion for the task of spotting, in which a predicted event at time t, is deemed a correct prediction given that it is within some tolerance δ of the ground truth event.

Datasets

Deep learning-based models scale very well with large datasets. Ever since AlexNet [33], CNNs have shown increasingly good results when coupled with large datasets for training. In the domain of action recognition and detection, the great advancement in dataset quality and size is significant. Both for training and for testing. Some early datasets for action recognition [45, 34] made big contributions to the field. Results on UCF101 are now at 98% for modern models. These datasets were annotated manually, which is an expensive task. Later, Youtube Sports1-M, Youtube 8-M and moments in time [5, 30, 39] further increased the datasets by millions. In 2017, the popular dataset Kinetics-400 [31] was released, with 400 classes and about 350k samples. Since then, Kinetics-600 and Kinetics-700 have been released as an extension, containing more classes and data, now at about 650k samples. These datasets are general and contain a wide range of actions. It should be noted that there are strengths and weaknesses related to each dataset. Annotating at this scale is not cheap. Therefore the authors used different ways of annotation. Youtube 8-M uses metadata from a youtube video to automatically annotate, while both kinetics and sports 1-m use human annotator systems. It is a common strategy to sample videos from Youtube. This means that if a video is taken down, a sample is lost, which may lead to concerns about the repeatability of test results as such. Kinetics attempts to remove duplicate videos, since Youtube videos in the same categories may be frequently reuploaded by other users, training data may be present in the test data set as well. There is, however, an inherent strength in sampling youtube videos. As noted, early datasets were controlled, but in youtube videos, video quality can be drastically different, which means a model must be robust to poor quality, e.g., shaky camera, poor resolution, different light, and more. In Figure 2.6, we can see that the increase in data has been substantial.

In recent years, action detection has seen meaningful additions to the field. With popular datasets such as THUMOS14 and ActivityNet [29, 26]. Another recent and important dataset is AVA [23]. This notably has spatio-temporal localization as a challenge, which means that the challenge is to detect actions both temporally and spatially in the video. Both Thumos [29] and Multithumos [2]

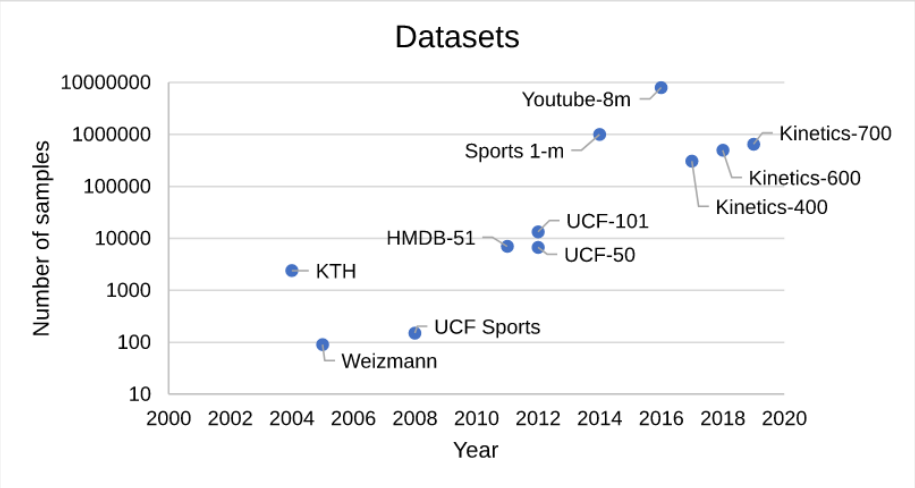


Figure 2.6: Number of clips available in recent action recognition datasets and their year of release.

feature a multilabel problem. Meaning that a given time interval may have multiple actions.

In soccer, a new dataset SoccerNet was released in 2018 [19]. This dataset comes with sparse events across 764 hours, manually annotated down to a resolution of 1 second. There are several challenges present in this dataset. The events are sparse, at 1 per 6.9 minutes on average. Since a soccer match is 90 minutes, any feasible model must be able to analyze a match within a reasonable amount of time. Another challenge is the similarity of events, and it may difficult to separate between an attempted goal and a goal.

2.7 Summary

The annotation of videos is expensive. It is boring and tedious work, and not available unless it is financially feasible. In this chapter, we describe essential machine learning concepts and terminology, such as supervised learning, classification, and regression. Next, we describe how datasets are commonly split into training, validation, and test set and why failing in this section can result in overfitting. We describe gradient descent and discuss the NN and CNN architecture. The problem of automatically annotating video is an active field of research. With increasing amounts of video data, there is a need for an effective and accurate way to annotate. In sports, manually annotating is time-consuming, and not feasible for all leagues. In order to solve this problem, we look into existing methods in deep learning with the goal of finding a method that may have the capability of understanding events based on relevant information where a human would be able to do the same. Therefore, we discuss action recognition and how deep-learning and better datasets have contributed to the field. Dense Improved Trajectories [51] worked as important features in action recognition by capturing spatio-temporal information. Two-stream networks [43] showed promise when combining the spatial capacity of CNNs with

the motion information of optical flow, strongly indicating that both features are important for action recognition. As datasets have gotten better, so have the results, with action recognition showing great promise [16, 47, 8]. In action detection, we found that the popular proposal-classify strategy from object detection is often used. Furthermore, RNNs have shown success when combined with CNNs as feature extractors. In soccer, there are both traditional and new deep-learning-based approaches. However, there has been a lack of datasets, leading to manually generated datasets that make it hard to compare against. In 2018, SoccerNet [19] was introduced, which provides the field with a way to properly compare results for soccer event detection. Soccer provides a case where the appearance is similar for many events, such as goal attempts and goals. Furthermore, events are sparse, with a variety of camera angles used. The baseline was based upon fixed feature extractors. We want to see how state-of-the-art action recognition models can perform when trained directly. Furthermore, we want to find a suitable configuration for a given model. In the next chapter, we propose a selection of action recognition models to configure to best perform in the case of soccer.

Chapter 3

Methodology

3.1 Introduction

Annotating videos is expensive. It is tedious and time-consuming work. With increasingly large amounts of video available, it is not always possible to use people for this task. This leads to either lackluster event annotation in cases where it is not financially lucrative to have accurate annotations or limitations in which events are described. The problem of automatic event detection in soccer has many different approaches. During the last decade, there has been a rise of deep learning-based models which increasingly perform better with the availability of data. In the last chapter, we found that new, larger, and more challenging datasets such as Kinetics-400 [31] greatly help in the development and testing of action recognition and detection. For event detection in sports, there is a challenge in that many events are sparse and difficult to separate. We want to approach the problem of detection in soccer with a data-driven solution that captures can easily be adapted to other events given data. In this chapter, we introduce state-of-the-art models that we want to test for the task of event detection. First, we formally define the task of spotting, which is used for the final evaluation for event detection. We provide a detailed description of the dataset, which we use for training and testing in a separate classification task. Next, we show how transfer-learning-based methods can be useful. We further investigate the effect of different size temporal inputs for 3D CNNs as well as different spatial resolutions. Several different models are tested and evaluated, with different configurations. Finally, based upon our results, we choose a single model for spotting.

3.2 Dataset description

SoccerNet [19] contains 500 games from different professional soccer leagues annotated. It contains untrimmed broadcast videos from each half of the game, meaning that the full dataset has 1000 videos with a length of about 45 minutes each. The dataset contains a video with audio, and 506,137 commentaries at 1-second resolution from online sources. Both audio and commentaries could be used for future work, but this thesis focus on the video frames. The dataset was collected by Giancola et al. [19] by first collecting videos from online sources.

Next, the authors synchronized the videos by detecting the game clock with optical character recognition (OCR). Finally, they get free online annotations with a 1-minute resolution, which they manually annotate down to a 1-second resolution. Each event in this dataset is annotated with a single temporal anchor. This is in contrast to the more traditional approach where a temporal interval is annotated, indicating that the event in question lies within this interval.

Three classes are present in this dataset: '*Card*','*Substitution*' and '*Goal*'. Giancola et al. [19] define the three events as follows.

- '*Card*' is defined as the moment where the referee shows the yellow/red card to the player.
- '*Substitution*' is defined as the moment the new player enters the field.
- '*Goal*' is defined similarly to the official IFAB rules, which is the moment that the ball crosses the goal line.

	Season			
League	14/15	15/16	16/17	Total
EN - EPL	6	49	40	95
ES - LaLiga	18	36	63	117
FR - Ligue 1	1	3	34	38
DE - BundesLiga	8	18	27	53
IT - Serie A	11	9	76	96
EU - Champions	37	45	19	101
Total	81	160	259	500

Table 3.1: The number of full soccer games with respect to season and league. Reprinted from Giancola et al [19].

In Table 3.1, we can see that all samples come from high-end leagues in recent years. Therefore, we can expect professional broadcast video that contains multiple different views, replays, and a variety of standard video production techniques such as game timer with score, slow-motion, and animations. Furthermore, since all leagues are popular, there are likely high-quality fields and an audience present. For lower leagues, video statistics may be different, and player behavior may change as well. For example, when celebrating a goal. Another practical case is personal videos, which may contain poor quality video with rotation and shaking. It is essential to be aware of such bias, since any analysis may not be representative for other cases.

Dataset (224p version)	
Frames per second	25
Resolution (HxW)	224x398
Duration (Hours)	784
N games	500
N training games	300
N validation games	100
N test games	100

Table 3.2: General information about a compressed version offered by the authors. Adapted from Giancola et al [19].

Table 3.2 shows some general metadata. It should be noted that the dataset is both available in high-quality and a compressed version. In some recent activity recognition papers [8, 43, 47], the resolution during training varies

between 112x112 and 224x224. Typically, a clip is resized first and then cropped. With ∼784 hours in total at 25 frames per second (fps), we have approximately 784*60*60*25 = 70.560.000 frames to analyze. If we increase our spatial resolution SxS, we quickly get limited. Modern deep-learning models such as CNNs 2.5.6 rely on GPU's during training and inference. The GPU has limited memory to work with. In practice, this means that if we increase our input size, we use more GPU memory and also require more time to process a sample. Due to both current research and limitations in both space and computational capacity, we use the compressed version. The 500 games are randomly split into a training set, validation set, and test set with 300, 100, and 100 games, respectively.

Class	Train	Validation	Test	Total
Card	1296	396	453	2145
Substitution	1708	562	579	2849
Goal	961	356	326	1643
Total	3965	1314	1358	6637

Table 3.3: The number of samples for classes for each split.

The dataset contains 6637 annotations for the three classes. Table 3.3 shows the dataset split between training, validation and test for each class. We can see that there is some imbalance between the different classes. '*Card*' represents 32.3% while '*Substitution*' and '*Goal*' represent 42.9% and 24.8%, respectively. It is essential to be aware of such imbalances since metrics used during the evaluation of results may be misleading when based on the assumption that the classes are equally distributed.

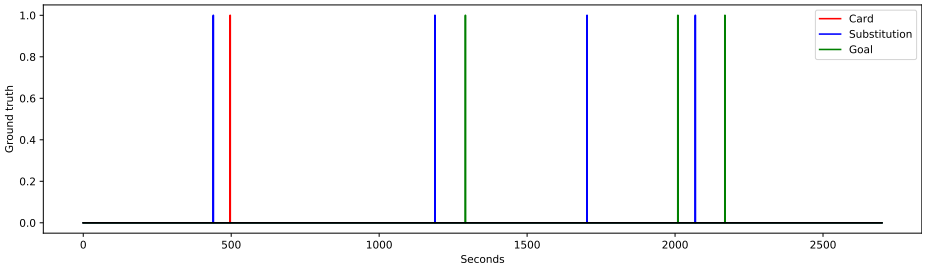


Figure 3.1: A timeline of a sample soccer game half with annotated events

Figure 3.1 shows an example of a 45-minute soccer game half with annotated events. We can note the sparsity, with 8 events spread out over 2700 seconds. If we assume that a given event lasts 5 seconds, then we have annotations for 5*6637 seconds of the total 784*60*60 seconds. This adds up to 1.17%, with the other 98.83% seconds containing something else.

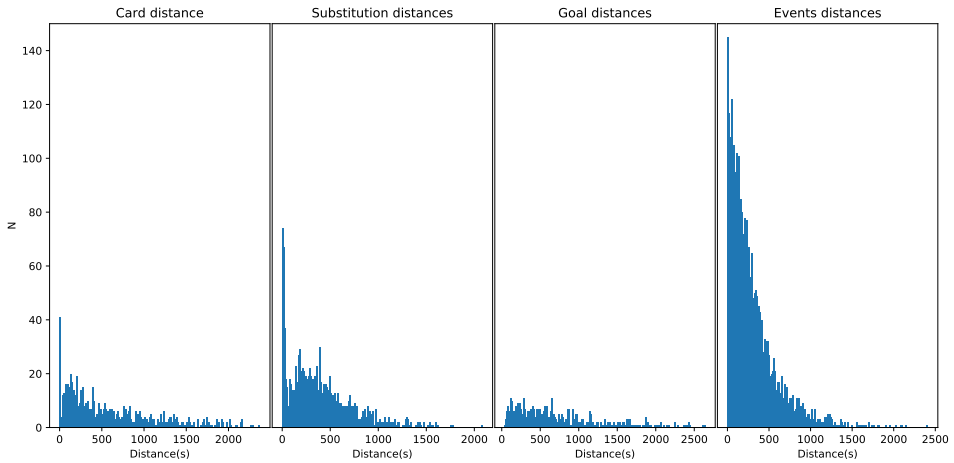


Figure 3.2: Distribution of the distances between samples within the same class in seconds. **Events distances** shows distance distribution for distances between any events. Number of bins = 200. They are calculated from the training set.

Distance	min	max	mean
Card	0.0s	2373.0s	577.0s
Substitution	0.0s	2086.0s	391.6s
Goal	40.0s	2642.0s	734.6s
All	0.0s	2410.0s	290.2s

Table 3.4: Minimum, maximum, and mean distance in seconds between events for each class from the training set. 'All' contain distances regardless of class. Calculated from training set.

For each game half in the training set, we get ordered temporal anchors for each class independently. Let a, b be temporal anchors in seconds for a class, where $a_t < b_t$. We calculate the distance in seconds as $d(a, b) = b - a$.In Table 3.4, we see the minimum, maximum, and mean distance measured between events of the same class and for all classes combined based on the training set. We can see that both '*Card*' and '*Substitution*' have events that happen at the same instant. This would occur if multiple players are substituted at the same time, or a referee showing the card to multiple players. Looking at '*Goal*' minimum distance, we find a minimum distance of 40 seconds. When a team scores in a soccer match, the timer will continue, but the players must be re-positioned to their own side of the field, after which the referee will restart the game from the middle of the field. Figure 3.2 shows us a histogram of the distances for each class separately, as well as for distances between any events.

Threshold (s)	T=20s	T=40s	T=60s	T=80s	T=100s
Card (%)	5.51	6.51	9.26	11.64	14.52
Substitution (%)	10.11	14.20	15.84	17.47	19.85
Goal (%)	0.00	0.00	0.20	2.05	3.69
All (%)	6.59	11.88	17.34	22.51	27.42

Table 3.5: Percentage of samples for each class, which has a distance of less than a threshold T seconds between samples in its own class training set. 'All' denote distances between any events regardless of class.

We investigate distances further in Table 3.5 by calculating the percentage of events for each class that has a distance less than a threshold T. For '*Card*' and '*Substitution*', it can be hard to impose strict assumptions since a relatively high percentage of the events lie close together. The class '*Goal*', however, may benefit from a rule stating that if a goal is scored at time t, then there is likely no goal in the interval [t-40,t+40].

3.3 Event detection

In many action recognition or detection datasets [29, 26, 31], an event is defined within a certain time interval. For classification tasks, this is often present in the form of trimmed clips of, for example, 3-10 seconds. For detection tasks, it is often required to predict a class along with a start and stop interval in time. A prediction is then considered correct if the class is correct, and the temporal Intersection over Union (tIoU) is higher than some threshold T. The events are annotated with a temporal anchor, which means that there is no directly annotated interval in time. Instead, spotting use the temporal anchors, and add a tolerance δ, where if a model correctly predicts an event within $\pm$ δ of the ground truth, it is considered a true positive.

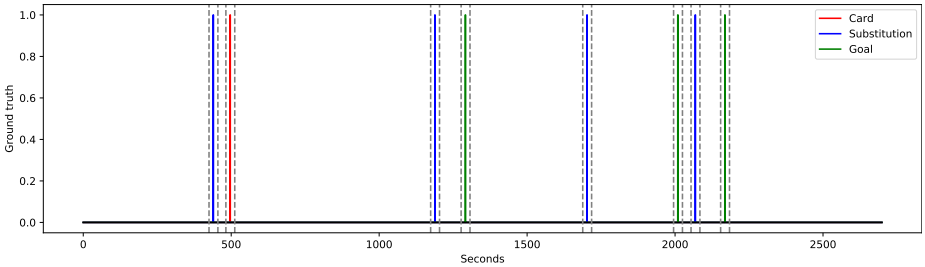


Figure 3.3: Illustrates region in which a correct prediction would be considered as a true positive for $\delta = 15s$

Figure 3.3 shows an example of ground truth with $\delta = 15s$. This means that any predictions within this interval, with the correct class, is considered correct.

3.4 Data preprocessing

This section describes how the data is organized and pre-processed during and after training. In order to train a model, we must first define some task which we can train under. We sample N frames locally with the temporal anchor

as the center. This, however, leaves us with a biased model considering all the background data. To better represent full videos, we generate background samples. We annotate a background set using the following rule: If the distance in seconds between 2 consecutive events a,b in a soccer game half is greater than 180s, then we annotate a new event labeled '*Background*' at $\frac{a_t+b_t}{2}$.

Class	Train	Validation	Test
Card	1296	396	453
Substitution	1708	562	579
Goal	961	356	326
Background	1855	636	653
Total	5820	1950	2011

Table 3.6: N number of samples for classes.

Table 3.6 shows our new dataset that contains '*Background*'. The dataset is slightly unbalanced, as noted previously in Section 3.2, about 98% of a soccer match is indeed background. Therefore this is not a perfect representation. There are some weaknesses as well since we automatically generate new samples. '*Background*' may be annotated during replays of any of the three events, while statistically, it is likely rare, we expect some 'bad' samples to be present.

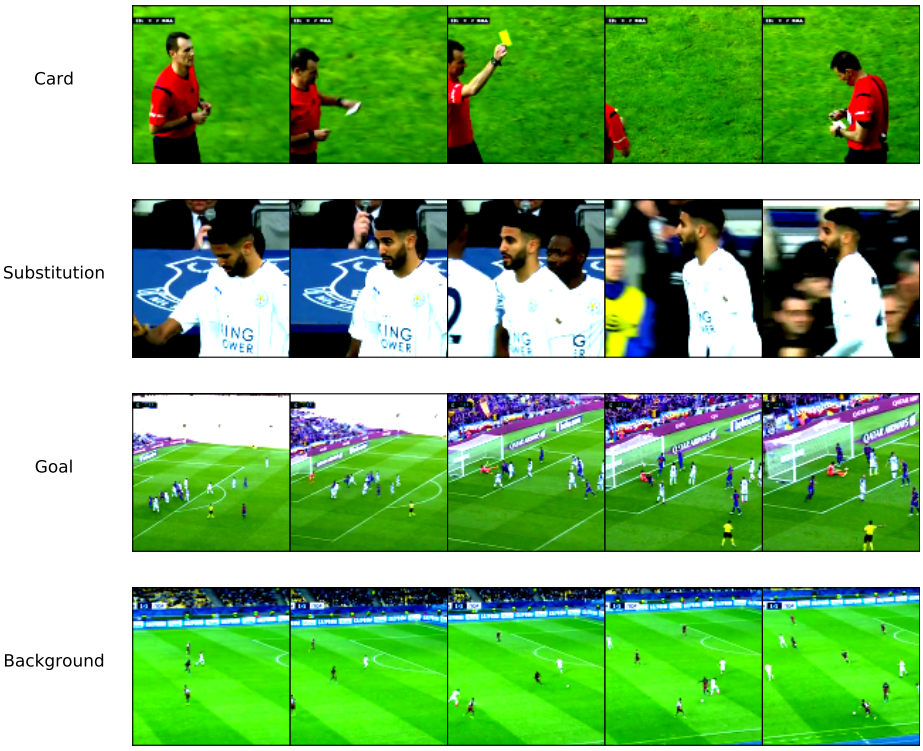


Figure 3.4: Sample frames visualized for a sample of 128 frames. The middle frame is at the annotated time.

During training, we pre-process the clips on the fly. First, we resize clips to a resolution of 112 x 199, followed by normalization of the data. Then, we

randomly crop a 112 x 112 clip. Finally, we randomly flip each frame with a probability of 0.5. Considering that soccer fields are symmetric around that center, therefore this may make the model more robust to events on either side. In Figure 3.4 we see some typical samples of the 4 classes.

3.5 Overview of models

To successfully detect events, we want models that can capture both spatial and temporal information. ResNet 3D 18 (R3D), ResNet Mixed Convolution 18 (MC3) and ResNet (2+1)D 18 (R (2+1)D) [47] are some strong models that rely only on RGB input. These models are available pretrained on Kinetics-400 using 16x112x112 inputs from PyTorch. Many models use optical flow [8, 43] in order to capture temporal information. There are some disadvantages to this approach, such as the need to pre-process optical flow. Instead of using an optical flow-based approach, we test the SlowFast architecture [16]. SlowFast has similar ideas but a different approach.

3.5.1 Model architectures

The models R3D, MC3, and R (2+1)D have similar structures. They all have a total of 17 convolutional layers with batch-normalization followed by global average pooling and a fully connected layer. For R3D, MC3, and R (2+1)D, we use 2 convolutional layers per block.

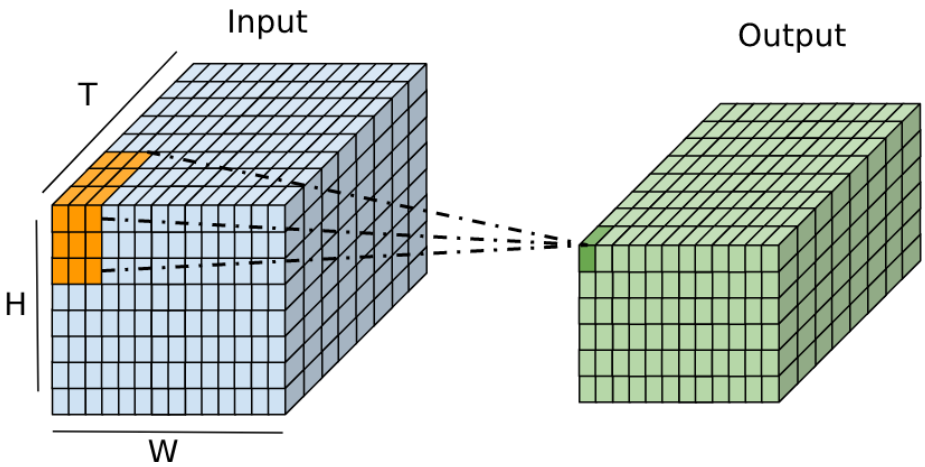


Figure 3.5: llustrates 3D convolution where W = width, H = height, T = frame

3D Convolution block We want to learn feature representations that are able to capture spatial information such as edges, colors, and more. However, we also want to capture temporal information, such as motion. 3D convolution can learn spatio-temporal features that help us achieve this goal. In Figure 3.5, we see an illustration of 3D convolution. We can consider a video clip as a stack of frames in the shape of C x T x H x W where C denotes the RGB channels, T the number of frames in the video and H, W the height and width

in pixels respectively. The input in this example shows a kernel containing learnable parameters with dimensions 3 x 3 x 3 producing the weighted sum for the output. The dimensions of the kernel is of the form *depth* x *height* x *width*.

