Department of Informatics

Technical University of Munich

Master's Thesis in Informatics

Pose Estimation and Analysis for American Football Videos

Ludwig Dickmanns

Department of Informatics

Technical University of Munich

Master's Thesis in Informatics

Pose Estimation and Analysis for American Football Videos

Posenschätzung und -analyse für American Football Videos

Author:	Ludwig Dickmanns
Supervisor:	Univ.-Prof. Dr. Christian Mendl
Advisor:	Dr. Felix Dietrich
Submission date:	November 15th, 2021

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

November 15th, 2021 Ludwig Dickmanns

Acknowledgments

First up, I want to thank my advisor, Dr. Felix Dietrich, for his guidance and inspiration. Throughout the entire project, he supported me with valuable ideas and tips, for instance, for the individual pipeline steps. Furthermore, Felix supported me with proofreading towards the end of the work. Secondly, I am grateful that Felix and the Chair of Scientific Computation in Computer Science at the TUM Department of Informatics gave me the opportunity to write my thesis about two topics I am highly passionate about: American football in the problem domain and deep learning for computer vision in the solution domain. Thirdly, I want to thank Rob Hess and Alan Fern from Oregon State University for kindly providing me with their data set. Fourthly, my cousin Jonas helped me in the final stages of the thesis with proofreading, which I am highly thankful for. Fifthly, I want to express my thankfulness to my partner Laura for covering my back, especially during the final weeks of the project. Finally, I am deeply grateful for the support with proofreading through my father.

Abstract

Strategic analysis of American football is of high importance for live commentary, fans, coaches, and players. For the latter, this analysis is essential to prepare appropriately for the next match. Conventionally, this strategic analysis is done manually; that is, people watch and analyze videos. This is a tedious and time-consuming task. To alleviate some of the challenges related to manual strategy analysis, this thesis describes an approach that identifies defensive coverage schemes from single image frames. To increase robustness while reducing learning effort for classifiers, the approach includes several pre-processing steps. Professional game analysts often use the so-called "all 22" camera angle. "All 22" means that the camera is placed in a high position filming all 22 players on the field. The approach in this thesis uses the "all 22" camera angle as well. In order to be less susceptible to different lighting and weather conditions, a pose estimation network already robust against those conditions extracts body key points of the players. Most of the time, the camera is situated in the middle of the playing field. Therefore, plays and corresponding player key points on the camera's left side look different from plays on the right since camera angles are very different. Thus, domain knowledge is incorporated to transform poses into a 3D model in two steps. In the first step, a projective transformation is calculated to get a top-down view of the playing field. This projective transformation is then applied to the key points in the second step. Additionally, the projected key points are stacked in layers and centered above the center of the feet. This transformation results in a global 3D model of the playing field and players for a single frame, which is entirely independent of the initial frame's camera angle, resolution, and aspect ratio. Finally, this 3D model is used for two sample tactical analyses. For both analyses, a baseline, a convolutional neural network, and a custom ResNet were trained. The first task was to distinguish between man and zone coverage. The best-performing model predicts the correct coverage on the validation and test data set with 85% accuracy. For the second analysis, four more specific coverage classes were analyzed. In this case, the best model predicts the correct class with a 58.07% (validation) and 46.00% (test) accuracy. Overall, the work described in this thesis resulted in a robust and modular pre-processing pipeline, which instead of end-to-end learning, incorporates modeled domain knowledge to facilitate learning of labels.

Contents

1 Introduction

1.1 Motivation

In an article from 1959 in the IBM Journal of Research, the term “machine learning” was popularized by Arthur Samuel [1]. Due to computational restrictions, however, training sophisticated models was impractical. Fortunately, recent CPU and, especially, GPU advancements alleviated this restriction. Overall, this development opened up the path for artificial neural networks [2]. Today, neural networks consist of millions and even billions of parameters. In May 2020, Tom B. Brown et al. published a paper introducing a language model called Generative Pre-trained Transformer 3 (GPT-3), consisting of of 175 billion parameters [3]. All this shows that complex machine – and especially deep – learning models can now be trained to address complex tasks due to the developments in processor technology. One central research area of deep learning is computer vision. Here, models are used to solve image classification, object detection, and semantic segmentation tasks. Another task possible with today’s hardware is human body pose estimation, where models are trained to estimate the location of different joints, such as ankles, knees, hands, or eyes. This yields into the topic of this thesis: Applying pose estimation in combination with some transformations to analyze American football videos.

1.2 American Football: Concepts and Terminology

Since the application area of this thesis is American football, some central concepts and terminology need to be introduced. As for soccer, two teams of eleven players face each other on a rectangular field of play. For American football, this field is 120 yards long and 53 ⅓ yards wide. On either of the long side’s ends, there is a field goal and a ten-yard long end zone, as shown in Figure 1.1. In contrast to the figure, however, the field goals are at the end of the end zone and not inside. Further, there are rows of single-yard markers on both sidelines and two inside of the field. Those two inside the field are called hash marks. Moreover, there are lines distanced five yards from one another. Every second line has a number indicating the distance to the next closest end zone.

In contrast to more dynamic sports like basketball or ice hockey, American football is played play-by-play. That is, the offense of one team executes a run or a pass play. The play ends when a defender tackles the ball-carrying player successfully or when the ball carrier leaves the field of play. A play can also end if the passer – usually the quarterback (QB) – throws an incomplete (not caught) pass or a defender intercepts the ball. For the

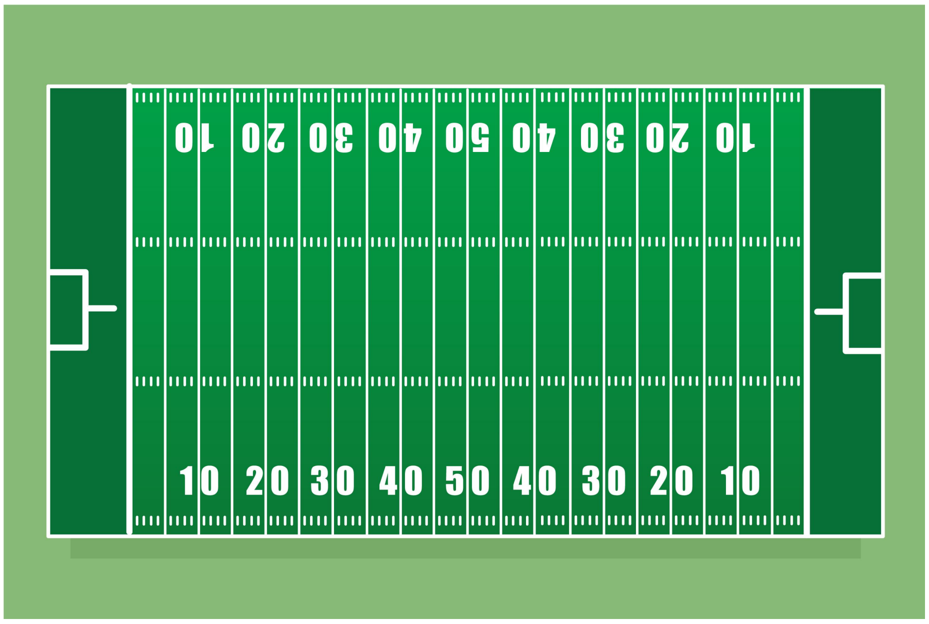


Figure 1.1: Schematic illustration of an American football field (Image: public domain).

