Cost-Optimized Event Detection in Football Video

Joakim Arpe och Richard Ericsson

Master's thesis
2020:E36

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

In this thesis we aim to investigate if it is possible to detect events in football video using deep learning methods. The performance of a few different models using video input of varying frame rates is evaluated to find the most promising approach. We also evaluate if a proprietary dataset consisting of estimated player positions and movement can be used to detect events as it would be cheaper than processing raw video. The footage provided comes from two stationary cameras aimed at the left and right side of the pitch respectively which distinguishes our work from most previous research which is done on broadcast video where models can find events by detecting differences in zoom, replays, etc. We also provide novelty by implementing our system in an online fashion to be used in real-time.

To limit the scope of the thesis we constrain the events to corners, kickoffs and free kicks which are relatively similar in their positional structure across games. The results we get are very promising. The frame-based models get decent results, while the models using the player position data achieves extraordinary results while being significantly cheaper to use.

Contents

1 Introduction

1.1 Background

During the twenty-first century football has evolved from a social movement into the billion-dollar-business it is today. The 2017/2018 season of European football saw an increase in revenue of about 11% from the year before up to an immense total of 28.4 billion euro according to the Deloitte annual review of football finance 2019 [1].

With more money and more attention comes the need for more professional and scientifically based methods of analyzing the game. These analysis methods include everything from team performance, player fitness and game statistics to social media analytics. The in recent years highlighted *expected goal* statistics is only the tip of the iceberg of what can be, and is, being done with data analysis in football. The sport analytics market was valued at 0.78 billion USD in 2019 with an expected forecast to reach a value of 3.07 billion USD in 2024 [2].

With this background in mind and considering the explosive rise in popularity machine learning has had in recent years it is not surprising that various machine learning models are being applied within this area [3].

1.2 Motivation

This thesis work is done at Spiideo, a Swedish sports video analysis company based in Malmö. They sell camera systems to sports teams across the world and many different sports to enable analysis of games and practices. The scope is to build a deep learning-based machine learning model that can perform event recognition on live football video, classifying events such as corners, free kicks, penalties, goals etc. The goal is to pave the way for automatic commentary, highlight reels or to enable coaches to analyze events without having to manually tag them. However, this system is supposed to be cheap enough to be used for practices and youth games as well. Therefore, we have to take computational constraints into account for this to be economically viable for both Spiideo and their customers.

1.3 Outline of Thesis

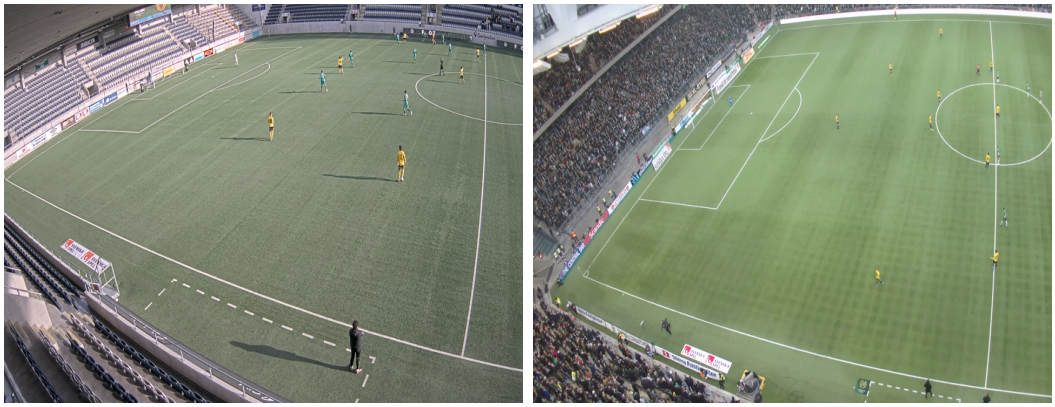
The first part of this project was to collect and correctly annotate the data. The video streams available to us consisted of two cameras per arena showing the left and the right half of the pitch respectively. Both cameras records at 4K resolution (3840x2160) and 25 fps. The recorded games used are from the 2019 season of the two top Swedish leagues Allsvenskan and Superettan. Games from three arenas of different sizes, corresponding to different camera positions on the arenas were extracted as our dataset. Figure 1.1a and 1.1b shows the most notable difference between camera angles in the dataset. The games have tags with timestamps from SEF (Svensk Elitfotboll) for the most important events: goals, kickoffs, free kicks etc.

The second phase was to implement the different models to try and classify the events. Video recognition using machine learning models introduces interesting new challenges compared to ordinary image classification. Beyond the need for good spatial feature extraction from single images also temporal features of a stream of images need to be accounted for. This requires new network constructions that can deal with the additional information and this also brings a much higher computational and memory cost. Since the camera images used in this project are stationary and not zoomed-in on the play we cannot downsample these images too far without losing all interesting information in them, and on the other hand they cannot maintain too high resolution for computational reasons.

Our approach will be to base our machine learning models mainly on deep learning and especially on convolutional neural networks because of their previous success in similar classification tasks. Even though the camera angles vary between the different arenas it is safe to assume that football pitches in general have quite distinct spatial features. This in combination with the fact that only relatively small parts of the images are changing between each frame, even with low sampling frequencies, leads us to believe that focusing mainly on spatial features will be enough to achieve high accuracy in this task. Because of the mentioned limitations we will focus on relatively simple events that should not require too much temporal context to correctly classify them: corner kicks, free kicks, and kickoffs.

The models will be evaluated mainly in three ways:

- F1 score will be used to evaluate how well the model is able to accurately classify the different events.
- Receiver operation characteristic (ROC) curves will be used to evaluate the models' ability to distinguish between the events.
- Prediction time on a reference GPU will be used to compare the computational requirements for each model.



(a) Gavlevallen (b) Tele2 Arena

Figure 1.1: Difference in camera angles for two arenas in the dataset.

2 | Previous Work

Many attempts have been made to find good techniques for recognizing events in sports and more generally, to recognize actions in video. Applications on football video often uses broadcast material and focuses on finding replays, when the camera view changes to a more focused view of an event to discover when something interesting has happened [4]. Other works [5] also uses broadcast video and realizes that videos of goal events are blurry because the camera moves faster and are more zoomed in than during other events.

Before deep learning became feasible due to increased computing power, an approach based on finding distinct image features on each frame and representing them as a so called "Bag of Features" followed by a classical machine learning classifier algorithm such as an SVM could be used to classify frames into categories. The full video was then classified by forming an average over the classification result on its frames [6].

An early paper using deep learning that tried to utilize temporal relations between frames was [7]. They experimented with different setups that were based on using pretrained 2D-networks to find features from individual frames and fusing the results together at different stages. Building on these results, Two Stream Networks was introduced in [8] where they used precomputed Optical Flow to capture the temporal context in one stream of the network and another to find spatial relations using single frames and then combined the predictions using an SVM. Optical Flow is an estimation of motion between frames [9]. LSTMs have also been used to model temporal relations between frames using CNNs as feature extractors, for example in [10].

Using 3D convolutions for video classification was introduced in [11], where the idea was to overcome the large computational constraints of training 3D convnets by pretraining on a large dataset (Sports1M) and then using the network as feature extractors on other smaller datasets. Another approach building upon that method is called I3D [12], where they leverage 2D models pretrained on ImageNet and inflate their weights into 3-Dimensional convolutions.

After the success of methods relying on 3D convolutions a team of Facebook researchers presented another approach in [13] where they factorize the 3D op-

eration into a spatial part and a temporal part. Both parts are 3D convolutions with the former being a $(1 \times d \times d)$ convolution and the latter being $(t \times 1 \times 1)$ instead of the full $(t \times d \times d)$ where t stands for the number of time steps and d the width and height of the input. They argue that the decomposition results in a network easier to train and that they should have better performance per parameter due to the number of nonlinearities being effectively double that of a regular 3D CNN. Their experiments show that they achieve better results than regular 3D CNNs.

3 Theory

3.1 Deep Learning and Artificial Neural Networks

Deep learning is a class of machine learning that uses artificial neural networks (ANNs) to learn in a way inspired by how biological brains process and distributes information. This report will be written in a way that assumes that the reader is familiar with the basics of deep learning and the most common network structures and therefore only a brief introduction of the basics to the various aspects will be given. For a more thorough introduction to deep learning the reader is recommended to read *The Deep Learning Book* by Goodfellow et al. [14].

3.1.1 Convolutional Neural Networks

A convolutional neural network is a type of network that uses mathematical convolutions to process data. This is especially effective for data in matrix form where local features benefit the goal of the network. The input can for example be a 1-D time series where data have been sampled at some time interval or, as is more well known, a 2-D image that can be represented as a matrix of pixel values. We are dealing with video data which can be seen as stacked images or a 3D matrix where 3D-convolutions can find both spatial and temporal features.

A convolution calculates so called feature maps S from input by sliding a kernel K, a $m \times n$ matrix, over the input I. Each element in I is then mapped to S by multiplying its neighbourhood with the kernel and summing the results, see Equation 3.1 for a 2D convolution which is computed for each pixel (i, j). Strictly speaking that equation is a cross-correlation since the kernel must be flipped to be a convolution but since the kernel is learned from data it does not matter and most deep learning libraries implement cross-correlation [14]. Figure 3.1 is a sketch of how video can be used as input to a model as a stack of frames, the model then receives a tensor of shape $I = (batch\ size, N, H, W, 3)$ where N is the number if images, H, W the height and width of an image and

the 3 represents the red, green, and blue channels of RGB images.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{3.1}$$

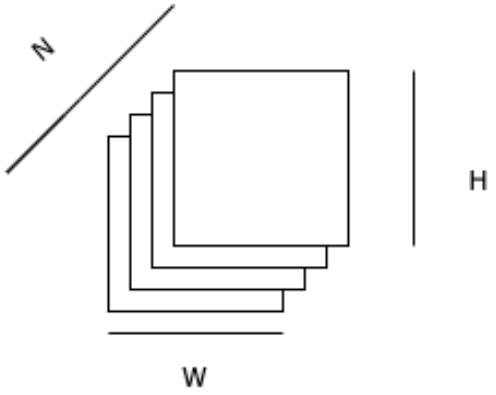


Figure 3.1: Visualization of input to a 3D-convolutional network.

3.1.2 Residual Neural Networks

Residual neural networks are a form of network architectures introduced in 2015 by He et al, [15]. They proposed the use of connections between nodes in non-consecutive layers commonly referred to as skip connections or shortcuts. These skip connections makes it much easier to have deep network architectures since the gradient can flow more easily through the network, because it skips some non-linear operations such as ReLU described in Section 3.1.4 and Batch Normalization described in Section 3.1.5 etc, and thereby negates the problem of vanishing or exploding gradients.

Another type of architecture used in deep networks to reduce the amount of computations is the bottleneck residual block. It is built upon the technique of separable convolution together with a skip connection, see Figure 3.2. Separable convolution means splitting up two ordinary convolutional layers into a block of three layers instead. The first and the third layer is a 1x1 convolution used to control (reduce and restore) the dimensionality. This way both the input and output connected to the second layer, a 3x3 convolutional layer, can be dramatically decreased in dimensionality. Since the number of computations are directly proportional to the dimensionality this method can be used to greatly decrease the amount of computations needed while still retaining its accuracy [15]. The original structure proposed in [15] was for 2-D convolutions, but in recent years the possibility of using the same structure with 3D-convolutions have been explored with great success [16].

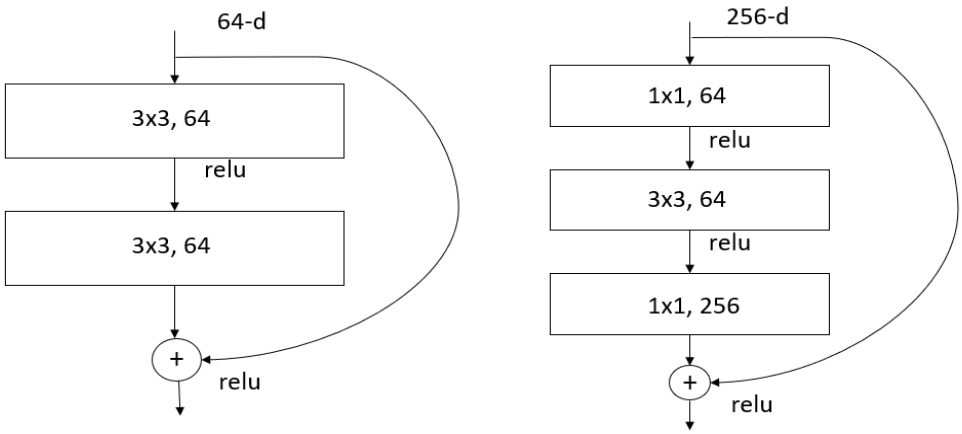


Figure 3.2: Visualization of two 3x3 convolutions with at skip connection vs a bottleneck block structure.

3.1.3 LSTM

The Long short-term memory (LSTM) is a type of recurrent neural network (RNN). RNNs are used to deal with sequences of data such as language or video. LSTM was introduced to better model long-term relations in data while retaining short-term capabilities. It is also meant to deal with the problem of exploding and vanishing gradients that RNNs suffer from during training. An LSTM unit keeps a cell state, C_t, that can be seen as the features that the LSTM has found in the data. There are three so called gates that alter the states in different ways. The gates use a sigmoid activation function, σ, that outputs numbers between 0 and 1 which decides what should be let through the gate. The forget gate, f_t, looks at the output of the previous timestep, h_{t-1} and the current input, x_t to decide which parts of the cell state should be removed. The input gate, i_t, also uses h_{t-1} and x_t to decide which cell states to update. The update is made by combining h_{t-1} and x_t and running them through a tanh layer. The output gate then decides what to output and finally the output is scaled to between -1 and 1 using a tanh layer [17]. See Equations 3.2-3.7 where W denotes weight matrices and b denotes biases for an idea of how these updates work.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{3.2}$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{3.3}$$
$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{3.4}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.5}$$
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{3.6}$$
$$h_t = o_t * \tanh\left(C_t\right) \tag{3.7}$$

3.1.4 ReLU

Activation functions are used in neural networks to introduce non-linearity between layers so that more complex functions can be learned. The rectified linear unit (ReLU) activation function, defined in Equation 3.8, is widely used in deep learning because it is cheap to compute and comparatively easy to optimize since it is nearly linear [18].

$$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{3.8}$$

3.1.5 Batch Normalization

When training deep neural networks, a common occurring problem is that when the parameters of a layer changes the distribution of the input to all following layers changes as well. This means that the training often requires a low learning rate and carefully initialized parameters. One way to handle this problem is to use the batch normalization transform defined in Equations 3.9 to 3.12 which aims to scale the output to be zero-mean and have unit variance [19].

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \tag{3.9}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_i - \mu_{\mathcal{B}}\right)^2 \tag{3.10}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{3.11}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}\left(x_i\right) \tag{3.12}$$

The equations above shows the steps of the BN transformation where x_i is a mini-batch output from a layer and y_i the corresponding mini-batch input to the next layer. The parameters γ and β are learnt in the optimization process.

3.1.6 Optimization

When training neural networks one must decide how to navigate the so called loss landscape, most commonly using gradient based methods. All these methods are based on calculating the gradient of the loss function, J, with regards to the weights of the network, θ, based on the training dataset and then update the weights to minimize the loss. This is the actual learning part of the machine learning algorithm. The most basic method is Stochastic Gradient Descent [20], SGD, where an update is performed each time a batch of training examples of inputs $x^{(i)}$ and labels $y^{(i)}$ has gone through the network, see Equation 3.13 for how the parameters are updated. A learning rate parameter, η, controls how much each batch changes the weights, usually this is configured to decay over time.