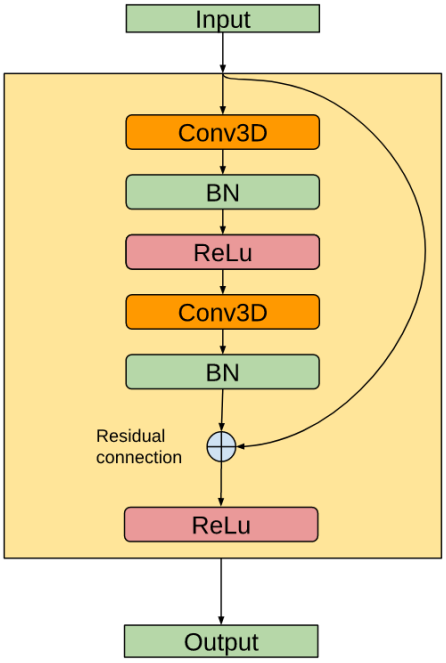


Figure 3.6: Illustrates a residual 3D block

3D convolution blocks contain 2x convolutional layers with batch normalization and ReLu, followed by a residual connection. Figure 3.6 illustrates this structure.

2D Convolution Block 2D Convolution blocks are built the same way as 3D Convolution blocks. 3D convolution is applied with kernels of dimensions 1 x S x S, where S denotes the dimension used spatially. This is equivalent to performing 2D convolution on each frame individually with the same kernel.

(2+1)D Convolution Block (2+1)D convolutional can be interpreted as 3D convolution broken into two separate steps. First, we use 3D convolution with a kernel of dimensions 1 x S x S like in 2D convolution blocks. This is followed by batch normalization and ReLu. Finally, a second 3D convolution is performed using a kernel with dimensions T x 1 x 1, where T denotes the temporal dimension. In other words, how many frames at each point. This effectively doubles the number of ReLu activations.

Stem The stem is the first layer, R3D and MC3 use the same basic stem, which is a 3D convolution followed by batch normalization and ReLu. This outputs 64 channels using a kernel of size 3 x 7 x 7 and stride of 1 x 2 x 2. R (2+1)D and SlowFast, the stem is different. For R (2+1)D, it is similar to the basic stem, but use 2+1D convolution.

Average pool Global average pooling is used after the last convolutional block. This reduces a tensor of dimensions C x T x H x W to C x 1 x 1 x 1. For R3D, MC3, and R (2+1)D, this results in 512 features that are fed into a linear output layer. We can note that the dimensions of the feature tensor into the global average pooling layer will change depending on the resolution or number of frames, but this does not interfere with our output layer since we average the values over time and space. This means that we can use arbitrary resolution or number of input frames.

3.5.2 Models

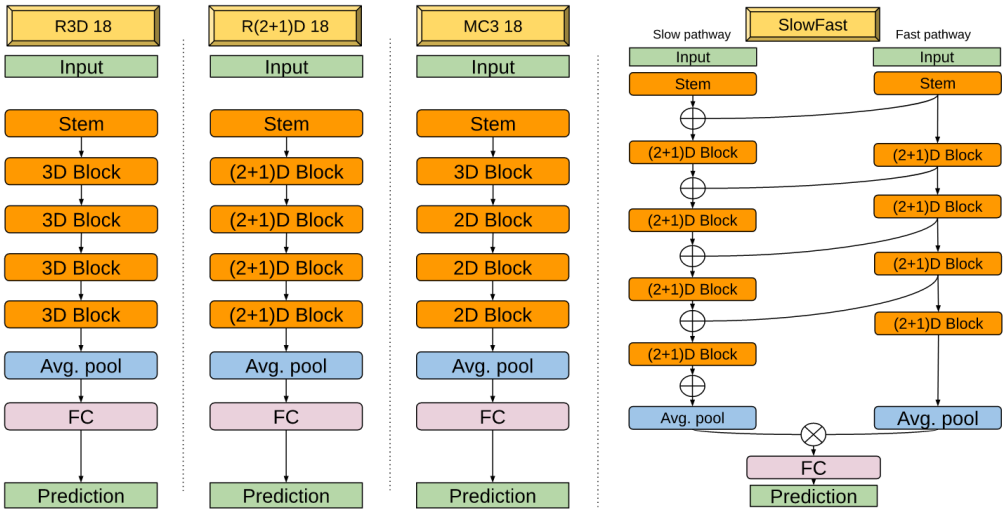


Figure 3.7: Figure illustrates the basic architectures of the models used [16, 47]. Here, $\bigoplus$ represents elementwise sum, and $\bigotimes$ represents concatenation.

In Figure 3.7, we see four different models that meet our criteria. R3D and MC3 all use 2x blocks for each layer, for a total of 4 convolutional layers per layer. R (2+1)D Technically have 4 convolutional layers per block. The stem contains for each R3D, and MC3 contains 1 single convolution-batch norm-ReLu layer, while R (2+1)D and SlowFast use a (2+1)D convolution.

Overview of models	
Model	Parameters
R3D	33,168,324
MC3	11,492,292
Res (2+1)D	31,302,177
SlowFast	12,684,816

Table 3.7: Parameters for the different models

Table 3.7 shows the number of trainable parameter for each model. We see that MC3 and SlowFast have about a third of the parameters in R (2+1)D and R3D.

R3D R3D uses a stem with a single-layered 3D convolutional block with a 3 x 7 x 7 kernel and stride 1 x 2 x 2 followed by multiple two-layered blocks

with 3 x 3 x 3 kernels throughout the network before avg.pooling and the linear output layer. This is a straightforward approach and enables the model to learn spatio-temporal features.

R (2+1)D R (2+1)D use (2+1)D convolutional stem followed by multiple two-layered 2+1D blocks, resulting in higher cost during training mainly due to double the number of convolutions, batch norm, and ReLu when compared to R3D. Tran et al. [47] argue that it may be easier to learn spatial features and temporal features separately.

MC3 MixedConvolution 3D uses both 3D and 2D convolution blocks. It should be noted that 2D convolution blocks are actually 3D convolutions, with 1 x S x S kernels. This model uses the basic stem as in R3D, followed by a layer of two 3D convolutional blocks. After which three layers of 2x 2D convolutional blocks are used before avg.pool and a linear output layer. The authors in Tran et al. [47] note that the idea here is that it may be best to get temporal features in the early layers, focusing on spatial features deeper in the network.

SlowFast We implement a model that is inspired by the state-of-the-art architecture for action recognition called SlowFast [16]. This builds upon the popular idea of having multiple pathways, such as in Two-Stream networks [43]. The main idea is to use one high-capacity 'slow' pathway, which is focused on temporally low-resolution spatial features, and a 'fast' low capacity pathway which focus on temporally high-resolution features. The number of channels between each pathway is related by a factor β. Here we use 1/8 as in the original version. Temporally, the pathways are related via a factor α, where the temporal stride of the stem for the slow pathway is defined as αT x H x W, with T x H x W being the stride for the slow pathway. We use $alpha = 8$, resulting in 8 times more frames in the fast pathway. We illustrate the architecture in Figure 3.7. First, we take inputs of size C x 64 x 224 x 224, which we input into each separate 3D stem. The slow pathway uses a kernel with dimensions 1 x 7 x 7 and a stride of 16 x 2 x 2 for dimensions T x H x W denoting frames, height, and width, respectively. This effectively downsamples from 64 frames to 4 frames in the slow pathway stem. The fast pathway uses a kernel of 5 x 7 x 7 for the stem with a stride of 2 x 2 x 2. We now have two separate streams with a temporal dimension of 4 and 32. Next, we use layers with one 2+1D block for each pathway. After each block, we fuse information from the fast pathway to the slow pathway. We do this by summation. Since the dimensions are incorrect throughout the network, we 3D average pool outputs from the fast pathway to fit the temporal dimension of the slow pathway. Next, we average across feature channels, after which we inflate the result to fit the slow pathway. Finally, we avg. Pool both pathways and concatenate the results which we feed into our linear output layer.

3.5.3 Implementation

We implement using Cuda 10.1 python 3.6.9 with PyTorch 1.3.1 and torchvision 0.4.2. For evaluation metrics, sklearn was used, while NumPy and Pandas were used when appropriate. The development was mainly done locally on a PC

with 16gb RAM and a GTX 2080Ti. The code is available on GitHub when requested.

Pytorch

PyTorch is an open-source machine learning framework that has tools for a variety of situations in data analysis. Both higher-level problems that are common such as creating dataset loaders or simple pre-processing steps are available, and lower-level computations using PyTorch Tensors, which is their version of n-dimensional arrays. The main reason to use PyTorch is that it is optimized towards single or multi-GPU systems. Torchvision is a PyTorch package that contains datasets, models, transformations, and more with a focus on vision tasks such as image or video classification or detection.

Nvidia Automatic-Mixed-Precision

Nvidia Apex 0.1 is a PyTorch extension that supports Automatic-Mixed -Precision (AMP). During a forward and backward pass of a batch, the GPU has to hold large amounts of data, which means that with video, we can quickly get limited by GPU-memory. By default, weights, activation values, gradients are typically stored as Floating Point 32 (FP32). While it is possible to use FP16, which will effectively half the GPU-memory requirements and result in fewer computations, it can cause problems with numerical stability and end up hurting accuracy. AMP [38] will use both FP32 and FP16 where appropriate, with some added tricks to help numerical stability. The result is a cheaper way to train models, which enables us to use larger batch sizes or inputs in general during training.

DGX-2

DGX-2 is a deep learning system that was used for training and testing in this thesis. It holds 16 Nvidia Tesla V100 GPU's with 512GB GPU memory total. It also has 1.5TB system memory, which can be helpful when first loading batches. With a single video clip holding 128 frames, 3 x 128 x 112 x 112, R (2+1)D required about 5GB GPU memory during training. With AMP, this is reduced to about 3.3 GB. Results can suffer if the batch size is too small during training, hence the need for GPU memory. Noteworthy, we need about triple the amount when we increase our resolution to 224 x 224. Since R (2+1)D holds double the number of activations, it has higher memory requirements than R3D and MC3. For 224 x 224 resolution at 128 frames with AMP, R3D, and MC3 require about 4GB and 4.8 GB, respectively. During training, multiple GPUs are used to work in parallel, where batches are equally distributed across GPUs.

3.6 Evaluation of Models: Definition of metrics

When we create a model, it is essential to ask the simple, yet sometimes tricky question, how good is our model? The correct way to evaluate a model is case dependant. Often, there are different costs for different errors. For example, a self-driving car incorrectly predicting a human as the road can have dire consequences and may lead to a tragic accident, while Netflix's recommendation

model incorrectly predicting the viewer's taste in movies will hopefully only lead to mild annoyance.

To properly evaluate a model, we need metrics that we can interpret relative to the specific case at hand.

A multi-class classification problem can be interpreted as a one-vs-all binary classification for each class in which *true positive* (TP) is when a model predicts the correct class, a *false positive* (FP) is when a classes are incorrectly predicted, a *true negative* (TN) when a class is correctly rejected, and a *false negative* (FN), where a class was incorrectly rejected.

$$Accuracy = \frac{N_{Correct}}{N_{Total}} \tag{3.1}$$

Where N_{Total} is the total number of samples, and $N_{Correct}$ is the number of correctly predicted samples. Accuracy is a simple yet effective metric to use. High accuracy will often indicate a good result. However, it can often be misleading.

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

A precision score holds a value between 0 and 1, where 1 would be considered the best results. If we have at least 1 correct prediction, precision will go to 1 as FP goes to 0. Therefore, if we are interested in how valuable our predictions are for a particular class, precision may be a useful metric.

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

Recall is another metric, where we focus on true positives and false negatives. Similar to precision, the score will land between 0 and 1, where higher is better. As the number of false negatives goes to 0, we see that recall goes to 1 under the assumption of at least 1 true positive. Therefore, it does not directly matter if we have multiple false positives. If we always predict a class as positive, we will get a recall of 1 for that class, which means that we will never incorrectly predict a negative for that specific class. We should note, however, that this will likely result in many false positives and a poor precision score.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

F1-score is the harmonic mean of precision and recall. It is a measure that will typically land between the precision and recall scores, thereby punishing significant differences.

$$Avg(S) = \frac{\sum_{i=1}^{C} S_i}{C} \tag{3.5}$$

S_i is the calculated precision,recall or F1-score where i denotes the i'th class for $i \in \{1, 2, ..., C\}$ for C classes.

$$Weighted\ Avg(S) = \frac{\sum_{i=1}^{C} S_i * N_i}{N_{Total}} \tag{3.6}$$

Where N_i is the number of samples for the i'th class. To get an overall picture of the different metrics, we average the scores across classes. One crucial factor is

an imbalance in the dataset. Therefore, we calculate both average and weighted average.

A common way to evaluate results in skewed data is through the precision-recall (PR) curve, which is precision and recall calculated at different confidence thresholds.

$$Average\ Precision(AP) = \sum_n (R_n - R_{n-1})P_n \tag{3.7}$$

Where R_n and P_n is the recall and precision at the n'th threshold. AP is related to the precision-recall and can be calculated as the area under the curve. This is useful as it reduces the PR-curve to a single comparable numeric value.

$$mean\text{-}Average\text{-}Precision = \frac{\sum_{i=1}^{C} AP_i}{C} \tag{3.8}$$

Where AP_i is AP calculated for the i'th class for C classes, mean-average-precision (mAp) is the mean AP calculated over all classes.

3.7 Model selection

This section describes the selection of both the model and parameters. We want to find a model that works well for event detection. In section 3.7 we describe several candidate models. Furthermore, we want to find the best parameters for these models. Therefore, before attempting to detect events in the roughly 150 hours present in the validation and test set, we perform several experiments to find the best model and configuration. First, we describe the different hyper-parameters which are the same for most models and the effect these may have. Next, we define different metrics that are used to evaluate the models. We show the effectiveness of transfer learning by comparing the results of R3D, MC3, and R (2+1)D with and without pretraining. The pretrained models are available from PyTorch and have been trained using the Kinetics-400 dataset [31], while the others were initialized with Kaiming He initialization [25]. Based on the results, we continue with pretrained models and further investigate the effect of both spatial resolution and temporally extended inputs. Reducing the input dimensionality may be useful in practical applications, we test model performance with grayscale input which also may provide insight into whether or not RGB colors are necessary for the model to classify correctly.

3.7.1 Hyperparameters

Hyperparameters are parameters that we set before training. The specific parameters change depending on methods use, but generally, one might argue that for neural network optimization, the minimum needed is a learning rate. Initial tests during development showed similar results across models for different learning rates, batch sizes, and the number of epochs. For our experiments in this chapter, we use a learning schedule similar to Tran et al. [47] for finetuning with the same initial learning. We use SGD with 0.9 momentum and with an initial learning rate of 0.001, reducing it by a multiplicative factor of 0.1 every 10 epochs, as illustrated in Figure 3.8. , with a batch size of 64. We keep hyperparameters the same during experiments for comparability unless otherwise

stated. A mini-batch size of 64 is used for all experiments in this section. For comparison Tran et al. [47] use a minibatch-size of 32 while Feichtenhofer et al. [16] use a minibatch size of 1024 when training on Kinetics-400 [31]. The use of large minibatch can come with challenges both in terms of GPU memory and performance [22, 32]. We, therefore, believe that 32 or 64 are both reasonable choices for this case and move forward with 64. During our experiments, we save both model weights, which results in the highest validation accuracy and the result. We use the model parameters based on the highest validation accuracy during all evaluations. This can be thought of as a simplified version of early stopping.

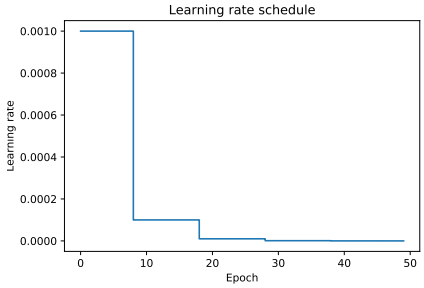


Figure 3.8: Learning rate schedule used

3.7.2 Kinetics-400 pretrained models

Transfer learning has shown excellent results for generalization, as discussed previously in Section 2.5.7. Parameters for R3D, R (2+1)D, and MC3 are available pretrained on Kinetics-400 through PyTorch. The models were trained with 16 frame inputs at 112x112. The models are trained on the Kinetics-400 [31], a sizeable popular action recognition dataset with 400 classes. Kinetics-400 holds a variety of classes, such as, with varying degrees of similarity to soccer. For example, '*Crying*','*Eating carrots*' and '*Assembling computer*' are included. There are 25 classes that contain ball sports, with '*Shooting goal (soccer)*' and '*Kicking soccer ball*' having 444,544 samples, respectively. For the three models, we replace the last fully connected layer to the output to fit our 4 classes and finetune with hyperparameters mentioned in Section 3.7.1. In these experiments, we use 16 frames as inputs with a spatially resized input to 112 x 199 and cropped at 112 x 112. For finetuning, the cropping is randomly selected, while the center crop is used during the evaluation of the validation set. What we observe is that pretrained models consistently outperform training from scratch.

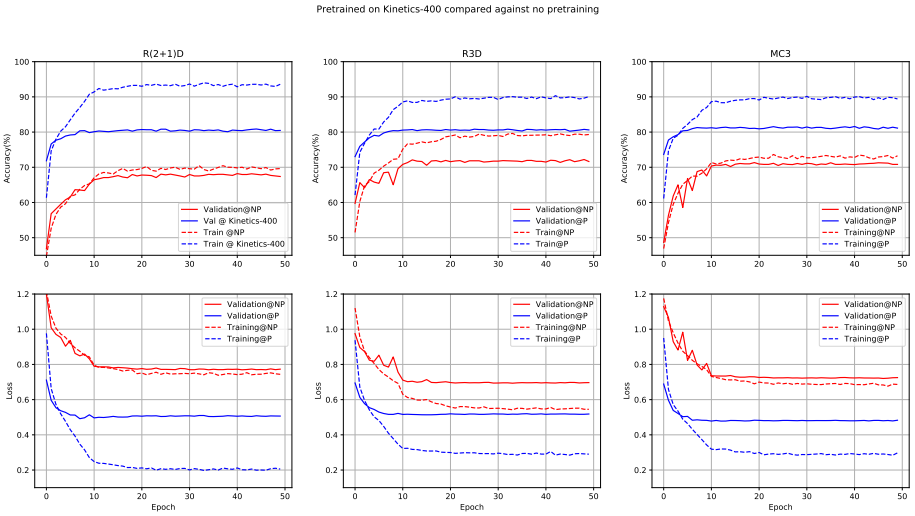


Figure 3.9: Accuracy and loss for each model.

In Figure 3.9 we see similar results across models. One difference we see is that for R3D non-pretrained, the accuracy and loss between validation and training evaluation seem to diverge. At the same time, for R (2+1)D and MC3, this effect is not as prominent. For all pretrained models, we observe that accuracy and the loss for keeps increasing by about 10 % while validation accuracy and loss reaches a plateau. This can indicate overfitting and is stopped by the learning schedule.

Pretrained vs from scratch validation				
Model	**Class**	**Precision**	**Recall**	**F1-score**
R (2+1)D NP	Card	0.583	0.523	0.551
	Substitution	0.783	0.512	0.619
	Goal	**0.884**	0.514	0.650
	Background	0.522	**0.836**	0.643
R3D NP	Card	0.775	0.566	0.654
	Substitution	0.691	0.870	0.770
	Goal	0.856	0.770	0.811
	Background	0.681	0.678	0.679
MC3 NP	Card	0.715	0.601	0.653
	Substitution	0.712	0.836	0.769
	Goal	0.801	0.860	0.829
	Background	0.706	0.638	0.671
R (2+1)D	Card	0.744	0.770	0.757
	Substitution	0.851	0.804	0.827
	Goal	0.881	0.913	0.897
	Background	0.766	0.770	0.768
R3D	Card	0.811	0.793	0.802
	Substitution	0.837	0.870	0.853
	Goal	0.858	**0.952**	**0.903**
	Background	**0.812**	0.745	0.777
MC3	Card	**0.825**	**0.808**	**0.816**
	Substitution	**0.867**	**0.890**	**0.878**
	Goal	0.869	0.916	0.892
	Background	0.802	0.769	**0.785**

Table 3.8: Per-class precision, recall and F1-score. Higher is better, best result per class in bold. Both models are trained using 16 frame inputs.

Kinetics-400 pretrained vs from scratch on validation set					
Model	**Avg. method**	**Precision**	**Recall**	**F1-score**	**Accuracy (%)**
R (2+1)D NP	Unweighted	0.693	0.596	0.616	62.051
	Weighted	0.675	0.621	0.619	—
R3D NP	Unweighted	0.751	0.721	0.729	72.718
	Weighted	0.735	0.727	0.724	—
MC3 NP	Unweighted	0.733	0.734	0.730	72.821
	Weighted	0.727	0.728	0.724	—
R (2+1)D	Unweighted	0.810	0.814	0.812	80.615
	Weighted	0.807	0.806	0.806	—
R3D	Unweighted	0.830	0.840	0.834	82.872
	Weighted	0.827	0.829	0.827	—
MC3	Unweighted	**0.841**	**0.846**	**0.843**	**83.846**
	Weighted	**0.837**	**0.838**	**0.838**	—

Table 3.9: Weighted and unweighted precision, recall and F1-score. Higher is better, best result per metric in bold. Both models are trained using 16 frame inputs.

First, we are interested in any significant disparities between classes for each metric. Next, we look at the relationship between precision and recall, and finally, how does pretraining compare to training from scratch. We make the following observations in table Table 3.8. R (2+1)D NP has the highest '*Goal*' precision, but also the lowest score for 'textitCard' and '*Background*' classes. Noteworthy, the event '*Goal*' has 0.884 precision and 0.514 recall, which tells us that the model has many false negatives, effectively missing out on many goals while having some success on samples predicted positive. For background, the opposite happens, high recall and low precision, which means that there are many false positives for '*Background*'. The other models trained from scratch seem to achieve more balanced scores. In Table 3.9, we see that the pretrained models achieve better results, with the best results from MC3 landing at 83.8% accuracy compared to its from-scratch counterpart at 72.8% accuracy. Based on these results, we conclude that pretrained models generalize better to this task. Therefore we move forward to our other experiments using pretrained models unless otherwise stated.

3.7.3 Effect of different temporal inputs

How long should the temporal interval be for the input of action recognition models? There are no widely accepted answers to this question at the time of writing. In Tran et al. [47], they find that accuracy peaks at 32 frames. Our videos are all at 25fps, which means that 32 frames are 1.28s of video. In order to find the best suited temporal input-size for soccer, we train and test our models with 8,16,32,64 and 128 frames. We stop at 128 frames due to the increasingly high requirements for GPU-memory. We use pretrained models due to the results in Section 3.7.2.

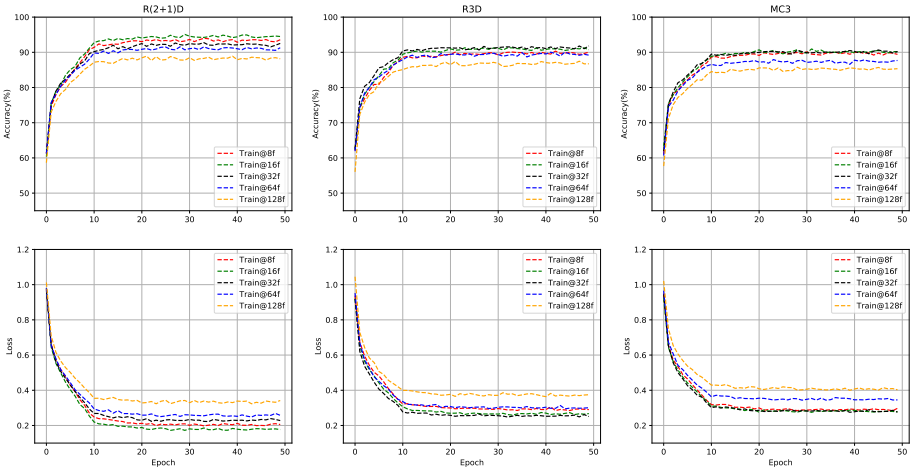


Figure 3.10: Accuracy and loss during training on training set

In Figure 3.10, we can see that during training, accuracy and loss are similar for all models. It seems that when we increase our frames, we also converge to lower accuracy and higher loss. This may indicate that it performs worse, but it could also indicate that the model is less prone to overfitting.