latter case, ball possession changes. Then, the next play is carried out. Each play starts at the so-called line of scrimmage (LOS), which is at the position of the ball when the previous play terminated. For the subsequent play, the ball is placed on the LOS. The teams align on their respective side of the LOS. The offensive line, consisting of five players aligning close to each other and close to the LOS, protects the QB. The middle offensive lineman, the center, holds the ball on the LOS with one hand before the play. Notably, offensive linemen are not allowed to receive a pass. On a signal of the QB, the center hands or throws the ball through his legs to the QB. This is called the snap or snapping the football, and it starts a play. After the snap, both teams can cross the LOS, but not before. The goal of the team in possession of the ball is to advance the ball down the field into the end zone of the opposing defense. To achieve this goal, the offense has four chances – also called downs – to advance the ball ten yards. If they manage to gain those ten yards, they get a new set of downs. Otherwise, ball possession changes. This article [4] provides more information on the rules of football and in-depth information on football terminology can be found in this glossary [5].

1.3 Outline

The approach introduced in this research work uses 2D pose estimation combined with a projective transformation to extract 3D poses of American football players. This 3D data about the players is then used to analyze defensive pass coverage. Therefore, the remainder of this thesis is structured as follows: Chapter 2 introduces recent and relevant research work on American football analysis, deep learning in computer vision, and pose estimation. Chapter 3 explains the classification pipeline starting with an overview of the benefits over end-to-end learning in Section 3.1. Then, Section 3.2 through Section 3.7 describe the individual pipeline steps and how data was obtained and labeled. Finally, Chapter 4 concludes the thesis with a summary, a discussion of the results, and an outlook for future work.

2 State of the Art

This part of the thesis is split up into three parts. Firstly, Section 2.1 discusses relevant work in the problem domain: American football analysis. Secondly, the required concepts in the solution domain – deep learning for computer vision – are introduced in Section 2.2. Thirdly, Section 2.3 focuses on relevant work and techniques for pose estimation.

2.1 Analysis of American Football

The analysis of American football is as old as the sport itself, as teams need to prepare for their upcoming opponents. These investigations translated into research analyzing the sport from various perspectives over the recent decades. In the following, I want to lay out a categorized summary of the relevant scientific publications on the analysis of American football. I categorized the publications as follows: Firstly, there is work – primarily by researchers supporting their university's football team – on tools helping teams and coaches with player evaluation and decision making. Secondly, are papers on situational analysis. Thirdly, there are publications focusing on projecting or registering the field into a model or reconstructing the playing field. Those were particularly helpful for my work in that area. Further and fourth, I summarize the work on tracking and recognizing players. Moreover, there is work analyzing individual players and their contribution to a team's success. Moving closer to what I did in my final tactical analysis, there is research on, sixthly, formation identification and, seventhly, play classification. For those two, however, most of the publications focus on the offensive side, whereas my focus is on defensive play classification. This leads to the last category – defensive analysis – where I found just one relevant publication.

2.1.1 Supportive Tools

To support the decision-making of the American football team of the University of Virginia, the researchers around Harry Elkins developed a suite of data analytics tools and published the companion paper in 2017. Those tools support coaches with in-game decisions and help them with scouting opponents. All models are based on textual inputs or data from the IT department. Furthermore, those models were developed to be used intuitively by non-data scientists [6].

A year after [6], Jack Corscadden et al. published a similar paper, further supporting the team of the University of Virginia. They extended some of the previous work of Elkins et al.. Furthermore, they added a player performance model and vision-based tools helping to scout opposing offenses [7].

Also in 2018, another paper on supportive tooling was released. The researchers R. Yurko, S. Ventura, and M. Horowitz provided four contributions: Firstly, they implemented a package for the programming language R providing access to NFL play-by-play data. Secondly, they developed an approach to estimate the expected points for a play. Thirdly, they developed a model, which estimates the win probability using the aforementioned expected points. Finally, they implemented a framework that can measure the individual contribution of offensive players [8].

2.1.2 Situational Analysis

The research work laid out in this subsection is about recognizing certain situations during an American football game. They are relevant for this thesis since they analyze football with a vision-based approach. Furthermore, they could be used in future work as a preprocessing step to extract plays and relevant information from the raw video material of a football game.

One of those studies – "Camera View-based American Football Analysis" by Yi Ding and Guoliang Fan – focuses on classifying which camera or camera angle was used in a shot. For this classification, they use a set of extracted and selected field features. This approach can help to find videos in the proper camera angle for my analysis [9].

Since a typical American football match lasts up to three hours for 60 minutes of playing time, automatic video summarization can support the process of cutting raw material. In that regard, several approaches have been introduced. First, in 2001, B. Li and M. Ibrahim Sezan presented their framework, which is capable of detecting events and summarizing sports broadcast videos. For event detection and summarization, their framework has two approaches: a rule-based and a Hidden Markov Model (HMM) based approach, where the rule-based approach outperformed the HMM-based alternative. Their experiments also include two other sports: sumo wrestling and baseball [10].

A second paper on play start detection was published four years later, in 2005. The authors developed a feature extractor for the field color or field marks. To select features and make decisions, they use a boosting chain. [11]. Furthermore, they compare their results to the results from [10]. While their field color detection accuracy was just 3% to 5% higher than the one in [10], their overall play detection was much more accurate [11].

Lastly for play start detection, there is the work by Mahasseni et al. from 2013.Their paper describes, discusses, and evaluates their approach for detecting the start of an American football play. However, in contrast to [10] and [11], they only detect the start of a play without detecting the end of it [12].

After the play has started, the approach by K. Kin, D. Lee, and I. Essa from 2012 alleviates the process of finding regions of interest on the camera image. To find such regions, they employ an approach based on camera motion [13].