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right) \tag{3.13}$$

There are also more advanced optimizers like Adaptive Moment Estimation, Adam [21] where each parameter is given their own learning rate which changes based on how the learning progresses. Adam also keeps an exponentially decaying average of past gradients, m_t, and past squared gradients, v_t, defined in Equations 3.14 and 3.15 that are estimations of the mean and uncentered variance of the gradient, g_t, respectively. These additions are supposed to help guide the convergence of training and avoid bad local minima. Both terms are bias-corrected, see Equations 3.16, 3.17. Finally, the update rule is given in Equation 3.18. The paper's proposed default values of the parameters are $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$ [21].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \tag{3.14}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \tag{3.15}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.16}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.17}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t \tag{3.18}$$

3.1.7 Imbalanced Classification

A common classification problem is that one or more classes dominates the dataset. This can be a problem because the network might be biased towards

the dominant class/classes and measuring performance by simply reporting accuracy is no longer representative. One way to handle this is to weigh the loss function using Equation 3.19 where m is total number of samples, m_i is the number of samples in a class and μ is a constant that controls how much the imbalance should affect the weights. In [22] $\mu = 0.7$ is empirically found to work for action recognition in hockey.

$$w_i = \log(\mu \frac{m}{m_i}) \tag{3.19}$$

3.1.8 Hard Negative Mining

A technique for making models more robust and better at differentiating between events and non-events is called Hard Negative Mining. The thought behind it is that after training a model on initial training data one extracts lots of non-event examples and run them through the model. You then add the non-events that the model with high probability predicts as an event to the training data and retrain the model with the updated dataset. This should hopefully make the model better at distinguishing between events and non-events and because of that generate fewer false positives. Our exact approach is described in greater detail in section 4.3.

3.1.9 Label Smoothing

Label smoothing is a widely used strategy in state-of-the-art image recognition models because it increases generalization and reduces the effect of overconfidence where the model is very certain even when it is wrong. Usually the contribution of one sample $\boldsymbol{y}$ to the cross-entropy loss is

$$H(\boldsymbol{y}, \boldsymbol{p}) = \sum_{k=1}^{K} -y_k \log(p_k) \tag{3.20}$$

where p_k is the probability output of a classification layer with a softmax function [23]. When using label smoothing the contribution is altered according to

$$y_k^{LS} = y_k(1 - \alpha) + \alpha/K \tag{3.21}$$

where K is the number of classes, α a hyperparameter and y_k is 1 for the true class and 0 for the rest [24]. Equation 3.22 shows an example with three classes and $\alpha = 0.1$.

$$LS_{\alpha=0.1}([1, 0, 0]) = [0.9333, 0.0333, 0.0333] \tag{3.22}$$

3.2 Metrics

This section describes the most important metrics used to evaluate the models in this thesis.

3.2.1 F1-score

When measuring the result of a test the first question that needs to be answered is what problem this model is supposed to solve. In our case when measuring on a whole match of 90 minutes the model could possibly reach a prediction accuracy of 99% by simply predicting negative all the time, since at almost all times this will be correct. The problem here becomes that one can be lead to believe that the model is fantastic when in reality it does nothing of what we want it to. Therefore when having an uneven class distribution, we need a more robust type of measurement. The F1-score is a measurement of test accuracy defined as a function of the two metrics precision and recall defined as

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positive}}{\text{True Positive+False Positive}} \\ \text{Recall} &= \frac{\text{True Positive}}{\text{True Positive+False Negative}} \end{aligned} \tag{3.23}$$

As can be interpreted from Equation 3.23 these two metrics gives more information both of how accurate the models is in its prediction (precision) as well as the cost of falsely classifying test samples (recall). The F1 score is the function that calculates the harmonic mean of the precision and recall as

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.24}$$

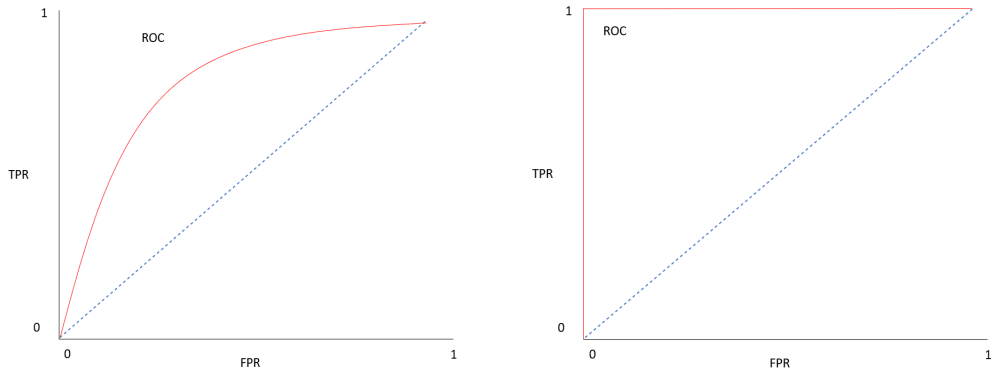
We have chosen F1-score as the measurement used when evaluating our model in the training phase for validation and when testing.

3.2.2 Receiver Operating Characteristic Curve

The receiver operating characteristic curve (ROC) or AUC-ROC (Area under curve - receiver operating characteristic) curve is a measurement of a model's ability to distinguish between two classes [25]. The axes of the plots in Figures 3.3a and 3.3b correspond to the true positive rate (TPR) on the y-axis and the false positive rate (FPR) on the x-axis. The TPR is the same thing as the recall defined in Section 3.2.1 and the FPR is correspondingly defined as the same metric but for the false positives. Compare Equations 3.23 and 3.25 to see the connection.

$$\text{FPR} = \frac{\text{False Positive}}{\text{True Negative} + \text{False Positive}} \tag{3.25}$$

Every point on the ROC curve represents a connection between the TPR and the FPR for various thresholds in the classification. The area under the ROC curve (AUC) is a good measure to compare how well different models can distinguish between classes.



(a) Example ROC of a better than random classifier.

(b) Example ROC of a perfect classifier.

Figure 3.3

Figure 3.3a shows a typical ROC curve where the orange line shows classifier performance and the dotted blue line shows the curve for when TPR and FPR is equal at every threshold. If the classifier is on the dotted line the AUC becomes 0.5 and this means that the model has no ability at all to distinguish between the classes and is no better than random. Figure 3.3b shows the optimal ROC curve where the AUC is equal to 1 and the model is perfectly able to distinguish between the classes.

The way we will make use of the ROC curve in this thesis is to see if there is a difference between how well our models detect different events. We do this by 1-vs-all, i.e. simulating a binary classification where the negative class is all other classes.

3.3 Computational costs

There are many factors that affect the computational cost of running a deep learning system in an online setting. For example, the complexity of the model, the structure of the model, how often predictions are made, and how large the input is both in terms of image size and the length of the temporal dimension.

We will present different settings and measurements to see how each of these factors affect the inference time.

3.4 Player Position Data

In a previous master thesis at Spiideo, a method to estimate player positions and movements was developed. It uses gradient-based foreground estimation of downsampled grayscale images to estimate the probability of belonging to foreground (non-stationary objects) for each pixel. To reduce the dimensionality of the data, the result is then binned into a grid of $n_x \times n_y$ smaller parts of the playing field where each grid element gets the value of the proportion of its pixels belong to foreground. Players are modelled as cylinders and are assumed to be located where projections of these cylinders onto the camera contain enough foreground pixels. Speed in the x- and y-dimension is then estimated by tracking players and using a weighted average of first-order differences, putting more emphasis on recent samples. For a much more detailed description of the method, see [26].

4 | Method

This section will describe the data provided to us and how it has been used in this thesis.

4.1 Data

Here we describe the two different datasets used and how the data is labelled.

4.1.1 Videos

The data provided consists of 33 games from the two top Swedish leagues, Allsvenskan and Superettan, from three different arenas, see table 4.1. Every game consists of two data streams of video recorded on the left and right half of the pitch respectively. The video streams have been recorded in 4k (3840x2160) resolution at 25 fps and with an average length of about two hours. The games also contain official statistics from the Swedish professional Football Leagues (SEF) in form of event tags at certain timestamps.

The events we will focus on in this thesis are corners, free kicks and kickoffs and the event data have been summarized in table 4.2 in the section below. The *Other tags* category includes the rest of the tags that are annotated for the games such as substitutions, goals, shots, yellow cards etc.

By looking at the video streams provided we quickly noticed that most of the time the ball is in play on one side of the pitch only, especially considering cases where tagged events occur. This means that looking at the video on the other camera yields no valuable information for the classification of said event. Spiideo provides a function that auto-follows where there is action on the pitch, i.e. where the ball is in play, based on player positions. Making use of this information we could retrieve the coordinates of the center of attention at a certain timestamp and use this to determine which camera stream we want to look at. This way we scale down the data needed to be analyzed in the training phase by half while retaining essentially all interesting information.

We decided that we would keep the images as they were and try to create a network that were able to generalize between the different camera angles from the arenas. The reasoning behind this being that one of the main focuses of the thesis is that the final model should be able to run quickly and introducing a preprocessing step that geometrically re-projects all images to the same angle relative to the pitch is a heavy computational demanding step.

The games were split up in a 80:10:10 ratio between train, validation and test, making sure that all arenas were represented in each dataset see table 4.1 With a more sizeable dataset it could be beneficial to split the data by arenas but since that would limit the amount of training data too much we decided that the second best option was to do the split by games.

	Tele2 Arena	Gavlevallen	Eleda Stadion	Games/Dataset
Train	12	6	7	25
Validation	2	1	1	4
Test	2	1	1	4
Games/Arena	16	8	9	33

Table 4.1: Train/val/test split of games and arenas.

4.1.2 Player position data

Working with full images often includes a a great deal of processing of data that is unnecessary for the task at hand. This is especially relevant considering the images used in this thesis and how small the difference can be between what is considered an event and what is not. Spiideo has provided us with a second dataset that they have extracted from the images that is already in use in their system and is also of much lower dimensionality than images. The features in this new dataset is represented by a $17x28$ grid of rectangles each representing an equally large part of the pitch. Each of these grid elements, or pixels, consists of 3 channels where the first is the probability of that element containing foreground, i.e. players, see figure 4.1 for visualization. The second and third channels represent horizontal and vertical movement respectively. This new feature data can be seen as normalized images with pixel values $I^{(t)} : \Omega \rightarrow [0, 1]$ where 0 indicates that an element/pixel belongs to the background and 1 meaning it is confidently a moving non-stationary object which in our case is most likely a player.

0	0.1	0	0	0.2	0	0	0	0	0
0	0	0.3	0	0.2	0	0	0	0.1	0
0	0	0.2	0	0	0	0.3	0	0	0
0	0	0	0	0	0	0	0	0	0
0.1	0	0	0.1	0	0	0.2	0	0	0
0	0	0	0	0	0	0	0	0.1	0
0	0	0	0	0.3	0	0	0	0	0

Figure 4.1: A visualization of player position data as an image of probability of foreground in different areas of a football pitch.

4.1.3 Tags

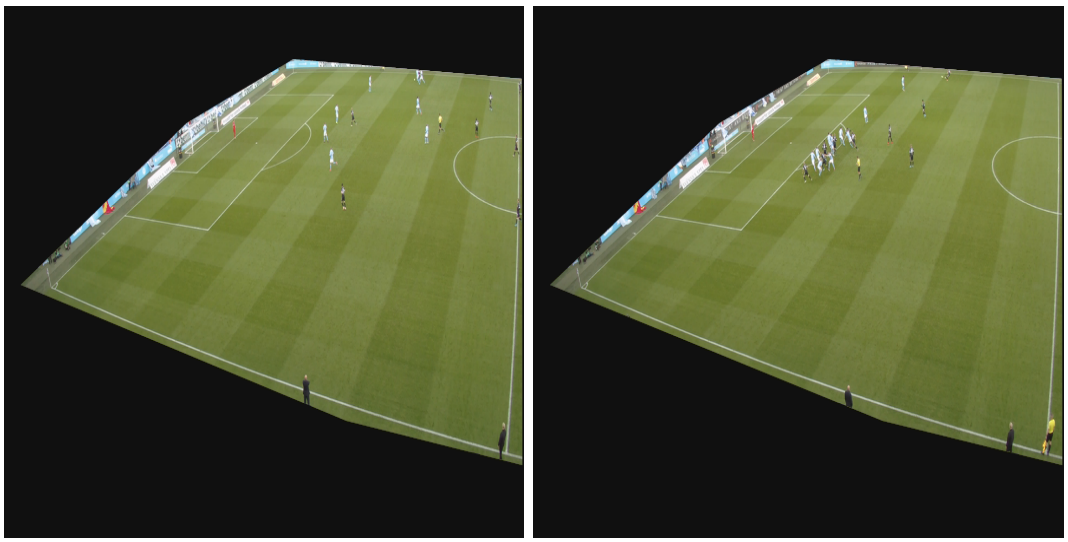
When analyzing the tag data provided we quickly realized two things. Firstly, that some tags that were necessary for our classifier were not set for all games, for example the Corner tag was first available in games from October 2019 and thus we needed to annotate events in some games ourselves. Secondly the available tags were mostly annotated when the referee makes the call and not when the actual event occurs, i.e. the ball is kicked. Since the time between the call is made and when the ball is kicked can sometimes range up to minutes and generalizing a classifier to determine when the referee makes different calls would be too difficult, these timestamps needed to be changed. We annotated the video streams using Spiideo's interface for tagging.

To deal with the first problem we used minute-by-minute match statistics data from Aftonbladet live score to find and annotate corners and free kicks in the games where these tags were missing. For the second problem we added new tags for those events and since we did this manually, we could decide where the timestamp for an event should be set. We choose the moment the ball was kicked since sampling around these timestamps would then both yield some frames of the setup for the set piece and also some frames of the movement following the ball being kicked. See figure ref 4.2a and 4.2b for the difference between the initial placement of a tag and with our modification.

When analyzing the *free kick* tags, we decided to limit which types of free kicks that were of interest for us. The reasoning behind this was that about half of the free kicks were simply a player taking the ball and passing it sideways to the nearest teammate. Since we are removing a lot of the temporal

information contained in a single example these free kicks would be very hard, if not impossible, to distinguish from a non-event clip. Apart from this we also decided to remove free kicks that were on the defensive half for the team with the free kick. The reasoning behind this was that our models (which are using frames and not player position data) are trained by only seeing one half of the pitch during an event. This means that a lot of these free kick events would not even contain the set piece itself and therefore be very hard to distinguish from a goal kick. In summary this means that we decided to limit the free kicks we were interested in, to free kicks that could be defined as a set piece situation. Limitations included that they should be:

- On the offensive half for the team taking the free kick
- Executed after a considerable amount of time after the referee makes the call
- Not simply being moved sideways to the closest teammate



(a) (b)

Figure 4.2: Difference between tagging when the referee makes the call (4.2a) and when a free kick is kicked (4.2b).

In table 4.2 we present how many annotated events that are available of each type on the different arenas. We also include a column called *Negative tags* which are tags that we have generated randomly to use as counterexamples to our event classes. They have been generated such that they do not overlap events. The *Other tags* category includes tags that are pre-annotated but we are not trying to classify. These includes events such as yellow and red cards, substitutions, shots, goals. We will be using these tags as well as the *Negative tags* as counter examples for training.