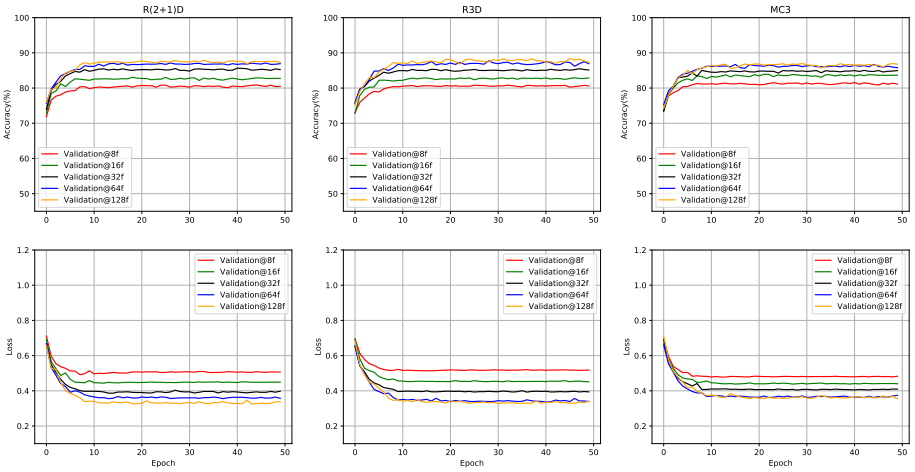


Figure 3.11: Accuracy and loss during training on validation set

In Figure 3.11, we see that validation accuracy for all models improves with more frames. It seems that the model reaches a plateau in training after about 10-15 epochs. We should note the difference seems to peak at around 128 frames.

Results based on number of frames as input				
Model	**Class**	**Precision**	**Recall**	**F1-score**
R (2+1)D 8f	Card	0.700	0.795	0.745
	Substitution	0.854	0.769	0.809
	Goal	0.887	0.840	0.863
	Background	0.741	0.766	0.753
R (2+1)D 16f	Card	0.744	0.770	0.757
	Substitution	0.851	0.804	0.827
	Goal	0.881	0.913	0.897
	Background	0.766	0.770	0.768
R (2+1)D 32f	Card	0.742	0.874	0.803
	Substitution	0.888	**0.849**	0.868
	Goal	0.896	**0.947**	**0.921**
	Background	**0.853**	0.766	0.807
R (2+1)D 64f	Card	**0.815**	0.859	**0.836**
	Substitution	0.914	0.815	0.862
	Goal	**0.940**	0.885	0.912
	Background	0.785	**0.860**	0.821
R (2+1)D 128f	Card	0.768	**0.909**	0.832
	Substitution	**0.938**	0.810	**0.869**
	Goal	0.928	0.907	0.918
	Background	0.826	0.841	**0.833**

Table 3.10: Accuracy, precision, recall and F1-score per class for R (2+1)D pretrained on Kinetics-400 with different number of consecutive frames as input

Results based on number of frames as input				
Model	**Class**	**Precision**	**Recall**	**F1-score**
R3D 8f	Card	0.789	0.758	0.773
	Substitution	0.827	0.852	0.840
	Goal	0.856	0.916	0.885
	Background	0.772	0.741	0.756
R3D 16f	Card	0.811	0.793	0.802
	Substitution	0.837	0.870	0.853
	Goal	0.858	0.952	0.903
	Background	0.812	0.745	0.777
R3D 32f	Card	0.836	0.823	0.830
	Substitution	0.861	0.915	0.887
	Goal	0.894	0.947	0.920
	Background	0.838	0.772	0.804
R3D 64f	Card	0.903	**0.846**	0.874
	Substitution	0.873	**0.920**	0.896
	Goal	0.893	**0.963**	0.927
	Background	**0.856**	0.811	0.833
R3D 128f	Card	**0.917**	0.838	**0.876**
	Substitution	**0.880**	0.916	**0.898**
	Goal	**0.912**	0.955	**0.933**
	Background	0.852	**0.844**	**0.848**

Table 3.11: Accuracy, precision, recall and F1-score per class for R3D pretrained on Kinetics-400 with different number of consecutive frames as input

Results based on number of frames as input				
Model	**Class**	**Precision**	**Recall**	**F1-score**
MC3 8f	Card	0.817	0.755	0.785
	Substitution	0.828	0.865	0.846
	Goal	0.859	0.927	0.892
	Background	0.777	0.748	0.762
MC3 16f	Card	0.825	0.808	0.816
	Substitution	0.867	0.890	0.878
	Goal	0.869	0.916	0.892
	Background	0.802	0.769	0.785
MC3 32f	Card	**0.881**	0.806	0.842
	Substitution	0.856	**0.911**	0.883
	Goal	0.865	0.919	0.891
	Background	0.819	0.788	0.803
MC3 64f	Card	0.860	**0.871**	0.866
	Substitution	0.880	0.909	0.894
	Goal	**0.878**	0.949	**0.912**
	Background	**0.847**	0.777	0.811
MC3 128f	Card	0.869	**0.871**	**0.870**
	Substitution	**0.909**	0.884	**0.896**
	Goal	0.860	**0.952**	0.904
	Background	0.840	**0.808**	**0.824**

Table 3.12: Accuracy, precision, recall and F1-score per class for MC3 pre-trained on Kinetics-400 with different number of consecutive frames as input

Results based on number of frames as input					
Model	**Avg. method**	**Precision**	**Recall**	**F1-score**	**Accuracy (%)**
R (2+1)D 8f	Unweighted	0.796	0.792	0.792	78.615
	Weighted	0.792	0.786	0.788	—
R (2+1)D 16f	Unweighted	0.810	0.814	0.812	80.615
	Weighted	0.807	0.806	0.806	—
R (2+1)D 32f	Unweighted	0.845	0.859	0.850	84.462
	Weighted	0.849	0.845	0.844	—
R (2+1)D 64f	Unweighted	0.864	0.855	0.858	85.128
	Weighted	0.857	0.851	0.852	—
R (2+1)D 128f	Unweighted	0.865	0.867	0.863	85.795
	Weighted	0.865	0.858	0.859	—
R3D 8f	Unweighted	0.811	0.817	0.813	80.821
	Weighted	0.807	0.808	0.807	—
R3D 16f	Unweighted	0.830	0.840	0.834	82.872
	Weighted	0.827	0.829	0.827	—
R3D 32f	Unweighted	0.857	0.864	0.860	85.538
	Weighted	0.854	0.855	0.854	—
R3D 64f	Unweighted	0.881	0.885	0.882	87.744
	Weighted	0.877	0.877	0.877	—
R3D 128f	Unweighted	**0.890**	**0.889**	**0.889**	**88.410**
	Weighted	**0.884**	**0.884**	**0.884**	—
MC3 8f	Unweighted	0.820	0.824	0.821	81.59
	Weighted	0.815	0.816	0.815	—
MC3 16f	Unweighted	0.841	0.846	0.843	83.846
	Weighted	0.837	0.838	0.838	—
MC3 32f	Unweighted	0.855	0.856	0.855	85.077
	Weighted	0.851	0.851	0.850	—
MC3 64f	Unweighted	0.866	0.877	0.871	86.564
	Weighted	0.865	0.866	0.864	—
MC3 128f	Unweighted	0.869	0.879	0.874	86.923
	Weighted	0.869	0.869	0.869	—

Table 3.13: Validation accuracy as well as weighted and unweighted average precision, recall and F1-score. Higher is better, best result per class in bold.

In Tables 3.10-3.13, we see that the models generally seem to improve with more frames. All models seem to greatly benefit from 8 frames to 16 frames with about 2% improvement in accuracy. The same jump is seen from 16 frames to 32 frames, with about 3% improvement for R (2+1)D and R3D. 64 frames

perform better, but interestingly, it seems that R (2+1)D does not improve as much with more frames when compared to R3D and MC3. With almost 8% higher validation accuracy from 8 frames to 128 frames, it seems that R3D scales best with a longer temporal window. At 128 frames with 25fps, we have a 5.12-second input. As we increase our window, we will likely have too much irrelevant information. Intuitively, around 5 seconds seems reasonable for a human annotator to understand what event was present, at 20-30-40 seconds however, there may be many other events present, which makes it less discernible. During validation, we use the same number of frames as the model was trained on. It may be that a larger window during validation is simply an easier task for the model. Therefore, we briefly test R3D trained on 64 frames on a 128 frame validation set, and R3D trained on 128 frames on 64 frames.

Models evaluated with different input size than during training					
Model	**Avg. method**	**Precision**	**Recall**	**F1-score**	**Accuracy (%)**
R3D 64f	Unweighted	**0.880**	**0.877**	**0.878**	**87.3**
	Weighted	**0.874**	**0.873**	**0.873**	—
R3D 128f	Unweighted	0.874	0.863	0.868	86.3
	Weighted	0.865	0.863	0.864	—

Table 3.14: Results for models trained on 64 and 128 frames when tested with 128 and 64 frames respectively.

In Table 3.14, we can observe that both models have a drop in performance. However, the 64 frame version is less negatively impacted. R3D 128f drops more than 2% in validation accuracy. It may be that a higher number of frames makes training more stable. Intuitively, we can say that it makes sense for the model to perform worse with smaller inputs. We conclude that the 128f version performs the best overall, and is, therefore, the best choice for our final tests.

3.7.4 Reduced input

There are many benefits with data reduction, such as storage and possibly computation depending on the application. Here, we test our model capabilities with regard to reduced inputs. When reducing RGB to gray-scale, we remove two-thirds of the information. We use the same models, but alter the stem layer such that it takes single-channel video, the models are not pretrained. We use 16 frame inputs for all models at 112 x 112 resolution.

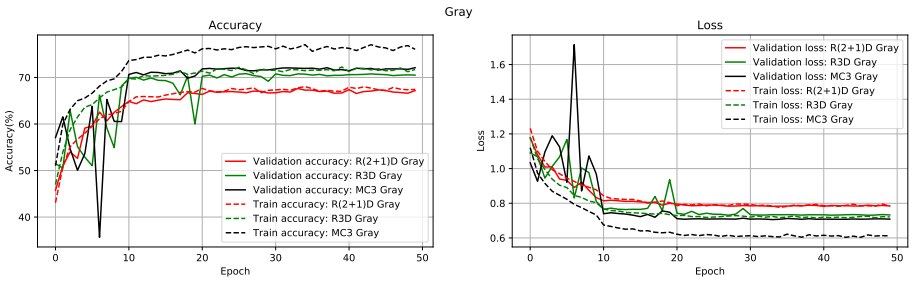


Figure 3.12: Accuracy and loss for gray input frames for training and validation set. Models are trained on 16 frames

We can see that training seems a bit unstable during training. For the first 20 epochs, there are several moments where validation accuracy and loss suddenly jumps. It may be that the learning rate is too high during the initial epochs for this case. Interestingly, R (2+1)D seems to be much more stable when compared to R3D and MC3. It is not clear why this is the case, and we can only speculate. If we look back at Figure 3.9 in Section 3.7.2, we see some similarities with more stable results through training comparatively.

Results of gray-scale input				
Model	**Class**	**Precision**	**Recall**	**F1-score**
R (2+1)D + Gray	Card	0.681	0.538	0.601
	Substitution	0.676	0.779	0.724
	Goal	0.721	0.843	0.777
	Background	0.670	0.604	0.635
R3D + Gray	Card	**0.677**	0.614	0.644
	Substitution	**0.705**	**0.794**	**0.746**
	Goal	**0.812**	**0.888**	**0.848**
	Background	**0.705**	**0.631**	**0.666**
MC3 + Gray	Card	0.675	**0.624**	**0.648**
	Substitution	0.704	0.767	0.734
	Goal	0.765	**0.888**	0.822
	Background	0.692	0.608	0.648

Table 3.15: Results per class for models trained on gray-scale inputs

Results of gray-scale input					
Model	**Avg. method**	**Precision**	**Recall**	**F1-score**	**Accuracy (%)**
R (2+1)D + Gray	Unweighted	0.687	0.691	0.684	68.462
	Weighted	0.683	0.685	0.680	—
R3D + Gray	Unweighted	**0.725**	**0.731**	**0.726**	**72.103**
	Weighted	**0.719**	**0.721**	**0.718**	—
MC3 + Gray	Unweighted	0.709	0.722	0.713	70.821
	Weighted	0.706	0.708	0.705	—

Table 3.16: Results average over all for models trained on gray-scale inputs

If we compare results in Table 3.15 and Table 3.16 to the results on non-pretrained models in Section 3.7.2 we can note that the difference is small. R (2+1)D performed worse on RGB inputs when trained from scratch, with only about 62% validation accuracy. Overall we still get slightly worse results with a drop of about 0.6% in validation accuracy for R3D and a drop of about 2.8% validation accuracy for MC3. However, these results indicate that color is not a necessity for this kind of classification task.

3.7.5 SlowFast results

SlowFast is a state-of-the-art model, noticeably winning the AVA challenge 2019 [23], which is a benchmark for spatio-temporal action detection. The architecture is described in Section 3.5. Here, we test the model at 224 x 224 with 64 frame inputs using two different learning rate schedules. First, we test the model using the same parameters as described in Section 3.7.1. An initial learning rate of 0.001, divided by 10 every 10 epochs. Next, we use the learning rate schedule used by the authors of SlowFast [16]. We use warm restarts as described in [36].

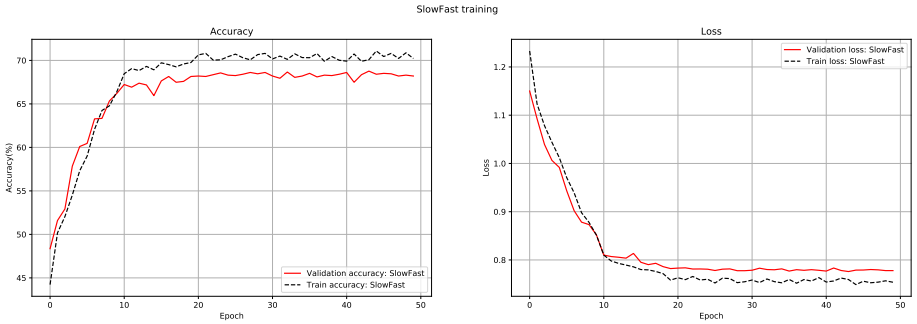


Figure 3.13: Figure shows accuracy and loss for SlowFast 2+1D

In Figure 3.13, we see that the model struggles to improve both on the training set and validation set after 10 epochs. It may be that the model requires a higher learning rate in order to converge appropriately. We, therefore, train the model with warm restarts for 100 epochs.

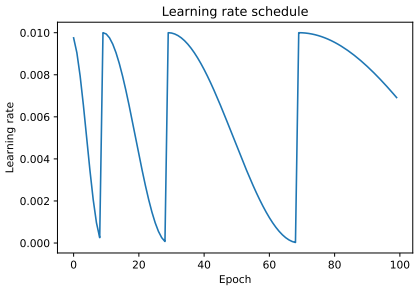


Figure 3.14: Learning rate schedule with warm restarts

In Figure 3.14, we see the learning rate schedule used for training during 100 epochs. We set the initial learning rate to 0.01, with a minimum learning rate of 0.00001.

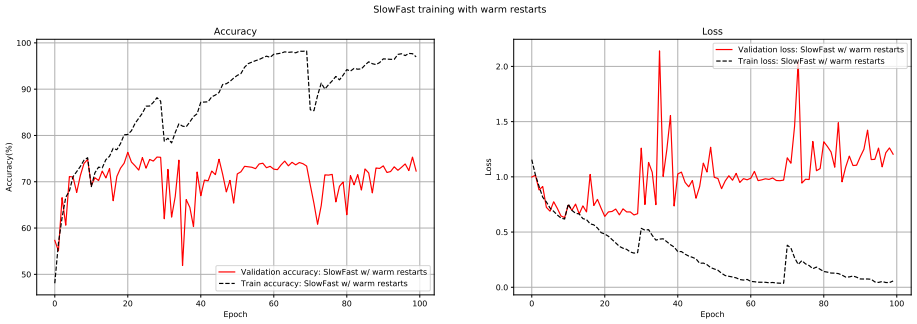


Figure 3.15: Accuracy and loss for training and validation set when using warm restarts

During training, we can observe that the model starts overfitting and reaches

a limit at about 77%. It seems that the model is unable to improve past this point. This may be due to limited data, or the model not responding well to the implementation.

Results based on number of frames as input				
Model	**Class**	**Precision**	**Recall**	**F1-score**
SlowFast warm	Card	**0.739**	**0.687**	**0.712**
	Substitution	**0.777**	**0.829**	**0.802**
	Goal	**0.842**	**0.916**	**0.878**
	Background	**0.750**	**0.701**	**0.725**
SlowFast warm	Card	0.638	0.609	0.623
	Substitution	0.693	0.744	0.718
	Goal	0.756	0.809	0.782
	Background	0.670	0.619	0.644

Table 3.17: Per class metrics for SlowFast 2+1D trained using two different learning rate schedules

Results for SlowFast 2+1D					
Model	**Avg. method**	**Precision**	**Recall**	**F1-score**	**Accuracy (%)**
SlowFast warm	Unweighted	**0.777**	**0.783**	**0.779**	**77.436**
	Weighted	**0.772**	**0.774**	**0.772**	—
SlowFast	Unweighted	0.689	0.695	0.691	68.769
	Weighted	0.686	0.688	0.686	—

Table 3.18: Average scores weighted and unweighted for SlowFast 2+1D trained with two different learning rate schedules.

In Table 3.17, we see that the results seem balanced for both cases. The use of warm restarts achieved better performance across all evaluation metrics. Table 3.18 shows us more clearly that we achieve an added 9% when we use warm restarts and save the model with the highest validation accuracy. In Section 3.7.2, we saw the best performance by non-pretrained models from MC3 with a validation accuracy of 72.821%, compared to 77.436% here. However, these results are not comparable due to different inputs in time and space. In Section 3.7.3 we show how much this can impact the results. We conclude that the learning rate schedule can significantly impact results, but that in the end, we should move forward using pretrained models.

3.7.6 Input resolution

This section investigates the effect of different input resolutions. We use 32 frame inputs due to the high GPU-memory requirement when increasing resolution. Increasing the resolution will intuitively improve finer-grained details such as the soccer ball, or body movement and human annotators may perform better. For action recognition, it is not clear to what extent the input resolution will affect results. We test each model at 224 x 224 resolution against its 112 x 112 resolution counterpart.

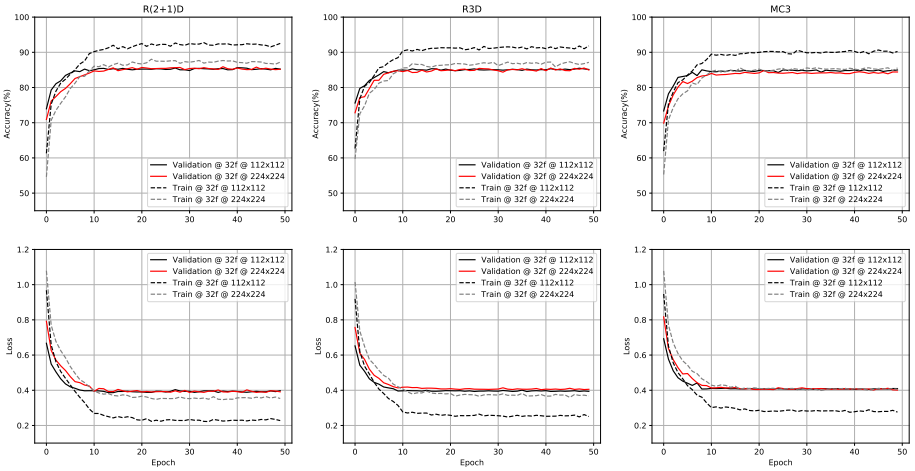


Figure 3.16: Accuracy and cross entropy loss during training

In Figure 3.16 there are similar results across all models. The models trained on higher resolution does not seem to be as prone to overfitting as models trained on a lower resolution. There is no noticeable difference in validation accuracy or loss during training, and it may be the case that with different hyperparameters during training, the higher resolution model performs better.

Results based on number of frames as input				
Model	**Class**	**Precision**	**Recall**	**F1-score**
R (2+1)D 112x112	Card	0.742	**0.874**	0.803
	Substitution	**0.888**	0.849	0.868
	Goal	**0.896**	0.947	0.921
	Background	**0.853**	0.766	0.807
R (2+1)D 224x224	Card	0.781	0.838	0.809
	Substitution	0.886	0.872	0.879
	Goal	0.872	**0.972**	0.919
	Background	0.849	0.767	0.806
R3D 112x112	Card	0.836	0.823	0.830
	Substitution	0.861	**0.915**	**0.887**
	Goal	0.894	0.947	0.920
	Background	0.838	0.772	0.804
R3D 224x224	Card	0.860	0.808	0.833
	Substitution	0.846	0.907	0.876
	Goal	0.869	0.947	**0.923**
	Background	0.834	0.788	**0.810**
MC3 112x112	Card	**0.881**	0.806	**0.842**
	Substitution	0.856	0.911	0.883
	Goal	0.865	0.919	0.891
	Background	0.819	0.788	0.803
MC3 224x224	Card	0.873	0.795	0.832
	Substitution	0.863	0.875	0.869
	Goal	0.858	0.966	0.909
	Background	0.817	**0.794**	0.805

Table 3.19: Comparison of results for 112 vs 224 resolution

Results based on number of frames as input					
Model	**Avg. method**	**Precision**	**Recall**	**F1-score**	**Accuracy (%)**
R (2+1)D 112x112	Unweighted	0.845	0.859	0.850	84.462
	Weighted	0.849	0.845	0.844	—
R (2+1)D 224x224	Unweighted	0.847	0.862	0.853	84.923
	Weighted	0.850	0.849	0.848	—
R3D 112x112	Unweighted	0.857	**0.864**	0.860	**85.538**
	Weighted	0.854	**0.855**	**0.854**	—
R3D 224x224	Unweighted	**0.860**	0.862	**0.861**	**85.538**
	Weighted	**0.855**	**0.855**	**0.854**	—
MC3 112x112	Unweighted	0.855	0.856	0.855	85.077
	Weighted	0.851	0.851	0.850	—
MC3 224x224	Unweighted	0.853	0.858	0.854	84.923
	Weighted	0.849	0.849	0.848	—

Table 3.20: Average results weighted and unweighted for 112 vs 224 resolution

In Tables 3.19 and 3.20, we can see just how close the results are. The cost of using higher resolution both in computation time and GPU-memory requirements is great. For R (2+1)D, we need about 3 times the amount of GPU-memory. This may be less prone to overfitting and worth pursuing more. However, there is a trade-off between the number of frames and resolution. We, therefore, conclude that the number of frames up to a certain point is the better choice.

3.8 Summary

In this chapter, we first describe the dataset we use and how we use it. Next, we define the different metrics that are used in order to choose the best performing models. We describe the architecture used by R3D, R (2+1)D, MC3, and SlowFast and hyperparameters that we use during training. We investigate the effect of transfer learning by testing pretrained models with and without weights based on Kinetics-400 training. We show that using gray-scale inputs has a relatively small effect on performance when compared to non-pretrained models. As one of the steps towards finding the most suitable model, we test SlowFast 2+1D using two different learning rate schedules and find that while training seems unstable and prone to overfitting, an improvement of 9% is observed when using warm restarts. To decide spatially and temporal inputs best suited for our task of event detection in soccer, we evaluate models at inputs ranging from 8 frames to 128 frames, concluding that 128 frames perform best and that R3D seemingly scales better with more frames. Another critical parameter to choose is the input resolution. We test R3D, MC3, and R (2+1)D with both 112x112 and 224x224 and a negligible difference in results. We do, however, observe that higher resolution does not seem to perform as well on training data when compared to a lower resolution, possibly indicating less chance of overfitting. We conclude, however, that due to the high cost of using a higher resolution, our best choice is the move forward with R3D using 128 frames as input at 112 x 112 spatial resolution. In the next chapter, we move forward with R3D trained on 128 frames at 112 x 112 resolution. We further test the model on the full dataset for event spotting using a sliding window approach.