In 2010, Li and Chellappa developed an approach to segment group motions. That is, they separate the 22 players into offensive and defensive players based on their trajectories. In contrast to the pipeline developed in this thesis, their model is not invariant to different viewing angles [14].

Finally, in 2014, Chen et al. published a paper on the automatic annotation of play situations. They differentiate five situations: offense, defense, field goal, punt, and kickoff. As for the previous studies, this approach is entirely vision-based [15].

2.1.3 Field Registration, Projection, or Reconstruction

This subsection discusses research work in the area of registering or projecting the playing field into a model. Such work is especially relevant since I use a similar transformation in my pipeline as well. To illustrate such a projection, an example of this transformation from my work is shown in Figure 3.8 in Section 3.4.

In their 2007 paper, Rob Hess and Alan Fern explain their approach for registering a playing field into a model. In contrast to other methods, they developed their approach to work without sufficient distinctive image features. To tackle this issue, they introduce a concept called local distinctiveness. With that, they obtained sufficient matches between image and model for robust playing field registration. Moreover, their technique can be fully automated [16]. In two other papers [17, 18], they reuse their approach for further analyses. Those analyses, however, fall into another category and hence will be discussed at a later point of the state-of-the-art. In his dissertation, Rob Hess included the work of [16] and discussed an approach for automatically generating the reference set for video registration [19].

Similarly, Bernard Ghanem and Narendra Ahuja released a paper on their framework for video registration in 2012. While other work focuses on matching explicit structures, they register the video frames with image or image patch matching [20].

Additionally, approaches of other papers [7, 21, 22, 23, 24] transform video frames into a top-down view. However, those do not focus on the transformation but rather use it as a pre-processing step.

2.1.4 Player Tracking and Recognition

Another subcategory of American football analysis found comparatively often is player tracking and recognition. The earliest work I found in that regard is "Tracking Using a Local Closed-World Assumption: Tracking in the Football Domain" by Stephen S. Intille from 1994. In contrast to conventional trackers at that time, he incorporates contextual information – i.e. knowledge about American football – into tracking to improve performance [25]. In a later publication, Intille and Bobick use player tracking as a pre-processing step for play classification [26].

Besides their work in playing field registration, Rob Hess and Alan Fern likewise worked on multiple player tracking with discriminatively trained particle filters [27]. As for field registration, Rob Hess discusses this approach in his dissertation [19]. A similar approach – discriminatively trained particle filters – was also used in the pipeline of [24] to control a simulator from video observations. Another multiple player tracking approach based on particle filters was introduced in 2012 by T. Zhang, B. Ghanem, and N. Ahuja [28]. In order to further tracking accuracy, they incorporate their video registration developed in [20].

In his Master's Thesis, Amit Bawaskar developed an interactive tool to track a player. Interactive means that tracking errors are corrected by human interaction to improve tracking performance. In contrast to [19, 27, 28], his work uses an online multiple instance learning tracker. However, he demonstrates tracker independence by using a particle filter as well [29].

Four years later – for the 2018 MIT Sloan Sports Analytics Conference –, Omar Ajmeri and Ali Shah submitted a paper on vision-based player tracking. For tracking players, they use a color-based analysis. Player location and tracks are then used to identify formations, calculate distance ran and speed of players, and compare pass receivers' route-running skills. Overall, however, the descriptions of their approach are not very detailed [30].

2.1.5 Individual Contribution

As mentioned in Subsection 2.1.1, Yurko, Ventura, and Horowitz developed a framework to isolate and estimate the individual contribution of a player. Additionally, their framework can estimate the win above replacement (WAR) for a player. Thus, the name of the framework is *nflWAR* [8].

As the name suggests, "Going Deep: Models for Continuous-Time Within-Play Valuation of Game Outcomes in American Football with Tracking Data" provides a framework for game outcome valuations, even within a play. Most other databases, in contrast, compute their analysis on a play-by-play basis. To do so, they incorporate player tracking data. Moreover, the researchers developed a model for ball carriers, which estimates the yards a player is expected to gain. This estimate is based on the positions and trajectories of all 22 players on the field [31].

Another paper, which already appeared in Subsection 2.1.1, is [7]. Here, the researchers developed a model for player performance based on grades by a company called Pro Football Focus and sensor data from Catapult wearable vests [7].

The earliest relevant paper from the MIT Sloan Sports Analytics Conference is from 2016. In his submission, Jeremy Hochstedler discusses an approach to evaluate the decision-making of a quarterback from player location tracking data. His approach incorporates the "openness" of a receiver, that is, the distance between a receiver and the closest defenders [32]. Similarly, three years later, another relevant paper was submitted to the 2019 MIT Sloan Sports Analytics Conference by Brian Burke. This work focuses on assessing a quarterback's decision-making and performance from tracking data as well. However, both analyses and descriptions are more in-depth. For the implementation of his approach, Burke uses deep neural networks [33].

A paper focusing on the analysis of wide receivers was submitted to and won the "Open

Entry" category of the 2019 Big Data Bowl. In his paper, Nathan Sterken uses a convolutional neural network (CNN) in order to classify routes run by wide receivers. As an input of the CNN, he uses plots of the receiver's tracking data. In a second step, the classified routes are used to estimate optimal combinations of receiver and route using the metric "win probability added" from [8]. However, the second step did not reveal significant insights, as mentioned by the authors [34].

2.1.6 Formation Analysis

In this category of American football analysis, I summarize papers that analyze a team's formation. One paper in that area is [30], which already appeared in Subsection 2.1.4 because the researchers track players to analyze formations and route combinations. With regards to formation assessment, they carried out two analyses. Firstly, they classify the quarterback position into three categories with an 86.5% accuracy. Secondly, they classify entire formations with a 72.3% accuracy. For both analyses, they evaluated five different machine learning algorithms with "Classification and Regression Trees" performing best for both cases.

Besides their work in supportive tools and player performance evaluation, Corrscadden et al. also addressed formation analysis. Their approach includes a projective transformation, player detection, and a decision tree to classify the formation [7].

As the title suggests, "Automatic Recognition of Offensive Team Formation in American Football Plays" from 2013 focuses on classifying offensive formations. One of the authors is Bernard Ghanem, whose work from [20] was reused as a pre-processing step. Their process for formation recognition includes four steps. Firstly, they identify the frame in which the players align in the formation. Secondly, the line of scrimmage is detected. Thirdly, they identify the offensive team in order to use all previously acquired knowledge to, fourthly, classify the offensive formation. For accuracy, the formation frame is detected in 95% of the videos. The line of scrimmage is detected at an even higher 98%. Formation classification, however, only reached a 67% accuracy [21]. In 2014, [21] was reworked and appeared in a computer vision book from Springer [22].