Arena	Matches	Corners	Kickoffs	Free kicks	Other tags	Negative tags
Tele2 Arena	16	145	92	108	529	320
Gavlevallen	9	86	46	59	316	180
Eleda Stadion	8	78	31	58	277	160
Average/game	1	9,4	5,1	6,8	34	20
Total	33	309	169	225	1122	660

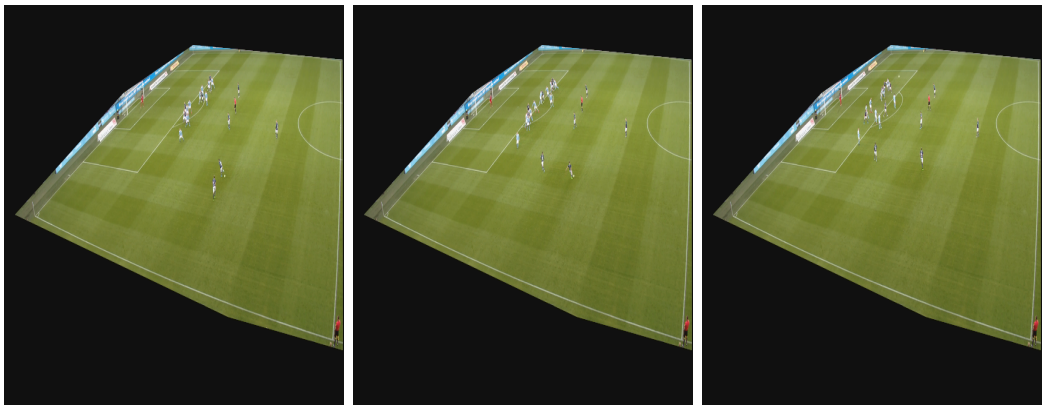
Table 4.2: Distribution of events. Other tags are available tags for events that we are not using.

4.2 Preprocessing

This section describes the different preprocessing steps performed on the data.

4.2.1 Sampling & Downsampling

Dealing with videos in 4K resolution and 25 fps is difficult because of memory and computational constraints. To be able to use deep learning methods on video the amount of data needs to be scaled down or it will not be commercially viable at the scale that Spiideo operates. It is also probably not necessary to use all this data since temporal variations between frames are small and players are still visible in lower resolution. An example of an event downsampled to 1 fps can be seen in figure 4.3.



(a) (b) (c)

Figure 4.3: Three frames of a Free Kick on Eleda Stadion sampled at 1 fps.

Formally, we uniformly sample N WxH frames per event, see figure 3.1, where we define an event as T seconds of video around its tagged timestamp. N and T are parameters that we will vary to find an optimal combination. W and H will be downsampled to 540 and 960 from the 4K input and our hope is that we can scale down the amount of data to reduce computational cost

while retaining enough spatial and temporal information that we are able to find discriminative features that separate our classes. Figure 4.4 shows how the full data of 25 frames covering 1 second is sampled to 1 respectively 3 fps where the blue frames represent the sampled frames. When downsampling we use the ffmpeg scale filter with the lanczos algorithm which is more effective than simple interpolation [27]. Additionally, pixel values are scaled to between -1 and 1 before sent to the network.

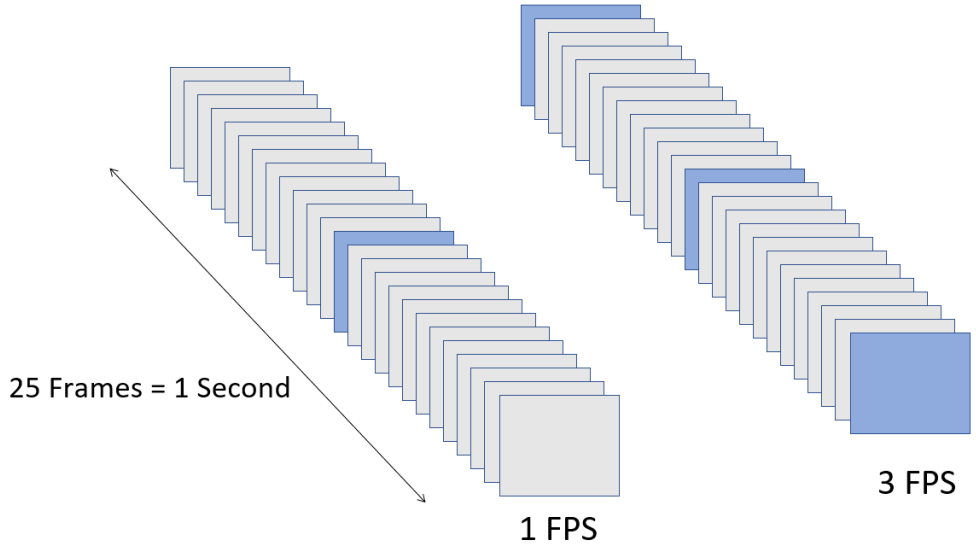


Figure 4.4: Illustration of how the native frame rate of 25 fps is downsampled to 1 and 3 fps respectively.

4.2.2 Masking

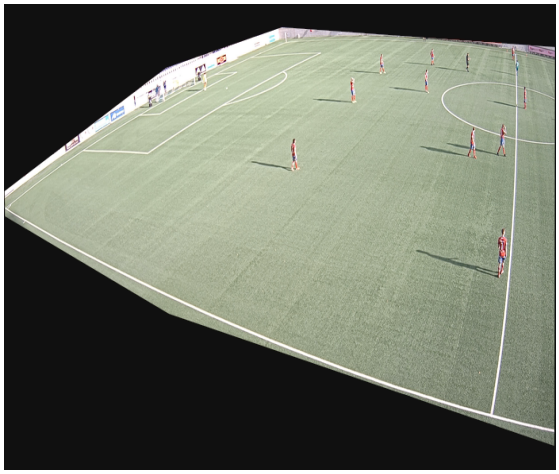
The audience in the different arenas takes up a fair amount of the camera images see figure 4.5. Since we are only interested in what happens on the pitch and these parts of the images would bring considerable noise to the network, we realized that we had to remove it. One approach would be to crop the images to both get rid of the noise and at the same time downscale the resolution. The problem with this approach is that the various arenas have significantly different camera angles. We would therefore be required to crop them individually and thereby introduce variable resolutions in our data. To avoid this issue while still solving the noise problem we decided to create individual masks for the different arenas and put the masks on each image, see figure 4.5. These masks were created manually in GIMP by creating images that were black outside the field and transparent on it.

(a) (b)

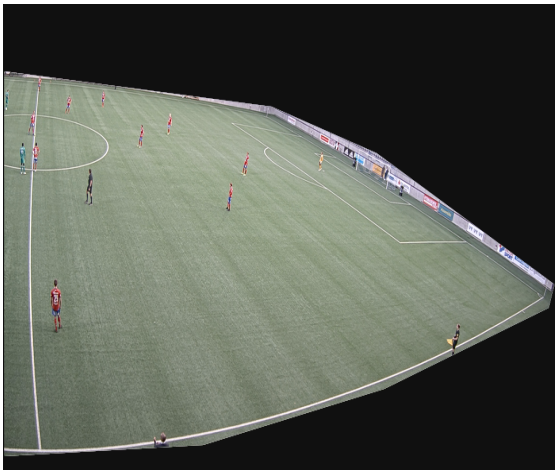
Figure 4.5: Example of how a mask is used to remove unnecessary information from images.

4.2.3 Data Augmentation

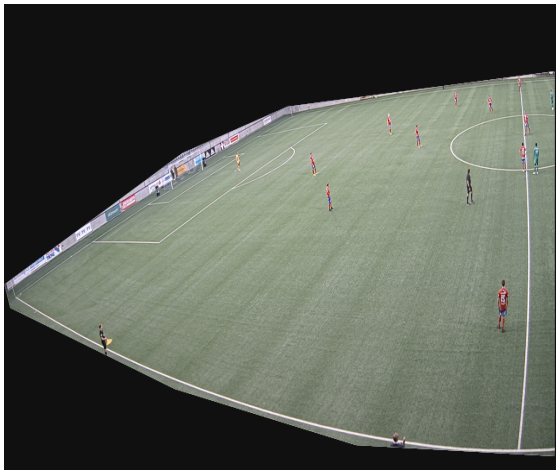
Random augmentation of training data is often a good idea to artificially increase the size of the dataset to get better generalization of the model. Examples of augmentations are random cropping, flipping, rotations, color changes of different kinds etc. These augmentations help when the input varies a lot and you need to make sure that your model for example recognizes a dog or a cat even if the fur is of a different color or the image is taken from a slightly different angle. In our case the cameras are fixed, and we need to see the entire field so cropping and rotating will most probably not help. We also decided not to do any color augmentations even though it theoretically could help to reduce the effect of different shirt and grass colors because we thought that it would be negligible. We did however use random horizontal flipping for both frame data and feature data, see figure 4.6 for an example. We believe that the difference between a right-side image and a flipped left-side image should be comparable to the differences between arenas and thus it might lead to better generalization between arenas to include flipped images in the dataset. For feature data we flip vertically as well since that does not distort the shape as it would with images. Even if the precision in calculating the features might be worse on the far side of the pitch it should still add some value.



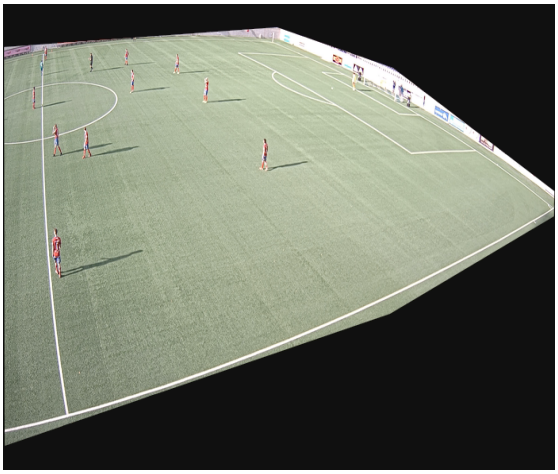
(a) Left side image of Kickoff in Gavlevallen.



(b) Right side image of Kickoff in Gavlevallen.



(c) Flipped version of (b).



(d) Flipped version of (a).

Figure 4.6: Comparison of images of a kickoff event and their flipped versions.

4.2.4 Temporal Jittering

A method that can be used to reduce overfitting and also increase the temporal dimensionality of the models is temporal jittering. This means that when training a model with N frames as input, we will extract $2N-1$ frames around the timestamp of an event. In each training batch a start frame will be randomized and then the subset of N consecutive frames following the chosen start frame will be extracted, with the condition that each of these subsets contains the frame with the event timestamp. This means that each training example can be varied in N different subsets. Figure 4.7 shows the frames extracted for the second subset when the model is trained with $N = 3$ frames.

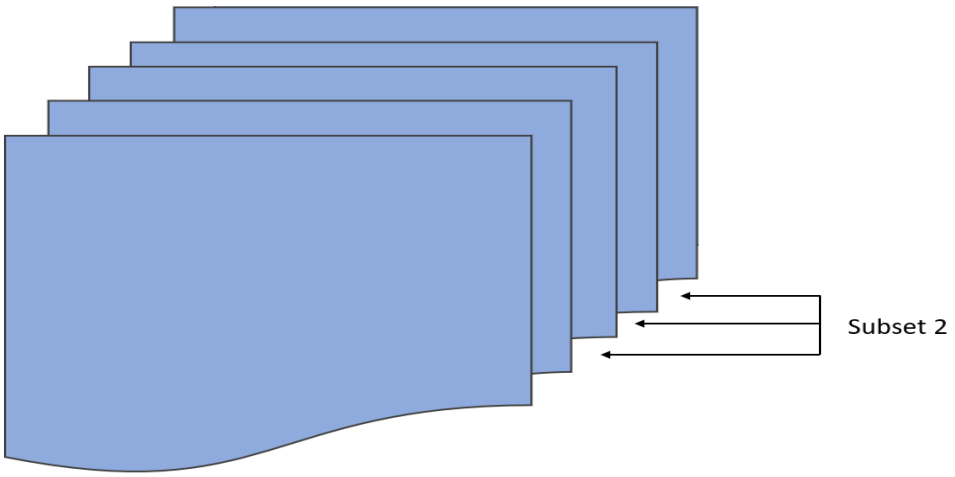


Figure 4.7: An illustration of 5 frames extracted around a timestamp when using 3 frames as input as well as which frames will be used in the second subset of the temporal jittering.

As we can see in Fig 4.7 this method provides us with N (3) variations of the same event. This obviously increases the amount of training examples and thus reduces the risk of overfitting. This will hopefully also extend the duration of where the network will be able to predict events.

4.3 Models

The models used in this thesis is all based on convolutional neural networks with either a dense head (fully connected layers) or an LSTM layer as classifier. A brief description of the architectures of each model will be given. For visualization see figures 4.10, 4.11 and for exact configuration see appendix A. We will from now on refer to models trained on video frames as "frame models" and models trained on player position data as "feature models".

4.3.1 Frame Models

Conv3D Baseline

This model performs two 3D convolutions with kernel size ($frames \times 3 \times 3$) with max pooling on the input. It is based on the MCx model from [13]. We use striding and pooling so that after these operations there is only one frame left in the temporal dimension. Thereafter we use 2D convolutions to capture spatial information and finally a dense head as classifier.

R(2+1)D

This model has the same architecture as the Conv3D-based model but with factorized 3D convolutions ((2+1)D) instead of ordinary 3D convolutions in the first two layers.

Our implementation of R(2+1)D networks is based on [28] but ported to Keras. Figure 4.8 shows the R(2+1)D convolution on an input consisting of 5 stacked RGB-frames of size 540x960. The first 3D convolution handles spatial information and uses a stride of two to downsample the image while the temporal convolution uses a stride of one.

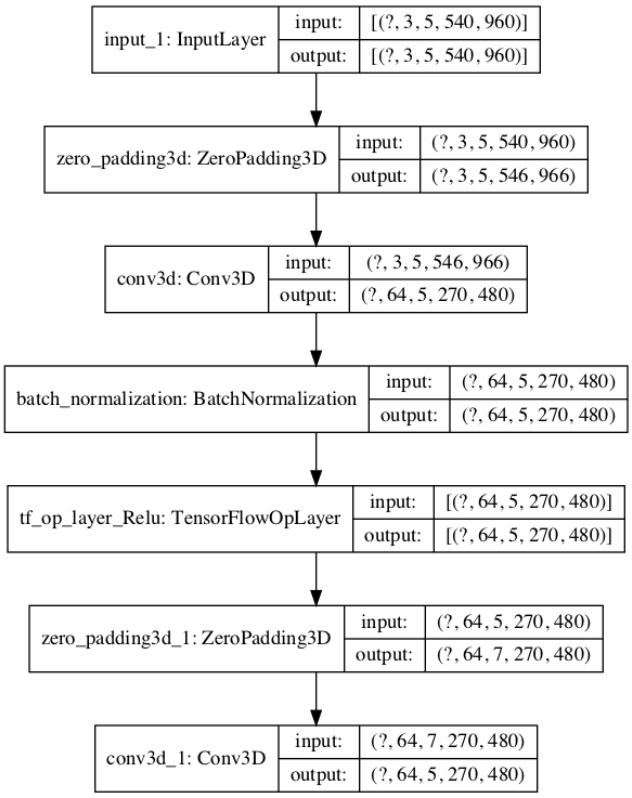


Figure 4.8: Example of a factorized 3D convolution in R(2+1)D.

Figure 4.9 shows how these convolutions are put together in layers with skip connections. These layers can either include or not include downsampling. If the layer is a downsampling layer as in this example the skip connection needs to downsample the input as well, otherwise it just contains an ordinary skip connection.

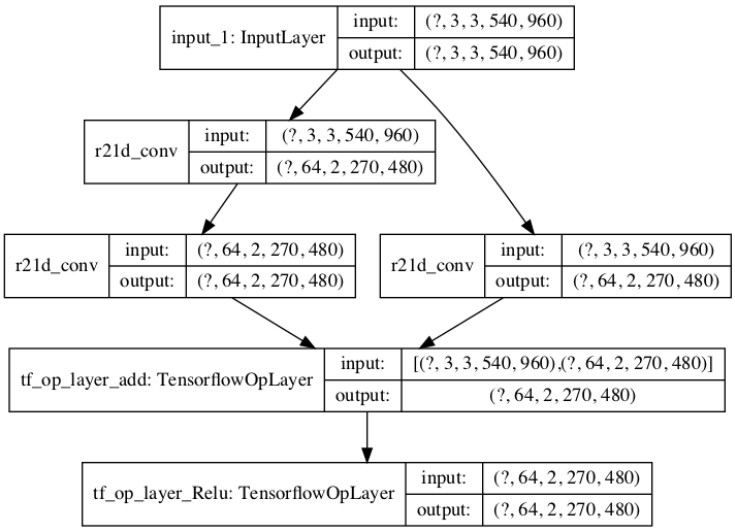


Figure 4.9: Example of a downsampling convolutional layer in R(2+1)D.

