Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Football shot detection using convolutional neural networks

JONATHAN ERIKSON

Football shot detection using convolutional neural networks

JONATHAN ERIKSON

Degree Programme in Computer Science and Engineering
Date: September 24, 2024

Supervisors: Stefano Markidis, Alex Jorge
Examiner: Bo Peng
School of Electrical Engineering and Computer Science
Host company: Football Analytics Sweden AB
Swedish title: Detektion av skott i fotboll med hjälp av convolutional neural networks

© 2024 Jonathan Erikson

Abstract

The application of computer vision in football is becoming more and more attractive. Computer vision can track players and actions allowing statistics and analysis of the sport. This study aims to recognize shots in football video data using computer vision.

Recognizing actions in video data has been explored using various techniques. Convolutional Neural Networks, CNNs, have dominated computer vision by proving great at understanding image data. However, extending CNNs to understand video data is a difficult task. In recent years 3D CNNs have been developed to understand the temporal information in video data, showing promising results in the field of action recognition.

In this study, 3D CNNs are compared to standard CNNs in their ability to predict shots in football video data. Football Analytics Sweden AB collects and owns the data which consist of short clips of shots and non-shots.

3D CNNs were found to outperform 2D CNNs in recognizing shots, achieving an accuracy of 85% on the test dataset. The results show that 3D CNNs have the potential to distinguish between different actions in football video data and could be used in further analysis of football matches using computer vision.

Keywords

Machine Learning, Computer Vision, Action Detection

Sammanfattning

Användning av maskininlärning och datorseende inom fotboll blir alltmer attraktiv. Datorseende möjliggör detektion av spelare och händelser, vilket kan användas för analys och statistik inom sporten. Denna studie avser detektion av skott i fotbollsvideodata med hjälp av datorseende.

Att känna igen handlingar i videodata har undersökts med olika tekniker. Convolutional Neural Networks, CNNs, har dominerat datorseende genom att vara mycket effektiva på att förstå bilddata. Att utöka CNNs för att förstå videodata är dock en svår uppgift. Under de senaste åren har 3D-CNN utvecklats för att förstå hur datan ändras över tid i videodata, och har visat lovande resultat inom detektering av handlingar.

I denna studie jämförs 3D-CNN med vanliga CNN i deras förmåga att detektera skott i fotbollsvideodata. Football Analytics Sweden AB samlar in och äger data som består av korta klipp av skott och icke-skott.

3D-CNN visade sig överträffa 2D-CNN i att känna igen skott och uppnådde en noggrannhet på 85% på testdatan. Resultaten visar att 3D-CNN har potential att skilja mellan olika handlingar i fotbollsvideodata och kan användas i vidare analys av fotbollsmatcher med hjälp av datorseende.

Nyckelord

Maskininlärning, Datorseende, Handlingsdetektion

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

In recent years, the analysis of shots in football has gained significant popularity, particularly with the advent of the Expected Goals (xG) metric. This method, adapted from similar shot prediction models used successfully in basketball [1], provides a probabilistic estimate of whether a shot will result in a goal. By accounting for various factors such as shot location, angle, and game context, xG models offer a more nuanced understanding of shot quality than traditional metrics. The increasing use of xG has allowed analysts to better evaluate team and player performance by focusing on the process of shot creation rather than solely on goal outcomes.

Understanding shots in football is therefore a crucial task when producing statistics in the field. Recognizing when shots are taken is the first step in this process.

1.1 Aim

This research is done in collaboration with Football Analytics Sweden AB. It aims to investigate possibilities for automating the process of extracting shots from football video data. This is a labor-intensive task that could be solved using computer vision methods. Recent developments in 3D Convolutional Neural Networks have led to great results in video action recognition tasks and could be very effective in this specific task. Using machine learning for this task would reduce the workload and costs of producing statistics.

1.2 Problem Statement

Using state-of-the-art transfer learning computer vision models, the study will answer the following research question:

- *How accurately can shots in football be detected using computer vision?*

1.3 Delimitation

This study will use transfer learning to fine-tune pre-trained models on a custom dataset. There are not enough data or hardware resources to train models completely from scratch.

1.4 Structure of the thesis

In chapter 3, other studies of similar topics are discussed. The relevant topics and ideas that might be necessary to understand the study are explained in chapter 2. In chapter 4, the method used in the study is explained. The results of the study are presented in chapter 5. In chapter 6, the results and implications of the study are discussed, along with some recommendations for future work in the field. The thesis and study are concluded in chapter 7, along with some notes on ethical aspects.

Chapter 2

Background

In this section, the architecture of CNNs and LSTMs is briefly explained to give an understanding of the models used in this study.

2.1 Artificial Neural Networks

Artificial neural networks (ANN) [2] are computational models inspired by the structure and function of the human brain. They consist of interconnected nodes, or neurons, organized in layers: input, hidden, and output layers. Each neuron processes input data using assigned weights and activation functions to produce an output. Through a process called training, ANNs learn to recognize patterns and make decisions based on data. This training involves adjusting weights to minimize the error between predicted and actual outputs, typically using the backpropagation algorithm. Due to their ability to model complex, nonlinear relationships in data, ANNs are widely used in various fields, including image and speech recognition, natural language processing, and predictive analytics.

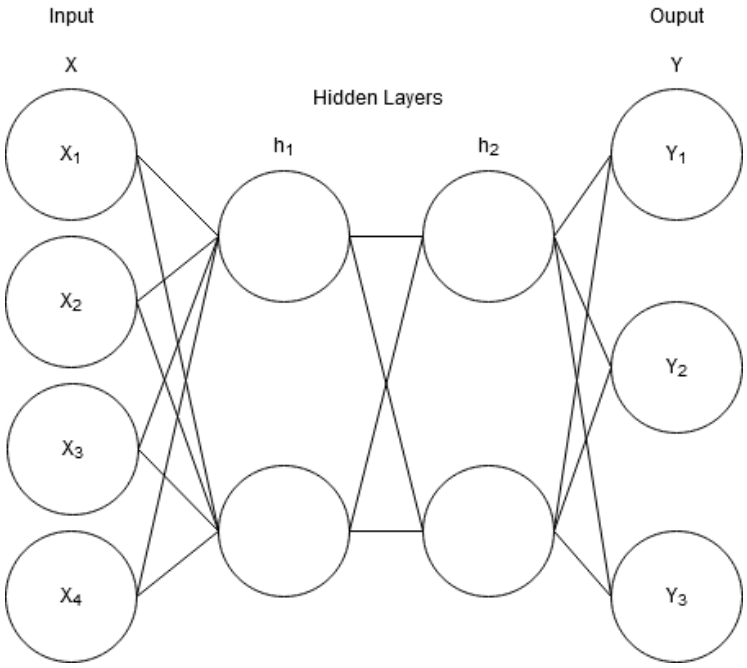

Figure 2.1: Figure showing an artificial neural network with 4 input nodes, 2 hidden layers with 2 nodes each, and an output layer with 3 nodes.

2.1.1 Forward Pass

The forward pass algorithm is the process of calculating the output given an input. One layer is calculated at a time, from the input to the output. Each layer contributes to the next layer by the following equation:

$$X_n = f(W_n \cdot X_{n-1} + b_n)$$

Where X_n is the nth layer, layer 0 is the input layer, W_n is the weight matrix of layer n and b_n is the bias for layer n. Each layer has weights and biases that change the correlation between the layers, these are the parameters learned during training. The function f is an activation function, often a sigmoid or rectified linear unit (ReLU) function, which introduces non-linearity into the model.

2.1.2 Backpropagation

Backpropagation, short for ”backward propagation of errors,” is an algorithm used to train artificial neural networks. It involves adjusting the weights of the

network to minimize the error in its predictions. The process consists of the following steps:

- **Compute the Loss**:

  After the forward pass, the network output is compared to the actual target values using a loss function. This loss quantifies the difference between the predicted and actual values. Two of the most used loss functions are mean squared loss for regression and cross-entropy loss for classification.