Chapter 4

Results

4.1 Introduction

In Chapter 3, we experimentally found that using pretrained models performed better than training from scratch. Further, we showed that the resolution did not significantly change the results. However, the temporal extent of the input provided a significant boost to validation results, ultimately leading us to choose the model R3D with 128 frame inputs for further tests.

In this chapter, we analyze the model R3D using 128 frame inputs using multiple different approaches. We investigate the behavior of R3D for each different event by looking at the model response with a sliding window approach locally around events. To better understand what the model responds to, we investigate the features which the model used for classification. Next, we look at the results for the test set and validation set in untrimmed SoccerNet clips. Finally, we test cross-dataset generalization by using 617 clips from Norwegian Eliteserien and Swedish Allsvenskan, where 533 contained goals and 84 contained goal attempts.

4.2 Datasets

Here, we use SoccerNet [19] as in Section 3.7 to investigate our model behavior. We also evaluate how well the model performs on the full untrimmed videos in the validation set and test set. Additionally, we downloaded 617 clips from the Norwegian Eliteserien and the Swedish Allsvenskan. 317 clips from Swedish Allsvenskan, where 233 contain goals and 84 that contains goal attempts. We also have 300 clips from Norwegian Eliteserien, which contains goals. In Table 4.1, we see the distribution of goals and goal attempts for the different datasets. All clips are in the same format where the events '*Goal*', and '*Attempted Goal*' happen at roughly 25 seconds. Here, we think of a Goal as happening the moment the ball crosses the goal-line. After manually inspecting parts of the data, we see that the goal generally happens close to 25 seconds as expected, while some samples have the goal prior at 21 seconds, and some as late as at 30 seconds. Based on that, we assume that the goal lies within 20 and 30 seconds into the clip. Furthermore, we observe that the samples contain multiple replays in 25-50 seconds after the events.

Dataset	N
Goal Allsvenskan	233
Goal Eliteserien	300
Goal attempt Allsvenskan	84
Total	617

Table 4.1: Dataset statistics for Allsvenskan and Eliteserien clips

4.3 Behaviour of model R3D

This section investigates the behavior of R3D. First, we look at sliding-window and how we can use this to predict full videos. Next, we experiment to both understand how R3D reacts to temporal shifts in input as well as to help decide hyperparameters for our sliding window approach. Next, we look at how our models behave in a longer time-window. We manually check the samples for goals to look for replays after the goal. Finally, we look at the class-activation tubes (CAT), which is a weighted sum of the features before prediction. This may give us indications of which part of the input is reacted to both spatially and temporally.

4.3.1 Sliding window

To annotate events in full videos, we decide to use a sliding-window approach. The sliding-window approach means that we 'slide' our model over our inputs, hence producing local predictions. This means that our model, which takes inputs in the form of 3 x 128 x 112 x 112, predict for a given video, the frames [0,128], followed by [0+s*n,128+s*n] where s is our stride and n is the n'th step. It is not obvious what our stride should be. If we have the minimum stride at 1 frame, our predictions significantly overlap and will be computationally expensive. If our stride is too large, we will potentially 'slide' past events and fail to detect them. Furthermore, how accurate we can be with our annotations is also be limited by our choice. Therefore, we experiment to understand better how the model behaves with temporal shifts.

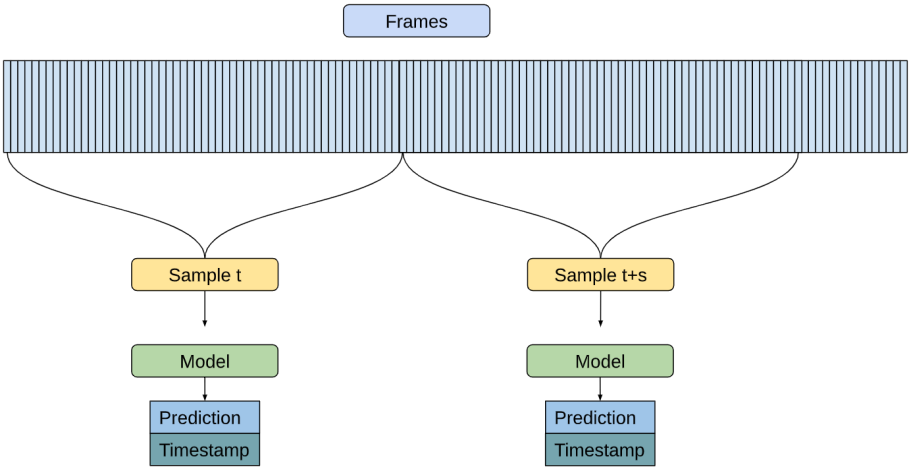


Figure 4.1: Illustration of sliding-window approach

In Figure 4.1, we see an illustration of how the model would sample frames throughout the video and pair timestamps with softmax predictions. Since we have a temporal window of 5.12 seconds for each prediction sample, we define the given timestamp as the temporal center of the sample.

4.3.2 How many frames are needed?

We are interested in finding out how our model behaves locally around events. Furthermore, we want to find the correct stride. Therefore we perform the following experiment. First, we randomly sample 8 correctly predicted event samples from the validation set. Next, we pad the input with 130 zero frames before, and after, hence, we now have a 3 x 386 x 112 x 112 tensor. Finally, we do a sliding-window approach, where we use a stride of 1 frame, densely sampling predictions. It may be that some classes have longer temporal windows of which a correct prediction is made. We, therefore, use AUC for comparison between classes.

Local behavior

We want to know:

1. How often must we sample our predictions to avoid missing events?
2. How much of an event must be seen by a model for a positive prediction?
3. Are there any differences depending on the event class?

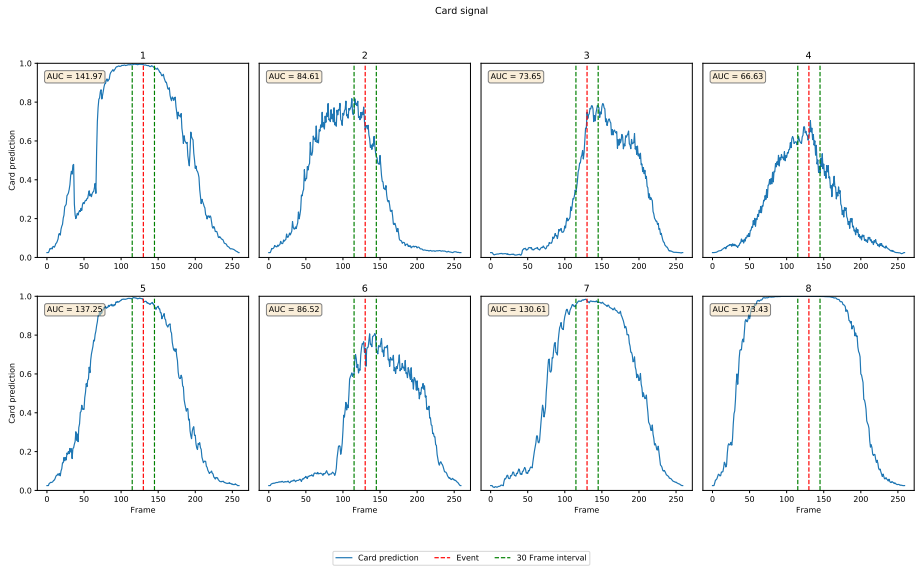


Figure 4.2: Softmax output for the event *Card* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction where

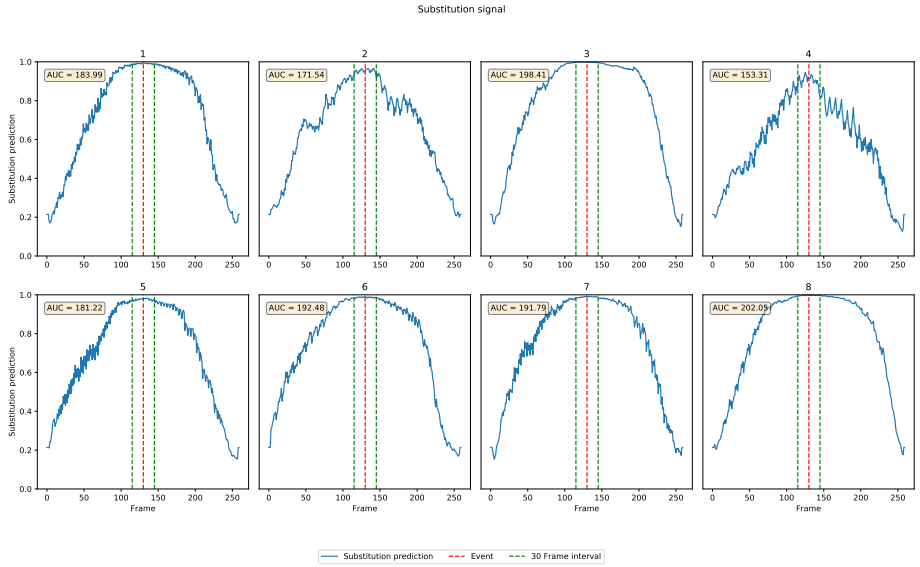


Figure 4.3: Softmax output for the event *Substitution* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction where

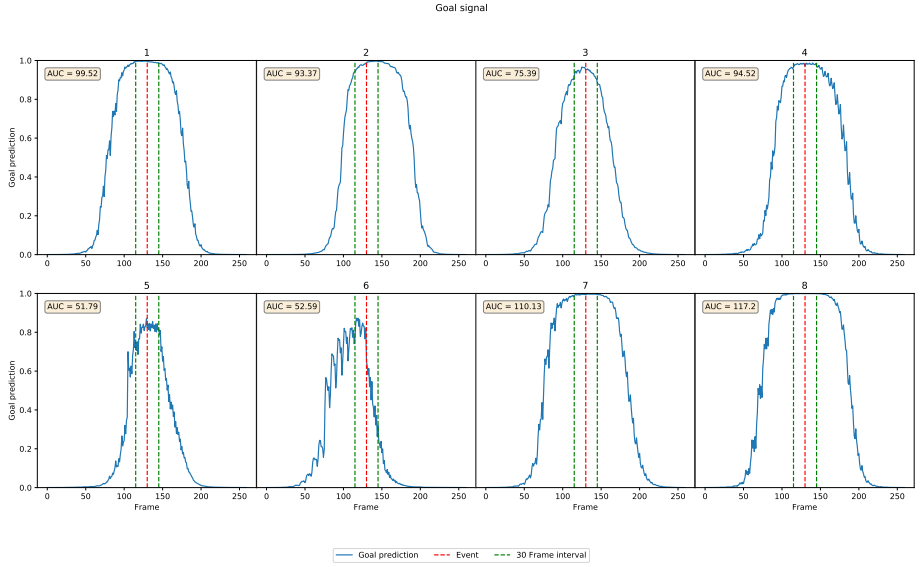


Figure 4.4: Softmax output for the event *Goal* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction for the 128 frames of the event.

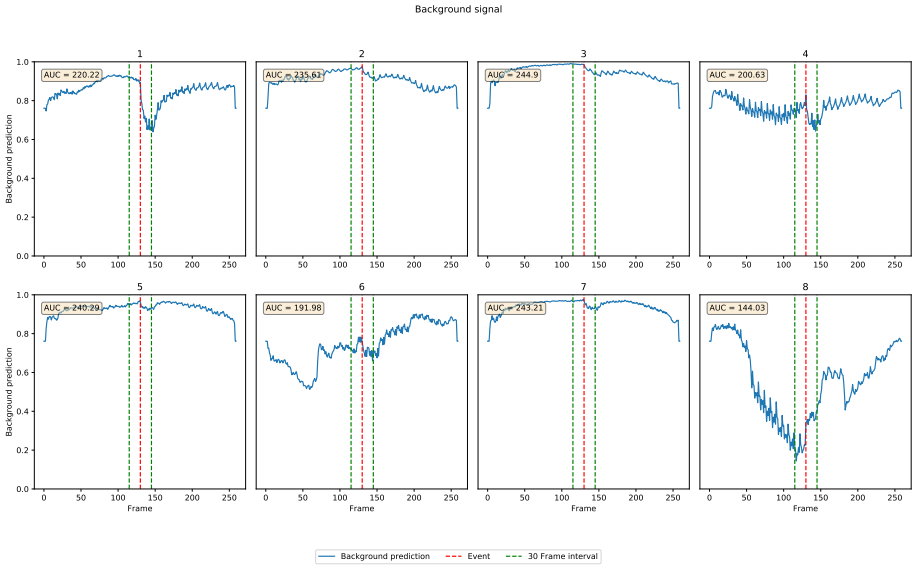


Figure 4.5: Softmax output for the event *Background* with a stride of 1 frame over 260 frames. Samples are correctly predicted validation samples. Red dotted line shows the output prediction for the 128 frames of the event.

In Figures 4.2-4.5, we visualize the softmax output from our models for each class. We have 386 frames in total, where 260 are zero-padded frames, meaning that our initial 3 predictions are between [0,128], [1,129], [2,130] where all frames are zero-padded frames. At 130, we have overlapped perfectly with our sample in the interval of [130,258]. This is indicated with a red dotted line. Since a natural stride to use for our final experiments is one second or 25 frames, we visualize a 30 frame interval around the point at which we entirely overlap with our sample. We can note that many classes can have more than 50% of its input as zero-padded frames while still prediction correctly, albeit with a smaller degree of confidence. We can see that as our model receive less zero frames and more relevant frames, the prediction gets higher. '*Goal*' generally seems to have smaller AUC than both '*Substitution*' and '*Card*' indicative that, for this class, it is more important to have the full input. For our '*Background*' class, we can see that prediction performance is reduced with more data. We can see that the model sees zero frames as '*Background*', which is essential to know since it will affect all other results. Since we use zero-padded frames that are not representative of any real case, these results are not a perfect representation of how the model reacts locally to an event. With that said, we have a controlled environment where we get some insight into how robust our model is to change in inputs. It is of particular interest that there are cases such as in Figure 4.3 where more than 50% of the input can be zero-padded frames while still resulting in high confidence. This is indicative that the inputs generate high activation values relative to the zero-frames. In conclusion, we show differences between classes, which may indicate that '*Goal*' is more reliant on particular information. It may be that the model needs to see the actual goal, which does not overlap at the beginning or end of this experiment. While we should be aware that this experiment has bias due to zero-padded frames, we argue that it still indicates

that a stride of 1 second is reasonable.

Model behaviour in larger temporal windows

To further investigate both local behavior and behavior in a more considerable temporal extent, we conduct another experiment. We repeat the process of finding samples that are correctly predicted in the validation set. We sample 25*128 frames where our event is in the center and zero-pad as in Section 4.3.2. This adds up to 3200 frames + 260 zero frames. With 25fps, this means a total of 128 + 10.4 seconds of zero frames. The temporal extent of our model is 128 frames or 5.12 seconds. With about 64 seconds before and after an event, we are interested in the following:

1. How is the event '*Goal*' affected by replays?
2. Is there noise in our signal?
3. Is there a difference between event classes?

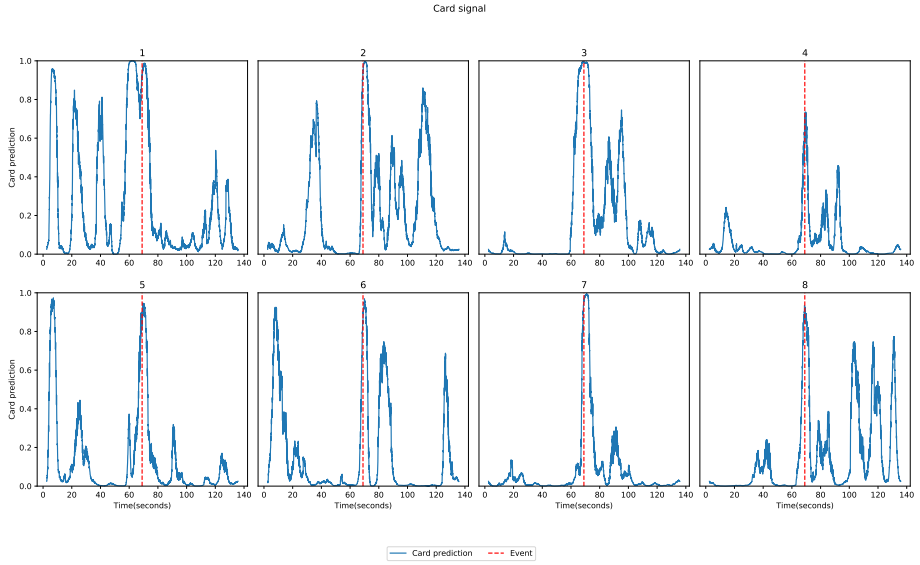


Figure 4.6: Densely sampled output prediction over 138.4 seconds around event '*Card*'. Red dotted line represents the temporal anchor of the event.

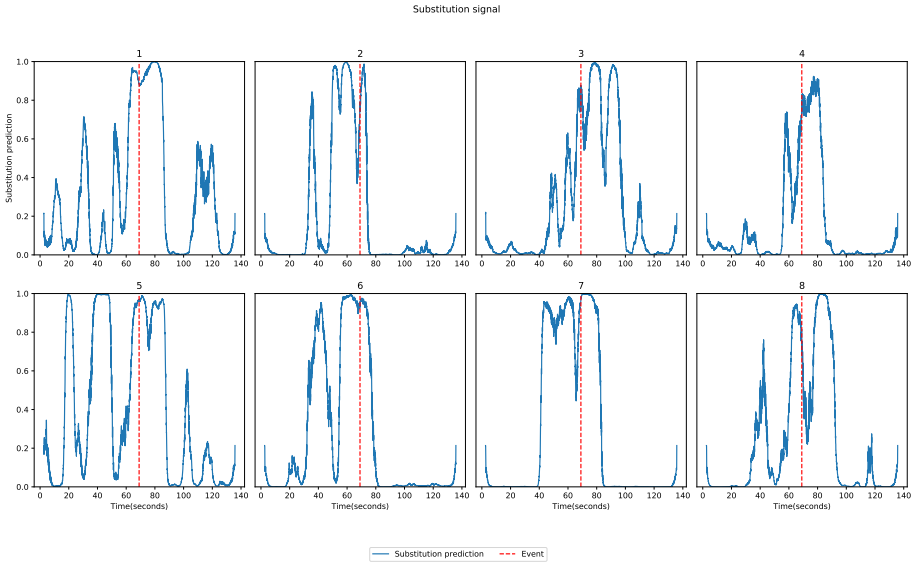


Figure 4.7: Densely sampled output prediction over 138.4 seconds around event '*Substitution*'. Red dotted line represents the temporal anchor of the event.

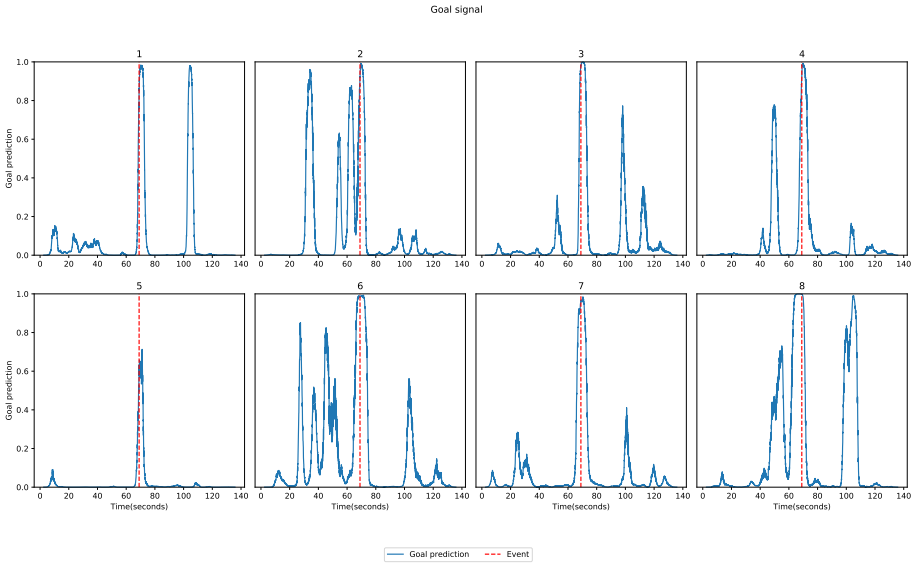


Figure 4.8: Densely sampled output prediction over 138.4 seconds around event '*Goal*'. Red dotted line represents the temporal anchor of the event.

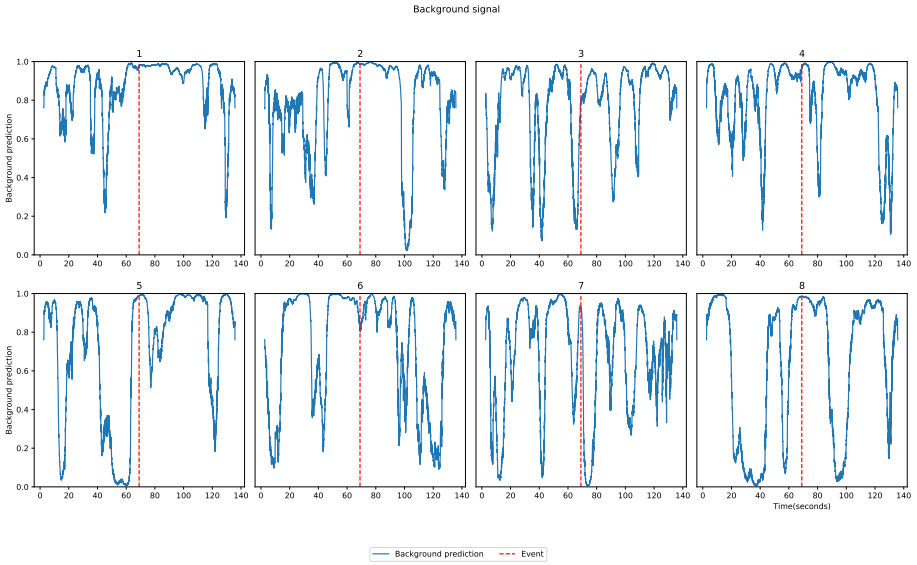


Figure 4.9: Densely sampled output prediction over 138.4 seconds around event '*Background*'. Red dotted line represents the temporal anchor of the event.

Here, we note that locally around each event, the data represents a more realistic case rather than zero-padded frames. We can see that the results are consistent with our previous experiment in that we mostly see similar peaks locally around each annotated event. Figure 4.6 shows the results for the event '*Card*', and it is clear that there is noise present. Some preprocessing steps are necessary for final predictions. We can apply a moving average filter, which would average local samples and smooth the signal. Furthermore, using a high threshold such as 0.9 or 0.95 can be used to remove the vast majority of false positives. In Figure 4.7, we can see that even a high threshold would result in many false positives. This may result in high recall and low precision for our full dataset results. The samples in Figure 4.8 contain multiple replays of the goal 30-60 seconds after the annotated events. Most of these first occurs 40 seconds after, with multiple angles shown. This seems to be reflected in some of the results, in sample number 1, there is a clear replay at 40 seconds that is clearly reflected in our results. For the background signals in Figure 4.9, the result is noisy, while mostly keeping a high response. After when looking through the sampled videos, we find that it dips consistently with changes in view and with goal attempts. For example, in sample 8, there is a continuous close-up view on coaches and benched players during 20-40 seconds. In sample 7, there is a goal attempt at about 40 seconds. This leads us to believe that what the model has learned to recognize as '*Background*' are particular common views and that the model is not necessarily robust to certain parts of the field. The consequence of this would likely be many false positives.