R(2+1)D Extended

This model builds upon and extends the R(2+1)D model. It uses one R(2+1)D convolution of size (3,7,7) and three downsampling R(2+1)D layers with kernel size (3,3,3) followed by further 2D convolutions to extract more spatial information.

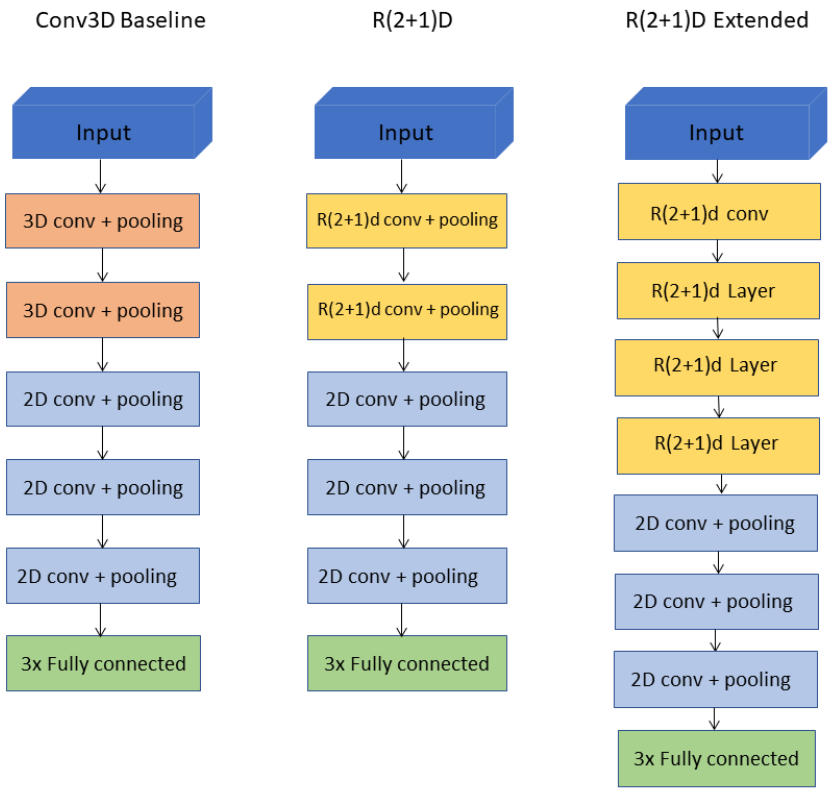


Figure 4.10: Visualization of the architecture of the models trained on video frames.

4.3.2 Feature Models

Conv3D features

This model is based on the architecture of the Conv3D model for frames but with only one 2D convolutions as the input is smaller.

Conv3D+LSTM

This model adds one more 3D convolution instead of the 2D convolution and uses an LSTM instead of fully connected layers as classifier.

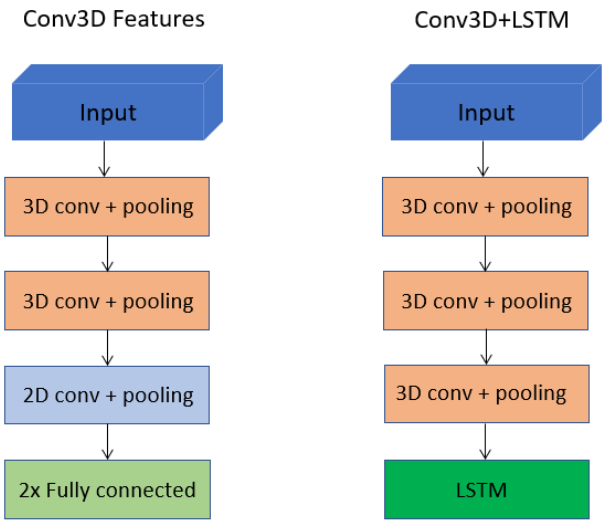


Figure 4.11: Visualization of the architecture of the models trained on player position features.

4.4 Experimental Setup

Here we will describe how the trained models have been evaluated and what experiments have been performed.

4.4.1 Computation & Tools

We have exclusively been running our trainings and tests on AWS, using the g4dn.xlarge instances running *Deep Learning AMI (Ubuntu 18.04) Version 27.0*. For reference when we discuss cost, the cost of using these instances are about 5 SEK per hour. It's also possible to pay considerably less, around 1.5 SEK per hour, by renting so called spot instances which can be interrupted during use.

We built our models using Keras 2.2.4-tf with TensorFlow 1.15.2 as backend. The reason for not using TensorFlow 2.X was that there were performance issues compared to 1.X. We believe that our work can easily be used with TensorFlow 2.X when these issues have been ironed out. The ffmpeg version we used was 3.4.6-0ubuntu0.18.04.1.

4.4.2 Online Evaluation

The standard procedure for evaluating neural networks is to evaluate them on a holdout test set to get a measure of how the network generalizes to unseen data. In our case this is not enough as this is a much easier problem than recognizing events in a stream of data. When testing on the test set the network sees the same amount of frames as during training and selects the most probable event whereas running live on a game means that the network outputs a stream of probabilities which does not trivially translate to classification since we need to hinder multi-tagging of the same event, handle uncertainty, etc.

In this thesis we handle this by creating a rule-based classifier using the stream of predictions as input which outputs start and stop times of events. We call this our online classifier. An event starts when the probability of said event surpasses a start threshold and it lasts until the probability falls under a lower exit threshold. New predictions are frozen for 5 seconds to further negate false positives. To be able to compare different such schemes and also different networks we decided that an event classification is regarded as a true positive if it overlaps the true label where we have set the temporal width of a true label to 20 seconds somewhat arbitrarily. If a true event has no overlapping predictions it is a false negative and a prediction that does not overlap any true events is a false positive. If a prediction would overlap more than one true label, only the maximum overlap is counted. If several predictions overlap the same true label, we merge predictions of the same label and count all of them which means that the same true label could be counted as both a true positive and a false positive simultaneously. From this we are able to form precision, recall and f1-scores as detailed in section 3.2.1.

The start and exit thresholds of predictions mentioned above are regarded as hyperparameters that we attempt to find optimal values for by performing a grid search over our validation games. Additional hyperparameters are moving average window size and the minimum length of an event, defined as the time passing between reaching the initial threshold and falling below the exit threshold. See algorithm 4.1 for a rough sketch of how this is calculated.

Algorithm 4.1 Grid Search Hyperparameters

```
for Threshold in (75, 80, 85, 90) do
    for Exit Threshold in (10, 20, 30) do
        for Minimum event time in (1, 2, 3) [s] do
            for Moving average window size in (1, 3, 5) do
                for Game in Validation Games do
                    Perform online classification
                    Calculate True positives, False positives and False Negatives
                Calculate combined F1-score for this set of hyperparameters
                if current F1 > max F1 then
                    max F1 = current F1
Store parameters corresponding to max F1 for use on test set ▷
```

Another aspect is that we can vary how often we make predictions, i.e. the stride we take in frames after making a prediction. In figure 4.12 the blue frames represent the first three frames extracted from a video when using three fps as temporal resolution and three frames as input. If a stride of one were to be used the green frames would be used to make the second prediction and then the first set of orange frames and so on. If a stride of two is used the models first sees the blue frames and then skips directly to the orange frames. Here it becomes quite obvious that using a higher stride dramatically decreases the number of predictions that needs to be computed. If the results are good enough while using a larger stride a lot of performance gains can be accounted for. Since different strides means that the classifier sees quite different inputs, we perform separate grid searches per stride and the results can be seen in appendix B. We will evaluate our models using strides from 1 to 25 with steps of 6. To reduce the time needed to perform these experiments and also for frame models to be more comparable to feature models we use video downsampled to 12.5 fps and therefore the actual stride in frames is doubled for frame models.

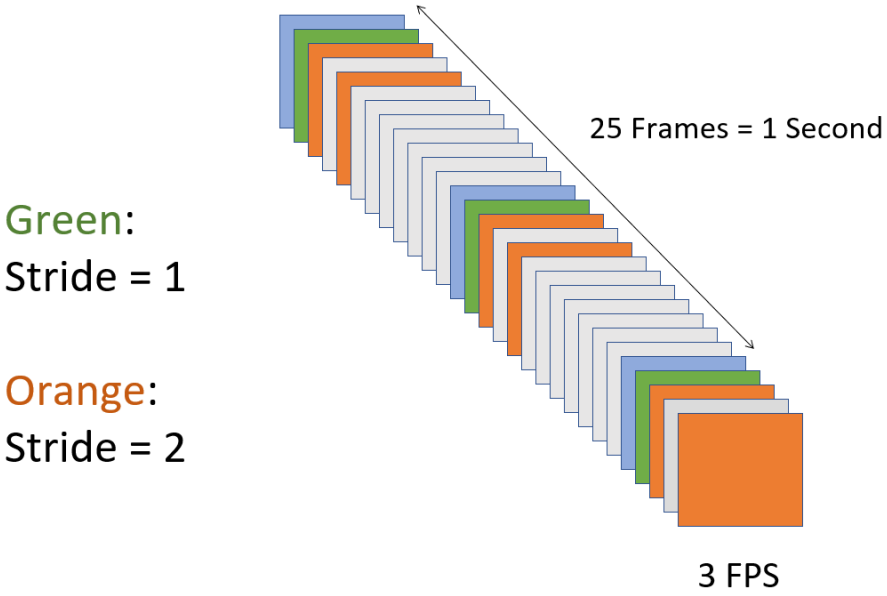


Figure 4.12: An illustration of which frames are used as input to the network depending on the stride.

4.4.3 Experiments

The experiments we carry out mean to compare a few model architectures on different sampling rates and temporal resolution. We will train models that see images sampled at 1, 3 and 5 frames per second and trained on 3 and 5 stacked frames. All settings have 9 frames that is available for temporal jittering described in section 4.2.4. In table 4.3 you can see what the different settings means in terms of what the network will see during training. Both frame and feature models will be trained and evaluated using the same settings according to algorithm 4.2 and the total number of parameters for each model and setting is found in table 4.4.

#Frames	FPS	Input Timespan (s)	Total Timespan (s)
3	1	3	5
	3	1	1.67
	5	0.6	1
5	1	5	9
	3	1.67	3
	5	1	1.8

Table 4.3: How FPS and number of frames affect the total amount of time the models see during training.

Frame Models	**Frames**	**Num Parameters**
Conv3D Baseline	3	2,731,013
	5	2,881,925
R(2+1)D	3	2,642,501
	5	2,683,461
R(2+1)D Extended	3	7,996,357
	5	7,996,357
Feature Models	**Frames**	**Num Parameters**
Conv3D Features	3	917,509
	5	1,068,421
Conv3D+LSTM	3	493,765
	5	679,813

Table 4.4: Number of parameters of the different models and settings.

Algorithm 4.2 Training Scheme

1: Initial Training - 100 epochs
2: Hard Negative Mining, see algorithm 4.3
3: Recalculate class weights and retrain the model on the updated dataset for 30 epochs
4: Hard Negative Mining, see algorithm 4.3
5: Recalculate class weights and retrain the model on the updated dataset for 30 epochs
6: Choose best weights based on validation result
7: Predict on test set
8: Grid-search for optimal online classifier parameters on validation games, see algorithm 4.1.
9: Online Evaluation on test games

Algorithm 4.3 Hard Negative Mining Algorithm

"*Negative examples are defined as non-events that are allowed to overlap each other but not real events. A 25 second block is used before and after real events.*"

for Game in Training & Validation Games **do**
    Extract 1000 Negative examples
    for Example in Negative examples **do**
        Use trained model to predict on example
        if Probability of any class $\geq .9$ **and** class != Negative **then**
            Add this example to the training/validation data

5 | Result

5.1 Empirical findings

In this section we present some observations we have made when working on this thesis.

5.1.1 Optimizer

We have noticed that the training of our models on video frames using the Adam optimizer does not converge in a reasonable time. SGD with a learning rate in the region of 0.001-0.02 have empirically been found to work the best, exact number depends on the model being trained. However, feature models converge just fine using ADAM with default parameters.

5.1.2 Models

The models used for the experiments have been modified and altered in various different ways and those who are presented are those that were able to learn the training set and also generalize to validation. Some architectures we tested but rejected are for example time-distributed 2D-convolutions and deeper 3D-convolutional networks. Since the deeper 3D networks did not achieve adequate results, neither with or without bottleneck structures, we decided that the trade-off between accuracy and computations were not worth it and therefore did not use the bottleneck architecture in our 3D convolutions.

5.2 Experiments

This section will present the results of the experiments performed. Results on the test set and the online classification will be reported for all models and settings described. One-vs-all ROC curves on test set and confusion matrices for both test set and online classification will be reported for the best performing setting on the online classifier using the lowest stride.

5.2.1 Frame models

Conv3D Baseline

Table 5.1 shows that the best performing setting based on F1 score on the test set was 5 fps and 3 frames. However, the online classifier that achieved the best F1 score with stride 2 was with 1 fps and 5 frames as seen in figure 5.1c.

Settings		Results		
#Frames	FPS	Precision	Recall	F1 score
3	1	0.8200	0.7693	0.7764
	3	0.8581	0.8729	0.8582
	5	0.9220	0.8302	0.8619
5	1	0.8353	0.7965	0.8017
	3	0.8774	0.8248	0.8415
	5	0.8867	0.8714	0.8600

Table 5.1: Results for the Baseline model on the test set evaluated with the different settings.