  - Mean squared loss:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.1}$$

  - Cross-entropy loss:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right) \tag{2.2}$$

- **Backward Pass**:

  The backward pass involves propagating the error from the output layer back through the network to update the weights. This is done using the chain rule of calculus:

  1. **Calculate Gradients:** For each neuron, calculate the gradient of the loss concerning its weights. The gradient represents how much the loss would change with a small change in the weights.
  2. **Update Weights:** Adjust the weights in the direction that reduces the loss. The update rule for weights w is generally:

$$w \leftarrow w - \eta \cdot \frac{\partial L}{\partial w} \tag{2.3}$$

     where η is the learning rate and $\frac{\partial L}{\partial w}$ is the gradient of loss L with respect to weight w.

- **Iterate**:

  The forward and backward passes are repeated for many iterations (epochs) over the training data, continuously updating the weights to minimize the loss.

Backpropagation is essential for training deep neural networks, allowing them to learn complex patterns in data. By iteratively adjusting the weights, the network improves its predictions and generalizes better to unseen data.

2.2 Computer Vision

Computer vision [3], a dynamic field within artificial intelligence, focuses on enabling computers to interpret and make decisions based on visual input from the world. From its early days of basic image processing to today's sophisticated, deep learning-driven models, computer vision has evolved remarkably, driven by advancements in both hardware and algorithms. This introduction delves into the core concepts, techniques, and applications of computer vision, providing a comprehensive overview of how machines learn to see and understand the visual world.

At its heart, computer vision involves tasks such as image classification, where computers learn to categorize images into predefined classes, and object detection, where specific objects within an image are identified and located. These tasks have wide-ranging applications, from improving medical diagnostics through precise image analysis to powering autonomous vehicles that navigate and interpret their surroundings safely.

2.2.1 Transformations

Transformations are actions applied to an image that alters it in some way. Some common transformations are as follows:

2.2.1.1 Resize

The size of the image is changed to a given size and the contents of the image remain intact.

2.2.1.2 Crop

The size of the image is changed to a given size by removing parts of the image.

2.2.1.3 Color normalization

The colors of the image are altered to a new color space, often to a normal distribution with a given mean and standard deviation.

Figure 2.2: An image from the football video dataset before applying transformations.

Figure 2.3: The same image as in Figure 2.2 after a resize transformation.

Figure 2.4: The same image as in Figure 2.2 after a crop transformation.

Figure 2.5: The same image as in Figure 2.2 after color normalization.

Figure 2.6: The same image as in Figure 2.2 after applying resize, crop, and color normalization.

2.2.2 Image classification

Image classification refers to the problem of classifying images into two or more classes. This is done by feeding images through a machine-learning model and training on pre-labeled classes.

Before images are fed to the model, they are often pre-processed using different transformations to standardize the inputted images.

The architecture most used for image classification is Convolutional Neural Networks (CNNs).

2.2.3 Video classification

Video classification is similar to image classification, but it classifies videos instead of images. The video is often deconstructed into several frames from the video that are transformed and then fed into a model for training.

Video classification is inherently more complicated than image classification due to the added dimension of time. It therefore requires more advanced architectures such as 3D-CNNs or a combination of 2D-CNNs and a Recurrent Neural Network (RNN).

2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) [4] is a type of neural network that is specifically designed to work with structured inputs, such as images. They are made up of neurons with learnable weights, similar to regular neural networks, but they explicitly assume that inputs have specific structures, such as images. CNNs use a series of layers, including convolution layers, pooling layers, and fully connected layers, to transform the activations or outputs of the previous layer through differentiable functions.

CNNs can be broken down into four components [5]

- Input layer, holding the data to be processed
- Convolutional layers, processing the input with kernels resulting in a new representation
- Pooling layer, responsible for downsampling the representation
- Fully connected layers, standard ANN layers responsible for learning and transforming the representation to the correct output.

2.3.1 2D Convolutional Neural Network

2D CNNs, as the name suggests, take 2D inputs and use 2D convolutional layers to learn representations of the input.

The 2D convolutional layers are composed of 2D kernels gliding through the input, activating based on the pattern in the data. The kernel starts in the top left corner of the matrix and then moves across the matrix, calculating convolutions for each position. The *stride* paramater defines the step size of the moving kernel.

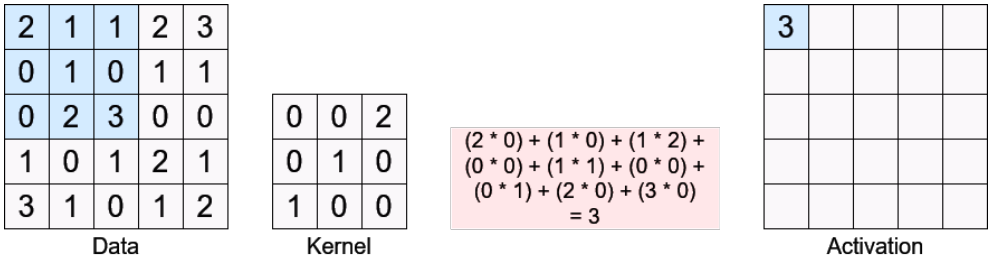

Figure 2.7: Figure showing a convolution, a 3x3 kernel is activated on the data, and the activation is calculated with a weighted sum.

Kernels are typically small square matrices, and their values are learnable and thus altered in the network training process. A CNN can have multiple kernels; each kernel detects different features from the input data, such as edges, textures, or shapes. The output of the layer consists of multiple feature maps, with each map representing the features learned by a specific kernel. This allows CNN to extract more diverse and complex patterns, improving its ability to recognize and classify objects. Additionally, multiple kernels enhance the network's expressive power by enabling it to capture various aspects of the input data at different levels of abstraction. However, this also increases computational demands.

After transforming the data with convolutional layers, the data is downsampled with a pooling layer. The pooling layer divides the data into non-overlapping windows, and an aggregation function such as max or average pooling is applied to each window, generating a single value. The aggregated values are collected into a downsampled version of the input.

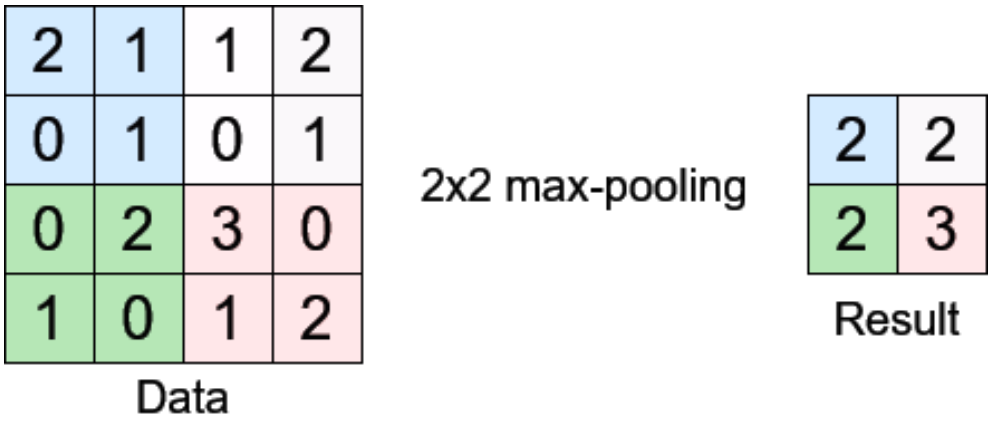

Figure 2.8: Figure showing a 2x2 max-pooling layer, for each 2x2 block of the data the maximum value is extracted.

The downsampled data is then fed into fully connected layers, see section 2.1, with neurons connected with learnable weights to generate an output.

2D CNNs are often used with images due to their ability to recognize patterns in large structured 2D data. Kernels are often structured hierarchically, early layers often detect simple patterns such as edges, while deeper layers are set up to recognize higher-level features like corners or even entire objects.

2.3.2 3D Convolutional Neural Network

Videos are represented as 3D data consisting of stacked 2D frames. 3D CNNs can be applied to videos to extract patterns in both the image and temporal space.

3D CNNs, works on 3D data by applying 3D kernels across the input [6]. The pooling layer downsamples the data using an aggregation over all three dimensions.