In 2007, two similar papers were published that rather focus on the computer vision techniques employed. However, they apply those techniques to identify players and classify formations [17, 18]. As for the other work by Rob Hess, he discusses the findings in his dissertation [19].

Within two years, from 1998 until 1999, Lazarescu et al. published three papers on formation classification. Those three publications refer to formation classification with play classification, which can be misleading at times. In terms of employed techniques, their work combines a vision-based approach with a natural language processing-based approach [35, 36, 37].

Finally, there is the work by Stracuzzi et al. from 2011. They use observations of video material to control a simulator. Besides playing field registration and player tracking, they incorporate formation recognition into their approach [24].

2.1.7 Offensive Play Classification

This subsection addresses the classification of offensive plays. The earliest work I found in this area was published in 2001 by Intille and Bobick. Their approach uses visual inputs to recognize a predefined set of plays [26].

The most recent work, in contrast, was published in 2020. Here, Cameron Taylor uses situational information combined with a single pre-snap image only of the play to forecast both yardage outcome and the play call. Play call forecasting has two classes – run and pass – with the best model having a 61.4% accuracy [38].

A more in-depth analysis of offensive playcalling was introduced in 2003 by Lazarescu and Venkatesh. Their approach uses the camera motion to identify seven classes of football plays – short and long pass, short and long run, quarterback sack, punt, and kickoff. To train and evaluate their classifier, they used 782 plays – 68% of which were classified correctly [39].

In 2009, Li et al. introduced a paper explaining their approach for classifying offensive plays. As an intermediary step, they introduce a discriminative temporal interaction manifold (DTIM) to represent the cooperation between multiple objects compactly. With regards to experiments, they classified 56 plays into the following five categories: Combo Dropback, HITCH Dropback, Middle Run, Wideleft Run, and Wideright Run. Their best model yielded an 83.7% recognition accuracy [40]. A year later, in 2010, Li published another paper with Chellappa. They classify the same data into three more general classes: Dropback, Middle&Right Run, and Wideleft Run. The final recognition accuracy was a worse 70% [41].

In 2009, Swears and Hoogs wrote a short publication on the recognition of American football plays [42]. In 2012, they followed up on this work with a more in-depth paper. Their approach employs an HMM model to classify plays into both four and seven classes. They classified 25 pass plays for the four-class classification and achieved a probability of correct classification (Pcc) of 76%. For the more specific case – seven classes – their dataset consisted of 78 plays. However, Pcc declined to 57.7% [43]. Figure 2.1 depicts the four pass plays for the four-class classification (left) and the additional three run classes for the seven class classification.

In their 2009 paper, Siddiquie, Yacoob, and Davis discuss a vision-based, sparse multiple kernel learning approach to classify 78 offensive American football plays. Similar to [42, 43], they analyze a less and a more complex case on the same dataset. They classify the two classes run and pass for the less complex case with an 89.4% recognition rate. For the more complex case, plays are classified into the seven labels from [42, 43]: short pass, option pass, rollout pass, deep pass, run left, run right, and run middle. However, with a (compared to [42, 43]) much higher, 71.9% accuracy [23].

Finally for offensive play analysis, there is the conference paper by J. Varadarajan et al., which discusses a supervised topic model approach to analyze offensive plays. They were able to classify run and pass with a 88% accuracy. Furthermore, they analyzed more specific labels. That is, run or pass combined with a direction (left, mid, right). Here, they

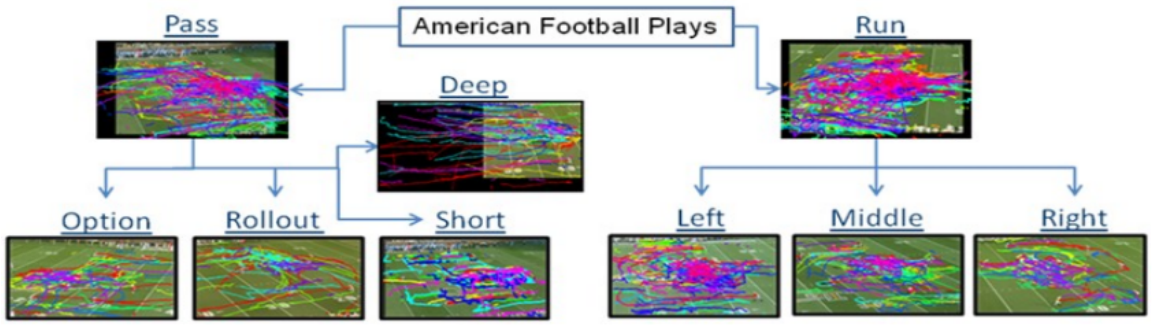


Figure 2.1: Four pass classes (left) and three run classes (right) used by Swears and Hoogs in [42, 43].

reached 70% accuracy [44]. The following figure illustrates an overview of their approach and their classification:

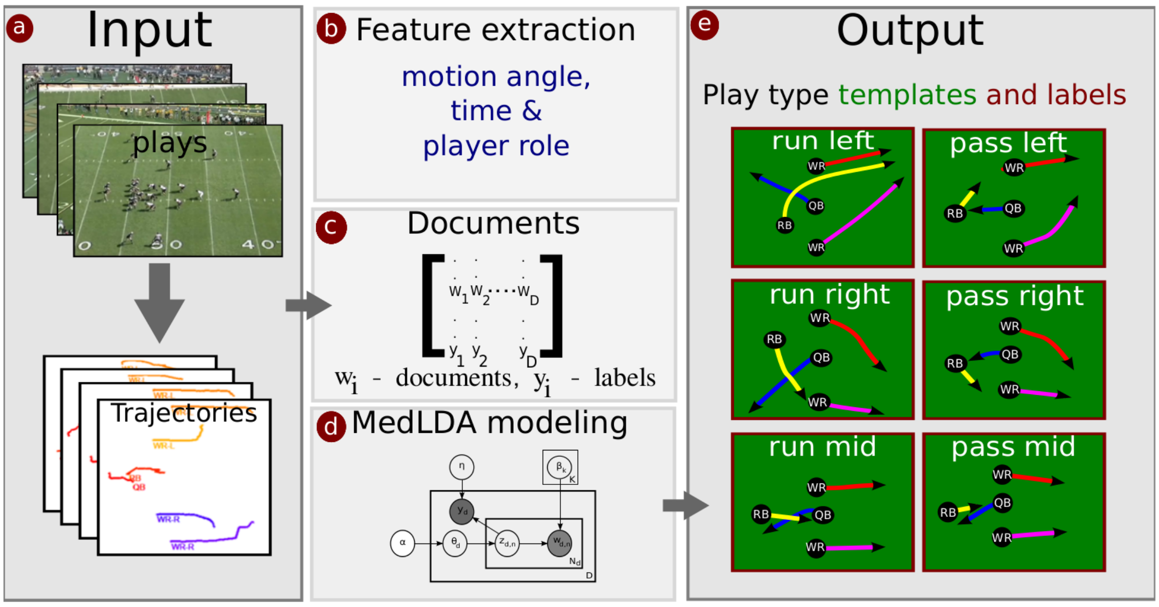


Figure 2.2: The approach of Varadarajan et al. to classify offensive plays [44].