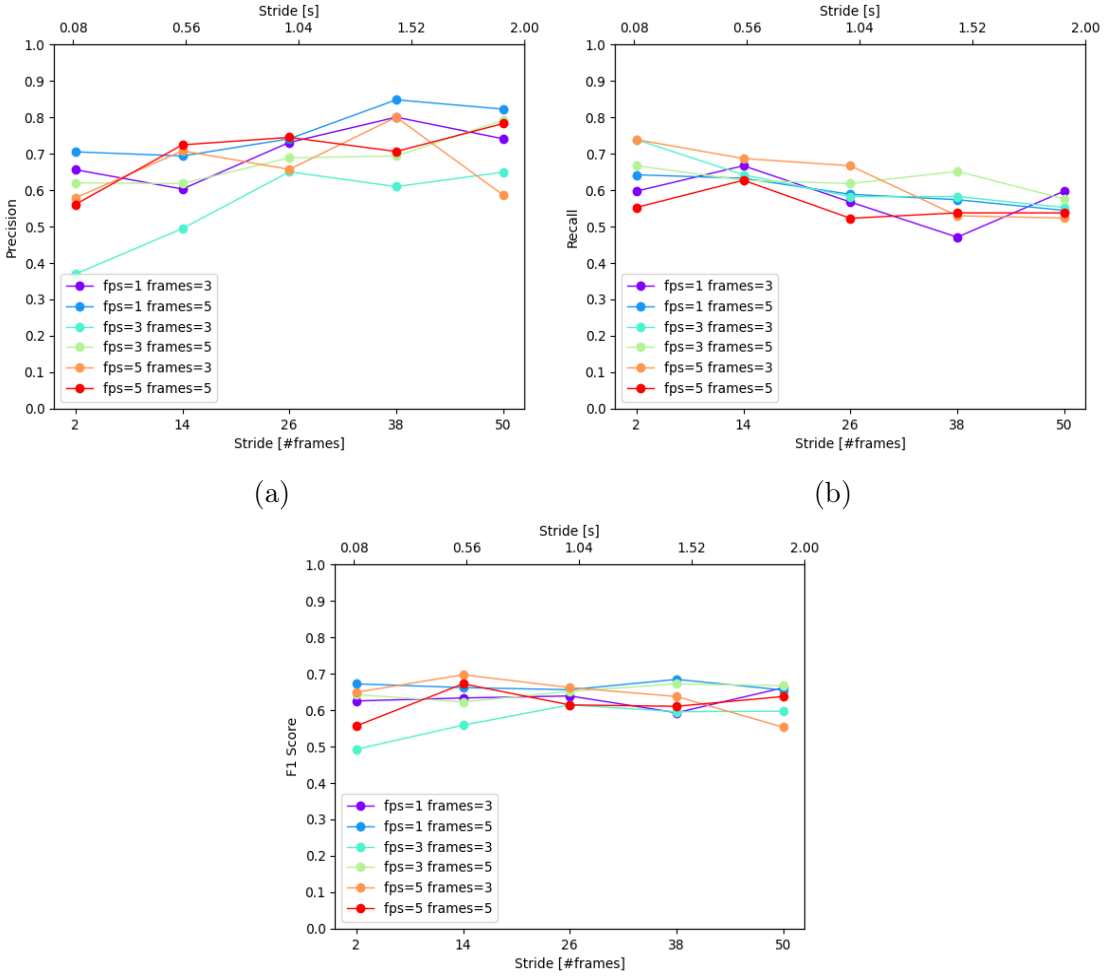
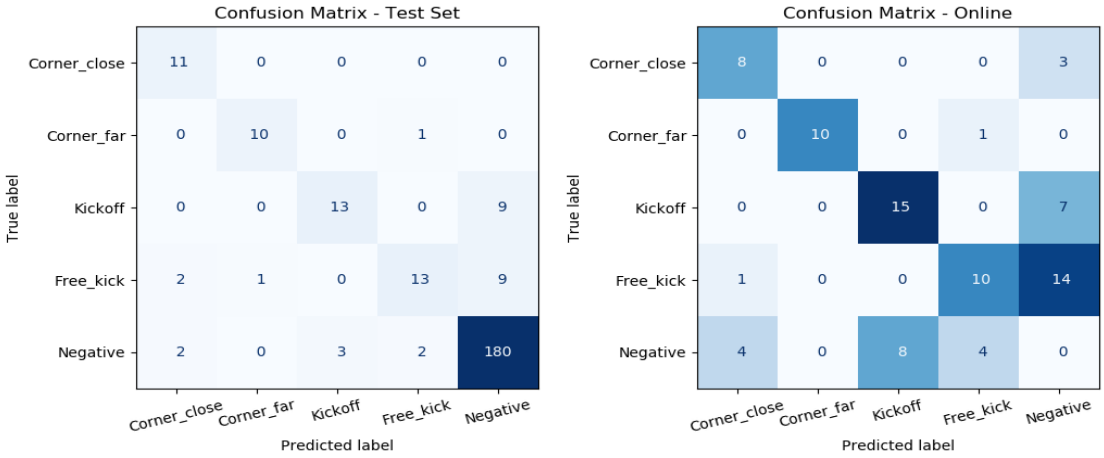


Figure 5.1: Precision, Recall, and F1 score for online classification with the Conv3D model for different settings of FPS, number of frames, and stride.



(a) Confusion matrix on the test set.

(b) Online Confusion matrix, note that true negatives are not counted.

Figure 5.2: Confusion matrices for the Conv3D model using 1 fps and 5 frames.

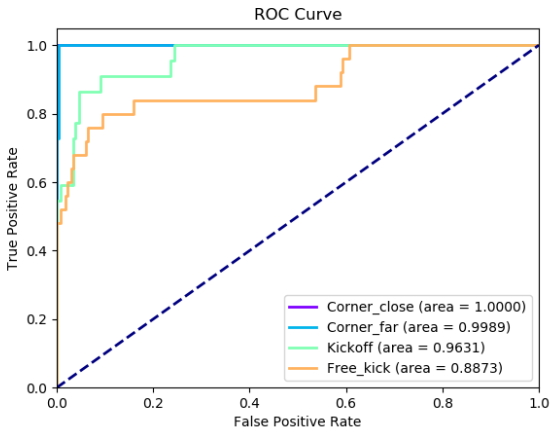


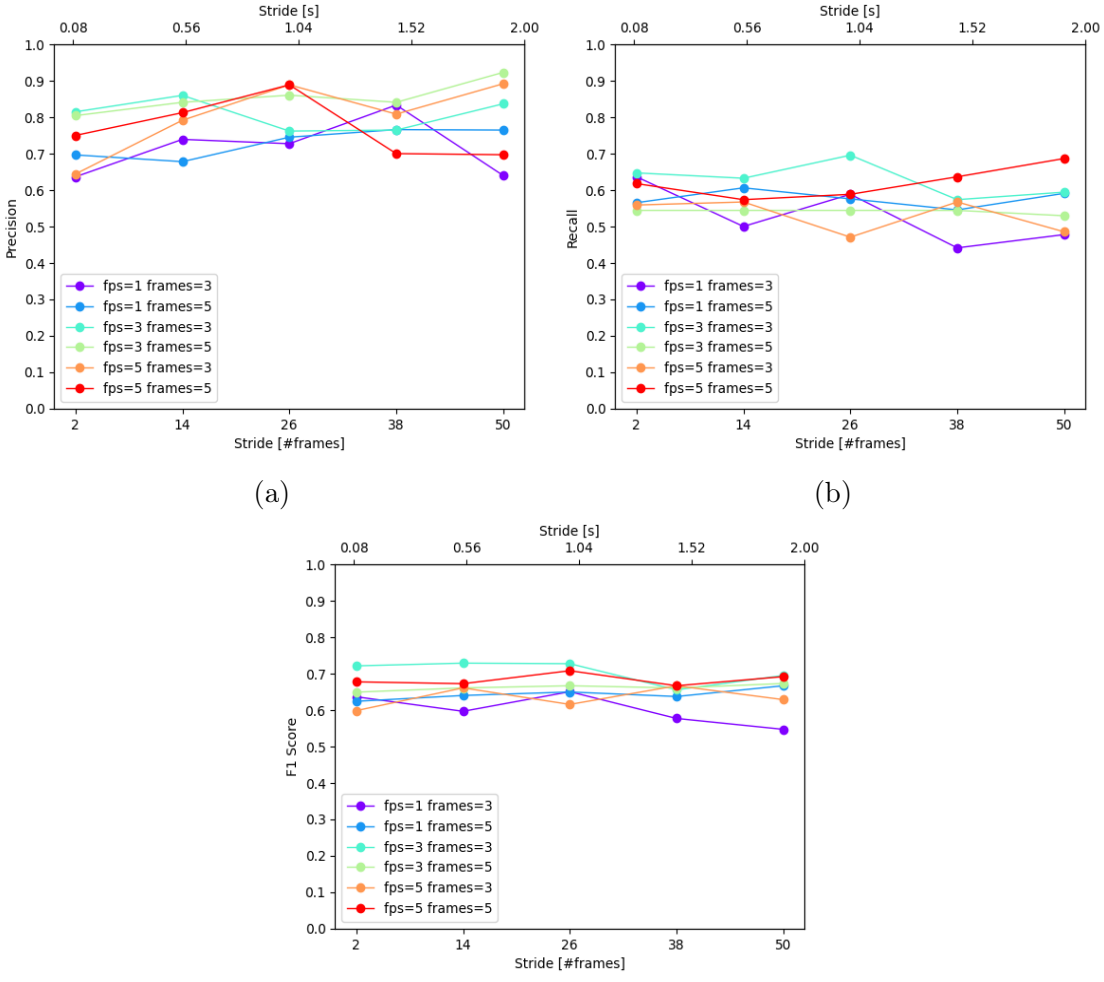
Figure 5.3: One-vs-all ROC curves on the test set for the Conv3D model using 1 fps and 5 frames.

R(2+1)D

Table 5.2 shows that the best performing setting based on F1 score on the test set was 5 fps and 5 frames. However, the online classifier that achieved the best F1 score with stride 2 was with 3 fps and 3 frames as seen in figure 5.4c.

Settings		Results		
#Frames	FPS	Precision	Recall	F1 score
3	1	0.8812	0.8932	0.8836
	3	0.8614	0.8569	0.8557
	5	0.8542	0.8482	0.8443
5	1	0.8630	0.8868	0.8674
	3	0.8842	0.8761	0.8765
	5	0.9181	0.8975	0.8990

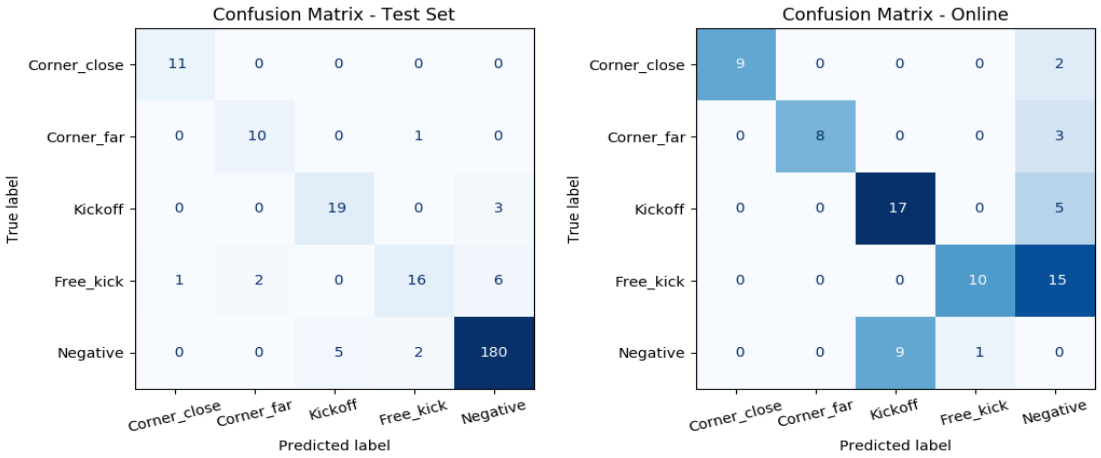
Table 5.2: Results for the R(2+1)D model on test set for different settings.



(a) (b)

(c)

Figure 5.4: Precision, Recall, and F1 score for online classification with the R(2+1)D model for different settings of FPS, number of frames, and stride.



(a) Confusion matrix on the test set.

(b) Online Confusion matrix, note that true negatives are not counted.

Figure 5.5: Confusion matrices for the R(2+1)D model using 3 fps and 3 frames.

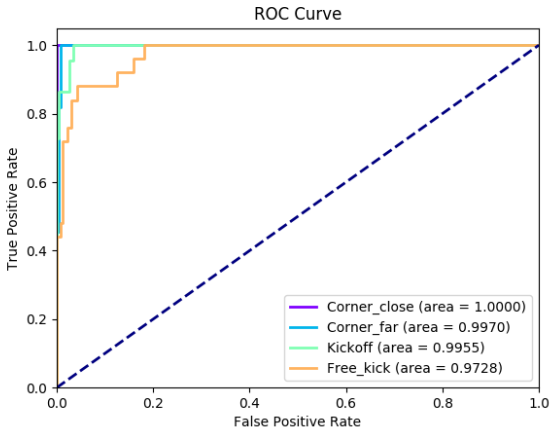


Figure 5.6: One-vs-all ROC curves on the test set for the R(2+1)D model using 3 fps and 3 frames.

R(2+1)D Extended

Table 5.3 shows that the best performing setting based on F1 score on the test set was 3 fps and 5 frames. Additionally, it also achieved the best F1 score with stride 2 during online classification as seen in figure 5.7c.

Settings		Results		
#Frames	FPS	Precision	Recall	F1 score
3	1	0.8025	0.8152	0.7992
	3	0.7547	0.7563	0.7402
	5	0.8145	0.9087	0.8532
5	1	0.8856	0.8841	0.8811
	3	0.8840	0.9103	0.8949
	5	0.8801	0.9092	0.8877

Table 5.3: Results for the R(2+1)D extended model on the test set for different settings.

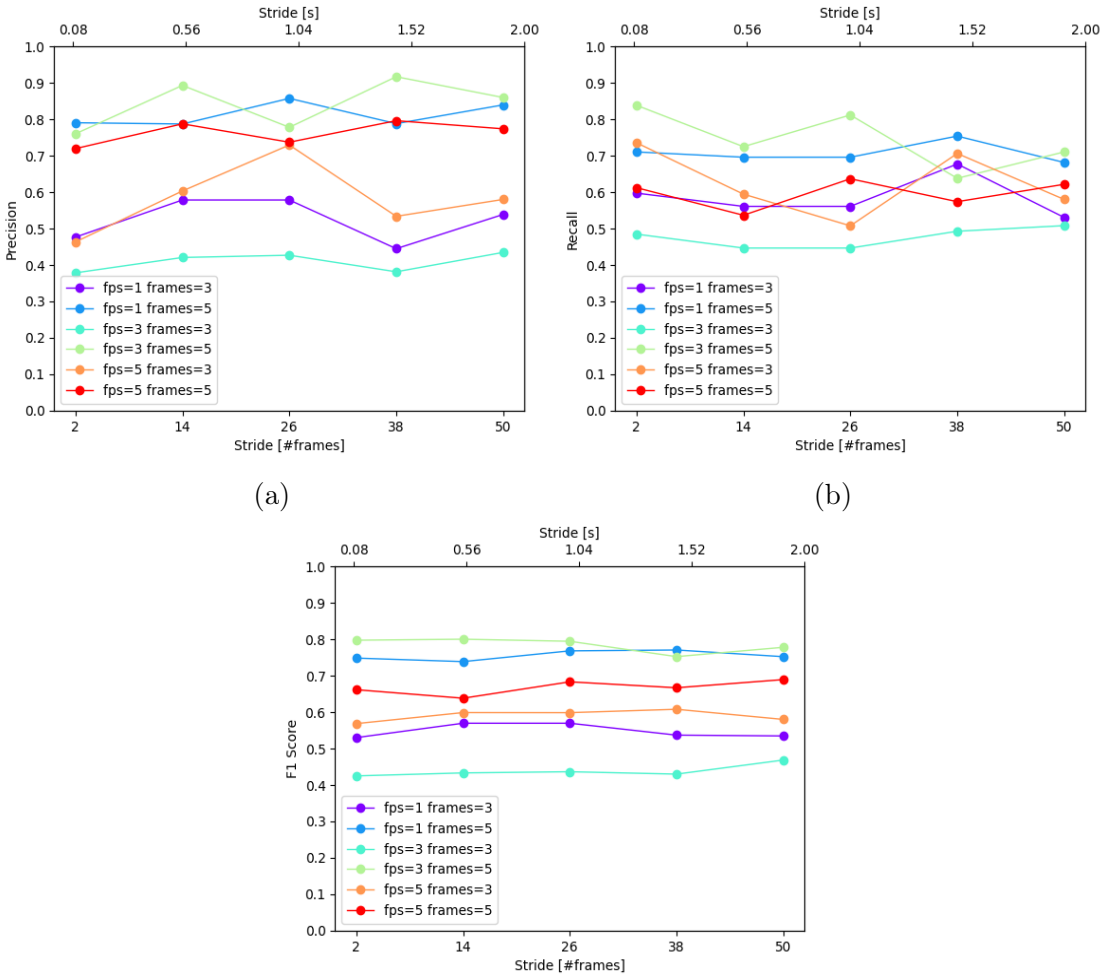
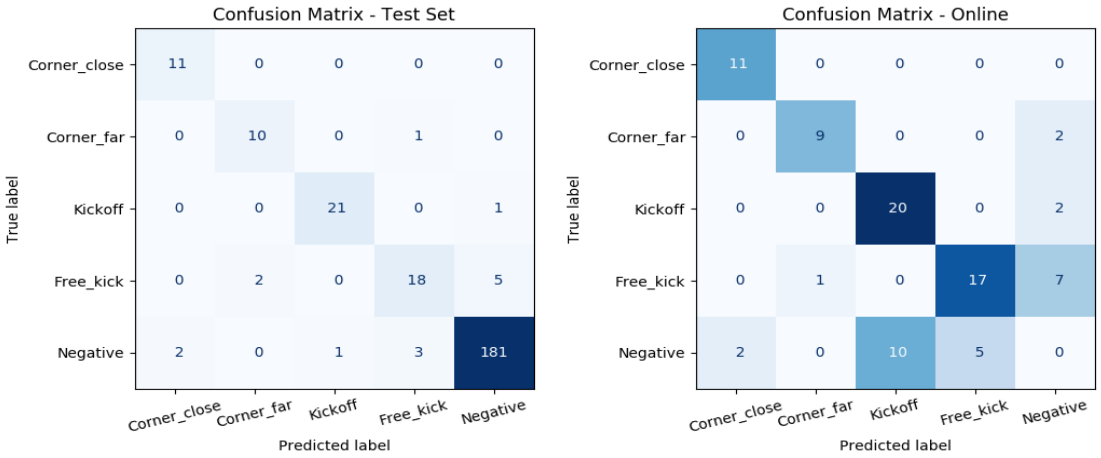


Figure 5.7: Precision, Recall, and F1 score for online classification with the R(2+1)D Extended model for different settings of fps, number of frames, and stride.