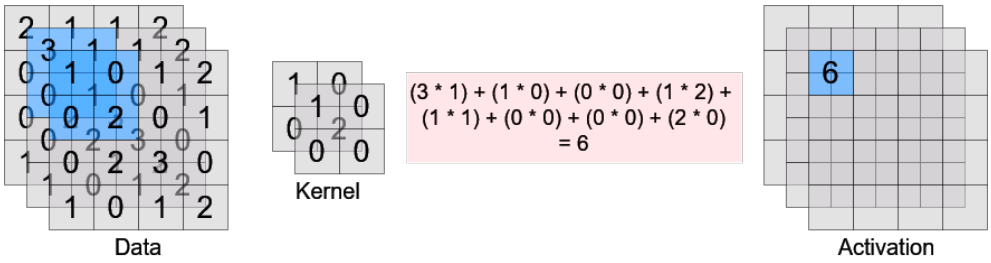

Figure 2.9: Figure showing a 3D convolution, a 2x2x2 kernel is activated on the data, and the activation is calculated with a weighted sum.

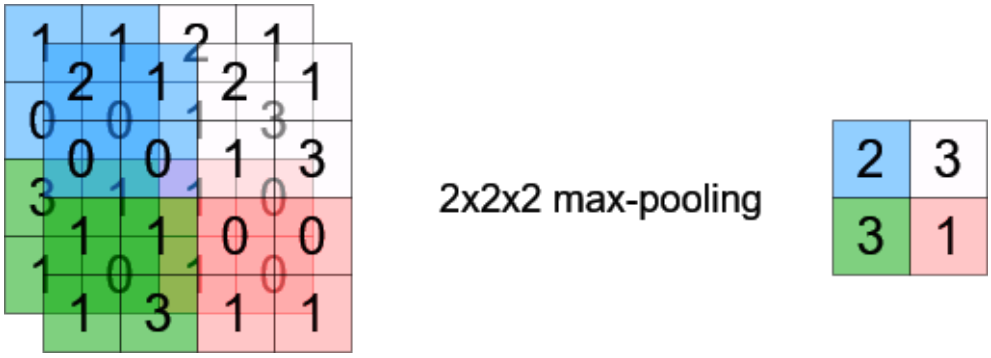

Figure 2.10: Figure showing a 2x2x2 max-pooling layer, for each 2x2x2 block of the data the maximum value is extracted.

Just like 2D kernels, 3D kernels capture patterns in the data such as edges or corners. But with the added dimension of time, they also capture the changes in patterns over time.

2.4 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data

[7].

A RNN is a type of neural network that handles sequential data by retaining information from previous inputs through internal memory. They are best suited for data where the order of inputs matter, such as time series prediction [8].

Unlike traditional RNNs, which struggle with retaining information over long sequences due to issues such as vanishing and exploding gradients, LSTMs use a unique architecture to mitigate these problems.

LSTMs consist of a series of cells, each containing three key components:

Input Gate: Controls the extent to which new information flows into the cell state. Forget Gate: Determines how much of the previous cell state information should be retained. Output Gate: Regulates the output of the current cell state to the next cell and the network.

These gates are controlled by learnable parameters and allow LSTMs to maintain and update information over long periods, making them effective for tasks such as language modeling, speech recognition, and time series prediction.

Since Hochreiter and Schmidhuber [9] introduced LSTMs they have been proven to be a very versatile architecture, achieving great results across different fields of machine learning.

Due to its underlying state architecture, it is often used in problems using data spanning across time, such as time series prediction, or video analysis.

Chapter 3

Related Works

In this chapter, previous work in the areas of video analysis, transfer learning, and sports analytics is discussed.

3.1 CNN Video analysis

CNNs primarily process 2D data (images), while video data presents an additional dimension: time. The main challenge of applying CNNs to video data is effectively exploiting temporal information.

One approach to model the temporal complexity of the data is to combine CNNs with Recurrent Neural Networks (RNN) such as LSTM networks. A CNN model is used to extract features from still frames of the video which are then fed into an RNN to model the temporal structure of the data. Such models have shown great performance at different video analysis tasks [10–13].

To further exploit the temporal information in video data, 3D convolutional neural networks (3D-CNNs) were introduced [14]. 3D-CNNs extract features from both spatial and temporal dimensions. It utilizes 3D convolutions to capture motion information. 3D-CNNs have been shown to outperform 2D CNNs on video analysis tasks on various datasets [6, 15].

3.2 Transfer Learning

Transfer learning is the concept of taking a machine learning model trained on one task and applying it to another task.

In recent years, pre-trained CNN models have become more widely available, having been trained on large-scale datasets, and can be fine-tuned to perform effectively on a variety of datasets.

ResNet [16], short for Residual Network, is a deep neural network architecture characterized by the innovative use of residual connections, allowing training of extremely deep networks by facilitating the flow of gradients and alleviating the problem of vanishing gradients. ResNet has shown great performance in different transfer learning tasks.

Tran *et al.* [14] introduced a pretrained 3D-CNN named C3D which is trained on the Sports-1M dataset [15] containing over 1 133 158 sports videos from YouTube.

Tran *et al.* [14] later introduced Res3D [17], a pre-trained 3D-CNN outperforming C3d in both accuracy and run-time.

Xie *et al.* [18] introduced the S3D architecture and showed that many of the 3D convolutions could be replaced by low-cost 2D convolutions, leading to a competitive performance with lower computational cost.

3.3 Convolutional Neural Networks in Sports Analytics

Rangasamy *et al.* [19] summarizes four different types of architectures used in sports video analysis:

- 2D-CNN
- 3D-CNN
- RNN
- LSTM

The big problem with video analysis is to capture the temporal structure of the data. 2D-CNNs perform excellently on image data but lack in performance when tasked with video data. RNNs and LSTM architectures have been tested due to their ability to understand temporal structure, but they lack performance when applied to image data when compared to 2D-CNNs.

Rangasamy *et al.* [19] explains that 3D-CNNs have been proven to be great at sports video analysis, the only drawback is the expensive training and fine-tuning of the models.

Tora *et al.* [20] used a combination of 2D-CNN and LSTM to classify hockey events. The CNN was used to extract features which were fed into the LSTM for classification. The combined architectures outperformed state-of-the-art 3D-CNN models on their hockey dataset.

Chapter 4

Methodology

In this chapter, the methodology used in the study is presented, from data collection and pre-processing to training models.

4.1 Data Collection and Pre-processing

The data set used comprises videos of football matches from the Swedish third division during the 2023 season. These videos are accompanied by data on all game actions, such as shots and passes, along with corresponding time-frames. The videos have a resolution of 1280x720 and a frame rate of 50 frames per second.

To enable machine learning model training on the data, clips of shots and non-shots must be extracted and labeled accordingly. A clip length of 1.5 seconds was chosen; longer clips would require more resources and would lead to the model being less precise when finding the exact timing of a shot.

From each match, 1.5-second video clips were extracted for each shot, where the middle of the clip is the shot moment. Random 1.5-second samples of non-shot video sequences were collected.

The resulting dataset consists of 9198 shots and 13414 non-shot actions.

Figure 4.1: Video frames from a shot clip.

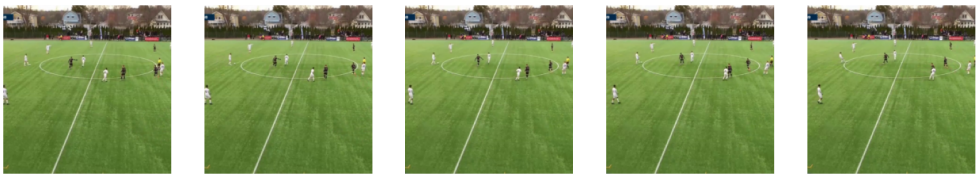

Figure 4.2: Video frames from a non-shot clip.

4.2 Model Architecture Design

Two different model architectures were chosen to be implemented and compared, a 2D-CNN used as a feature extractor combined with an LSTM classifier and a 3D-CNN.

4.2.1 Convolutional Neural Network with Long Short-Term Memory

The first architecture that was tested was a combination of a CNN feature extractor and an LSTM model. Such model architectures have been used successfully in other video analysis tasks [10–13].

Other classifiers other than LSTM could have been used, but LSTMs are often used in sequential data because of their ability to model the temporal structure of data.

Every fifth frame was extracted for each video, resulting in a new frame rate of 10 frames per second. A higher frame rate requires more resources to model and could be harder for a model to adapt to. A lower frame rate is easier for the model to adapt to and requires fewer resources but could lead to information being lost in the video.

Figure 4.3: Every fifth frame of a shot clip extracted.

Each frame of the video was fed through a ResNet-50 network, with 48 convolutional layers, one MaxPool layer, and one average pool layer. The final classification layer was removed, resulting in the output being 2048 features.