2.1.8 Defensive Coverage Classification

The most relevant paper I found was published recently in 2020 by Dutta, Yurko, and Ventura. In contrast to all the previously mentioned papers, it discusses the defensive side of American football. In particular, they developed an approach to identify defensive pass coverage. Therefore, it is the work closest to mine in the problem domain. However, my work is distinguished in two ways. First, they use NFL player location tracking data, whereas I use a vision-based approach – not requiring the location trackers. Secondly, they classify coverage per player as illustrated in Figure 2.3. My approach, in contrast, was developed to classify the coverage of the entire defense. In their work, they use an unsupervised learning approach. This learning approach does not require any training data. Instead, they defined a feature set distinguishing between man and zone coverage. For that, they employ clusters obtained from mixture models. Their approach indicates the importance of each feature used [45].

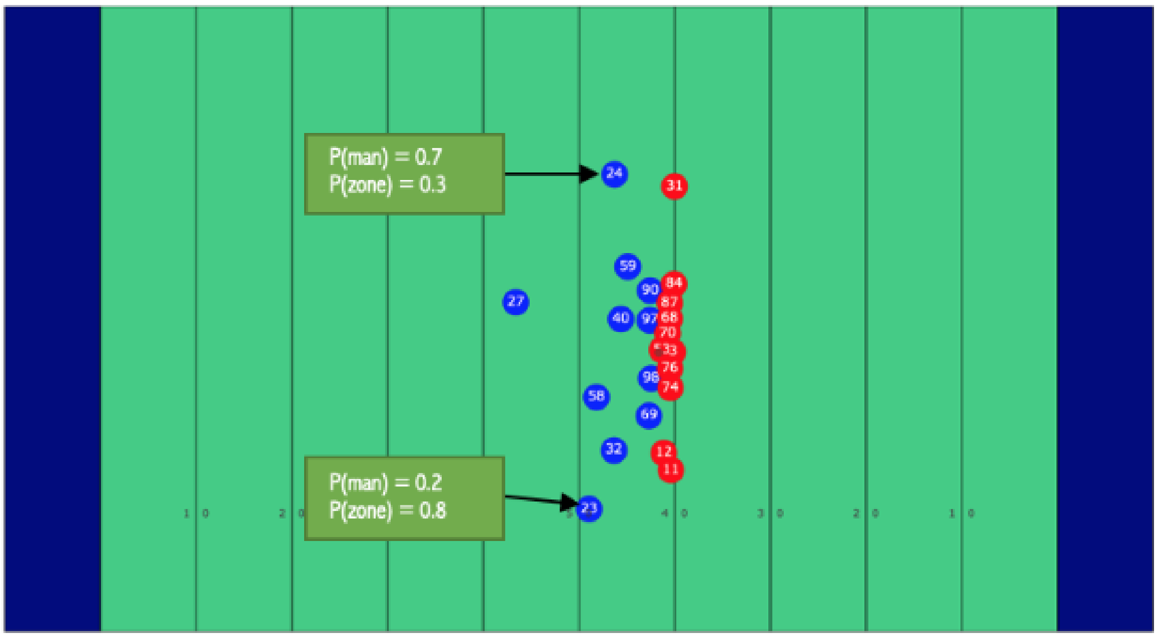


Figure 2.3: Probability for each player for playing man (m) or zone (z) [45].

2.1.9 Summary

The state-of-the-art American football analysis indicates that there is comparatively much work on supportive tools, situational analysis, field registration, and player tracking. Tactics of the offense in terms of formations and playcalling have been studied – again, comparatively – much. In contrast, defenses were not studied as extensively. Thus, one significant contribution of this thesis's final defensive tactical analysis is providing literature, a body of work, and inspiration for further studies of the defensive side of football.

2.2 Deep Learning for Computer Vision

Several technologies and techniques are required for the pose estimation at the beginning and the classification of defensive pass coverages at the end of the pipeline. However, as pose estimation can be seen as a separate field of research, it is discussed in a separate section – Section 2.3. This section focuses on deep learning concepts. The subsections discuss different building blocks for deep learning in computer vision. The first subsection discusses standard activation functions. A subsection on loss functions to determine the error in predictions follows. The subsequent two subsections explain how the error/loss is used to optimize neural networks with (1) backpropagation and (2) optimizers. Before the two subsections on convolutional neural networks and ResNets, I discuss two important topics – regularization and data augmentation. The final subsection concludes this section with an introduction to basic evaluation metrics.

2.2.1 Activation Functions

To introduce non-linearity into otherwise linear neural networks, several activation functions have been proposed. Typically, those activation functions are applied after the inputs of a layer have been multiplied with the layer weights. As recent references for the following functions, more in-depth explanations, and additional activation functions, consider [46, 47, 48].

Sigmoid Function

The Sigmoid or logistic function is a non-linear, bounded function that outputs a number in the interval $(0, 1)$. Therefore, it is often used to predict a probability, e.g. for a binary classification task (as in this thesis). The following formula defines it:

$$\frac{1}{1 + \exp(-x)}. \tag{2.1}$$

Softmax Function

Similar to the Sigmoid function, one can interpret the Softmax function's outputs as probabilities. In contrast to the Sigmoid function, however, Softmax can output a probability distribution for more than two classes. As a probability distribution, the sum of the output probability vector is one. Thus, the Softmax activation function is used in the output layers of multi-class, deep learning architectures. This formula defines the output Softmax activation function for the i-th class:

$$\frac{\exp(x_i)}{\sum_j \exp(x_j)}. \tag{2.2}$$

x_i is the i-th neuron's output, and the denominator sums up $\exp(x_j)$ over all neurons j.