4.3.3 Class activation tubes

What exactly did our model learn during training? To gain insight into what our model reacts to temporally and spatially, we inspect the class activation

maps (CAM) for correct and wrong predictions. We follow the methodology in Zhou et al. [59] in order to generate CAMs. The model R3D uses global average pooling before an FC layer after 17 convolutional layers, as explained in Section 3.5.

A tensor with shape C x T x H x W goes into the global average pool layer, resulting in C in a C x 1 x 1 x 1 output. For R3D with 128 frames specifically, we have a 512 x 16 x 7 x 7 tensor that is reduced to 512 x 1 x 1 x 1. These are the 512 features that are used to compute the final scores, followed by softmax. Let F_i denote the i'th feature channel for i $\in \{1, 2, 3, ..., N\}$. Then S_c is the pre-softmax class score, computed as a weighted sum the following way:

$$S_c = B_c + \sum_{i=1}^{N} w_i^c F_i \tag{4.1}$$

Where B_c is the bias term, and c denotes our four different classes '*Card*', '*Substitution*','*Goal*' and '*Background*'.

F_i is calculated as follows:

$$F_i = \frac{1}{K} \sum_{t,x,y} V_i(t, x, y) \tag{4.2}$$

V_i holds our N feature volumes with K elements each, which in our case is 16 * 7 * 7 = 784, where t denotes our temporal dimension, and x,y our spatial dimensions. Thereby:

$$\begin{aligned} S_c &= B_c + \sum_{i=1}^{N} w_i^c \frac{1}{K} \sum_{t,x,y} V_i(t, x, y) \\ &= B_c + \frac{1}{K} \sum_{t,x,y} \sum_{i=1}^{N} w_i^c V_i(t, x, y) \end{aligned} \tag{4.3}$$

In Equation 4.3, we see the relationship between the pre-average pool feature volumes and the weights and bias used to compute the class scores S_c. As in Zhou et al. [59], we ignore the bias term moving forward as it has little impact on the final results. We define a class feature tube (CAT) as a 3 dimensional equivalent of class activation maps as follows:

$$T_c(t, x, y) = \sum_{i=1}^{N} w_i^c V_i(t, x, y) \tag{4.4}$$

Going from Equation 4.4 to our class score S_c, we need to calculate the average over all elements and add the bias term. Since this is the case, we may find useful information both temporally and spatially that helps us gain insight into what our model reacts to.

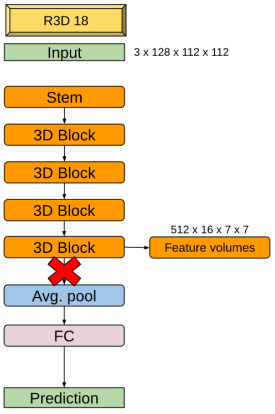


Figure 4.10: Illustration of feature volumes V_i

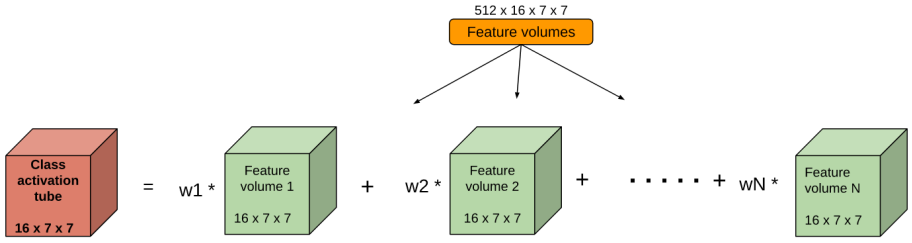


Figure 4.11: Illustration of calculation of class activation tubes

Figure 4.10 illustrates where we get our feature volumes, and Figure 4.11 shows how we weight our feature volumes to generate our CAT's. We use CAT's to understand spatio-temporal features by considering the spatial information at time t. In order to get a comparison to our input, we interpolate our 7 x 7 maps in $\{T_c(t=0,x,y), T_c(t=1,x,y)...T_c(t=T,x,y)\}$ using bicubic interpolation. We also use the CAT's to compute temporal signals to indicate where in time, our model reacts. This is achieved by average pooling T_c across spatial dimensions, resulting in a 1-dimensional signal. Due to the spatial relationship of for the spatio-temporal class activation features at time t, we refer to them as *class activation maps*. Furthermore, we refer to the 1-d the temporal signal as '*class activation signal*'.

We investigate each class with the following method.

Sample collection

To analyze the results, we decide to use samples from the validation set used during training. We evaluate both correctly predicted samples and wrongly predicted samples. We are interested in results from a single 128 frame input as well as the results as we get close to the annotated point in time. Therefore we first randomly sample 4*128 frames from the validation set where the event lies in the center at frame 256. Next, we get a subset containing correctly predicted

samples by classifying locally in the frame interval [192,320] for each class. The interval [192,320] contain the 128 frames that we used during evaluation in Chapter 3.

Wrongly predicted samples While analysis of what the model reacts to with correctly predicted samples, we might learn more by looking at wrongly predicted samples. What went wrong? Are there obvious reasons as to why a sample was misclassified? What can we learn from this? These are some questions that we have when analyzing samples that misclassified.

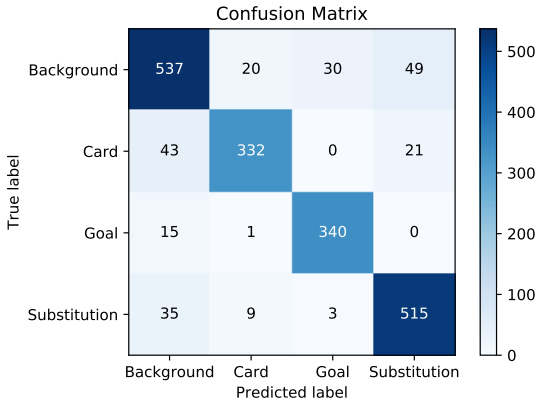


Figure 4.12: Confusion matrix for validation results using R3D with 128 frames

Figure 4.12 shows the confusion matrix for the validation results from Chapter 3 on model R3D with 128 frames. We can see in the confusion matrix that for the events '*Card*','*Goal*' and '*Substitution*', most bad cases end with a wrong prediction for the event '*Background*'. We also look at the cases where '*Background*' is wrongly predicted as one of the other three classes. The event '*Background*' was generated automatically through a method described in Section 3.2, and may contain replays of goals or other events which we may see here.

Dense CAMs

We use both positive and negative samples and visualize the spatial information within T_c. For each input, we sample frames with a stride of 8, starting at the fourth frame. This makes for a total of 16 frames, corresponding to the number of CAMs present in T_c when using 128 frame inputs. We compare these visually, side by side. This may grant insight into the decision process of our model.

Local temporal signal

Since we use a sliding window approach with a temporal window size of 5.12 seconds, it is of particular interest to see how our class activation signals are before and just after an event has occurred. Our experiment looks at how our signal changes by a sliding window approach with a stride of 16. We plot the results in order, with information on exactly where the event occurs, and what the center frame of the current signal is.

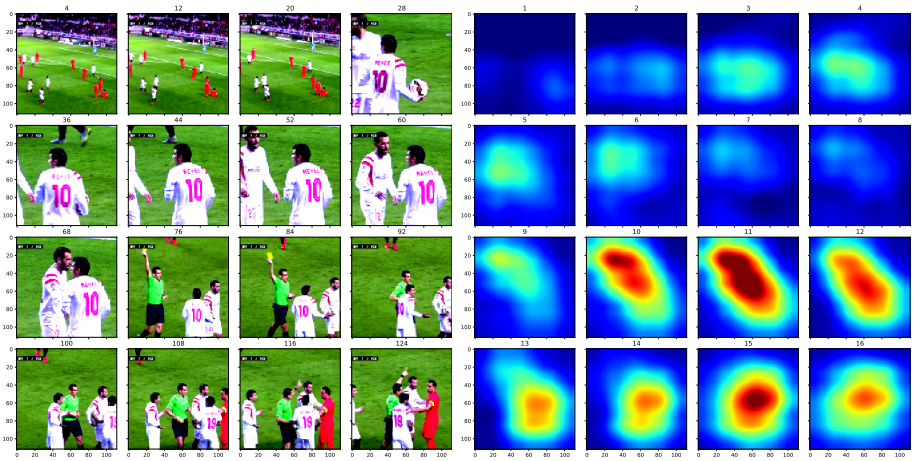
Temporal signal for large inputs

What happens when we increase our inputs to our model? Moreover, how is the class activation signal locally during an event? By zero-padding our 512frames both sides with 512, making for a total size of 512 + 512 * 2 = 1536 frames, we experiment with 512 sized inputs to our model and use a sliding window approach with a stride of 64. We plot our results in order where the annotated event is perfectly centered in the interval of [512,1024] at frame 768.

Sparse CAMs

To supplement local temporal signal and temporal signal for large inputs, we try to get insight into the spatial aspects as well. We slide our model across each sample with inputs of 128 frames, using a stride of 16 frames. For each location, we save the middle input image and the corresponding middle CAM at $T_c(8, x, y)$.

Results for event '*Card*'

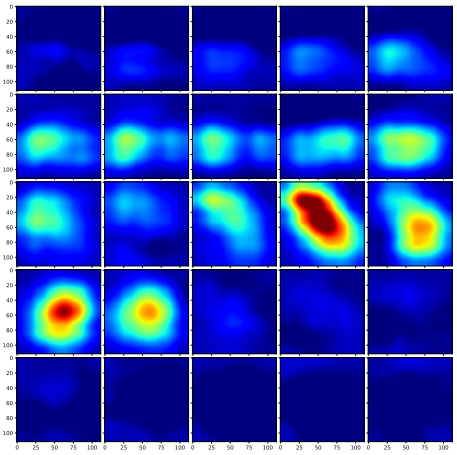


(a) Frames (b) Class activation maps

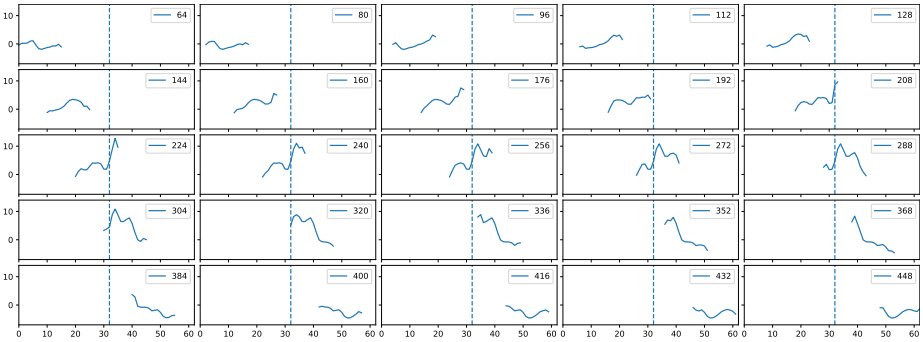
Figure 4.13: a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{card}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$

In Figure 4.13, we see different view angles with the latter half containing a referee showing the card and arguing with players. At corresponding CAMs in b) 10 and 11, we see strong activations when the card is shown. In b) 15, we also see a reaction, looking at the corresponding estimated input frame, it can seem like the referee is in the process of pointing somewhere. This may indicate that handwaving, in general, is something the model reacts to, perhaps with the combined information of the referee's unique clothing. While we cannot make hard conclusions based on this, the model seemingly reacts to reasonable information in this particular case.

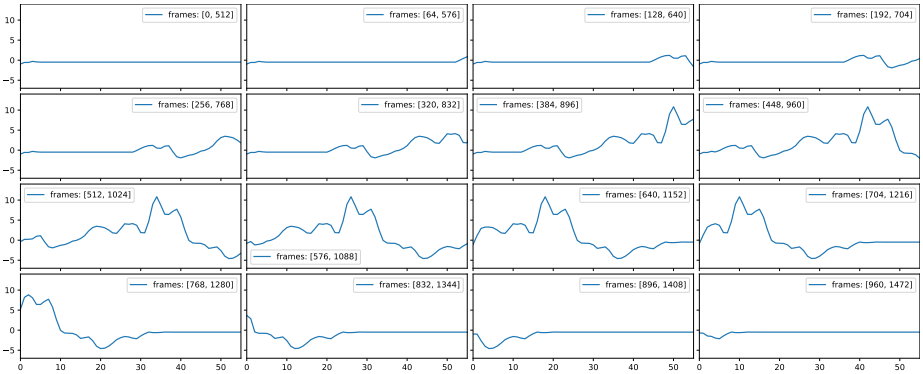
(a) Center frames



(b) Center CAM



(c) Class activation signal



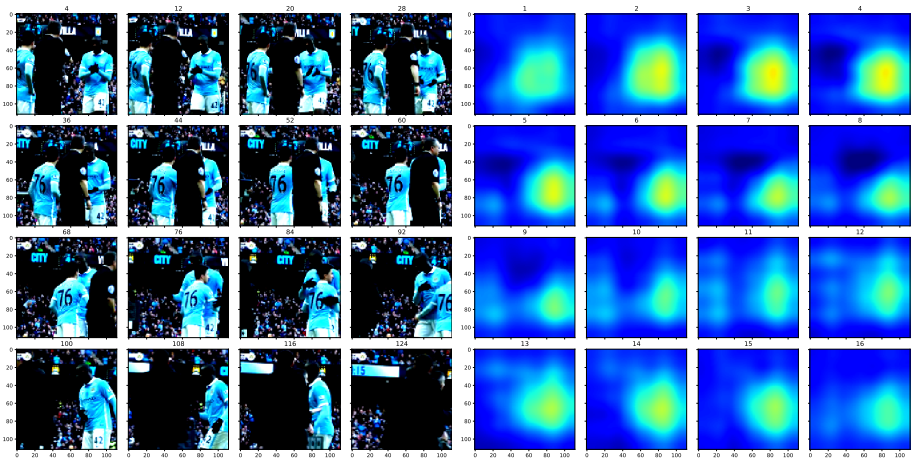
(d) Class activation signal with 512 frame input

Figure 4.14: Results from CAT's spatio-temporally and temporally for the event '*Card*'. a) Center input frame between interval [s*n,28+s*n] of 512 frame sample of event for the n'th sample. b) Corresponding center CAM $T_{card}(8, x, y)$ c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal for zero-padded 512 frame inputs at 64 frame stride.

In Figure 4.14a and Figure 4.14b, we see sparse CAMs calculated from 512-frame inputs, where we use a stride of 16, and the first frame is the 64th frame. Images and CAMs are in order, read row by row, from left to right. The sparse CAMs are upsampled to 112x112 using bicubic interpolation. We can note roughly how strong the activations are before and after the annotated event itself in the sparse CAMs. We see some activations early, but it does not react strongly until we see the referee. In Figure 4.14c, we see the class activation signal as it approaches the event. Here, the x-axis represents the corresponding time in feature space as the input over the full 512 frames, while the placement of the signal shows where in time it has been calculated. Particularly interesting is the position of the maximum value of the signal as it slides across the event as we see that the reactions roughly match the time of the event. This property could perhaps be used directly for temporally accurate predictions by mapping the position back to input space, granting an estimated point within our 5.12-second window.

In Figure 4.18d we see the class activation signal for 512-frame inputs. The original 512 frames have been zero-padded with additional 512 frames on each side, hence the larger temporal extent of the signal. Here, we use a stride of 64. We observe that the class activation signal has low responses as expected during the zero-padded frames. The signal seems equivariant in that the class activation signal shifts corresponding to our shift in input, locally remaining similar in the activated areas. This property could be used for temporal region proposal for a given class by taking maximum value positions after using temporally large inputs.

Results for event '*Substitution*'



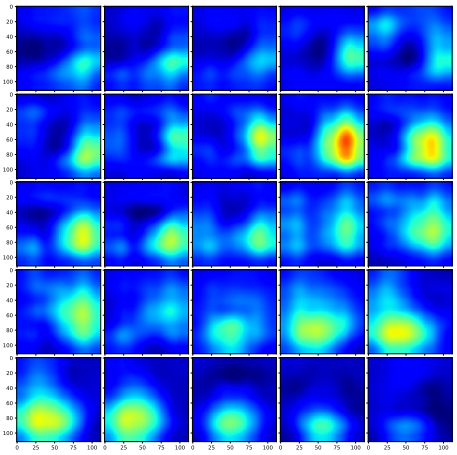
(a) Frames (b) Class activation maps

Figure 4.15: a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{substitution}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$

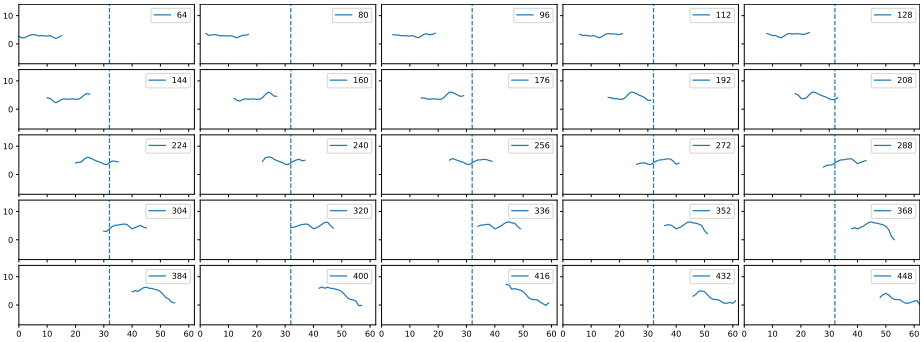
In Figure 4.15, there seems to be a continuous reaction to the players. It

is not very easy to precisely say what the response comes from. Possibly, the reaction is based on close up shots of players. The audience here seems to cause a locally low response to the class. While this is speculative, the results may indicate that we should expect false positives for close up shots of players.

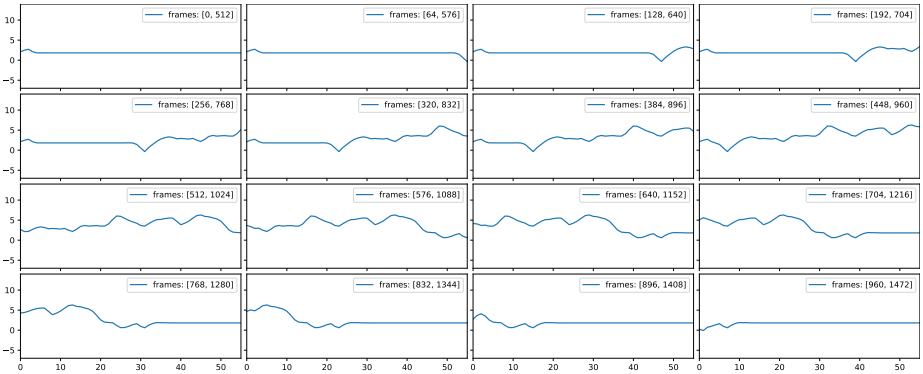
(a) Center frames



(b) Center class activation map



(c) Class activation signal

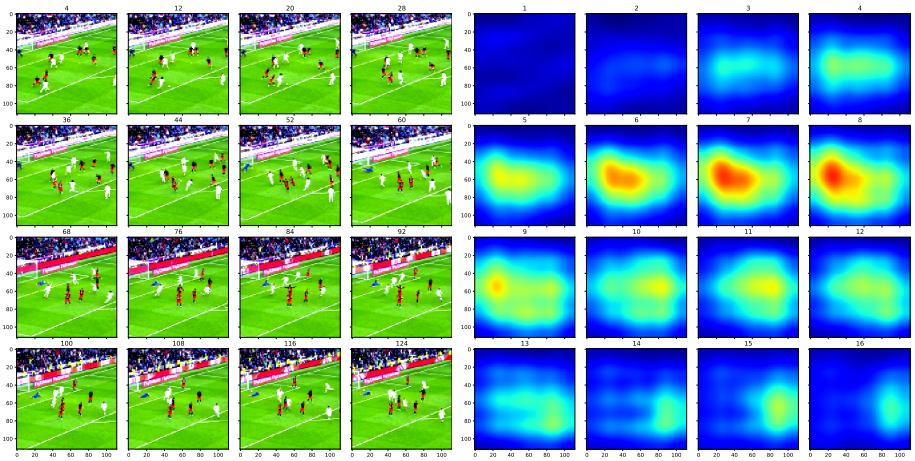


(d) Class activation signal with 512 frame input

Figure 4.16: Results from CAT's spatio-temporally and temporally for event '*Substitution*'. a) Center input frame between interval [s*n,128+s*n] of 512 frame sample for the n'th sample. b) Corresponding center CAM $T_{substitution}(8, x, y)$ c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal for zero-padded 512 frame inputs at 64 frame stride..

In Figures 4.16a-4.16b, we can observe much of the same. An interesting point is the last row in a) where we see responses dropping slowly as the player turns around. Looking at the local temporal signal in Figure 4.16c, we mostly see a relatively flat response. If we were to use this information to try to annotate the event more accurately, it could prove difficult, indicating that this approach might not work well with this class. The temporal signal for large inputs in Figure 4.14d leads us to the same conclusion. The model seems to have been unsuccessful in capturing the exact moment.

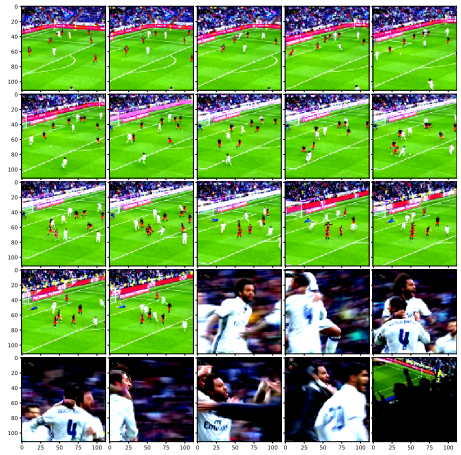
Results for event '*Goal*'



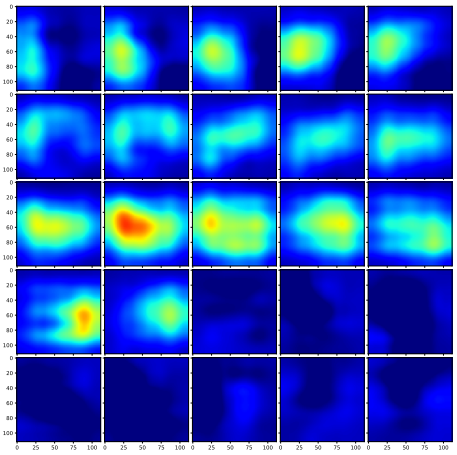
(a) Frames (b) Class activation maps

Figure 4.17: a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{goal}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$

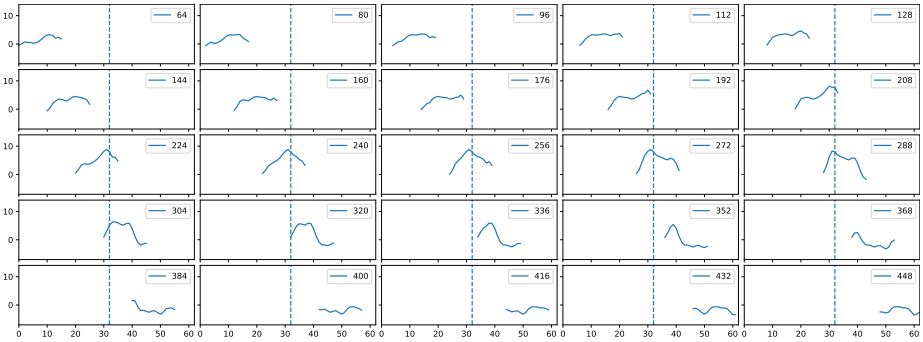
Figures 4.17a-4.17b show our dense CAMs for the event '*Goal*'. The goal occurs close to frame 60. We see strong reactions around that point, with a focus on the left side. We hope that the model reacts when the goal crosses the goal line. We should note, however, that while the '*Background*' class may contain some goal attempts, the model is not explicitly trained to separate goal attempts and actual goals.