The ResNet model requires a frame size of exactly 224x224 pixels. To accommodate this, the frames were first resized using different sizes and then cropped to 224x224 pixels.

An LSTM classifier model was created using PyTorch, feeding inputs of size 2048 into a variable number of LSTM layers and then through a linear classification layer generating a binary output, corresponding to shot or non-shot.

The features of each frame were collected into a tensor and fed into the LSTM model, generating a binary prediction for each clip.

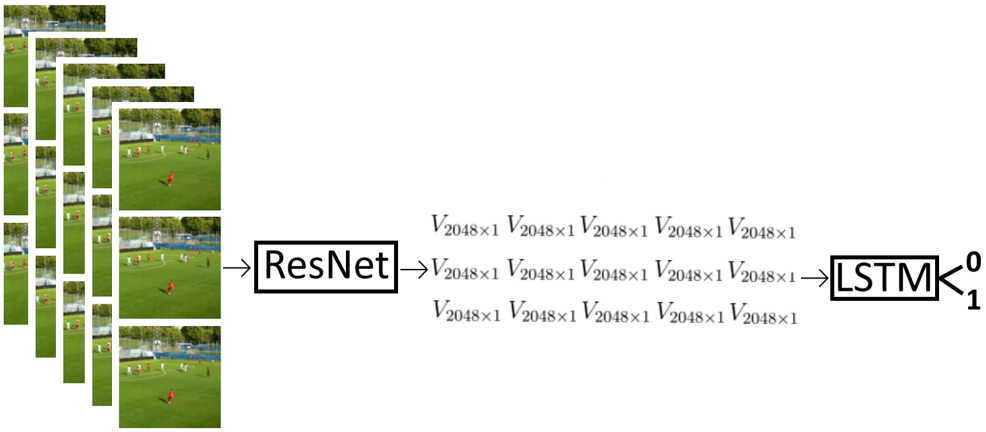

Figure 4.4: The combined CNN with LSTM architecture visualized from data to output.

4.2.2 3D Convolutional Neural Network

A pre-trained 3D-CNN model was imported through Pytorch. The chosen model was the S3D model [18]. The last classification layer was changed to a binary output layer.

The clips were transformed according to the S3D paper, to match the data on which the model has been trained, the following was the procedure:

The frame rate was changed to 15. The clips were resized to 256x256 pixels and then center-cropped to 224x224 pixels. The pixel values were normalized using these values:

$$\mu = (0.43216, 0.394666, 0.37645)$$
$$\sigma = (0.22803, 0.22145, 0.216989)$$

Figure 4.5: Sample video frame before transformation.

Figure 4.6: Sample video frame after transformation.

4.3 Training Procedure

The training procedure was different for the two different architectures.

4.3.1 Training environment

All training was done on Windows 10 with a high-end consumer-grade graphics card, Nvidia RTX 3070 with 8 GB of memory. Python version 3.9.16

was used alongside PyTorch 2.10.

4.3.2 Convolutional Neural Network with Long Short-Term Memory

The ResNet model was used to generate features for all clips in the dataset. The LSTM classifier was trained on the features using backpropagation with an Adam optimizer.

4.3.2.1 Hyperparameter tuning

The different hyperparameters to be tuned for this model were:

- LSTM hidden layers
- LSTM hidden layer size
- Learning rate

The values of the hyperparameters were found using two grid searches. First, a course grid search training for 15 epochs to get a general idea of sensible values for the different parameters. A second grid search was conducted, training for 100 epochs using early stopping to get the best-performing model with the best-fitted parameters.

4.3.3 3D Convolutional Neural Network

The S3D model was trained using backpropagation with an Adam optimizer, the most commonly used optimization algorithm for deep learning models.

4.3.3.1 Hyperparameter tuning

The best hyperparameters were found using a two-step search. The first step tested one parameter at a time on 10% of the training data. The parameters tested were:

- Learning Rate
- Dropout
- Frozen Layers
- Batch Size

The dropout parameter was applied before the last layer to randomly assign weights to zero with a probability of p. Values of $p = 0$, $p = 0.2$, and $p = 0.5$ were tested.

The frozen layer parameter refers to the number of frozen layers and, therefore, is not changed in the training process. Values of 0, 5, 10, and 15 were tested. A value of n means that the first layers of the model n are frozen.

The second hyperparameter tuning step was a grid search searching for a combination of batch size and learning rate parameters using 25% of the training dataset. Batch sizes from 2 to 16, the maximum batch that fits into the memory of the GPU, were tested.

The best-performing parameters of the hyperparameter search decided on minimum validation loss were used to train the final model. The model was trained on the whole training dataset for 10 epochs using early stopping.

4.4 Software implementation

In this section, the most important parts of the software implementation of the training process of the different architectures are presented and explained.

4.4.1 2D Convolutional Neural Network with LSTM

The first part of this model is to extract features using a ResNet feature extractor. This can be done by loading a ResNet model and removing the final classification layer. Images can then be fed through the network, generating features. A Python implementation of this can be seen in Listing 4.1

Listing 4.1: Python code for extracting features from a video frame dataset using a ResNet-50 model

```
from torchvision.models import resnet50

# Load the pre-trained ResNet-50 model and remove the last layer
resnet_model = resnet50(pretrained=True)
feature_extractor = nn.Sequential(*list(resnet_model.children())[:-1])

# Iterate through the partial dataset and extract features
train_features = []
train_labels = []
for frames, labels in frame_train_dataloader:
    # Move frames and labels to the GPU if available
    frames = frames.to(device)
    labels = labels.to(device)

    # Extract features from the frames, dont compute gradients
    with torch.no_grad():
        features = [feature_extractor(frame) for frame in frames]

    # Convert features to numpy arrays and store
    train_features.append(torch.stack(features))
    train_labels.append(labels)
```

The features can be used to train an LSTM model to predict the labels of frame sequences. A LSTM model is loaded with set hyperparameters. The training dataset is iterated over for a set number of epochs, for each batch making a prediction and using backpropagation to alter the model parameters. A Python implementation of this can be seen in Listing 4.2.

Listing 4.2: Python code for training a LSTM model

```
def train_model(model, train_loader, num_epochs=10, lr=0.001, hidden_size=128, num_layers=2):
    # Initialize the model
    input_size = 2048  # ResNet-50 feature size
    num_classes = 2

    model = LSTMClassifier(input_size, hidden_size, num_layers, num_classes)
    model.to(device)
    model.train()

    # Define loss function and optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(num_epochs):
        for features, labels in train_loader:
            features = features.to(device)
            labels = labels.to(device)
            # Forward pass
            outputs = model(features)
            loss = criterion(outputs, labels)

            # add loss to epoch loss
            epoch_loss += loss.item()

            # Backward pass and optimization
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

4.4.2 3D Convolutional Neural Network

The process of training a 3D-CNN, in this case a S3D model, is done by importing the S3D model and setting the hyperparameters. The training data is iterated over for multiple and backpropagation is used to tune the model parameters. A Python implementation of this can be seen in Listing 4.3.

Listing 4.3: Python code for training a 3D-CNN model

```
from sklearn.model_selection import ParameterGrid

import torch
import torch.nn as nn

# Load a pre-trained S3D model
from torchvision.models.video import s3d

# Define hyperparameters for grid search
param_grid = {
    'lr': [0.0001, 0.0005, 0.001],
    'dropout': [0, 0.2, 0.5],
    'batch_size': [2, 4, 8, 16],
}

num_epochs = 10

# Perform grid search
for params in ParameterGrid(param_grid):
    lr = params['lr']
```

```
dropout = params['dropout']
batch_size = params['batch_size']

# Create a DataLoader to iterate over the dataset
video_train_dataloader = DataLoader(reduced_video_train_dataset, batch_size=batch_size, shuffle=False)
video_val_dataloader = DataLoader(reduced_video_val_dataset, batch_size=batch_size, shuffle=False)

# Create a pre-trained S3D model
s3d_model = s3d(weights='DEFAULT', progress=True, dropout=dropout)

# Modify the classifier layer to output 2 classes
s3d_model.classifier[1] = nn.Conv3d(1024, 2, kernel_size=(1, 1, 1), stride=(1, 1, 1))
s3d_model.to(device)  # Move the model to the GPU if available

# finetuning all layers
for params in s3d_model.parameters():
    params.requires_grad = True

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(s3d_model.parameters(), lr=lr)

for epoch in range(num_epochs):
    for frames, labels in video_train_dataloader:
        frames = frames.to(device)
        labels = labels.to(device)

        # Forward pass
        outputs = s3d_model(frames)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Chapter 5

Experimental Results

The results of training and testing the different models are presented in this chapter.