Hyperbolic Tangent Function

The hyperbolic tangent function (tanh) is another bounded, non-linear activation function. However, it is zero centered in the interval $(-1, 1)$, and alleviates some of the problems of the Sigmoid function. The following is the formula defining the tanh function:

$$\frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}. \tag{2.3}$$

Rectified Linear Unit Function

The rectified linear unit (ReLU) function is a commonly applied activation function in state-of-the-art deep learning models. Due to its simplicity, it is much faster to compute than the tanh or the Sigmoid activation functions.

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

2.2.2 Loss Functions

Besides activation functions, there is a second set of functions relevant to training neural networks – loss functions. As explained in the following subsections, networks are trained by propagating the gradient of a loss of predictions backward through the network. This subsection addresses some loss functions used for classification tasks. For all loss functions, N is the number of samples, y_i is the actual label – either zero or one – of the i-th sample and $\hat{y}_i$ is the predicted label – which can be interpreted as a probability – of the i-th sample.

Mean Absolute Error

The following formula gives the mean absolute error (MAE) loss function:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|. \tag{2.5}$$

Mean Squared Error

The mean squared error (MSE) formula is the following:

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2. \tag{2.6}$$

Binary Cross-Entropy Loss

The binary cross-entropy (BCE) loss function is given by:

$$BCE = -\sum_{i=1}^{N}(y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log(1 - \hat{y}_i)). \tag{2.7}$$

Cross-Entropy Loss

The following formula defines the (categorical) cross-entropy (CE) loss:

$$CE = -\sum_{i=1}^{N}\sum_{k=1}^{K}(y_{ik} \cdot \log \hat{y}_{ik}). \tag{2.8}$$

Here, the additional variable K was added for different classes. It is also easy to obtain that CE is the generalization of BCE for multiple classes.

2.2.3 Backpropagation

Backpropagation is the common technique to compute the gradient of the loss function w.r.t all network weights and biases. It is called backpropagation because it employs the chain rule to propagate the gradient *backward* through the network. The gradient is calculated because it can be used to find (local) minima of the loss function w.r.t. weights and biases. Thus, optimizers can adjust network parameters to produce a smaller loss. Such optimizers are discussed in the following subsection. Backpropagation was popularized by Rumelhart, Hinton, and Williams [49, 50]. However, other researchers (e.g. [51]) discovered it independently as well.

2.2.4 Optimizers

This subsection introduces and discusses some relevant optimizers. For an extensive summary of gradient descent optimization algorithms, see e.g. [52, 53].

(Batch) Gradient Descent

Standard (batch) gradient descent computes the gradient of the loss function L w.r.t the network parameters θ for the *whole* training set. As the gradient points in the direction of the steepest increase of the loss, weights are adjusted in the opposite direction with a specified learning rate η:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta). \tag{2.9}$$

A significant drawback of batch gradient descent is that the entire dataset is used to calculate the gradients. Therefore, it can be slow or even impractical for large datasets. Another drawback is that incremental or online learning is not possible. Thus, optimization must be re-run to incorporate new training data [52].

Stochastic Gradient Descent

Stochastic gradient descent (SGD) addresses the two drawbacks of batch gradient descent. Instead of performing a gradient step over the entire dataset, SGD updates weights and biases for each individual training sample:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta; x_i, x_i). \tag{2.10}$$

x_i and y_i are the i-th sample – features and label, respectively – of the training set. The structure of SGD allows for much faster and online learning. However, SGD can also lead to noisy weight updates since it only uses a single data point to update weights [52].

Mini-Batch Gradient Descent

Mini-Batch Gradient Descent combines the advantages of the previous two gradient descent algorithms. Instead of a single sample, a mini-batch of n samples is used for optimization. This way, gradients are less noisy than those for SGD while still allowing for large datasets and online learning:

$$\theta = \theta - \eta \cdot \nabla_\theta L(\theta; X_i, Y_i). \tag{2.11}$$

X_i and Y_i are the features and labels of the i-th mini-batch, respectively. Again, there are also drawbacks to this optimizer. That is, gradients are not scaled individually across different directions. Therefore, a conservative learning rate is required in order to avoid divergence. Further, SGD can get stuck in plateaus surrounding saddlepoints [52].

Adam

To tackle the issues of previous optimizers, several other optimizers have been proposed. For an overview, see section 4 of [52]. A prevalent one of them is the adaptive moment estimation (Adam) algorithm. Adam was proposed by Kingma and Ba in 2015 [54]. The researchers designed this algorithm to compute an adaptive learning rate for each parameter. Thus, convergence is faster, and it alleviates the training of deep neural networks [52].

2.2.5 Regularization

Regularization is employed to make models less prone to overfitting, thus, generalizing better to unseen data. For my thesis, I worked and experimented with the following four: L1- and L2-Regularization, dropout, and batch normalization. For an extensive overview of regularization techniques, see [55].

L1-Regularization

L1-Regularization is a regularization technique, where a regularization loss is added to the loss function. In particular, the following term is added to the loss function:

$$R(\theta) = \sum_i \lambda_i \cdot |\theta_i|. \tag{2.12}$$

θ_i are the individual weights, and λ_i is the individual regularization factor of weight θ_i. Thus, high absolute network parameters values are penalized. Therefore, sparse weight matrices are favored by L1-Regularization, and it can be used for feature selection.

L2-Regularization

Similar to L1-Regularization, L2-Regularization adds a regularization term to the loss function:

$$R(\theta) = \sum_i \lambda_i \cdot \theta_i^2. \tag{2.13}$$

θ_i and λ_i are defined as for L1-Regularization. In contrast to L1-Regularization, L2-Regularization enforces similar values for all weights. Thereby, weights incorporate all information.

Dropout

Dropout is a regularization technique to prevent the neurons of a neural network from co-adaption. This is achieved by randomly omitting a percentage of a layer's neurons for each training batch. Thus, many sub-networks are optimized at training time. At test time, the mean of all of the sub-networks is used [56, 57]. The following figure from [57] illustrates a neural network before (left) and after applying a dropout of 50% (right):

Batch Normalization

In 2015, Sergey Ioffe and Christiean Szegedy introduced batch normalization (BN). It was explained as a technique to accelerate the training of deep neural networks. However, the authors also noted that it also regularizes models. BN can even render dropout unnecessary. Instead of using a sub-model for each training batch like dropout, BN normalizes layer inputs for each batch [58].