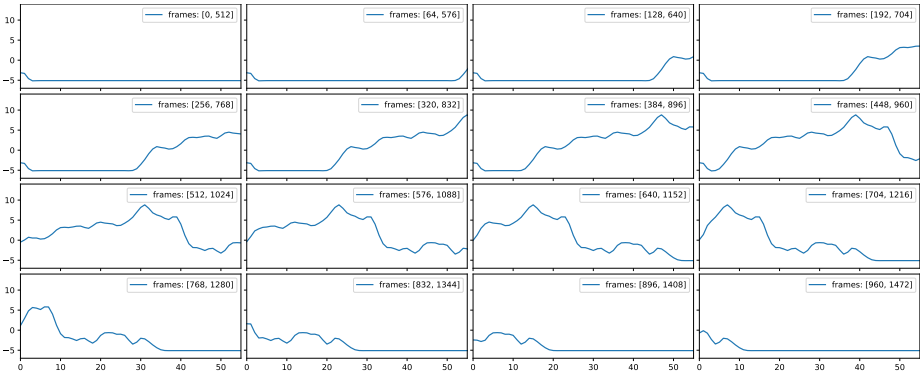
(a) Center frame



(b) Center class activation map



(c) Class activation signal

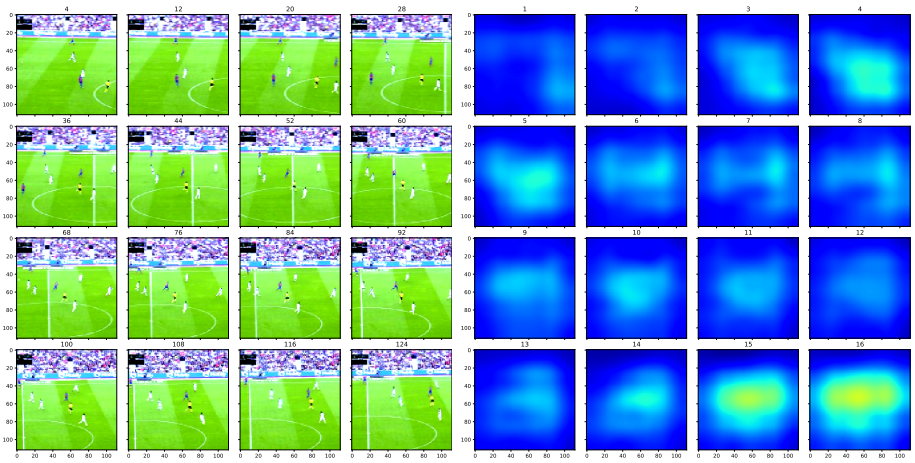


(d) Class activation signal with 512 frame input

Figure 4.18: Results from CAT's spatio-temporally and temporally for the event '*Goal*'. a) Center input frame between interval [s*n,128+s*n] of 512 frame sample for the n'th sample. b) Corresponding center CAM $T_{goal}(8, x, y)$. c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal signal calculated with 512 frame inputs at 64 frame stride.

In Figures 4.18a-4.18b, we see our sparse CAMs, showing some response long before the actual goal. As the view angle is switched to celebration by the players, we see that there is a weak response. The class activation signal in Figure 4.18c are interesting. It seems that the maximum value correctly follows the actual goal. In this particular case, using this information to set the temporal anchor accordingly would result in a more accurate estimate prediction in time. Looking at the temporal signal for large inputs in Figure 4.18d further shows that we get sharp peaks at the right moment.

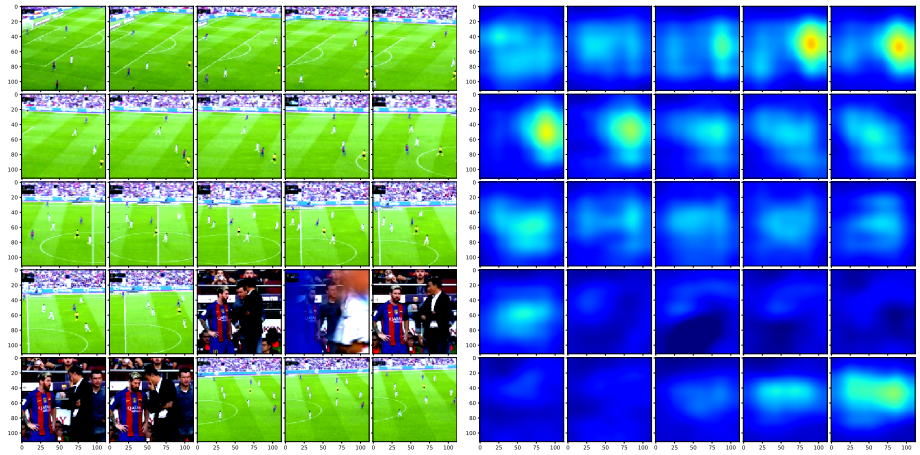
Results for event '*Background*'



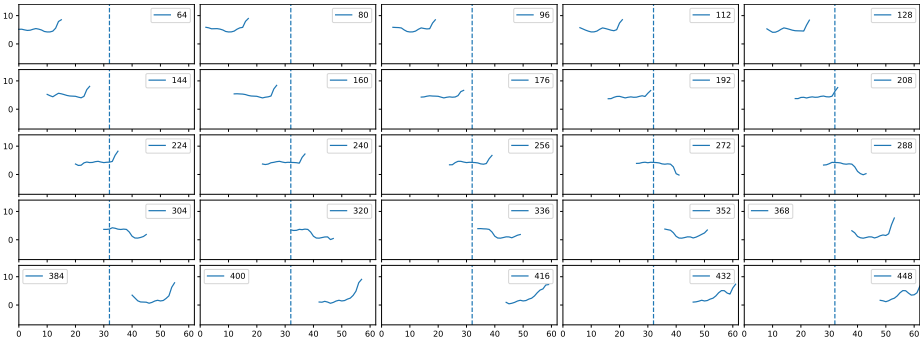
(a) Frames (b) Class activation maps

Figure 4.19: a) Frames during 128 frame input with a stride of 8, starting at the fourth frame. b) Corresponding class activation maps, $T_{background}(t, x, y)$ visualized for $t \in \{1, 2, 3, ..., 16\}$

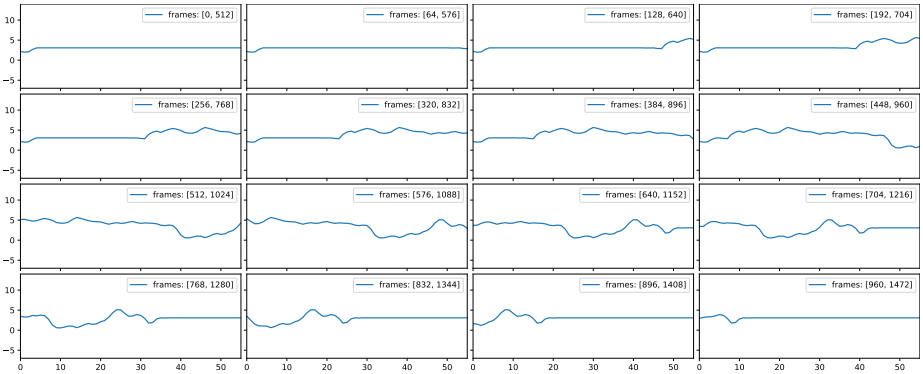
The background class is special. Due to the imbalanced relationship in soccer videos between specific events and background video, we use this class as a means to stop false positives. Since it has been automatically annotated with somewhat unpredictable content, it can be difficult to infer what the model learns and reacts to. Figure 4.19 shows a relatively consistent response except for the last few CAMs. The frames we see in Figure 4.19a is a common view in soccer, and it would not be surprising if the majority of the background samples hold similar views. Intuitively, it also makes sense that this is classified as background since nothing special in particular seems to happen in this case.



(a) Center frame (b) Center class activation map



(c) Class activation signal



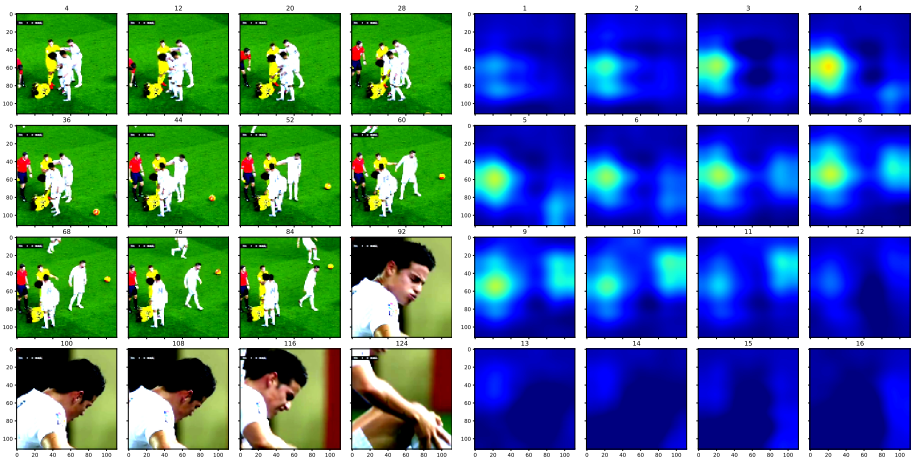
(d) Class activation signal with 512 frame input

Figure 4.20: Results from CAT's spatio-temporally and temporally for the event '*Background*. a) Center input frame between interval [s*n,128+s*n] of 512 frame sample of event '*Background*' for the n'th sample. b) Corresponding center class activation maps $T_{background}(8, x, y)$ c) Class activation signal. X-axis indicates the full 512 frame interval in feature space d) Temporal class activation signal signal calculated with 512 frame inputs at 64 frame stride.

When we look at the results in Figures 4.20a-4.20b we can observe that the model response is relatively even from a certain angle. However, the response is much lower when the view is switched to appears to be a player-coach discussion. Semantically, this part is similar to a substitution, which may be the reason for the lowered response. From the class activation signals and temporal signals for large inputs in Figures 4.20c-4.20d we see that with the exception of the player-coach discussion, the signal response is relatively flat. Indeed, our intention is not to annotate 'Background' a specific point in time, large spikes or high variance here would prove detrimental to our final results. We speculate that the model may have learned certain view angles as the background class and that it may be fooled if the broadcast changes to different views such as closeup views of players.

In order to further understand the model, we look at dense CAMs for samples where the model incorrectly predicted the wrong event. For events '*Card*','*Substitution*', and '*Goal*', we look at the input frames and the corresponding CAMs for the correct class, which means that we can look for what the model *does not* react to but should. For the special class '*Background*', we look at the input frames, along with the CAMs corresponding to the class that it incorrectly predicted. Thus, meaning that we can attempt to see what the model incorrectly reacted to. This may provide valuable insight that could help us in multiple ways. We could find contextual similarities in the data or uncommon views. The results may, for example, be used to target the annotation of new data specifically.

Events misclassified as background

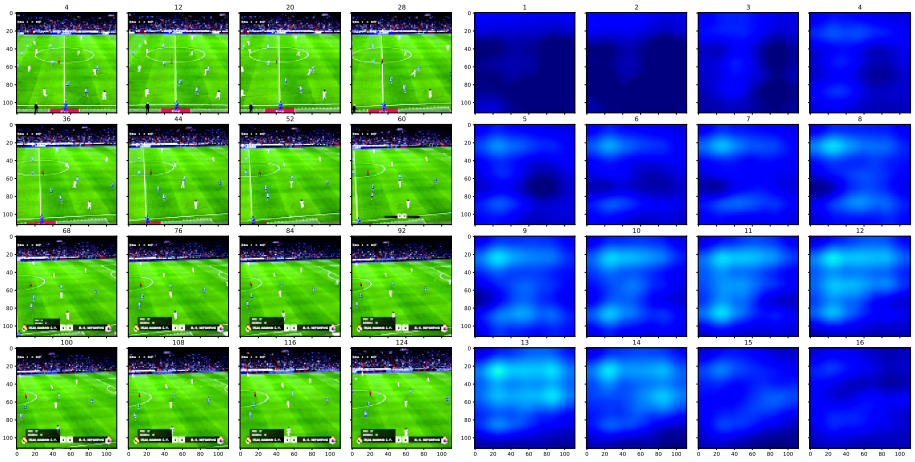


(a) Frames (b) Class activation maps

Figure 4.21: Model mistakes event '*Card*' as event '*Background*', figure shows samples from input clip alongside CAMs for event '*Card*'.

In Figure 4.21a, we do not see the referee hold the card up. What we observe is that there seem to be some soft responses to the spatial location of the referee. This may be an annotation that is slightly off, or that the card was given during the change of view. Another cause may be the cropping that is

used during data preprocessing. Looking at frames 76 and 84, we do not have a good view of the right arm of the referee. While we cannot make conclusions here, it may give cause to reevaluate the choice of center cropping from 112 x 199 resolution, perhaps a more generous width or a different choice for resizing should be considered.



(a) Frames (b) Class activation maps

Figure 4.22: Model mistakes event '*Substitution*' as event '*Background*', figure shows samples from input clip alongside features for event '*Substitution*'.

In Figure 4.22, there seems to be a long-distance view of a substitution, the scene looks similar to what expect most background samples to look like. Intuitively it makes sense for this sample to be misclassified considering the sample frames. This shows some limitations, as well. The model does not work correctly in some instances where we do not have the expected information within the inputs.

(a) Frames (b) Class activation maps

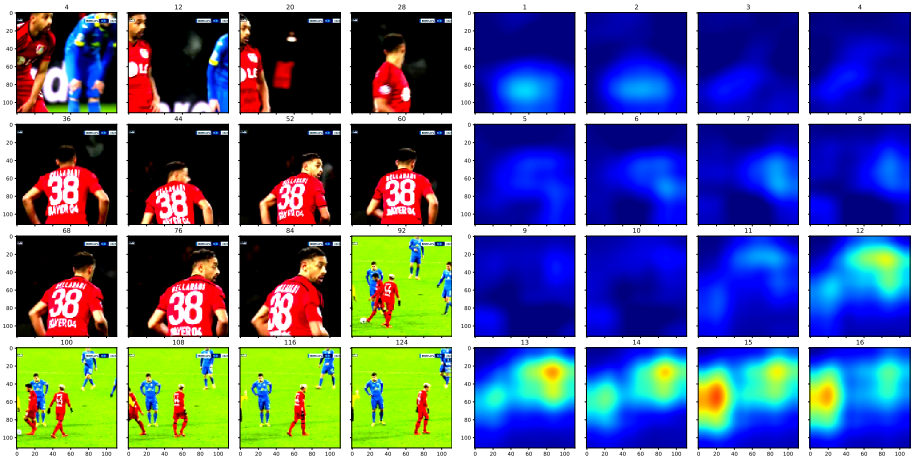
Figure 4.23: Model mistakes event '*Goal*' as event '*Background*', figure shows samples from input clip alongside features for event '*Goal*'.

Looking at the results in Figure 4.23, we see that the model hardly reacts. The model incorrectly classified the sample as '*Background*'. We take note that the goal seems to be a penalty, where the camera angle is somewhat unusual. Based on this, we speculate that there are not enough different camera views

for goals within the dataset, which would make it difficult for the model to be invariant to a wide range of camera angles. Another factor might also be that much of the scene is filled with audience members, which might be associated with background samples.

Background misclassified

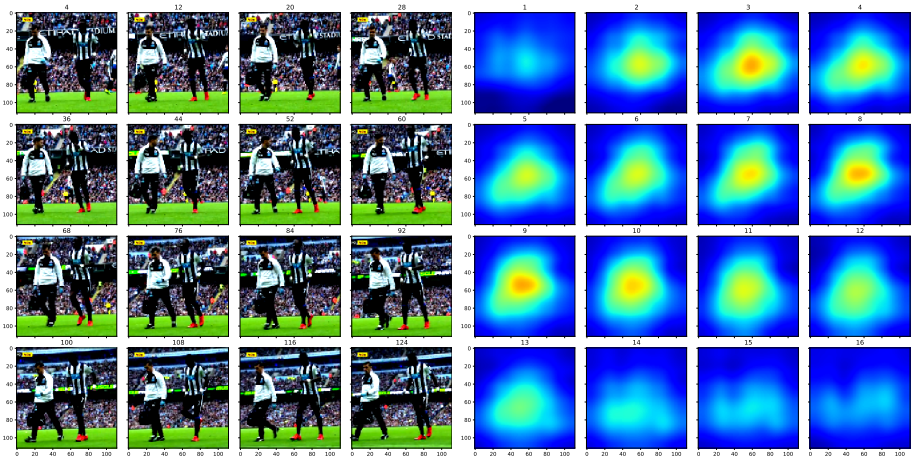
Considering the nature of our background samples as automatically annotated, we expect a variety of different scenes within the dataset. We briefly investigate background samples that were wrongly classified as another event. We pair the CAMs with the input frames to see if we can better understand what the model reacted to in this case.



(a) Frames (b) Class activation maps

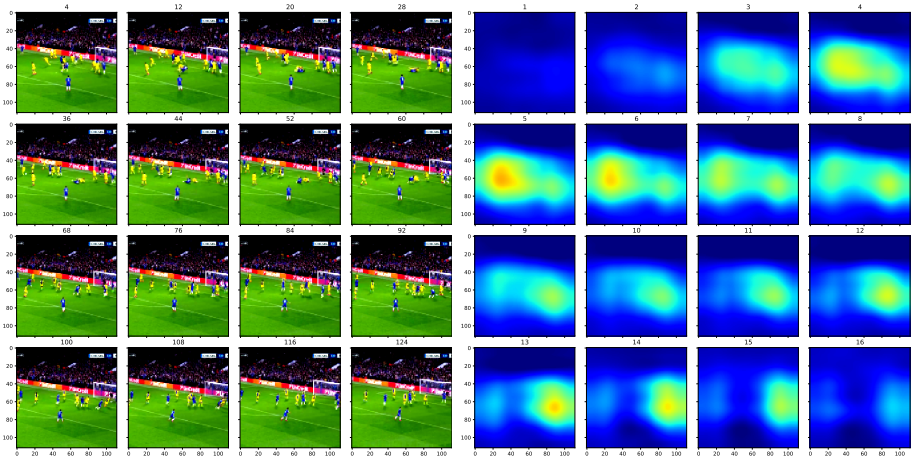
Figure 4.24: Model mistakes event '*Background*' as event '*Card*', figure shows samples from input clip alongside features for event '*Card*'.

We can see that, for the most part, there is little response in Figure 4.24. However, through frames 92-124, there are some reactions to the left side. Looking closely at the input frames, we can see a player standing still with a barely visible referee pointing. It may be that this is the reason, or it may be the player's stance.



(a) Frames (b) Class activation maps

Figure 4.25: Model mistakes event '*Background*' as event '*Substitution*', figure shows samples from input clip alongside features for event '*Substitution*'.



(a) Franes (b) Class activation maps

Figure 4.26: Model mistakes event '*Background*' as event '*Goal*', figure shows samples from input clip alongside features for event '*Goal*'.

In Figure 4.25, we see what appears to be a hurt player leaving the field. Interestingly, this is semantically similar to a substitution. Figure 4.26 seems to be a goal attempt triggering a false positive. This is interesting and increases our expectation of false positives around goal attempts.

Summary

In Section 4.3, we investigated model behavior to understand strengths and weaknesses better. Section 4.3.2 showed us that model response is different depending on the class, and that we could have as much as 50% zero-padded

frames as input while still get correct predictions. Furthermore, these results indicated that a stride of 1 second is reasonable. We further investigated behavior for larger temporal windows in Section 4.3.2, where we found that our model produces noisy signals and seemingly reacts in multiple cases to contextually similar scenes such as replays. To gain some insight into the spatio-temporal features used for model prediction, we looked at both class activation maps and class activation signals in Section 4.3.3. We find that while it remains unclear whether the model learns the correct features, the results indicated that contextually meaningful information results in a strong response. The class activation signals showed varying results depending on the class. For classes '*Goal*' and '*Card*', the result showed potentially useful information for increasing accuracy within the 5.12-second temporal window of the model. Further, we found that misclassified samples seemed to be caused by different factors such as rare camera angles, which could be resolved with more data. Another factor might be bad cropping, which could be fixed by changing preprocessing steps.

4.4 Full video detection

To evaluate the model in a practical scenario, we use both test and validation set from SoccerNet [19], which is discussed in detail in Section 3.2. With 200 untrimmed clips, each holding about 45 minutes each, the events are sparse. We first slide our model R3D with 128 frame inputs at a stride of 1 s with a sliding window approach, which results in about 550 thousand raw predictions for the validation and test set each. Evaluating the 200 games took about 30 hours to complete, roughly 9 minutes per game. Next, we pre-process each signal separately and evaluate the results with respect to spotting tolerance δ.

4.4.1 Processing output prediction

Moving average filter

One tactic in action recognition during testing is to sample multiple clips and average the predictions during testing [8, 47] for full video prediction. In our case with 128 frame input, 5.12 seconds, the model process the same frames multiple times due to our 1-second stride. We, therefore, apply a small moving average filter of size 3, with a kernel containing [1/3, 1/3, 1/3]. The goal is too smooth out the signal slightly to remove noise.

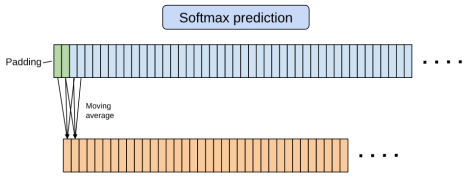


Figure 4.27: Illustration of moving average filter

Figure 4.27 illustrates how we apply the moving average filter. Here, we use symmetric padding at the beginning and end of the input.

Thresholding prediction

We are predicting at every second for 400 videos making for roughly 1080000 seconds. As seen in Section 4.3, we get noisy signals. In our validation set and test set, we have 1314 and 1358 samples, respectively, which together represent about 0.25% of our predictions made. To remove most of the noise, we threshold the signals with a high threshold T.

Removing duplicates

To remove temporally close and high predictions, we use a window of size 10, where we look for the time of the highest local prediction for a given class. This effectively means that there will be no predictions within 5 seconds of each other.

4.4.2 Spotting results

For spotting, we look at each class separative as a one-vs-all binary problem and consider a positive prediction as a true positive if it is within a tolerance δ of the ground truth event with higher confidence than our threshold. Formally, we use the following condition:

$$|gt_{spot} - p_{spot}| < \delta \tag{4.5}$$

Where gt_{spot} is a ground truth spot, and p_{spot} a predicted spot in seconds. For predictions where this is false, we consider it a false positive. If, for a given gt_{spot}, there are no predictions made where this condition holds, we consider it a false negative.

We look at recall, precision, and F1 scores over different thresholds for low tolerance of 2.5 seconds, consistent with our 5.12-second temporal window of our model R3D. First, we visually inspect some results.

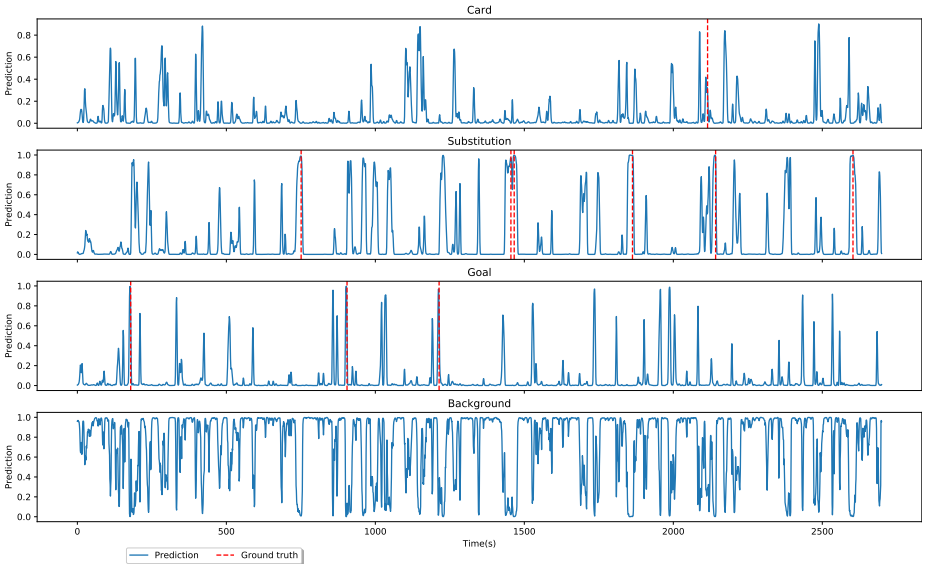


Figure 4.28: Softmax confidence for each class over 45-minutes with ground truth.