5.1 Convolutional Neural Network with Long Short-Term Memory

The results from tuning and testing a combined model using a 2D Convolutional Neural Network with a LSTM are presented in this section.

5.1.1 Hyperparameter Tuning

The features were extracted using a ResNet-50 model and a LSTM was used to predict labels. The number of hidden layers, the size of the hidden layers, and the learning rate were tuned using a grid search, training for 15 epochs on the full training dataset. The parameters yielding the lowest validation loss were chosen.

Hidden Layers	Hidden Layer Size	Learning Rate	Validation Accuracy	Validation Loss
1	1024	0.0001	0.756495	0.486397
1	1024	0.0005	0.754837	0.485954
1	1024	0.001	0.747651	0.490789
1	2048	0.0001	0.758154	0.486113
1	2048	0.0005	0.755943	0.495120
1	2048	0.001	0.755390	0.492687
2	1024	0.0001	0.757601	0.479910
2	1024	0.0005	0.758430	0.481629
2	1024	0.001	0.755390	0.485532
2	2048	0.0001	0.759536	0.485485
2	2048	0.0005	**0.762300**	**0.478882**
2	2048	0.001	0.746821	0.496102

Table 5.1: Results of grid search on the number of hidden layers, hidden layer size, and learning rate. Models were trained for 15 epochs on the full training dataset. The best validation accuracy and the lowest validation loss were achieved with 2 hidden layers of 2048 nodes, with a learning rate of 0.0005.

The best-performing parameters were used to train a model for 100 epochs using early stopping. Three different learning rates were tested for this experiment.

Hidden Layers	Hidden Layer Size	Learning Rate	Validation Accuracy	Validation Loss
2	2048	0.00025	**0.764234**	**0.479168**
2	2048	0.0005	0.762576	0.486907
2	2048	0.00075	0.757601	0.480667

Table 5.2: Results of training for 100 epochs on the full training dataset. The best validation accuracy and the lowest validation loss were achieved with 2 hidden layers of 2048 nodes, with a learning rate of 0.00025.

5.1.2 Final Model

The best model found had the following hyperparameters:

- LSTM hidden layers: 2
- LSTM hidden size: 2048
- Learning Rate: 0.00025

The model was trained for 100 epochs using early stopping and achieved the following metrics on the validation dataset:

Accuracy	0.76
Precision	0.69
Recall	0.76

Table 5.3: Metrics of the best performing combined CNN and LSTM model. The model was trained for 100 epochs using early stopping. Metrics are calculated after evaluation on the validation dataset.

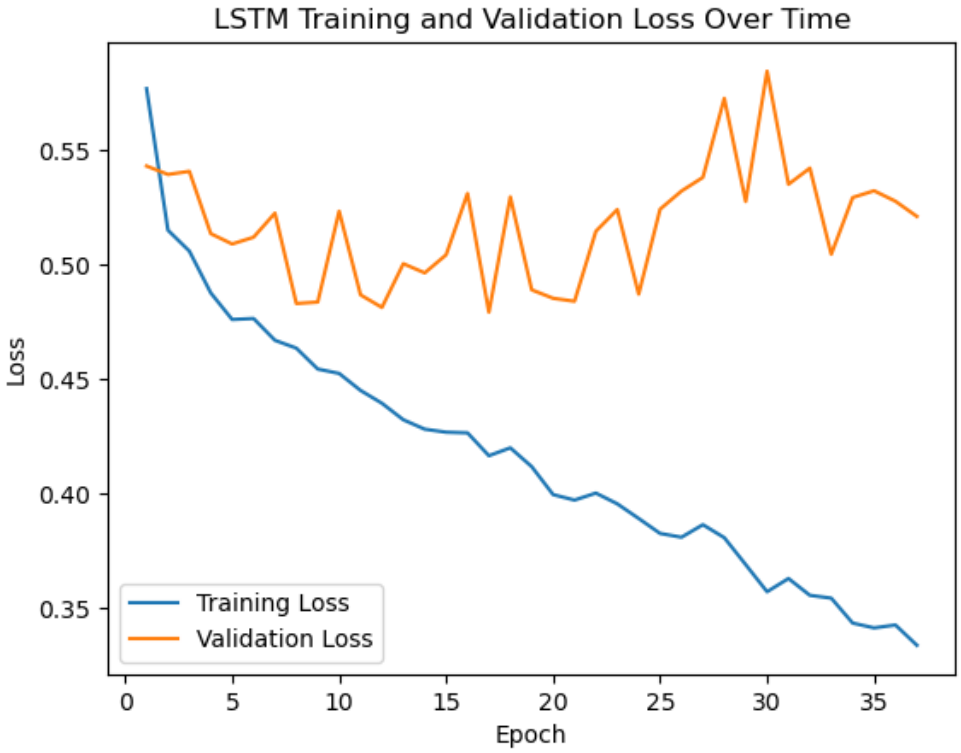

Figure 5.1: Training and validation loss over time for the best performing combined CNN and LSTM model. Epoch number on the x-axis and loss on the y-axis. The blue line is training loss and orange line is validation loss. The downward trend of the training loss with a non-stable oscillating validation loss shows that the model has a hard time generalizing to the data, and is overfitting.

5.1.3 Computational Cost

Regarding the feature extraction step, using a ResNet-50 model, the average time to extract the features for a video was 0.39 seconds. The whole training dataset could be feature extracted in 1 hour and 55 minutes.

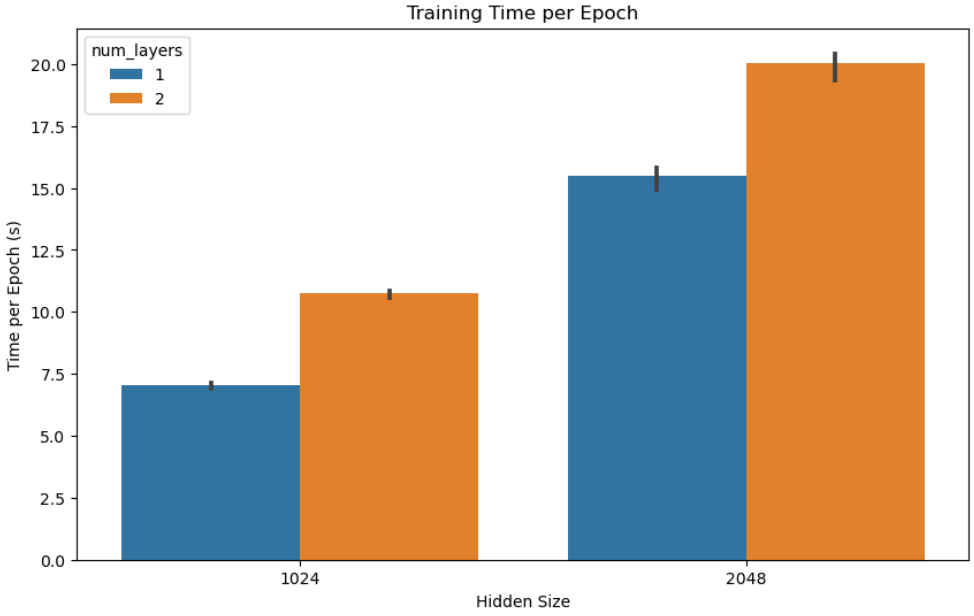

Figure 5.2: Training time per epoch for different LSTM hyperparameters. Training time was roughly doubled when increasing the number of hidden layers from 1 to 2. Training time was also roughly doubled when increasing the hidden layer size from 1024 to 2048.

Regarding training the LSTM models, the training time varied between different combinations of parameters, hidden layer size and the number of hidden layers had a big effect on training time, as seen in Figure 5.2.

Training the final model for 100 epochs took 33 minutes.

5.2 3D Convolutional Neural Network

The results from tuning and testing a 3D Convolutional Neural Network are presented in this section.

5.2.1 Hyperparameter tuning

The learning rate, dropout rate, and the number of frozen layers were tested one by one. As seen in Table 5.4, Table 5.5, Table 5.6 the results show that 0 frozen layers, a dropout of 0.2 and a learning rate of 0.000005 yield the best results.