(a) Confusion matrix on the test set.

(b) Online Confusion matrix, note that true negatives are not counted.

Figure 5.8: Confusion matrices for the R(2+1)D Extended model using 3 fps and 5 frames.

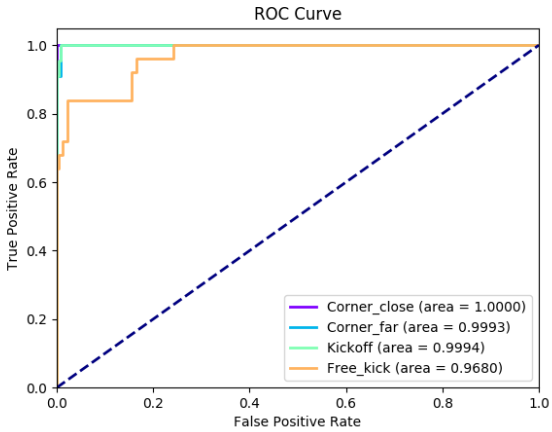


Figure 5.9: One-vs-all ROC curves on test set for the R(2+1)D Extended model using 3 fps and 5 frames.

5.2.2 Feature models

Conv3D Feature Model

Table 5.4 shows that the best performing setting based on F1 score on the test set was 1 fps and 3 frames. However, the online classifier that achieved the best F1 score with stride 1 was with 1 fps and 5 frames as seen in figure 5.10c.

Settings		Results		
#Frames	FPS	Precision	Recall	F1 score
3	1	0.9413	0.9397	0.9384
	3	0.9064	0.9044	0.8996
	5	0.9433	0.9077	0.9167
5	1	0.9147	0.9204	0.9139
	3	0.9397	0.9317	0.9319
	5	0.9463	0.9338	0.9337

Table 5.4: Results for the Conv3D feature model on the test set for different settings.

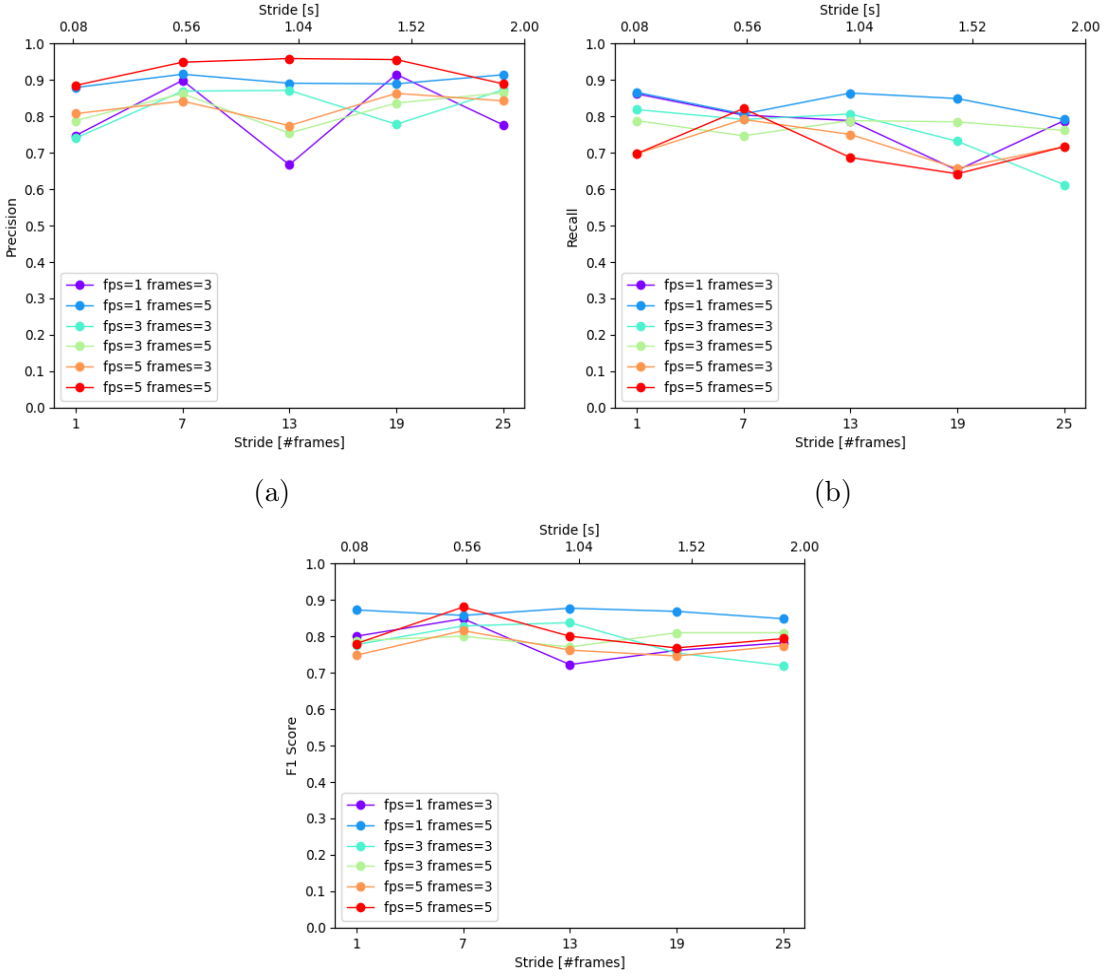
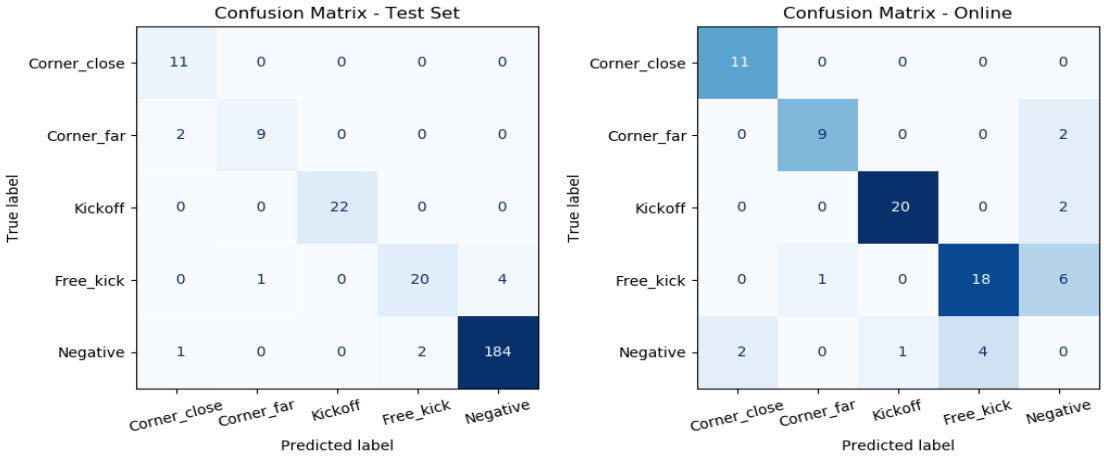


Figure 5.10: Precision, Recall, and F1 score for online classification with the Conv3D feature model for different settings of FPS, number of frames, and stride.



(a) Confusion matrix on the test set.

(b) Online Confusion matrix, note that true negatives are not counted.

Figure 5.11: Confusion matrices for the Conv3D feature model using 1 fps and 5 frames.

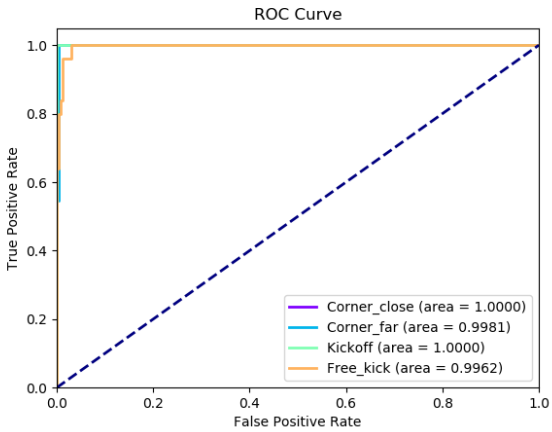


Figure 5.12: One-vs-all ROC curves on the test set for the Conv3D feature model using 1 fps and 5 frames.

Conv3D+LSTM

Table 5.5 shows that the best performing setting based on F1 score on the test set was 3 fps and 3 frames. However, the online classifier that achieved the best F1 score with stride 1 was with 1 fps and 5 frames as seen in figure 5.13c.

Settings		Results		
#Frames	FPS	Precision	Recall	F1 score
3	1	0.9273	0.8852	0.8972
	3	0.9347	0.9317	0.9283
	5	0.9106	0.8990	0.9006
5	1	0.9246	0.9284	0.9228
	3	0.9227	0.8942	0.9041
	5	0.8775	0.9183	0.8915

Table 5.5: Results for the Conv3D+LSTM model on the test set for different settings.

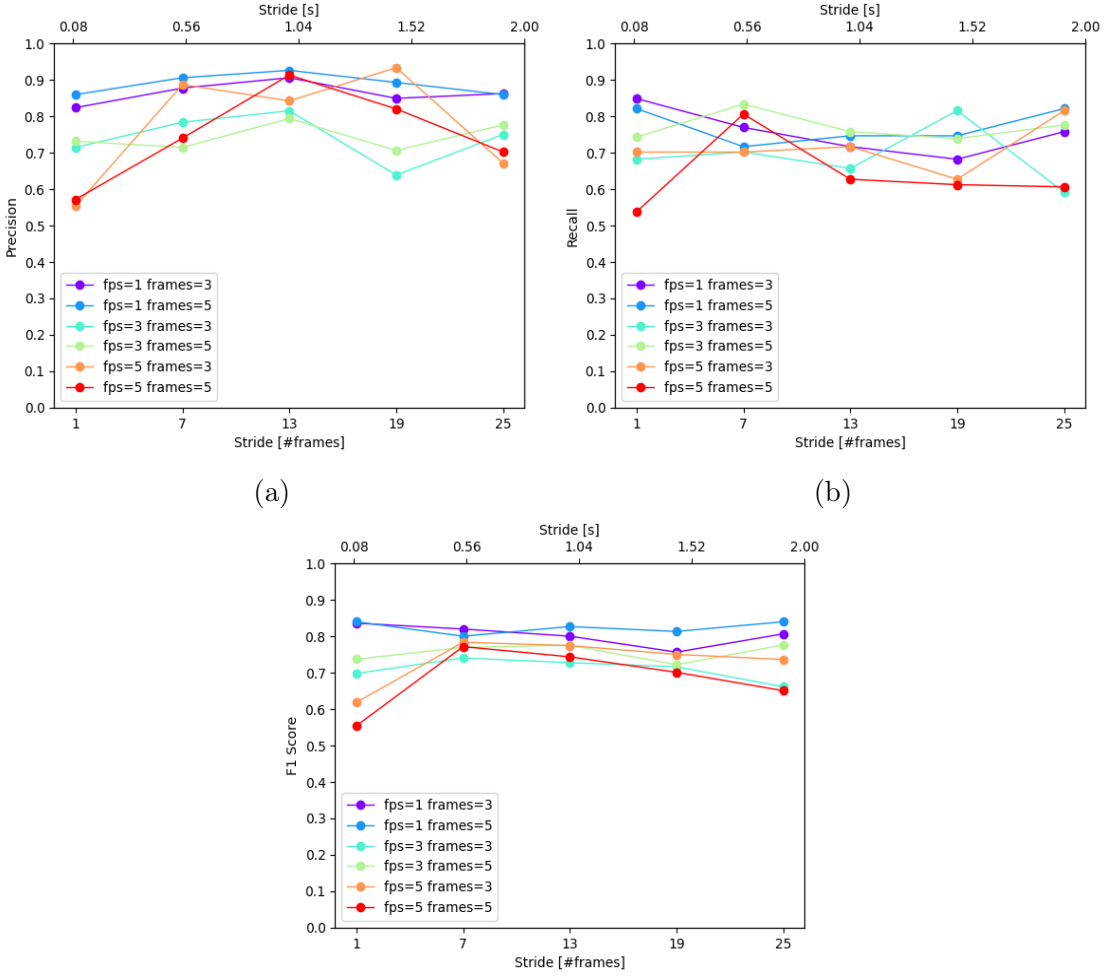
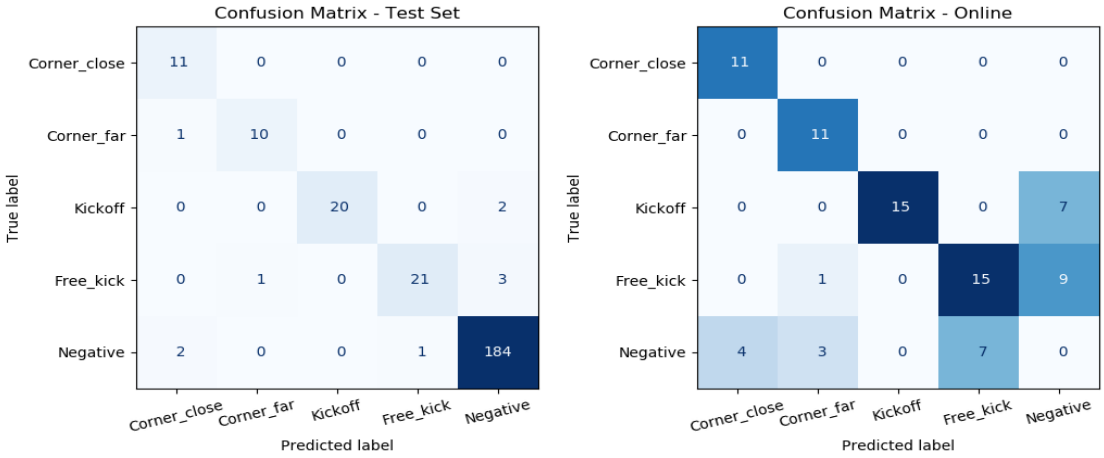


Figure 5.13: Precision, Recall, and F1 score for online classification with the Conv3D+LSTM feature model for different settings of fps, number of frames, and stride.



(a) Confusion matrix on the test set.

(b) Online Confusion matrix, note that true negatives are not counted.

Figure 5.14: Confusion matrices for the Conv3D+LSTM feature model using 1 fps and 5 frames.