In Figure 4.29, we see the softmax confidence for each class separately for a soccer half-game of 45 minutes. The background signal dominates most of the time. However, the signals are noisy and include multiple high responses at the wrong time.

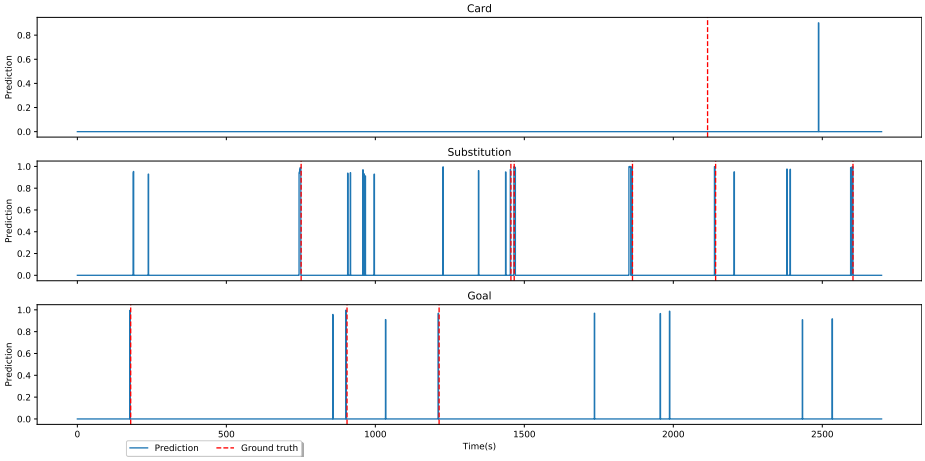
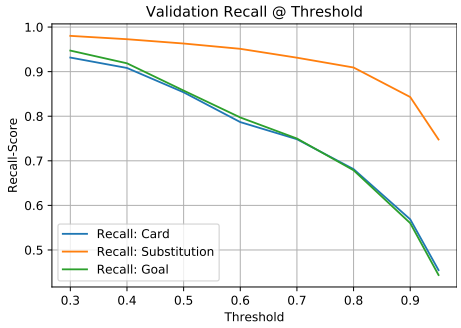
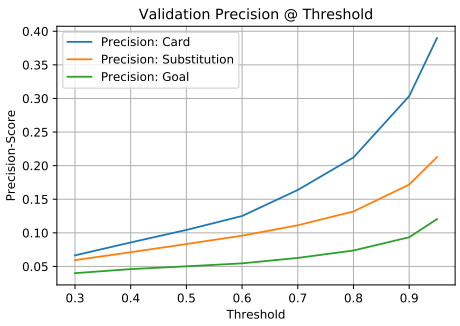


Figure 4.29: Prediction for each second after mean filtering, after thresholding. Red dotted line shows ground truth for each class.

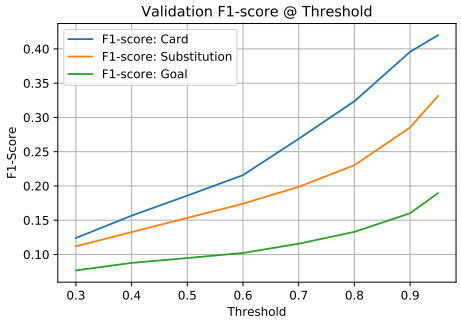
Figure 4.29 shows our final predictions. After applying a threshold of 0.9, we see that we have removed most of the noise. For the event '*Card*', we get a false positive long after the event itself. Looking at both 'Substitution' and 'Goal', we see that we get reasonable predictions close to the ground truth. However, we also end up with multiple false positives that are entirely unrelated.



(a) Recall

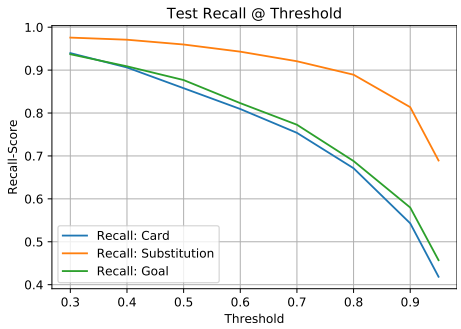


(b) Precision

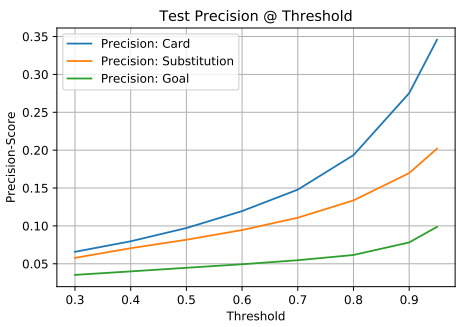


(c) F1-score

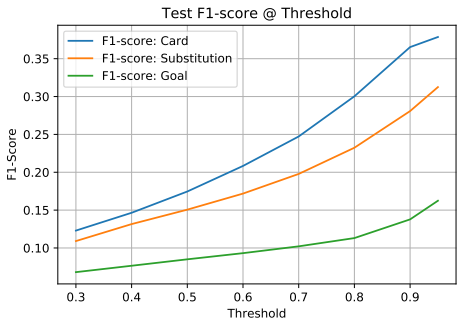
Figure 4.30: Recall, precision and F1-score of validation set over different thresholds using a tolerance $\delta = 2.5s$



(a) Recall



(b) Precision



(c) F1-score

Figure 4.31: Recall, precision and F1-score of test set over different thresholds using a tolerance $\delta = 2.5s$

Results for tolerance 2.5 on test set								
Threshold	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.95
mAp	0.21	0.22	0.22	0.22	0.23	0.24	0.26	0.3
Card AP	0.32	0.33	0.33	0.33	0.33	0.35	0.41	0.48
Substitution AP	0.23	0.24	0.24	0.25	0.25	0.26	0.26	0.28
Goal AP	0.09	0.10	0.10	0.10	0.10	0.11	0.12	0.15
Card Precision	0.07	0.08	0.10	0.12	0.15	0.19	0.28	0.35
Substitution Precision	0.06	0.07	0.08	0.09	0.11	0.13	0.17	0.20
Goal Precision	0.04	0.04	0.04	0.05	0.05	0.06	0.08	0.10
Card Recall	0.94	0.91	0.86	0.81	0.75	0.67	0.54	0.42
Substitution Recall	0.98	0.97	0.96	0.94	0.92	0.89	0.81	0.69
Goal Recall	0.94	0.91	0.88	0.82	0.77	0.69	0.58	0.46
Card F1	0.12	0.15	0.17	0.21	0.25	0.30	0.37	0.38
Substitution F1	0.11	0.13	0.15	0.17	0.20	0.23	0.28	0.31
Goal F1	0.07	0.08	0.09	0.09	0.10	0.11	0.14	0.16
N p_{card}	11892	8060	5615	4016	2800	1896	1159	835
N $p_{Substitution}$	27208	21265	16945	13374	10360	7515	4598	2900
N p_{Goal}	9706	8019	6735	5711	4737	3755	2555	1686

Table 4.4: Table with results on test set for mAp, AP, Precision, Recall, F1-score and number of predictions made at each threshold

Figure 4.31 shows recall, precision and F1-Score over multiple thresholds. To get what we believe are meaningful predictions, we use a low tolerance of 2.5 seconds. The event '*Goal*' seems to perform the worst. We believe that this is mainly due to two reasons. 1) Goal attempts can fool the model. 2) Replays can fool the model. This means that while we may find a large percentage of the events at a low threshold, as seen in Figure 4.31a, we end up with a large number of false positives in the process. Results on the validation set in Figure 4.30 shows similar results.

Tolerance	Card AP	Substitution AP	Goal AP	mAp
1	0.26	0.14	0.08	0.16
2	0.34	0.23	0.08	0.22
3	0.47	0.33	0.13	0.31
4	0.55	0.42	0.17	0.38
5	0.57	0.49	0.17	0.41

Table 4.2: Average-precision and mAp for classes on validation set

Tolerance	Card AP	Substitution AP	Goal AP	mAp
1	0.24	0.13	0.06	0.14
2	0.30	0.21	0.07	0.19
3	0.42	0.29	0.12	0.28
4	0.48	0.37	0.17	0.34
5	0.51	0.44	0.18	0.38

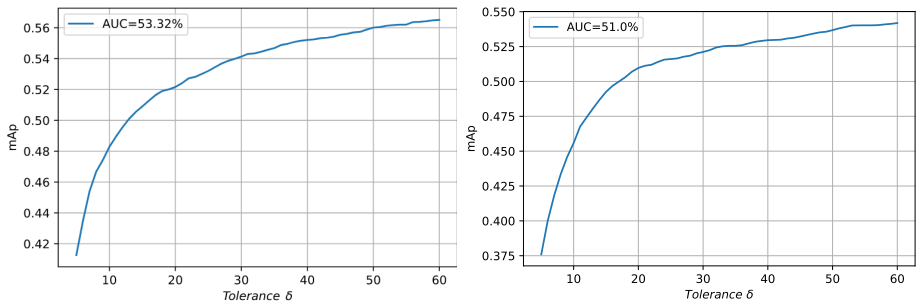
Table 4.3: Average-precision and mAp for classes on test set

In Table 4.2-4.3, we see the average-precision and mean-average-precision for the validation and test set for tolerances 1 through 5.

In Table 4.4 we see the results for a tolerance δ of 2.5. We can see that we reduce the number of samples drastically from the predicted 550 000, but that our precision suffers.

4.4.3 Comparison to baseline

We compare our results to the baseline in SoccerNet [19]. Starting at 5 seconds, we calculate average precision after thresholding at 0.9, where any missed predictions use the original prediction at that point.



(a) Validation set (b) Test set

Figure 4.32: mAp over different tolerances

The metric used is Average-mean-average-precision (Average-mAp), which is calculated by the AUC as a percentage using the mAp for tolerances ranging from 5-60. Giancola et al. [19] report an Average-mAp of 49.7% using 20-second windows for their approach. We report 51.0% Average-mAp on the test set and 53.32% on the validation set for comparison. It should be noted, however, that this metric is less suitable to our approach, as we believe that any improvement after a tolerance of about 5 seconds is due to noise and likely not meaningful predictions because of the temporal extent of our model. Figure 4.32 shows results for both test and validation set. For a more appropriate comparison for our model, we look at the result for the lowest reported tolerance. While not directly reported, it is clear from the spotting results in Giancola et al. [19], that mAp at the initial tolerance of 5 is less than 0.2. Our approach, reports 0.38 mAp on the test set, and 0.41 mAp on the validation set as seen in Tables 4.2-4.3.

4.5 Generalization to other datasets

Training a model on one particular dataset does not automatically mean that the same model generalizes well to other datasets containing the same classes. The dataset that we used for training was described in Section 3.2, and we noted that we might indeed have some bias such as video filming techniques, audience, video editing, and more. In this section, we try to evaluate how well our model generalizes to soccer from different leagues. We use two different datasets from Swedish and Norwegian leagues with clips ranging from 60-90 seconds long. These clips contain a goal at 25 seconds, with an estimated annotation error of ±5 seconds. We also know that roughly 40 seconds after the goal, we have replays. First, we investigate the prediction signal for goals by using a sliding window approach with a stride of 1 second. Next, we investigate whether or not we can fool our model by performing the same tests on goal attempts in the same format, with no actual goal. To measure performance, we evaluate accuracy at different thresholds since we may get false positives when outside

of 20-30 seconds, we evaluate the results when limited to the interval between 20-30 seconds, compared to results over the full clips.

4.5.1 Goals

We first look at our model response for the 233 samples in Allsvenskan and 300 samples in Eliteserien containing goals. First, we look at the prediction signals over time for eight random samples containing goals. Since all samples are similar in that the event occurs at the same temporal spot, we look at the average prediction signal over all samples. Finally, we look at accuracy at different thresholds.

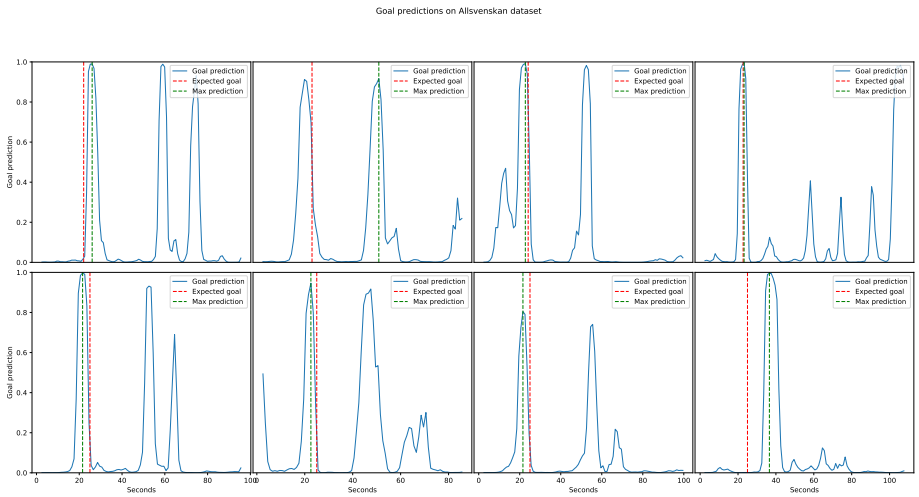


Figure 4.33: Goal prediction for randomly sampled clips from Allsvenskan

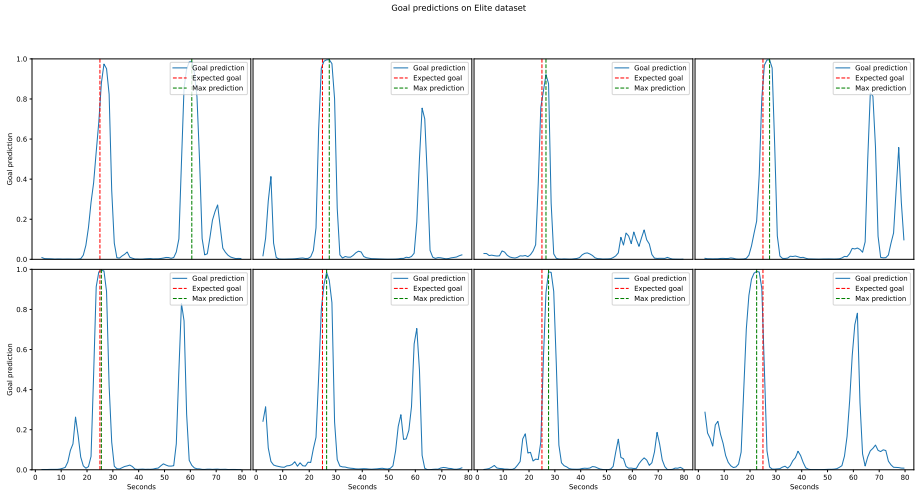
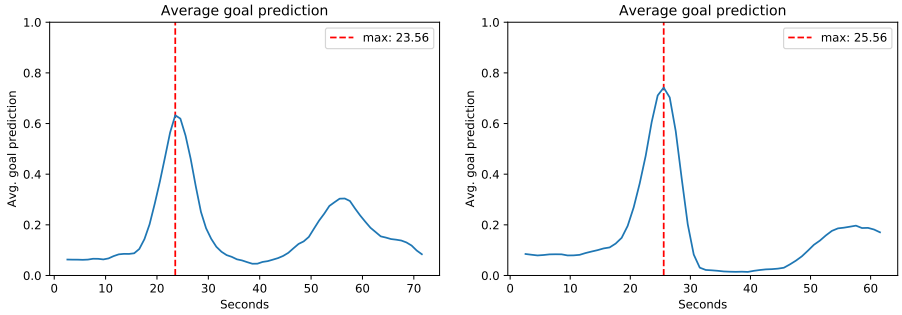


Figure 4.34: Goal prediction for randomly sampled clips from Eliteserien

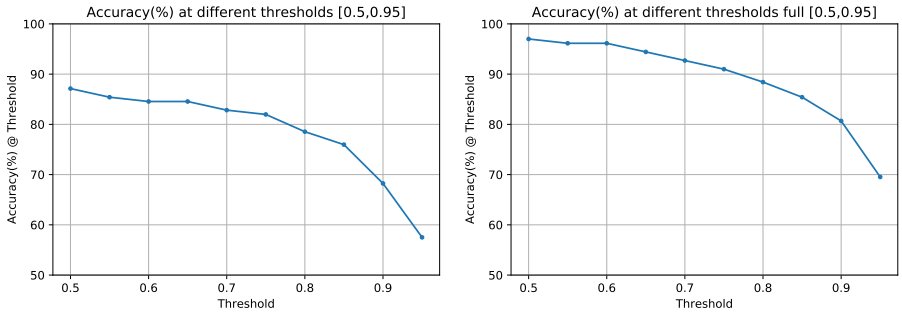
Figures 4.33-4.34 show the softmax predicted score for the class '*Goal*'. We mark the expected goal at 25 seconds, and the maximum prediction produced by the model. We generally see a peak of around 25 seconds as expected. Furthermore, the model seems to produce false positives 30-60 seconds after the goal. This is consistent with replays, which highlights a limitation for our model in that it can not inherently separate the actual goal from a replay. The results of this imply that we can expect the model to produce false positives for most goals, depending on factors such as replay speed and camera angle. It may be appropriate to use this information by assuming that there are no overlapping goals within a temporal window of length L centered at the max prediction.



(a) Swedish Allsvenskan (b) Norwegian Eliteserien

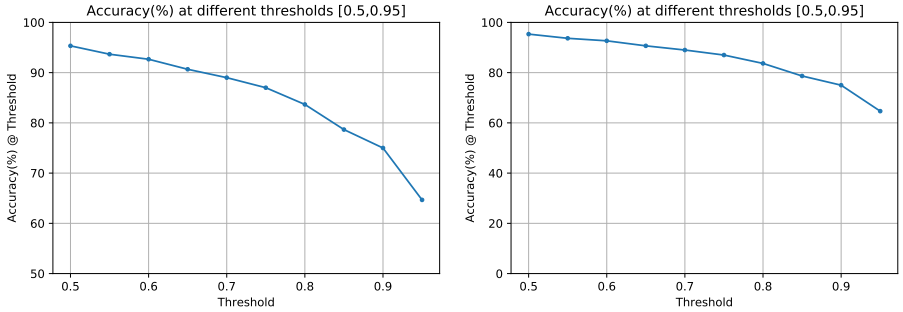
Figure 4.35: Avg. Prediction from Norwegian Eliteserien clips and Swedish Allsvenskan

In Figure 4.35 we see the average softmax prediction for '*Goal*'. We see that there is a high response around 25 seconds in both datasets. Interestingly, we get a higher response from Eliteserien clips. Some possible explanations for this can be differences in production techniques. It may be that different camera angles are used, or statistical differences in the position of goal shots or type of goal. We can see what appears to be false positives from replays after the fact at around 55 seconds. Further strengthening our belief that many replays are caught.



(a) Results in interval [20,30] s (b) Results full clip

Figure 4.36: Results from Allsvenskan dataset containing goals for both full clip and limited interval



(a) Results in interval [20,30] s

(b) Results full clip

Figure 4.37: Results from Eliteserien dataset containing goals for both full clip and limited interval

In Figures 4.36a-4.36b, we see the accuracy calculated by using different thresholds for the temporal interval of [20,30] and the full clip respectively. We see that the accuracy gradually gets worse as we increase our threshold. Comparing the results between the short interval and full clip, we see about 10% higher accuracy. This means that we are catching false positives, likely replays. The results from the Eliteserien dataset shows similar results in Figures 4.37a-4.37b. In the Eliteserien dataset, we see that as we increase our threshold, we also increase the difference in accuracy when comparing the short interval and full clip. Worth noting is the difference in accuracy between Allsvenskan and Eliteserien, which further fuels our suspicions of differences between the datasets as noted earlier.

4.5.2 Goal attempts

Is the model reacting to goals or scenes contextually similar, and can we fool our model? To test this, we use 84 from Allsvenskan containing goal attempts. These clips come in the same format, with the event '*Goal attempt*' occurring at 25 seconds. We perform the same tests by looking at several random samples, average prediction, and accuracy for goal predictions locally and globally. Some clips may contain replays of the goal attempt, and we observe that it may contain subsequent goals, or replays of earlier goals as well.

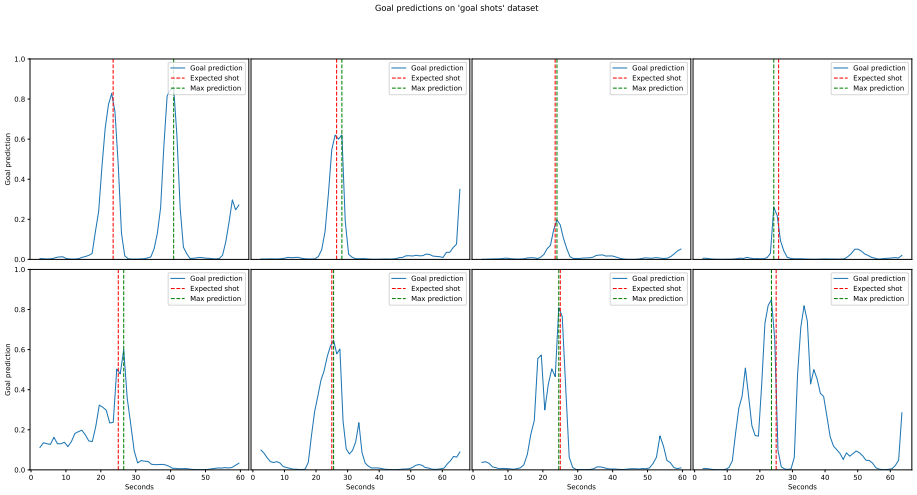


Figure 4.38: Randomly sampled from Allsvenskan 'goal attempts' clips

We see in Figure 4.38 that there is a trend of high responses around 25 seconds. However, we note that based on these samples, the prediction score is lower compared to actual goals. This indicates that we should expect many false positives at lower thresholds and that it is crucial to have a high enough threshold.

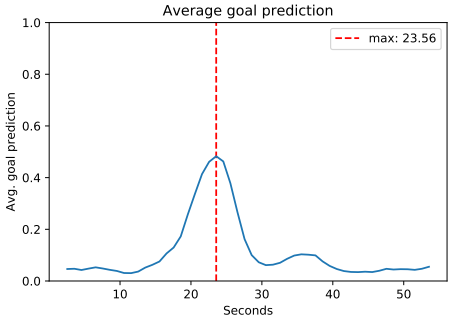
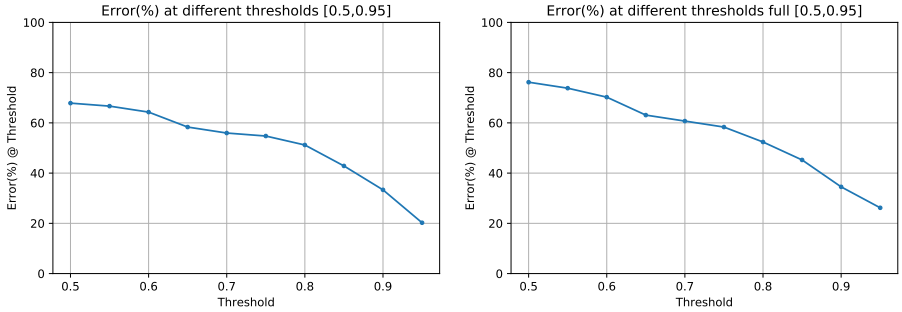


Figure 4.39: Avg. prediction from 'goal attempts' clips

Figure 4.39 further shows a smaller average peak and less response after the goal attempt. We believe that this is because it is less common to have replays after a goal attempt when compared to actual goals. Another common occurrence is multiple goal attempts close to each other, such as the keeper deflecting a shot, resulting in another attempt. This may explain the small increase in our average prediction at about 35 seconds.