Learning Rate	Minimum Val Loss
0.001	29.07
0.0005	3.18
0.0001	1.15
0.00005	1.07
0.00001	0.72
0.000005	**0.53**
0.000001	0.71

Table 5.4: Minimum validation losses for different learning rates. The 3D CNN was trained for 5 epochs on 10% of the data. The results show that a lower learning rate performed better, with a learning rate of 5×10^{-6} yielding the best results.

Dropout	Minimum Val Loss
0	0.54
0.2	**0.53**
0.5	0.56

Table 5.5: Best validation loss achieved for different dropout parameters. A 3D convolutional neural network was trained for 5 epochs on 10% of the training data using different dropout rates before the last classification layer. A dropout of 0.2 yielded the best performance.

Frozen Layers	Minimum Val Loss
0	**0.53**
5	0.55
10	0.62
15	0.62

Table 5.6: Best validation loss achieved for a different number of frozen layers. A 3D convolutional neural network was trained for 5 epochs on 10% of the training data, only tuning a few layers and freezing the rest. Freezing more layers yielded worse results, the best results were achieved with 0 frozen layers.

These results were used in a grid search to find an optimal combination of batch size and learning rate. The previous search found a learning rate of 0.000005 to be best, so the search was restricted to values around this.

Batch size	Learning Rate	Minimum Val Loss
2	0.00001	0.416
2	0.000005	0.452
2	0.000001	0.485
4	0.00001	0.409
4	0.000005	0.413
4	0.000001	0.433
8	0.00001	0.436
8	0.000005	0.425
8	0.000001	0.433
16	0.00001	0.514
16	**0.000005**	**0.400**
16	0.000001	0.463

Table 5.7: Best validation loss achieved for different combinations of batch size and learning rate. A 3D convolutional neural network was trained for 10 epochs on 25% of the training data. Different combinations of batch size and learning rates were tested. The best results were achieved with a batch size of 16 and a learning rate of 5×10^{-6}

The best parameters found where:

- Learning Rate: 0.000005
- Dropout: 0.2
- Batch Size: 16
- Frozen Layers: 0

These parameters were used for the final model trained on the full dataset.

5.2.2 Final Model

The hyperparameters found in the tuning phase were used to train a model on the full training dataset. It was trained for 10 epochs using early stopping, converging after only 2 epochs.

The model was tested on the test dataset and achieved the following metrics:

Precision	Recall	Accuracy
0.82 ± 0.017	0.83 ± 0.017	0.85 ± 0.01

Table 5.8: Precision, recall, and accuracy of the best 3D CNN model on the test dataset. The model was trained on the full training dataset for 2 epochs. Variance is calculated with a binomial proportion confidence interval.

The model achieved a precision of 82 %, a recall of 83 %, and an accuracy of 85 % on the test dataset.

As seen in Figure 5.3, validation loss started to increase after 2 epochs, and training was stopped after 5 epochs.

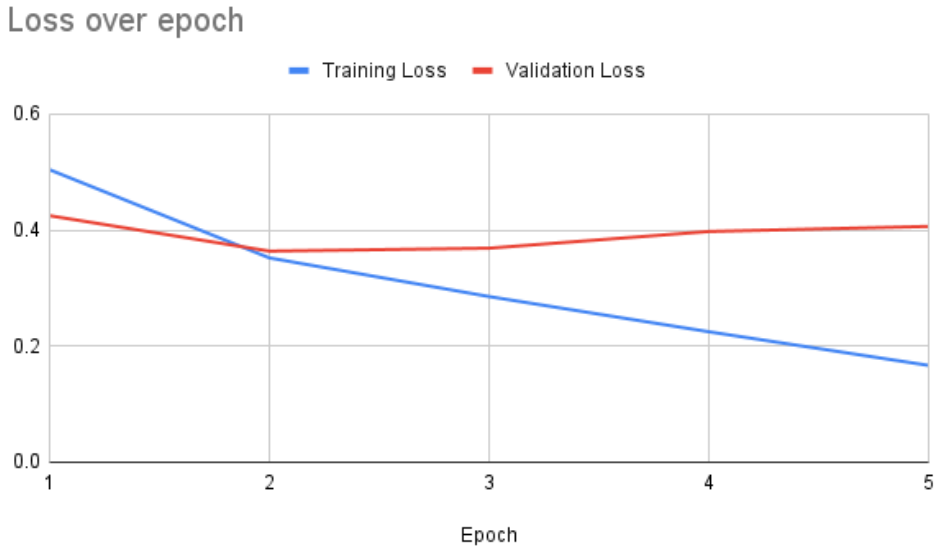

Figure 5.3: Training and Validation loss over epoch for S3D model trained on the full dataset. On the X-axis is the epoch, and on the Y-axis is the average cross-entropy loss. Training loss is colored blue, and validation loss is colored red. Validation loss was the lowest after 2 epochs, while training loss continued to decrease, indicating overfitting after 2 epochs.

5.2.3 Computational Cost

The training time varied slightly for different batch sizes as seen in Figure 5.4.

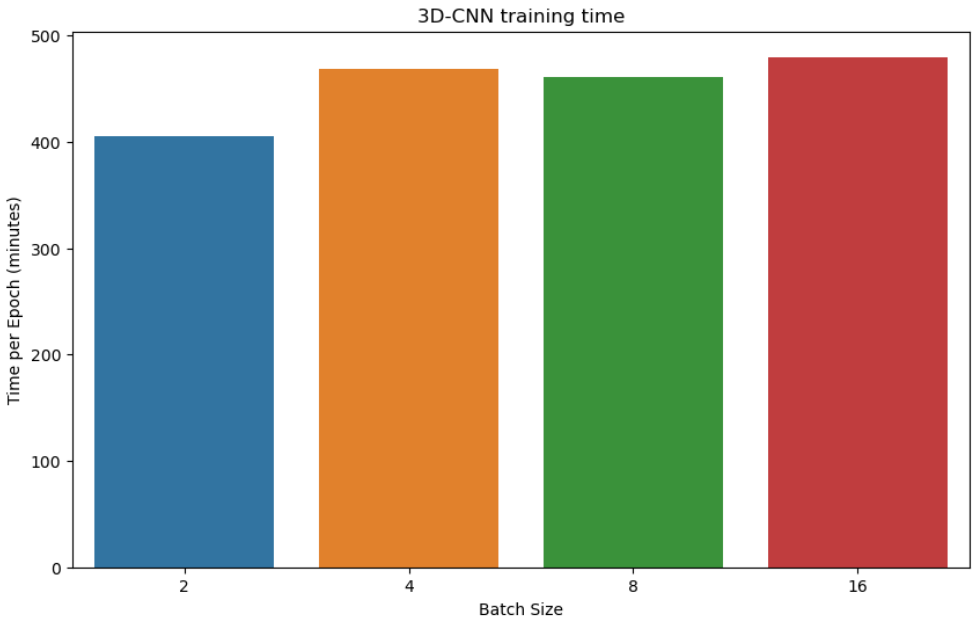

Figure 5.4: Training time per epoch for different batch sizes. Training time was slightly longer for larger batch sizes. A batch size of 16 yielded to longest training time.

The final model, trained on the full training dataset, averaged more than 6 hours per epoch, exceeding 60 hours for a 10-epoch training process.

5.3 Summary

The best results were achieved using a 3D-CNN architecture. 82% precision, 83% recall, and 85% accuracy were achieved on the test dataset.

Chapter 6

Discussion

In this chapter, the results are discussed and analyzed. Some limitations and potential future work are brought up.

6.1 Performance Analysis

The results made clear that a 3D-CNN architecture outperformed a 2D CNN architecture, which aligns with previous research suggesting that 3D-CNNs perform better on video analysis tasks. As discussed in chapter 3, 3D-CNNs are better at adapting to the temporal structure of video data and often outperform 2D-CNNs in video analysis tasks.

The performance of the best-performing model, achieving an accuracy of 85%, seems a bit lackluster. To use such a model in production scenarios, where longer sequences are analyzed and shots are extracted correctly, the performance needs to be better. Being wrong more than 10% of the time in short 1.5-second clips means that the model has a 80% probability of being wrong at least once in a 15-second clip where each segment is analyzed without overlap. It would therefore be of great use to have a better performing model. A model of accuracy greater than 99% would only be wrong 10% of the time in a 15-second context and could be a good benchmark to aim for.