2.2.6 Limited Data and Class Imbalances

Training neural networks can be problematic in the presence of limited or imbalanced data. This issue can be addressed by under- and oversampling and data augmentation. While

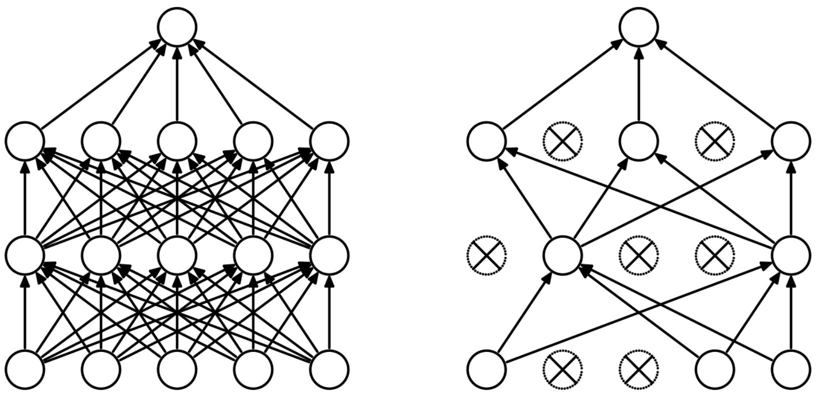


Figure 2.4: A neural network without (left) and with dropout (right) [57].

data augmentation can be seen as a data-based regularization technique [55], I decided to address it here because it has two benefits. As mentioned, data augmentation can be used to increase dataset size. Thus, it decreases the chances of overfitting (data regularization). However, data augmentation is also often employed to balance out a dataset. This is achieved by increasing the number of samples just for underrepresented classes. Before discussing data augmentation in detail, two other techniques to address class imbalance are discussed: Under- and oversampling.

Undersampling

As the name suggests, this technique balances a dataset by undersampling majority class samples to the number of samples of the minority class. Besides balancing the dataset, undersampling has another benefit. That is, by reducing the overall number of training samples, it decreases training time. However, this also introduces a drawback: It can remove important information from excluded samples of majority classes.

Oversampling

Instead of excluding samples of all majority classes, one can also increase the instances of the minority classes to match the number of samples of the single majority class. Typically, this is done by replicating instances of the minority classes. In contrast to undersampling, all information of the majority class is retained. However, a larger training set increases training time. Furthermore, it induces an increased risk for overfitting minority classes since samples are just replicated.

Data Augmentation

A standard method to alleviate the problems of oversampling is data augmentation. While data augmentation still increases training time due to more training samples, it also increases variance with transformed replications. Training samples in this thesis are images or image-like (3D tensors with width, height, and channels) data. Therefore, the focus is on a specific image data augmentation technique. For a comprehensive overview of image data augmentation, see this article [59] by Connor Shorten and Taghi M. Khoshgoftaar. The augmentation method of choice is the geometric transformation of flipping. For image data, flipping horizontally is more common than flipping vertically [59]. However, both are useful for my work. Figure 2.5 depicts a sample image of a cat (top-left), a horizontal flip (top-right), a vertical flip (bottom-left), and the combination of both horizontal and vertical flipping. A model trained on such augmented data can thus learn different representations of a cat.

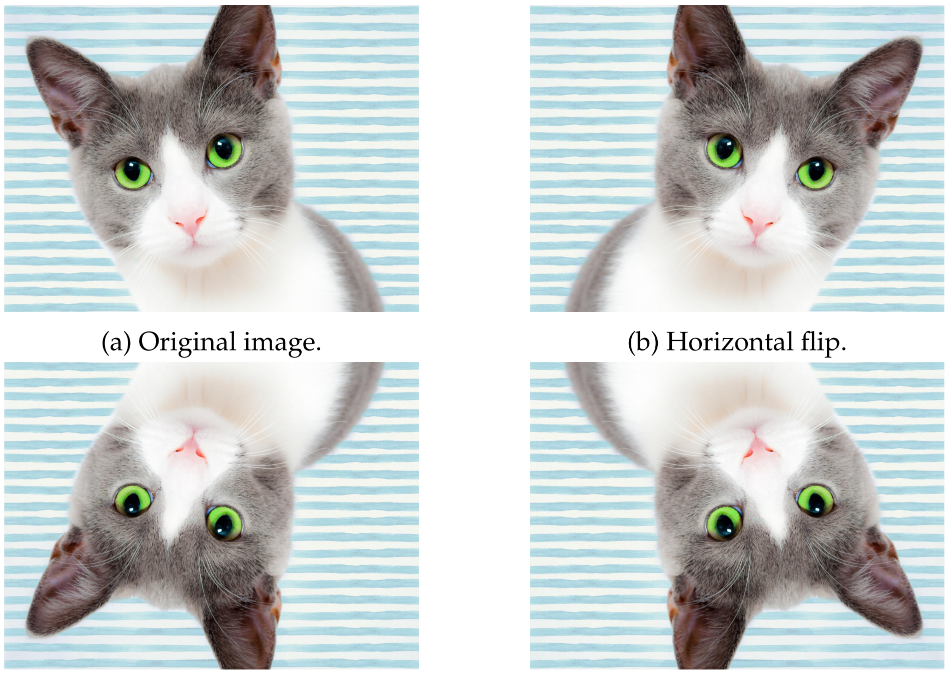

(a) Original image. (b) Horizontal flip. (c) Vertical flip. (d) Horizontal and vertical flip.

Figure 2.5: Horizontal and vertical flipping (Image: public domain).

2.2.7 Convolutional Neural Networks (CNNs)

Regarding state-of-the-art deep learning for computer vision, one of the most basic yet most important concepts is the convolutional neural network architecture. Inspired by Hubel and Wiesel's work on the visual nervous system [60, 61], Fukushima introduced the neocognitron in 1980 [62]. In literature, the neocognitron is often referred to as the predecessor of CNNs. Ten years later, in 1989/90, CNNs were introduced by Yann LeCun et al. [63, 64]. Since then, CNN architectures have evolved drastically. Here, a clear trend is towards more depth, with literature supporting this trend, e.g. [65, 66]. With increasing depth, however, networks become harder to optimize due to exploding or vanishing gradients [67]. Careful weight initialization [67, 68] and batch normalization [58] can alleviate those problems. However, for even deeper networks, the degradation problem was discovered [69, 70]: At some point during training, accuracy saturates and then degrades. However, this degradation is not due to overfitting as training accuracy degrades as well. An architecture resolving this issue and allowing for much deeper models is discussed in the following.