(a) Results in interval [20,30] s

(b) Results full clip

Figure 4.40: Results from Allsvenskan dataset containing goal attempts for both full clip and limited interval

We consider the results in Figures 4.40a-4.40b and see that we do indeed get a high error. Effectively fooling the model. However, we see that both local and global predictions drop significantly at higher thresholds. We note that the difference is small when comparing locally against the full clip. To further elaborate, we compare the results in Table 4.5 and find that while the model is fooled. It is not always the case, and that while actual goals have a drop of about 30% from threshold 0.5 to 0.95, goal attempts have a drop of about 47% in error.

Cross-dataset results in accuracy within interval 20 and 30 seconds			
Threshold	**Allsvenskan Goals**	**Eliteserien Goals**	**Goal attempts (Error)**
0.50	87.12	95.33	67.86
0.55	85.41	93.67	66.67
0.60	84.55	92.67	64.29
0.65	84.55	90.67	58.33
0.70	82.83	89.00	55.95
0.75	81.97	87.00	54.76
0.80	78.54	83.67	51.19
0.85	75.97	78.67	42.86
0.90	68.24	75.00	33.33
0.95	57.51	64.67	20.24

Table 4.5: Results for Allsvenskan and Eliteserien at different thresholds

Summary

We have investigated how our model generalizes to other datasets and found that it often find goals. Furthermore, we see that replays may hurt the performance by producing false positives. To evaluate the robustness of our model, we tried to fool our model with goal attempts, and the results indicate that our model is not robust to contextually similar events. The results indicate that on a large scale, the model will likely have high recall at lower thresholds while producing multiple false positives for the event '*Goal*'.

4.6 Discussion

In this section, we first discuss corrupt samples and how we dealt with this during training. Afterward, we look into known bugs that were discovered

after experiments were done. We further discuss recently published work that is relevant and how it affects this thesis. Finally, we discuss potential further improvements that can be made.

4.6.1 Corrupt samples

During model training and testing, occasional stutters were present. In some videos in SoccerNet [19], 59 samples in the training set, 14 in the validation set, and 103 in the test set. This resulted in 3-22% missing frames in these samples. Our solution was to pad these samples by duplication of the last observed frame.

4.6.2 Known bugs

We discovered a negligible bug that occurs during pre-processing steps for the action recognition task of locally sampled frames around events in SoccerNet [19] in the training process. Initially, the method was meant to provide data augmentation by temporally shifting the input by an amount based on a gaussian distribution. However, due to the bug, the model will either not shift at all, or shift 0.016 seconds left or right, this is less than a single frame. Therefore we determine that this has a negligible effect on the results.

4.6.3 Just published work

In late 2019, a new paper on spotting by Cioppa et al. [9] reported 62.5% Average-mAp for the spotting task in SoccerNet. The authors used a novel loss function for temporal action segmentation and achieved state-of-the-art results on THUMOS14 and ActivityNet [18, 29]. It should be noted that the authors here specifiy that they use the tolerance δ as a window such that for a sample to be positive it must no more than $\frac{\delta}{2}$ seconds away from the ground truth. This differs from our approach, as we calculated based on a tolerance δ in both directions.

4.6.4 Scope of the work

There are many different events, and ways to approach this problem. While there are datasets for action detection such as THUMOS14 and activitynet [29, 26], HMDB51, Kinetics [34, 31] for action recognition, or for AVA [23] for spatio-temporal action detection.

In this thesis we mainly focus to on the three events events '*Card*', '*Substitution*' and '*Goal*'. There are multiple other events both in soccer, and in sports in general, however due to the time consumption of manual annotation, we limited the scope to these three.

There are multiple approaches to action recognition and detection, such as through iDT features and more classical approaches. RNNs such as LSTM are also popular for event detection. We however, limited the scope to the 3D CNN based models selected.

4.6.5 Further improvements

During our visualization of CAMs, we found potential errors in our resize and cropping strategy. To avoid this issue, we recommend a different strategy, such

as resize with no cropping or resize without keeping the aspect ratio constant. This to prevent cases where vital parts of the video are cropped away.

In our results, we see a recurring problem with false positives. On our test set with a tolerance of 2.5 and threshold of 0.3, we reduce our predictions from about 550000 to 9706 for the event 'Goal' with a recall of 0.93, for 'Substitution' we show 0.97 recall and for 'Card' 0.93 as seen in Table 4.4. This compared to the 356, 562 and 396 samples respectively. We see that while our recall is decent, our precision is low, with only 0.07 for the event '*Card*'. For the Event 'Goal', this is especially evident, only reaching a precision of 0.1 at threshold 0.95 and corresponding 0.46 recall. This means that for every correct prediction we make, we get nine false positives. We believe, however, that this can be improved by, for example, the detection and removal of replays.

To further improve the current model, we recommend 1) A robust proposal method 2) A different resize and crop strategy 3) Adjustments to training process with regards to background data, for example, add goal attempts to training. This would likely help with the problem of false positives.

There are multiple approaches to event detection, such as the use of improved Dense Trajectories features, Motion boundary histograms, RNNs, and CNNs, both 2D and 3D. In this thesis, we focused on deep-learning based models to predict all events as a multi-classifier. However, it may be that a more appropriate approach is to use multiple different models to specifically targeted each event. For example, a goal may be more appropriately detected through a ball detection model coupled with other features. Another valuable addition would be replay detection, which would also remove many false positives from the '*Goal*' detections. In a practical setting, it is hard to say how useful this is. For now, it is more appropriate to use a human annotator when possible due to the number of false positives. However, there are some use cases where the model could be used as-is. Due to the relatively high recall, it may be viable to use as a proposal method, potentially reducing thousands of ours down to small clips that can be verified by human annotators and generate more massive datasets efficiently. Suppose a small clip of 20 seconds with a known event is available, then a use case would be to get a more accurate prediction locally.

4.7 Summary

In this chapter, we investigate the behavior of the ResNet 3D model using 128 as input with a 5.12-second temporal window. Section 4.3 investigates model behavior both locally around an event to find a reasonable stride and gain insight into how much information the model requires. Further, we find that during a larger temporal window of 138.4 seconds that the results are noisy and cause false positives during some replays. We further looked into what the model reacts to both spatially and temporally by using class activation maps and signals. The results indicated differences between classes, where the event '*Substitution*' seemed to have temporally more extended reactions. Indicative of the model not learning the defined event as the point a player enters the field, but rather contextual similarities such as closeup video of a player that is walking. For events '*Card*' and '*Substitution*', we found that information from class activation signals could increase accuracy within the time interval of 5.12 seconds. When looking at misclassified examples, we found possible weaknesses

with our pre-process center crop strategy. We also found that the model has difficulties with particular camera views that are rare. We test our model on SoccerNet [19] using a sliding-window approach in Section 4.4 and find that . In Section 4.5, we test our model on video clips from Swedish Allsvenskan and Norwegian Eliteserien with 533 clips containing goals, and 84 clips containing goal attempts. The results showed that the model R3D generally finds the events and reacts to replays with a varying degree. However, the results also show that the model is often fooled by clips containing goal attempts.

Chapter 5

Conclusion

5.1 Summary and contributions

Today, event annotation is performed manually. This is a tedious, expensive, and time-consuming task. Annotations are important for entertainment, analysis, and user-consumption in general. With an increased availability with internet access, smartphones, and sites such as Youtube, annotations become all the more important.

In this thesis, we tackled the problem of automatic event detection in soccer. Specifically, we targeted the three events *Card*, *Substitution* and *Goal*. Our objectives were 1) To research and find a general approach to automatic event detection in soccer video that can be easily adapted to new events. 2) Implement and find a good configuration for our selected approaches through experimental prototyping. 3) Analyze weaknesses and strengths of the selected approaches. 4) Compare experimental results to state-of-the-art.

For objective 1 we researched state-of-the-art methods in action recognition and detection and selected the four different architectures: ResNet 3D, ResNet (2+1)D, Mixed Convolution and SlowFast [47, 16].

In the context of objective 2, we used SoccerNet [19], which contains 6637 annotated events over 784 hours. We tested different configurations for our selected model in a classification task by sampling video-frames locally around events. We compared models pretrained on Kinetics-400 [31] and found that our best performing model, Mixed Convolution, achieved 83% accuracy on the validation set compared to its from-scratch counterpart with 72.8%. We further experimented with reduced input data by using gray-scale input rather than RGB and found that the results were similar to the use of RGB. To find our input resolution, we tested 112 x 112 against 224 x 224 and found similar results, leading us to choose 112 x 112 as input. We showed that as a classification task, performance increased when using a higher number of frames during training and testing, resulting in our best performing model ResNet 3D with 128 frame inputs at 88.4% validation accuracy.

For objective 3, we attempted to gain insight into the decision process and behavior of the now trained ResNet 3D model. We performed a sliding-window test locally over a zero-padded 128 frame input, which showed us that as much as 50% of the input data can be zero-padded frames, while still correctly pre-

dicting samples in some cases. Furthermore, we found that the results indicated that a stride of 1 second is appropriate. To better understand our model predictions, we performed another sliding-window test over 138.6 seconds that showed us that the model may produce noisy predictions and that replays often could result in false positives. We analyzed both correctly classified samples and wrongly classified samples by calculating the class activation maps, which we looked at both from a spatio-temporal angle, and a temporal angle. For objective 4, we tested our sliding-window approach on the test and validation set of SoccerNet [19], and we report 51% Average-mAp compared to the 49.7% on the test set. Furthermore, we tested cross-dataset generalization by downloading 617 clips from Norwegian Eliteserien and Swedish Allsvenskan that had goal attempts in 84 of the clips, with the remaining 533 clips containing goals. The results showed that we could generally classify events with 87% accuracy on Allsvenskan samples containing goals, and 95% on Eliteserien clips containing goals with a confidence threshold of 0.5. However, we observed that our model could be fooled by goal attempts, with 67% of goal attempts producing false positives at the same threshold.

5.2 Future work

For future work, we want to see the use of action proposals during spotting, for example, through the use of class activation signals. By using large input window and local subsampling procedures of the class activation signals, it may be possible to create an accurate detection model that does not rely on an overlapping sliding-window approach.

Robust training schemes to adequately capture background data is encouraged. We also encourage the use of both locally extracted features, combined with non-local information, such as through the use of long-term feature banks [54]. Intuitively, this could capture easier with the additional information temporal context. We believe this may help separate events such as 'Goal' and 'Attempted Goal'.

Another related task would be unsupervised learning for event detection. Can we locate events, for example, with clustering, given prior information of the number of occurrences of a given event in game-halves? It may be possible to extract features and perform clustering independent of the spotting task, or it may be used in combination with action detection methods. This would be interesting as, in many practical cases, this information is often easily accessible, e.g., if the event is a goal, one could apply OCR in the ending.

Bibliography

[1] *[1403.6382] CNN Features off-the-shelf: an Astounding Baseline for Recognition*. URL: https://arxiv.org/abs/1403.6382 (visited on 28/11/2019).

[2] *[1507.05738] Every Moment Counts: Dense Detailed Labeling of Actions in Complex Videos*. URL: https://arxiv.org/abs/1507.05738 (visited on 22/11/2019).

[3] *[1704.00389] Hidden Two-Stream Convolutional Networks for Action Recognition*. URL: https://arxiv.org/abs/1704.00389 (visited on 22/11/2019).

[4] *[1711.10305] Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks*. URL: https://arxiv.org/abs/1711.10305 (visited on 24/06/2019).

[5] Sami Abu-El-Haija et al. "YouTube-8M: A Large-Scale Video Classification Benchmark". In: *arXiv:1609.08675 [cs]* (Sept. 2016). arXiv: 1609.08675. URL: http://arxiv.org/abs/1609.08675 (visited on 14/11/2019).

[6] Jürgen Assfalg et al. "Semantic annotation of soccer videos: automatic highlights identification". In: *Computer Vision and Image Understanding* 92.2 (2003), pp. 285–305. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2003.06.004. URL: http://www.sciencedirect.com/science/article/pii/S1077314203001231.

[7] Zhe Cao et al. "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *arXiv:1611.08050 [cs]* (Nov. 2016). arXiv: 1611.08050. URL: http://arxiv.org/abs/1611.08050 (visited on 27/08/2018).

[8] Joao Carreira and Andrew Zisserman. "Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset". In: *arXiv:1705.07750 [cs]* (Feb. 2018). arXiv: 1705.07750. URL: http://arxiv.org/abs/1705.07750 (visited on 20/11/2019).

[9] Anthony Cioppa et al. *A Context-Aware Loss Function for Action Spotting in Soccer Videos*. 2019. arXiv: 1912.01326 [cs.CV].

[10] Navneet Dalal, Bill Triggs and Cordelia Schmid. "Human Detection Using Oriented Histograms of Flow and Appearance". en. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof and Axel Pinz. Vol. 3952. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 428–441. ISBN: 978-3-540-33834-5 978-3-540-33835-2. DOI: 10.1007/11744047_33. URL: http://link.springer.com/10.1007/11744047_33 (visited on 18/11/2019).

[11] Jia Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. en. In: (), p. 8.

[12] P. J. Denning et al. “Computing as a discipline”. In: *Computer* 22.2 (Feb. 1989), pp. 63–70. ISSN: 1558-0814. DOI: 10.1109/2.19833.

[13] A. Ekin, A. M. Tekalp and R. Mehrotra. “Automatic soccer video analysis and summarization”. In: *IEEE Transactions on Image Processing* 12.7 (July 2003), pp. 796–807. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.812758.

[14] Christoph Feichtenhofer, Axel Pinz and Richard P. Wildes. “Spatiotemporal Residual Networks for Video Action Recognition”. In: *arXiv:1611.02155 [cs]* (Nov. 2016). arXiv: 1611.02155. URL: http://arxiv.org/abs/1611.02155 (visited on 20/06/2019).

[15] Christoph Feichtenhofer, Axel Pinz and Andrew Zisserman. “Convolutional Two-Stream Network Fusion for Video Action Recognition”. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 1933–1941. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.213. URL: http://ieeexplore.ieee.org/document/7780582/ (visited on 18/11/2019).

[16] Christoph Feichtenhofer et al. “SlowFast Networks for Video Recognition”. In: *arXiv:1812.03982 [cs]* (Oct. 2019). arXiv: 1812.03982 version: 3. URL: http://arxiv.org/abs/1812.03982 (visited on 15/11/2019).

[17] FIFA.com. *2018 FIFA World Cup^{TM} - News - More than half the world watched record-breaking 2018 World Cup - FIFA.com*. en-GB. URL: https://www.fifa.com/worldcup/news/more-than-half-the-world-watched-record-breaking-2018-world-cup (visited on 19/11/2019).

[18] Bernard Ghanem et al. “The ActivityNet Large-Scale Activity Recognition Challenge 2018 Summary and Workshop Papers”. en. In: (), p. 112.

[19] Silvio Giancola et al. “SoccerNet: A Scalable Dataset for Action Spotting in Soccer Videos”. In: *arXiv:1804.04527 [cs]* (Apr. 2018). arXiv: 1804.04527. URL: http://arxiv.org/abs/1804.04527 (visited on 14/01/2019).

[20] Melvyn A. Goodale and A. David Milner. “Separate visual pathways for perception and action”. In: *Trends in Neurosciences* 15.1 (1992), pp. 20–25. ISSN: 0166-2236. DOI: https://doi.org/10.1016/0166-2236(92)90344-8. URL: http://www.sciencedirect.com/science/article/pii/0166223692903448.

[21] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[22] Priya Goyal et al. “Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour”. In: *arXiv:1706.02677 [cs]* (June 2017). arXiv: 1706.02677. URL: http://arxiv.org/abs/1706.02677 (visited on 26/06/2019).

[23] Chunhui Gu et al. “AVA: A Video Dataset of Spatio-temporally Localized Atomic Visual Actions”. In: *arXiv:1705.08421 [cs]* (May 2017). arXiv: 1705.08421. URL: http://arxiv.org/abs/1705.08421 (visited on 05/06/2019).

[24] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *arXiv:1512.03385 [cs]* (Dec. 2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385 (visited on 22/11/2019).

[25] Kaiming He et al. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *arXiv:1502.01852 [cs]* (Feb. 2015). arXiv: 1502.01852. URL: http://arxiv.org/abs/1502.01852 (visited on 05/02/2020).

[26] Fabian Caba Heilbron et al. “ActivityNet: A large-scale video benchmark for human activity understanding”. en. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, June 2015, pp. 961–970. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298698. URL: http://ieeexplore.ieee.org/document/7298698/ (visited on 20/11/2019).

[27] *https://www.youtube.com/about/press/*. Nov. 2019. (Visited on 19/11/2019).

[28] Haroon Idrees et al. “The THUMOS challenge on action recognition for videos “in the wild””. In: *Computer Vision and Image Understanding* 155 (2017), pp. 1–23. ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2016.10.018. URL: http://www.sciencedirect.com/science/article/pii/S1077314216301710.

[29] Y.-G. Jiang et al. *THUMOS Challenge: Action Recognition with a Large Number of Classes*. 2014. URL: http://crcv.ucf.edu/THUMOS14/.

[30] Andrej Karpathy et al. “Large-Scale Video Classification with Convolutional Neural Networks”. en. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. Columbus, OH, USA: IEEE, June 2014, pp. 1725–1732. ISBN: 978-1-4799-5118-5. DOI: 10.1109/CVPR.2014.223. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909619 (visited on 21/11/2019).

[31] Will Kay et al. “The Kinetics Human Action Video Dataset”. In: *arXiv:1705.06950 [cs]* (May 2017). arXiv: 1705.06950. URL: http://arxiv.org/abs/1705.06950 (visited on 11/04/2019).

[32] Nitish Shirish Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2016. arXiv: 1609.04836 [cs.LG].

[33] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (visited on 18/11/2019).

[34] H Kuehne et al. “HMDB: A Large Video Database for Human Motion Recognition”. en. In: (), p. 8.

[35] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Comput.* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: http://dx.doi.org/10.1162/neco.1989.1.4.541.

[36] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Restarts". In: *CoRR* abs/1608.03983 (2016). arXiv: 1608.03983. URL: http://arxiv.org/abs/1608.03983.

[37] Shugao Ma, Leonid Sigal and Stan Sclaroff. "Learning Activity Progression in LSTMs for Activity Detection and Early Detection". In: June 2016, pp. 1942–1950. DOI: 10.1109/CVPR.2016.214.

[38] Paulius Micikevicius et al. "Mixed Precision Training". In: *arXiv:1710.03740 [cs, stat]* (Feb. 2018). arXiv: 1710.03740. URL: http://arxiv.org/abs/1710.03740 (visited on 05/02/2020).

[39] Mathew Monfort et al. "Moments in Time Dataset: one million videos for event understanding". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019), pp. 1–1. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2019.2901464. URL: https://ieeexplore.ieee.org/document/8651343/ (visited on 20/11/2019).

[40] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv:1506.01497 [cs]* (June 2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497 (visited on 12/06/2018).

[41] Zheng Shou, Dongang Wang and Shih-Fu Chang. "Temporal Action Localization in Untrimmed Videos via Multi-stage CNNs". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, June 2016, pp. 1049–1058. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.119. URL: http://ieeexplore.ieee.org/document/7780488/ (visited on 23/06/2019).

[42] Gunnar A. Sigurdsson, Olga Russakovsky and Abhinav Gupta. "What Actions are Needed for Understanding Human Actions in Videos?" In: *CoRR* abs/1708.02696 (2017). arXiv: 1708.02696. URL: http://arxiv.org/abs/1708.02696.

[43] Karen Simonyan and Andrew Zisserman. "Two-Stream Convolutional Networks for Action Recognition in Videos". In: *arXiv:1406.2199 [cs]* (Nov. 2014). arXiv: 1406.2199. URL: http://arxiv.org/abs/1406.2199 (visited on 18/11/2019).

[44] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. arXiv: 1409.1556 [cs.CV].

[45] Khurram Soomro, Amir Roshan Zamir and Mubarak Shah. "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild". en. In: (Dec. 2012). URL: https://arxiv.org/abs/1212.0402v1 (visited on 23/04/2019).

[46] Christian Szegedy et al. "Going Deeper with Convolutions". In: *arXiv:1409.4842 [cs]* (Sept. 2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842 (visited on 21/11/2019).

[47] Du Tran et al. "A Closer Look at Spatiotemporal Convolutions for Action Recognition". en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, June 2018, pp. 6450–6459. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00675. URL: https://ieeexplore.ieee.org/document/8578773/ (visited on 11/04/2019).

[48] Du Tran et al. "Learning Spatiotemporal Features with 3D Convolutional Networks". In: *arXiv:1412.0767 [cs]* (Oct. 2015). arXiv: 1412.0767. URL: http://arxiv.org/abs/1412.0767 (visited on 21/11/2019).

[49] Grigorios Tsagkatakis, Mustafa Jaber and Panagiotis Tsakalides. "Goal!! Event detection in sports video". In: *Electronic Imaging* 2017 (Jan. 2017), pp. 15–20. DOI: 10.2352/ISSN.2470-1173.2017.16.CVAS-344.

[50] Heng Wang and Cordelia Schmid. "Action Recognition with Improved Trajectories". en. In: (Dec. 2013). URL: https://hal.inria.fr/hal-00873267 (visited on 24/06/2019).

[51] Heng Wang et al. "Dense Trajectories and Motion Boundary Descriptors for Action Recognition". en. In: *International Journal of Computer Vision* 103.1 (May 2013), pp. 60–79. ISSN: 0920-5691, 1573-1405. DOI: 10.1007/s11263-012-0594-8. URL: http://link.springer.com/10.1007/s11263-012-0594-8 (visited on 18/11/2019).

[52] Limin Wang, Yu Qiao and Xiaoou Tang. "Action Recognition with Trajectory-Pooled Deep-Convolutional Descriptors". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). arXiv: 1505.04868, pp. 4305–4314. DOI: 10.1109/CVPR.2015.7299059. URL: http://arxiv.org/abs/1505.04868 (visited on 24/06/2019).

[53] Limin Wang et al. "Temporal Segment Networks: Towards Good Practices for Deep Action Recognition". In: *arXiv:1608.00859 [cs]* (Aug. 2016). arXiv: 1608.00859. URL: http://arxiv.org/abs/1608.00859 (visited on 03/01/2019).

[54] Chao-Yuan Wu et al. "Long-Term Feature Banks for Detailed Video Understanding". In: *arXiv:1812.05038 [cs]* (Dec. 2018). arXiv: 1812.05038. URL: http://arxiv.org/abs/1812.05038 (visited on 20/06/2019).

[55] Yuanjun Xiong et al. *A Pursuit of Temporal Accuracy in General Activity Detection*. 2017. arXiv: 1703.02716 [cs.CV].

[56] Huijuan Xu, Abir Das and Kate Saenko. *R-C3D: Region Convolutional 3D Network for Temporal Activity Detection*. 2017. arXiv: 1703.07814 [cs.CV].

[57] Serena Yeung et al. *End-to-end Learning of Action Detection from Frame Glimpses in Videos*. 2015. arXiv: 1511.06984 [cs.CV].

[58] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *arXiv:1411.1792 [cs]* (Nov. 2014). arXiv: 1411.1792. URL: http://arxiv.org/abs/1411.1792 (visited on 18/11/2019).

[59] Bolei Zhou et al. "Learning Deep Features for Discriminative Localization". In: *CoRR* abs/1512.04150 (2015). arXiv: 1512.04150. URL: http://arxiv.org/abs/1512.04150.

Appendix A

Appendix