To achieve better performance, more data and more computer resources are needed. More resources could be used to improve training with better-optimized hyperparameters, tuning over larger proportions of data, and over more epochs. The data used in this study are only from one single league over a single season, and one could use data from multiple other leagues and other seasons. This would lead to a more diverse dataset, which could be harder to learn but would also lead to a more generalized model capable of

understanding the intricacies of different scenarios.

The 3D-CNN model was much more computationally demanding to train. Training on the entire data set took around 480 minutes, compared to LSTM taking only 20 seconds. This meant that the 3D-CNN model could not feasibly be tuned with the same precision as the LSTM model. This further strengthens the claim that the 3D-CNN model could probably perform better with more resources and is better performing than the combined 2D-CNN and LSTM model for this specific task.

In a production environment, the training time of the model is probably not very important, it only has to be trained once. Using the model is very fast when compared to training it. This further supports the claim that the 3D-CNN architecture is probably better than the 2D-CNN architecture for this task.

The 2D-CNN architecture achieved a validation accuracy of around 76%, this seems quite low and the model had a hard time adapting to the different videos. One reason for this could be that the pre-trained image feature extractor, in this case, a ResNet model, is not enough to distinguish between intricate football actions. This could also be the reason why the 3D-CNN model performed worse when fine-tuning fewer layers. In both cases, it seems that the general features extracted from the video data are not sufficient. Performance improved drastically when more layers were fine-tuned in the 3D-CNN architecture. Perhaps using a custom image feature extractor or a fine-tuned feature extractor trained on a football dataset could be needed to achieve better results with a 2D-CNN model.

6.2 Limitations and Future Work

This research only investigates the classification of shots and non-shots in video data. The results suggest that the model can differentiate between a shot and a nonshot, and further research needs to be done investigating the classification of other actions such as passes and throw-ins. More complex models could be created where multiple actions are classified.

The 3D-CNN architecture outperformed the 2D-CNN architecture, achieving promising results. There could however be room for improvement in the 3D-CNN model. Due to the very time-consuming training of the 3D-CNN model, hyperparameter tuning and model optimization could potentially lead to even better results. Using a high-end consumer graphics card, RTX 3070, training for a single epoch took over 6 hours. Using high-performance cloud computing clusters or supercomputers could be a way to get even better

optimization.

In this study, short sequences were analyzed. The question arises what happens when trying to analyze longer sequences, maybe whole matches? The architecture presented in this study could be applied to longer sequences by analyzing many shorter segments. The performance of the model in such a scenario is unknown.

The models investigated in this research are based on CNN and 3D-CNN architectures, as these architectures have been proven in the literature to perform great in action detection tasks, as pointed out in chapter 3. There could be other models that are suitable for similar data. Further research needs to be done training and testing other model architectures on classifying shots in football video data. One potential architecture that has shown promising results in similar tasks is transformers. Such models are quite new and have not been studied as thoroughly in the field of action detection and were therefore not included in this study.

Chapter 7

Conclusion

In this chapter, the thesis is summarized, the research question is answered, and some ethical aspects are discussed.

7.1 Summary

This research was done in collaboration with Football Analytics Sweden AB. The aim was to investigate possibilities to automate the process of extracting shots from football video data. This is a labor-intensive task that could be solved using computer vision methods.

A custom dataset composed of short 1.5-second videos of football matches from the Swedish third division was used, consisting of many shot and non-shot clips. Two different computer vision architectures were implemented and trained on the dataset, 2D-CNN combined with an LSTM, and also a 3D CNN. The 3D-CNN was shown to outperform the other architecture and achieved an accuracy of 85% on the test dataset.

7.1.1 Research Question Answered

- *How accurately can shots in football be detected using computer vision?*

Shots in football matches were correctly identified 85% of the time using a 3D-CNN architecture on the custom dataset. With further optimization using more data and more complex models, trained with better hardware, it is hypothesized that the results could be improved.

7.2 Ethical Aspects

7.2.1 Computer Resources

Using computer resources to train machine learning models has several ethical implications that should be carefully considered. The fair distribution of computing power is one such issue. Devoting significant resources to model training may unintentionally exacerbate already existing inequalities, since low-income individuals or organizations may not be able to afford access to cloud services or high-performance computing clusters. This calls into question the inclusion and fairness of technological innovation.

Furthermore, it is impossible to ignore how large-scale model training affects the environment. Extensive computational activities lead to carbon emissions and energy consumption that worsen the environment and accelerate climate change. As a result, ethical discussions need to consider sustainability and promote prudent resource management to reduce harmful environmental repercussions.

Ethical considerations go beyond the sustainability of the environment and access to include the exploitation of computing resources. Using resources to create algorithms that reinforce prejudice or discriminatory practices carries risks and can create social injustices. Ethical frameworks should stress transparency and accountability in resource allocation, ensuring that the benefits of machine learning research are equally dispersed while limiting harm to underprivileged people.

Another interesting discussion is whether research should be spent on analyzing football video data rather than other more important fields such as medicine. Researching problems in video analysis in one field, such as football, will be important for understanding how to apply such techniques in other fields as well. Analyzing seemingly unimportant problems could lead to better understanding and better results when researching more important problems.

7.2.2 Gambling

Football statistics and analysis are strongly connected to betting. Statistics is used by betting companies to analyze probabilities and outcomes when setting odds for different markets. Statistics produced using the techniques in this study could be used by betting companies or gamblers.

Making statistics available for everyone to use would lead to a more fair

playing field in the betting field. However, only selling statistics to the betting companies would create an advantage for the companies and could lead to gamblers being put into a disadvantageous situation where they have a larger chance of losing.

Sports betting is a huge industry and contributing to advancements in football analytics could in turn benefit the betting industry.

Gambling addiction, financial ruin, and exploitation of vulnerable individuals are common concerns with gambling. In addition, there are ethical dilemmas regarding the fairness of the results, especially when technology is used to gain an unfair advantage. Moreover, the promotion of gambling can contribute to social inequality and undermine the integrity of sports and games.

References

[1] J. Mead, A. O'Hare, and P. McMenemy, "Expected goals in football: Improving model performance and demonstrating value," *PLOS ONE*, vol. 18, no. 4, pp. 1–29, Apr. 2023. DOI: 10.1371/journal.pone.0282295. [Online]. Available: https://doi.org/10.1371/journal.pone.0282295.

[2] Z. Yang and Z. Yang, "6.01 - artificial neural networks," in *Comprehensive Biomedical Physics*, A. Brahme, Ed., Oxford: Elsevier, 2014, pp. 1–17, ISBN: 978-0-444-53633-4. DOI: https://doi.org/10.1016/B978-0-444-53632-7.01101-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780444536327011011.

[3] R. Klette, *Concise computer vision* (Undergraduate Topics in Computer Science), en, 2014th ed. Guildford, England: Springer, Jan. 2014.

[4] J. Teuwen and N. Moriakov, "Chapter 20 - Convolutional neural networks," in *Handbook of Medical Image Computing and Computer Assisted Intervention*, ser. The Elsevier and MICCAI Society Book Series, S. K. Zhou, D. Rueckert, and G. Fichtinger, Eds., Academic Press, Jan. 2020, pp. 481–501, ISBN: 978-0-12-816176-0. DOI: 10.1016/B978-0-12-816176-0.00025-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128161760000259 (visited on 04/03/2024).

[5] K. O'Shea and R. Nash, *An Introduction to Convolutional Neural Networks*, en, arXiv:1511.08458 [cs], Dec. 2015. [Online]. Available: http://arxiv.org/abs/1511.08458 (visited on 05/14/2024).

[6] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," en, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan. 2013, ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.5

9. [Online]. Available: http://ieeexplore.ieee.org/document/6165309/ (visited on 04/03/2024).

[7] Y. Yu, X. Si, C. Hu, and J. Zhang, "A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures," *Neural Computation*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019, ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. eprint: https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf. [Online]. Available: https://doi.org/10.1162/neco%5C_a%5C_01199.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[9] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[10] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, *Beyond Short Snippets: Deep Networks for Video Classification*, en, arXiv:1503.08909 [cs], Apr. 2015. [Online]. Available: http://arxiv.org/abs/1503.08909 (visited on 04/03/2024).