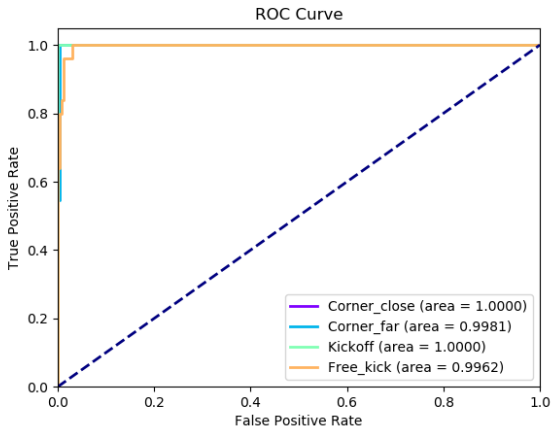


Figure 5.15: One-vs-all ROC curves on the test set for the Conv3D+LSTM feature model using 1 fps and 5 frames.

5.3 Analyze experiment

In this section we will take deeper look at the results of the experiments and how the models performed.

5.3.1 Computational Cost

Table 5.6 shows the difference in training time, including hard negative mining, between the models where the largest disparity can be found when comparing frame and feature models. As can be seen in figure 5.16 and 5.17 the inference

time is vastly decreased as well when using the feature data instead of video frames. Equation 5.1 estimates cost from inference time, note that the FPS in this equation is the framerate of the data predicted on and should not be confused with the framerate of the input to the models. Refer back to figure 4.12 and the associated paragraph.

$$
\begin{aligned}
T &= \text{Seconds in a game [s]} = 90 \cdot 60 = 5400 \\
P &= \text{Predictions per second [s}^{-1}\text{]} = \frac{\text{FPS}}{\text{Stride}} \\
C &= \text{AWS g4dn.xlarge cost per second [SEK s}^{-1}\text{]} = \frac{5}{3600} \\
I &= \text{Avg inference time per prediction [s]} \\
\text{Cost per game} &= T \cdot P \cdot C \cdot I = 5400 \frac{\text{FPS}}{\text{Stride}} \frac{5}{3600} I = 7.5 \frac{\text{FPS}}{\text{Stride}} I
\end{aligned} \tag{5.1}
$$

Model	**Frames**	**Average Training Time (100 epochs) [h]**
Conv3D Baseline	3	5
	5	8.5
R(2+1)D	3	5.5
	5	9
R(2+1)D Extended	3	10
	5	13.5
Conv3D Features	3	0.17
	5	0.3
Conv3D+LSTM	3	0.4
	5	0.4

Table 5.6: Comparison of training times including hard negative mining for different models.

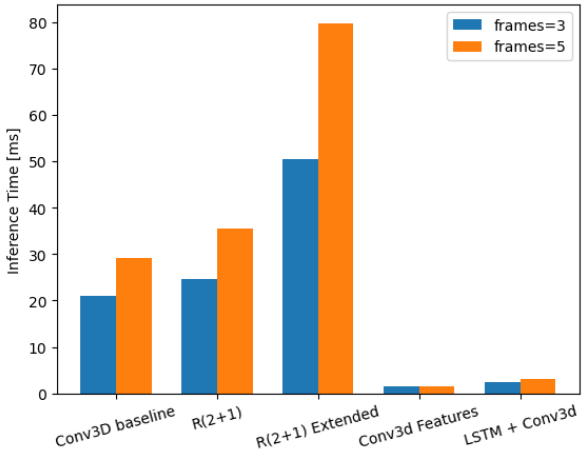


Figure 5.16: Comparison of average inference time per sample for frame and feature models.

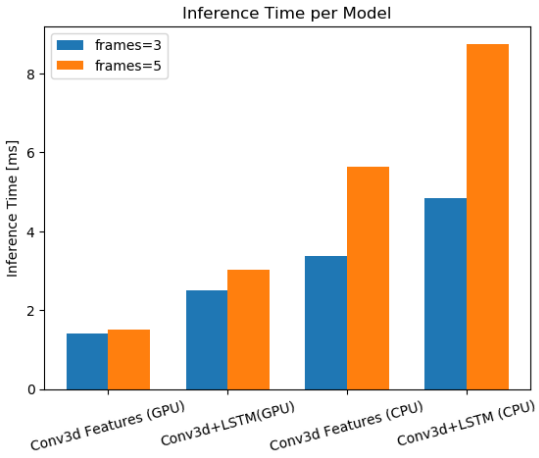


Figure 5.17: Comparison of average inference time per sample for feature models on GPU and CPU.

5.3.2 Hard Negative Mining

When analyzing hard negatives chosen by the different models we found that among the most used examples there were several that just as well could have been labeled as events since they were so similar to the actual event. For instance we found two examples, one in figure 5.18b, that were a kickoff but for some reason the players had to wait for the referee to start the game long enough that the time we blocked for hard negatives extraction (25 seconds before and after) was not enough. The same or similar situations were found to occur in the other examples discovered, another one is shown in figure 5.18a which depicts the build up to a free kick that also takes longer than the blocked amount of time.

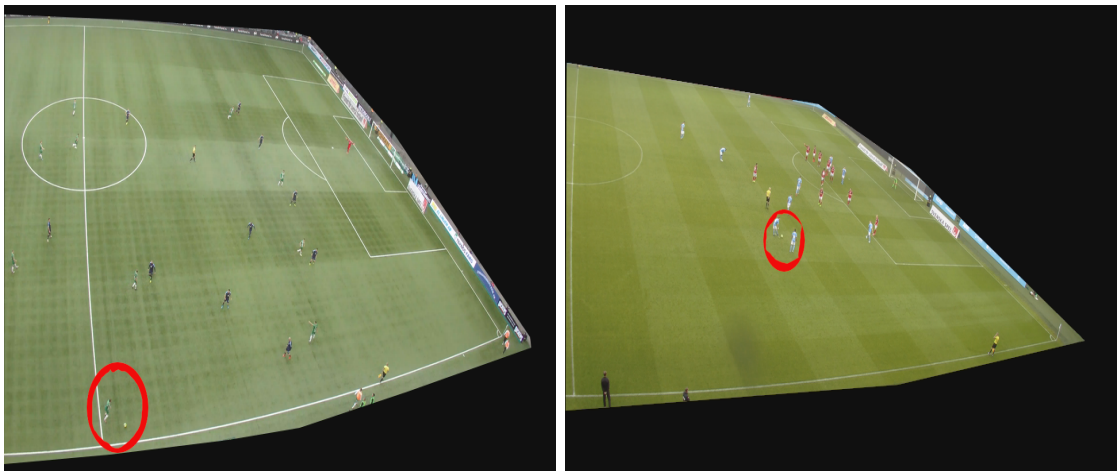
(a) Hard negative looking like a free kick. (b) Hard negative looking like a kickoff.

Figure 5.18: Two examples of hard negatives that are very similar to events.

5.3.3 Result on Test

When taking a closer look at what events our models had a hard time correctly classifying, we note that some events are more frequent than others. Figure 5.19 shows two such events where fig 5.19a shows a free kick in a situation that definitely resembles ordinary play and fig 5.19b shows a short free kick where the ball is passed to the closest player.

Apart from this we also found one example of a negative tag overlapping a corner in the same way as the examples from the hard negative mining in the analysis above.



(a) Free kick similar to ordinary play. (b) Short Free kick commonly labeled as negative.

Figure 5.19: Two hard examples in the test set.

Since the models are not perfect on the test set it is not reasonable to assume

that they would work flawlessly in an online setting. As can be seen in the comparison of the confusion matrices in figure 5.2, 5.5, 5.8, 5.11 and 5.14 it is evident that the online classification results in the model missing some events that were detected in the test set. One such event can be seen in figure 5.20 which shows a corner on the far side of the camera in Tele 2 Arena. This event was correctly classified by the Conv3D Feature model (using 1 fps and 5 frames) when predicting on the test set but missed by the online classifier. To get a better understanding on why this happens figure 5.21 shows the model's probability output for the different classes during 40 seconds, overlapping the corner shown in figure 5.20. Here it becomes clear that the model recognizes the corner in the set event interval but is never confident enough to reach the online threshold.

Figure 5.20: A corner on the far side on Tele 2 Arena.

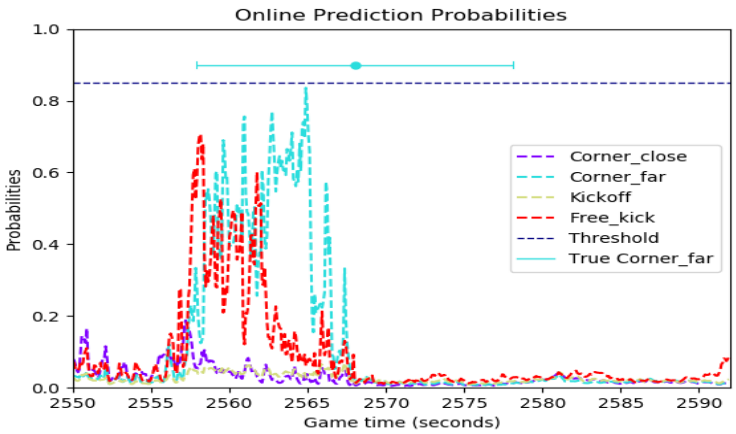


Figure 5.21: Probability distribution from the online classifier during 40 seconds, overlapping the corner shown in Figure 5.20.

6 | Discussion

One takeaway from this thesis is that using deep learning on video is still difficult. 4K video in 25 fps contains lots of redundant data but it is not obvious how many frames can be removed or how much images can be downsampled to reduce redundancy to speed up inference but not lose too much detail so that accuracy is degraded. When we started working, we hoped to evaluate our models on different image sizes as well and not just across different fps but we had to limit the scope to make running all tests feasible. Even if speed had not been a factor the sizes of inputs and models are still limited by GPU memory, for example our largest model can use a batch size of 4 when using 5 940×540 frames so there is not much room to increase the size of the input. As GPUs gets larger and faster this problem will of course be reduced but it is safe to say that when speed is key - smart computed features can still be very useful.

In figures 5.1, 5.4, 5.7, 5.10, and 5.13 where we show the online classification results of our models one might wonder why there is not a clear decline in performance when using large strides and skipping data. There are however a few explanations for that, the most prominent being that each stride has different hyperparameters in the online classifier, see appendix B. Since they are optimized individually on the validation set it is not trivial to compare different strides in that matter. Another explanation that is that while recall tends to decrease because more events are missed, precision tends to increase because the classifier has fewer chances to predict false positives and these effects seems to balance each other out. It might also be too short to stride with 2 seconds to notice a strong decline in performance since not even the models trained on 5 fps deteriorated significantly. This is partly explained by table 4.3 which tells us that with temporal jittering the time-spans of events that models see during training range from 1.8–9.0 seconds between 5 and 1 fps so in retrospect we probably should have tried some even larger strides. The important takeaway however is that it is possible to wait up to at least two seconds between predictions and still get comparable results.

Models trained on features got better results than frame models as evident in figures 5.1, 5.4, 5.7, 5.10, and 5.13c and tables 5.1, 5.2, 5.3, 5.4, and 5.5. This is most likely due to the fact that we only try to classify semi-stationary events

that depend heavily on the positions of players and how they interact compared to e.g. a card event that should not be detectable simply by looking at position and velocity of players. Since feature models are computationally cheap, see figure 5.17, and the feature data is already being generated in Spiideo's system this means that these events can be detected virtually for free.

Continuing on the subject of cost, it is very clear from figures 5.16 and 5.17 that our feature models are superior to the frame models as they are much cheaper to use and also get better results as discussed previously. Training times are also much shorter, see table 5.6, which is also a factor even if the inference time is more important. This comparison is of course a little skewed since we do not count any of the prepossessing steps included in extracting the feature data from video frames but as mentioned earlier this is already being done by Spiideo which makes the comparison more reasonable. It is also worth mentioning that because of the feature models being so much faster to train and evaluate, a more thorough optimization of the model structure and parameters could be performed. This has most likely contributed to the superior results of the feature models.

Another observation that can be made from the online results is that for frame models there is a much larger difference in results between different settings for the larger R(2+1)D Extended than for the smaller R(2+1)D and Conv3D models. In figure 5.7 we note that the difference in F1 score between the best and worst results with stride 2 is almost 0.4, both using 3 fps. Figures 5.1c and 5.4c show the largest difference to be about 0.18 and 0.12 for Conv3D and R(2+1)D respectively where the best and worst setting was 1 fps, 5 frames and 3 fps, 3 frames for Conv3D where for R(2+1)D it was 3 fps, 3 frames and 5 fps, 3 frames. For R(2+1)D Extended it is very clear that using 5 frames is better than 3, which is perhaps to be expected since it sees more of the frames surrounding events and compared to the other models it extracts more temporal information before moving to just 2D convolutions.

It can also be noted while comparing the results on the test set in tables 5.1, 5.2, 5.3, 5.4, and 5.5 to the online test results that the results on the test set are better than the online results and that it is not always the same settings that score best on both. The online test results are not as good because it is a much harder problem. In theory it should be possible to get the same results but during online testing there is a lot more non-events that needs to be avoided and build ups to events might be confusing for the network and cause it to classify something else and missing the real event that it might have found if the confusing part was ignored. Recall that if we classify an event then we do not classify any other events until the probability of the first event reaches an exit threshold and the 5-second-long freeze time has passed.

By looking at the ROC curves we see that the models generally are good at distinguishing between the different events. Though a clear difference can be

seen here as well showing that the frame models in figures 5.3, 5.6, and 5.9 are not as proficient as the feature models in 5.12 and 5.15. We can also note that Corners seems to be the easiest to generalize whereas Free Kicks not so unsuspectingly are the most difficult, scoring lowest in AUC for all models. This further confirms the argument that events where the variation in player locations is lower are easier to generalize.

The flaws found in our hard negative mining extraction is of course a problem that most likely increases the difficulty for the models to find good features. It should be noted however that we only found a few examples where this occurred in contrast to the twenty-nine thousand total negatives extracted. Also, these examples will all be relatively stationary since the ball is not in play. As mentioned we defined the ground truth timestamp of an event as the moment the ball is kicked, this means that a perfect model would be able to account for both the buildup and the movement following the ball being kicked. A perfect model could therefore in theory actually benefit from these examples as well.

Looking at events commonly misclassified in the test set, for example Figure 5.19, we can see that these examples borders on not even being defined as events according to how we defined free kicks in 4.1.3. We were interested in free kicks that could be defined as a set piece situation. Looking at theses individually 5.19a does not necessarily lead to anything as the ball is not kicked towards the goal. In 5.19b the ball is simply passed to the closest player, in this case the kick itself is not interesting but here it is rather a situation of a free kick in an interesting part of the pitch.

When evaluating the online classification results it is important to have in mind the result of the same model on the test set. If we for example look at Figure 5.11a which shows the confusion matrices for the best Conv3D Feature Model, we can see that it has four false negatives. Since predicting on the test set simply means taking the class with the highest probability, an example could theoretically have 21% and still be chosen (5 total classes). When transferring this to the online setting where we have set the lowest possible confidence threshold to 0.75 (see algorithm 4.1), an example that is not correctly predicted on the test set will in consequence never be correctly predicted online. If we keep this in mind and look at 5.11a again this actually means that the online classifier only has six additional false negatives over four games. One of them can be seen in figure 5.20 and 5.21 where the probability of a corner is accumulating over the course of a couple of seconds, but the threshold is never reached. A smarter online classifier could probably be able to find this event as well. The naive solution of simply lowering the threshold can fairly obviously be disregarded as that would produce false positives as well. The overall increase in false positives is of course a natural consequence when running the model on more unseen data.

As mentioned in section 2, most of the previous work we found on deep learning on sports video in general and football in particular has been done on broadcast video as the kind of footage we received from Spiideo is not open to the public. Furthermore, most video deep learning models use very low resolution and tries to classify many very different classes. These two factors made it difficult to find applicable research to base our work on, especially considering that a main goal of ours was to find models that are lightweight enough to be viable in production. We are however pleased that our frame models succeeded in learning features that are good enough to distinguish between the events we chose while not becoming so computationally expensive that they are infeasible to use. We were very pleasantly surprised by how well the feature-based models fared and the fact that they were even better than frame models is great news for Spiideo as they are so lightweight. With enough training data we believe that frame models should be able to at least be on par with feature models. It is obviously desirable be able to recognize more events than corners, free kicks, and kick offs and in the future work section we briefly discuss some ideas on how to do that.

7 | Future Work

We have evaluated and developed our algorithm under the assumption that it is done in an online setting. We therefore do not use the fact that if the future is known then it might be possible to make better decisions about which event to classify if one seems more probable than the first that reaches a threshold. This could be useful during offline processing where accuracy is more important than making predictions fast as images/feature data becomes available. One extension to a production system could be to mark uncertain events for review that can be used to improve the detector and also to find real events that were nearly classified.

We have thought of a few approaches for trying to improve the accuracy of our online classifier. While we use the prediction probability output from our models in a relatively simple way, one could imagine treating it as a time series classification problem where many methods can be applied. For example logistic regression or an RNN could be trained to output when to classify events, either a simple binary classifier that predicts timestamps for events or a more complex model that predicts start and stop times for events like our model.

Furthermore, the long lead times in training models with lots of data means that we have not explored hyperparameter optimization to a great extent and there might be accuracy gains to find with relatively small efforts if done properly. For example, we wanted to explore how Dropout layers would affect our frame models but we decided to focus more on the feature data. It would also be interesting to explore different label smoothing approaches, e.g. giving more weight to the negative class or a similar class than non-similar classes. Feature models that are much faster to train could most likely benefit from optimizing dropout probabilities, number of filters etc on the validation set.

When we grid search over our validation games to find optimal parameters for the classifier, we assume that they will generalize to the test set and also to new matches during production. However, our validation and test sets are quite small and this assumption might not hold. We believe that a larger amount of validation data is required to be able to properly tune the classifier. It could also be beneficial to search over a larger range of parameters or introduce new

ones, e.g. class-specific parameters. Additionally, more data from different weather conditions etc should be added to make the frame models more robust. Smart data augmentation, e.g. changing brightness of images to simulate different light conditions, could potentially also lead to the frame models being more generalized. Since the camera properties are known it is also possible to make projections of the images to simulate that the camera is in a different position which might lead to better generalization across arenas.

Hard negative mining is something that we believe is generally a good idea, and while we haven't explicitly evaluated the effects in this report we have observed that it reduces the number of false positives more than it makes true positives harder to detect. It might be better to do it by predicting on several entire games on the training set and extracting the hard examples there instead of sampling many random clips. This would take much longer than our approach but it is possible that it leads to better generalization to the online setting since examples that are hard to see online are used for training.

Extending our work to other events might not be trivial in general, but we have some ideas for how to approach some of them. We believe that penalties could be added and classified with little to no effort if more games are added to the data so that there are enough samples to learn from. We tried classifying shots but the models were thrown off by the variety in annotations and the manual work required to re-annotate them was much larger than with other events but there might be an opportunity there. Card events could possibly be found if the player tracking is extended to track the referee and then cropped images of the referee are used to classify whether a card is shown or not. Substitutions should be easy to find by using a small model on a cropped image of the area where the substitution board is shown. The training data should then include images of when the board is used to indicate stoppage time to reduce false positives. Goals are probably the hardest event of all and we are not sure if there is a good way to detect them without tracking the position of the ball which might be expensive. However, since our models often find kickoffs it is possible to deduce approximately when goals are scored.

Throughout this work we have always used F1 score as our main evaluation metric to get results that neither have too many false positives nor false negatives. However, we can imagine a system where two models are used together where the first is trained to maximize recall and the second model maximizes precision. The first model is then used to find event proposals for the second model which gives the final verdict. This opens up the opportunity to train heavier models with very high accuracy as using it on the proposals is a lot less work than on an entire game.

8 | Bibliography

[1] Dan Jones. Annual review of football finance 2019. https://www2.deloitte.com/uk/en/pages/sports-business-group/articles/annual-review-of-football-finance.html. Accessed: 2020-02-18.

[2] Grand View Research. Sports analysis market size, share & trends. https://www.grandviewresearch.com/industry-analysis/sports-analytics-market. Accessed: 2020-05-24.

[3] Victor Holman. Sports analytics methods - machine learning analysis. https://www.agilesportsanalytics.com/sports-analytics-machine-learning-analysis/. Accessed: 2020-05-24.

[4] Haohao Jiang, Yao Lu, and Jing Xue. Automatic soccer video event detection based on a deep neural network combined cnn and rnn. pages 490–494, 11 2016.

[5] Grigorios Tsagkatakis, Mustafa Jaber, and Panagiotis Tsakalides. Goal!! event detection in sports video. *Electronic Imaging*, 2017(16):15–20, 01 2017.

[6] Heng Wang, Alexander Kläser, Cordelia Schmid, and Liu Cheng-Lin. Action recognition by dense trajectories. *IEEE Conference on Computer Vision & Pattern Recognition*, 06 2011.

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

[8] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *CoRR*, abs/1406.2199, 2014.

[9] Laura Sevilla-Lara, Yiyi Liao, Fatma Güney, Varun Jampani, Andreas Geiger, and Michael J. Black. On the integration of optical flow and action recognition. *Pattern Recognition*, page 281–297, 2019.

[10] Joe Yue-Hei Ng, Matthew J. Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4694–4702, 2015.

[11] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks, 2014.

[12] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 07 2017.

[13] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. *CoRR*, abs/1711.11248, 2017.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[16] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. Video classification with channel-separated convolutional networks. 04 2019.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[20] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[22] Konstantin Sozykin, Adil Khan, Stanislav Protasov, and Rasheed Hussain. Multi-label class-imbalanced action recognition in hockey videos via 3d convolutional neural networks. 09 2017.

[23] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

[24] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? *CoRR*, abs/1906.02629, 2019.

[25] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145 – 1159, 1997.

[26] Robin Gustavsson. Region of interest prediction in football using artificial neural networks, 2017. Student Paper.

[27] Ken Turkowski. *Filters for Common Resampling Tasks*, page 147–165. Academic Press Professional, Inc., USA, 1990.

[28] Irhum Shafkat. R2plus1d-pytorch. `https://github.com/irhum/R2Plus1D-PyTorch`, 2018.

A | Model Setup

A.1 Conv3D Baseline

Layer	Kernel size	Stride	Filters/Units	Activation
Conv3D	(N, 3, 3)	(1,3,3)	64	ReLU
MaxPool3D	(1, 2, 2)	(1,2,2)		
Conv3D	(N, 3, 3)	(1,3,3)	128	ReLU
MaxPool3D	(N,3,3)	(N,2,2)		
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Conv2D	(3,3)	(1,1)	512	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Dense			256	ReLU
Dense			128	ReLU
Dense			64	ReLU
Dense			Classes	Softmax

Table A.1: Full model configuration for the Conv3D Baseline model.

A.2 R(2+1)D

Layer	**Kernel size**	**Stride**	**Filters/Units**	**Activation**
R(2+1)D	(N, 3, 3)	(1,3,3)	64	ReLU
MaxPool3D	(1, 2, 2)	(1,2,2)		
R(2+1)D	(N, 3, 3)	(1,3,3)	128	ReLU
MaxPool3D	(N,3,3)	(N,2,2)		
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Conv2D	(3,3)	(1,1)	512	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Dense			256	ReLU
Dense			128	ReLU
Dense			64	ReLU
Dense			Classes	Softmax

Table A.2: Full model configuration for the R(2+1)D model.

A.3 R(2+1)D Extended

Layer	Kernel size	Stride	Filters/Units	Activation
R(2+1)D Conv	(3, 7, 7)	(1,2,2)	64	ReLU
R(2+1)D Layer	(3,3,3)	(2,2,2)	64	ReLU
R(2+1)D Layer	(3, 3, 3)	(2,2,2)	128	ReLU
R(2+1)D Layer	(3,3,3)	(2,2,2)	256	ReLU
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Conv2D	(3,3)	(1,1)	512	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Dense			256	ReLU
Dense			256	ReLU
Dense			128	ReLU
Dense			Classes	Softmax

Table A.3: Full model configuration for the R(2+1)D Extended model.

A.4 Feature B3D

Layer	Kernel size	Stride	Filters/Units	Activation
Conv3D	(frames,3,3)	(1,1,1)	64	ReLU
MaxPool3D	(1,2,2)	(1,2,2)		
Conv3D	(frames,3,3)	(1,1,1)	128	ReLU
MaxPool3D	(frames,2,2)	(frames,2,2)	256	ReLU
Conv2D	(3,3)	(1,1)	256	ReLU
BatchNorm				
MaxPool2D	(2,2)	(2,2)		
Dense			256	ReLU
Dropout			0.5	
Dense			128	ReLU
Dropout			0.5	
Dense			Classes	Softmax

Table A.4: Full model configuration for the Feature B3D model.

A.5 Conv3D+LSTM

Layer	**Kernel size**	**Stride**	**Filters/Units**	**Activation**
Conv3D	(frames,3,3)	(1,1,1)	64	ReLU
BatchNorm				
MaxPool3D	(1,2,2)	(1,2,2)		
Conv3D	(frames,3,3)	(1,1,1)	128	ReLU
BatchNorm				
MaxPool3D	(1,2,2)	(1,2,2)		
Conv3D	(frames,3,3)	(1,1,1)	256	ReLU
BatchNorm				
MaxPool3D	(1,2,2)	(1,2,2)		
TimeDistributed(Flatten)				
LSTM			64	tanh
Recurrent Dropout			0.5	
Dense			Classes	Softmax

Table A.5: Full model configuration for the Conv3D+LSTM model.

B | Hyperparameters

Model	FPS \ Stride	1	7	13	19	25
Conv3d	1	0.85, 0.1, 20, 3	0.75, 0.1, 10, 5	0.75, 0.2, 30, 1	0.75, 0.1, 10, 3	0.75, 0.1, 20, 1
	3	0.85, 0.3, 30, 5	0.85, 0.3, 30, 5	0.85, 0.1, 10, 5	0.8, 0.1, 10, 5	0.75, 0.1, 10, 5
	5	0.85, 0.1, 10, 5	0.8, 0.1, 10, 3	0.85, 0.1, 20, 1	0.75, 0.3, 10, 3	0.75, 0.1, 20, 1
R(2+1)D	1	0.8, 0.3, 30, 1	0.75, 0.2, 30, 5	0.75, 0.1, 10, 3	0.8, 0.2, 20, 1	0.75, 0.1, 10, 3
	3	0.85, 0.1, 30, 5	0.8, 0.2, 30, 3	0.75, 0.3, 10, 3	0.85, 0.1, 10, 1	0.85, 0.2, 20, 1
	5	0.85, 0.2, 30, 5	0.8, 0.2, 30, 5	0.75, 0.2, 10, 5	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3
R(2+1)D Extended	1	0.85, 0.1, 20, 3	0.75, 0.1, 10, 5	0.75, 0.1, 10, 5	0.75, 0.2, 10, 3	0.85, 0.1, 10, 1
	3	0.85, 0.1, 30, 5	0.75, 0.1, 10, 5	0.75, 0.1, 10, 3	0.8, 0.1, 10, 1	0.8, 0.1, 20, 1
	5	0.85, 0.2, 20, 5	0.85, 0.1, 30, 3	0.85, 0.2, 10, 3	0.85, 0.1, 20, 1	0.85, 0.1, 20, 1
Conv3d Features	1	0.85, 0.1, 20, 3	0.75, 0.1, 10, 5	0.75, 0.2, 30, 1	0.75, 0.1, 10, 3	0.75, 0.1, 20, 1
	3	0.8, 0.1, 20, 5	0.75, 0.1, 10, 5	0.75, 0.1, 10, 3	0.8, 0.2, 20, 1	0.75, 0.1, 10, 3
	5	0.75, 0.1, 20, 5	0.8, 0.1, 10, 1	0.75, 0.1, 20, 1	0.75, 0.1, 20, 1	0.75, 0.1, 10, 1
Conv3d + LSTM Features	1	0.85, 0.1, 20, 5	0.8, 0.1, 10, 5	0.85, 0.2, 10, 3	0.75, 0.1, 10, 3	0.85, 0.1, 20, 1
	3	0.85, 0.1, 20, 5	0.75, 0.2, 10, 5	0.75, 0.1, 10, 3	0.75, 0.1, 20, 1	0.8, 0.1, 20, 1
	5	0.85, 0.2, 20, 5	0.8, 0.1, 10, 5	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3	0.85, 0.1, 10, 1

Table B.1: Online classifier hyperparameters for the models using 3 frames. Each cell contains values for Event Probability Threshold, Exit Threshold, Minimum Event Time and MA Window Size in that order. Note that for frame models the stride in frames is doubled since predictions are done on 12.5 FPS.

Model	Stride / FPS	1	7	13	19	25
Conv3d	1	0.85, 0.1, 30, 5	0.8, 0.1, 30, 5	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3
	3	0.85, 0.2, 30, 1	0.8, 0.3, 30, 3	0.8, 0.3, 10, 3	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3
	5	0.85, 0.2, 30, 5	0.8, 0.1, 10, 5	0.75, 0.1, 10, 5	0.75, 0.2, 10, 3	0.75, 0.1, 10, 3
R(2+1)D	1	0.85, 0.1, 30, 5	0.75, 0.1, 10, 5	0.75, 0.1, 10, 5	0.75, 0.1, 10, 5	0.75, 0.1, 10, 3
	3	0.85, 0.1, 30, 5	0.8, 0.1, 10, 5	0.75, 0.1, 10, 5	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3
	5	0.85, 0.1, 10, 3	0.75, 0.2, 10, 5	0.75, 0.1, 10, 3	0.8, 0.1, 10, 1	0.75, 0.1, 20, 1
R(2+1)D Extended	1	0.85, 0.3, 20, 1	0.8, 0.3, 30, 1	0.75, 0.2, 10, 3	0.75, 0.2, 20, 1	0.8, 0.3, 10, 1
	3	0.75, 0.1, 10, 5	0.8, 0.1, 30, 1	0.75, 0.1, 30, 1	0.8, 0.1, 10, 1	0.75, 0.1, 20, 1
	5	0.85, 0.1, 10, 1	0.85, 0.1, 10, 1	0.75, 0.3, 30, 1	0.8, 0.1, 10, 1	0.75, 0.2, 20, 1
Conv3d Features	1	0.85, 0.1, 30, 5	0.8, 0.1, 30, 5	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3
	3	0.8, 0.2, 10, 3	0.75, 0.2, 10, 3	0.75, 0.3, 20, 1	0.75, 0.2, 10, 1	0.75, 0.2, 10, 1
	5	0.85, 0.1, 30, 3	0.75, 0.1, 10, 5	0.8, 0.1, 10, 3	0.75, 0.1, 10, 3	0.8, 0.1, 20, 1
Conv3d + LSTM Features	1	0.85, 0.2, 30, 5	0.85, 0.3, 30, 5	0.75, 0.1, 10, 5	0.8, 0.3, 10, 3	0.85, 0.1, 20, 1
	3	0.85, 0.1, 20, 5	0.75, 0.1, 10, 3	0.75, 0.1, 10, 3	0.8, 0.1, 20, 1	0.85, 0.1, 10, 1
	5	0.8, 0.3, 30, 3	0.85, 0.1, 20, 3	0.85, 0.1, 10, 3	0.75, 0.1, 10, 3	0.85, 0.1, 20, 1

Table B.2: Online classifier hyperparameters for the models using 5 frames. Each cell contains values for Event Probability Threshold, Exit Threshold, Minimum Event Time and MA Window Size in that order. Note that for frame models the stride in frames is doubled since predictions are done on 12.5 FPS.

Master's Theses in Mathematical Sciences 2020:E36
ISSN 1404-6342

LUTFMA-3416-2020

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/