[11] J. Donahue *et al.*, *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*, en, arXiv:1411.4389 [cs], May 2016. [Online]. Available: http://arxiv.org/abs/1411.4389 (visited on 04/03/2024).

[12] A. Ullah, J. Ahmad, K. Muhammad, M. Sajjad, and S. W. Baik, "Action recognition in video sequences using deep bi-directional lstm with cnn features," *IEEE Access*, vol. 6, pp. 1155–1166, 2018. DOI: 10.1109/ACCESS.2017.2778011.

[13] X. Wang, L. Gao, J. Song, and H. Shen, "Beyond Frame-level CNN: Saliency-Aware 3-D CNN With LSTM for Video Action Recognition," *IEEE Signal Processing Letters*, vol. 24, no. 4, pp. 510–514, Apr. 2017, Conference Name: IEEE Signal Processing Letters, ISSN: 1558-2361. DOI: 10.1109/LSP.2016.2611485. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7572183 (visited on 04/04/2024).

[14] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, *Learning Spatiotemporal Features with 3D Convolutional Networks*, en, arXiv:1412.0767 [cs], Oct. 2015. [Online]. Available: http://arxiv.org/abs/1412.0767 (visited on 04/03/2024).

[15] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-Scale Video Classification with Convolutional Neural Networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, ISSN: 1063-6919, Jun. 2014, pp. 1725–1732. DOI: 10.1109/CVPR.2014.223. [Online]. Available: https://ieeexplore.ieee.org/document/6909619 (visited on 04/03/2024).

[16] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, arXiv:1512.03385 [cs], Dec. 2015. DOI: 10.48550/arXiv.1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385 (visited on 04/03/2024).

[17] D. Tran, J. Ray, Z. Shou, S.-F. Chang, and M. Paluri, *ConvNet Architecture Search for Spatiotemporal Feature Learning*, arXiv:1708.05038 [cs], Aug. 2017. DOI: 10.48550/arXiv.1708.05038. [Online]. Available: http://arxiv.org/abs/1708.05038 (visited on 04/03/2024).

[18] S. Xie, C. Sun, J. Huang, Z. Tu, and K. Murphy, *Rethinking spatiotemporal feature learning: Speed-accuracy trade-offs in video classification*, 2018. arXiv: 1712.04851 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1712.04851.

[19] K. Rangasamy, M. A. As'ari, N. Rahmad, N. F. Ghazali, and S. Ismail, "Deep learning in sport video analysis: A review," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 18, p. 1926, Aug. 2020. DOI: 10.12928/telkomnika.v18i4.14730.

[20] M. Tora, J. Chen, and J. Little, "Classification of Puck Possession Events in Ice Hockey," *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jul. 2017.

TRITA – EECS-EX 2024:0000
Stockholm, Sweden 2024

www.kth.se

€€€€ For DIVA €€€€

{
”Author1”: { ”Last name”: ”Erikson”,
”First name”: ”Jonathan”,
”Local User Id”: ”u741370”,
”E-mail”: ”jerikso@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
}
},
”Cycle”: ”2”,
”Course code”: ”DA231X”,
”Credits”: ”30.0”,
”Degree1”: {”Educational program”: ”Degree Programme in Computer Science and Engineering”
,”programcode”: ”CDATE”
,”Degree”: ”Degree of Master of Science in Engineering”
,”subjectArea”: ”Computer Science and Engineering”
},
”Title”: {
”Main title”: ”Football shot detection using convolutional neural networks”,
”Language”: ”eng” },
”Alternative title”: {
”Main title”: ”Detektion av skott i fotboll med hjälp av convolutional neural networks”,
”Language”: ”swe”
},
”Supervisor1”: { ”Last name”: ”Markidis”,
”First name”: ”Stefano”,
”Local User Id”: ”u100003”,
”E-mail”: ”markidis@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”DIVISION OF COMPUTATIONAL SCIENCE AND TECHNOLOGY” }
},
”Supervisor2”: { ”Last name”: ”Jorge”,
”First name”: ”Alex”,
”E-mail”: ”alex@playmaker.ai”,
”Other organisation”: ”Football Analytics Sweden AB”
},
”Examiner1”: { ”Last name”: ”Peng”,
”First name”: ”Bo”,
”Local User Id”: ”u1hwtn2r”,
”E-mail”: ”bopeng@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”DIVISION OF COMPUTATIONAL SCIENCE AND TECHNOLOGY” }
},
”Cooperation”: { ”Partner_name”: ”Football Analytics Sweden AB”},
”National Subject Categories”: ”10201, 10207”,
”Other information”: {”Year”: ”2024”, ”Number of pages”: ”1,41”},
”Copyrightleft”: ”copyright”,
”Series”: { ”Title of series”: ”TRITA – EECS-EX” , ”No. in series”: ”2024:0000” },
”Opponents”: { ”Name”: ”A. B. Normal & A. X. E. Normalè”},
”Presentation”: { ”Date”: ”2022-03-15 13:00”
,”Language”:”eng”
,”Room”: ”via Zoom https://kth-se.zoom.us/j/ddddddddddd”
,”Address”: ”Isafjordsgatan 22 (Kistagången 16)”
,”City”: ”Stockholm” },
”Number of lang instances”: ”2”,
”Abstract[eng]”: €€€€

The application of computer vision in football is becoming more and more attractive. Computer vision can track players and actions allowing statistics and analysis of the sport. This study aims to recognize shots in football video data using computer vision.

Recognizing actions in video data has been explored using various techniques. Convolutional Neural Networks, CNNs, have dominated computer vision by proving great at understanding image data. However, extending CNNs to understand video data is a difficult task. In recent years 3D CNNs have been developed to understand the temporal information in video data, showing promising results in the field of action recognition.

In this study, 3D CNNs are compared to standard CNNs in their ability to predict shots in football video data. Football Analytics Sweden AB collects and owns the data which consist of short clips of shots and non-shots.

3D CNNs were found to outperform 2D CNNs in recognizing shots, achieving an accuracy of 85\% *on the test dataset. The results show that 3D CNNs have the potential to distinguish between different actions in football video data and could be used in further analysis of football matches using computer vision.*

€€€€,

"Keywords[eng]": €€€€
Machine Learning, Computer Vision, Action Detection €€€€,
"Abstract[swe]": €€€€

Användning av maskininlärning och datorseende inom fotboll blir alltmer attraktiv. Datorseende möjliggör detektion av spelare och händelser, vilket kan användas för analys och statistik inom sporten. Denna studie avser detektion av skott i fotbollsvideodata med hjälp av datorseende.

Att känna igen handlingar i videodata har undersökts med olika tekniker. Convolutional Neural Networks, CNNs, har dominerat datorseende genom att vara mycket effektiva på att förstå bilddata. Att utöka CNNs för att förstå videodata är dock en svår uppgift. Under de senaste åren har 3D-CNN utvecklats för att förstå hur datan ändras över tid i videodata, och har visat lovande resultat inom detektering av handlingar.

I denna studie jämförs 3D-CNN med vanliga CNN i deras förmåga att detektera skott i fotbollsvideodata. Football Analytics Sweden AB samlar in och äger data som består av korta klipp av skott och icke-skott.

3D-CNN visade sig överträffa 2D-CNN i att känna igen skott och uppnådde en noggrannhet på 85\% *på testdatan. Resultaten visar att 3D-CNN har potential att skilja mellan olika handlingar i fotbollsvideodata och kan användas i vidare analys av fotbollsmatcher med hjälp av datorseende.*

€€€€,
"Keywords[swe]": €€€€
Maskininlärning, Datorseende, Handlingsdetektion €€€€,
}

acronyms.tex

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
% The following command is used with glossaries-extra
\setabbreviationstyle[acronym]{long-short}

\newacronym{KTH}{KTH}{KTH Royal Institute of Technology}

\newacronym{CNN}{CNN}{Convolutional Neural Network}

\newacronym{3D-CNN}{3D-CNN}{3D Convolutional Neural Network}
\newacronym{2D-CNN}{3D-CNN}{2D Convolutional Neural Network}
\newacronym{RNN}{RNN}{Recurrent Neural Network}
\newacronym{ANN}{ANN}{Artificial Neural Network}
\newacronym{LSTM}{LSTM}{Long Short-term Memory}
```