2.2.8 Residual Networks (ResNets)

Residual networks were first introduced by He et al. in 2014 [71]. As mentioned in the previous subsection on CNNs, deeper networks can perform better than their shallower counterparts. However, this also induces problems. One of these problems is the degradation of training accuracy with deeper models. Thus, ResNets were designed to alleviate this problem. To achieve this goal, they introduce a shortcut, identity mapping of the input of a convolution block to its output. This shortcut connection enables a direct backward flow of the gradient. The following figure depicts a standard residual block (left) and a bottleneck block (right):

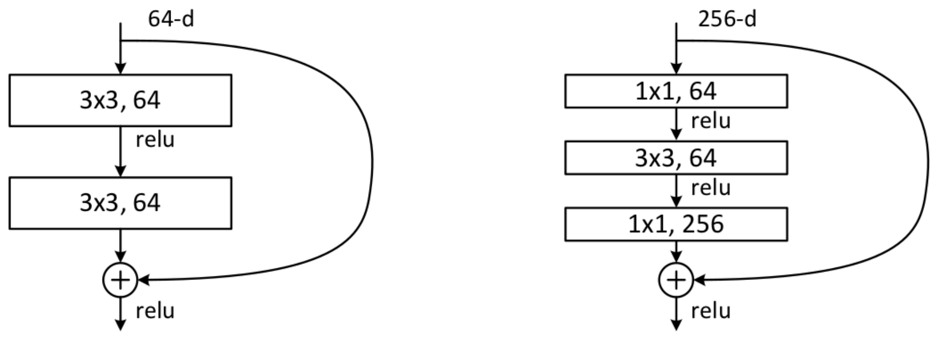


Figure 2.6: Standard residual block (left) and bottleneck block (right) [71].

ResNets employ a stride of two for the first convolution layer in a residual block to exchange image size for channel depth. However, since shrinking the image and increasing channel depth render a direct identity mapping impractical, the researchers propose two solutions. Either (1) padding the input with zeros or (2) using one-by-one convolutions. Both options use a stride of two for image dimension reduction. Furthermore, they discuss a bottleneck layer, which was introduced mainly for practical reasons. That is, they put a one-by-one convolution before and after the three-by-three convolution to reduce dimensions for the three-by-three convolution. However, the bottleneck design is susceptible to the degradation problem as well. Instead of dropout, ResNets use batch normalization –

before activation, right after convolutions. With those three building blocks, ResNets can be built by stacking residual blocks. The head of the standard residual network consists of a global average pooling layer after the residual blocks and a fully-connected layer with softmax activation for classification. The authors train the networks with SGD [71].

ResNets are highly relevant for this thesis, as the pose estimation network used in the approach uses a ResNet-50 backbone. Moreover, the network implemented to classify defensive coverage from 3D points is a ResNet as well.

2.2.9 Evaluation Metrics

In order to evaluate the models classifying the 3D point clouds, some evaluation metrics are discussed in the following. All metrics are discussed for a binary classification case. However, their multi-class extensions are straightforward.

Confusion Matrix

A confusion matrix compactly depicts the ground truth labels – negative and positive – as rows versus the predicted labels as columns. As obtainable from Table 2.1, true negatives (TN) are correctly classified as negative samples, while true positives (TP) are correctly classified as positive samples. In turn, mispredictions can be categorized as false positives (FP) – predicted positive; actually negative – and false negatives (FN) – predicted negative; actually positive. Besides their concise illustration of results, confusion matrices also facilitate the calculation of the other metrics.

	negative	positive
negative	TN	FP
positive	FN	TP

Table 2.1: Schematic confusion matrix.

Accuracy

One of the most basic evaluation metrics for classification is accuracy. It measures how man labels were predicted correctly. That is, the rate of TNs plus TPs divided by the total number of samples:

$$Accuracy = \frac{TN + TP}{TP + TN + FP + FN}. \tag{2.14}$$

Precision

Precision is a metric to evaluate the accuracy of positive predictions. Thus, the number of true positives is divided by the number of all positively classified samples:

$$Precision = \frac{TP}{TP + FP}. \tag{2.15}$$

Recall

Finally, recall describes the rate of detected positives to all actually positive samples. Hence, it is also called true positive rate (TPR). TPR is computed by dividing all positive classifications by the total number of actually positive samples:

$$Recall = \frac{TP}{TP + FN}. \tag{2.16}$$

2.3 Pose Estimation

Apart from implementation-based categorization, human body pose estimation can be categorized into single-person pose estimation and multi-person pose estimation. For this thesis, the latter is more relevant. For a recent survey on the former, see e.g. [72] by Zhang, Zhu, and Wang. Moreover, pose estimation (PE) can be further distinguished in terms of their implementation. That is, conventional vs. deep learning-based pose estimation. For the latter, there is a very recent survey [73] as well, which also lists some surveys on conventional methods for human pose estimation. However, the approach developed for this thesis employs a deep learning-based person pose estimation network, particularly a deep learning-based multi-person 2D pose estimation. Thus, the remainder of this section focuses on those. For single-person and 3D PE, [73] also provides an overview. Regarding 2D multi-person PE, [73] uses a common distinction for state-of-the-art approaches: bottom-up vs. top-down. Both approaches are discussed in more depth in the following two subsections.

2.3.1 Bottom-up

As the name suggests, bottom-up approaches start by detecting individual joints. Then those joints are assembled into full human bodies in a second step. One prevalent pose detection system using a bottom-up approach is OpenPose [74, 75, 76, 77]. OpenPose is very fast, offering even real-time capability. However, drawbacks outlined in [73] are problems with both low resolution and occlusions.

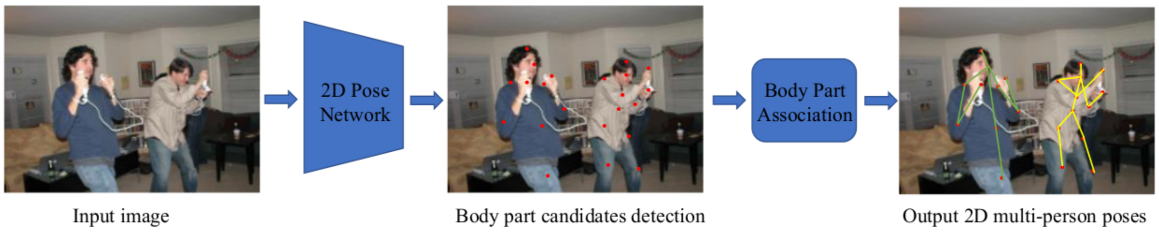


Figure 2.7: Bottom-up pose estimation (figure adapted from [73]).

2.3.2 Top-down

In contrast to bottom-up pose estimation, top-down PE approaches the problem in the opposite direction. An object detector first detects the persons, and then a single-person pose estimator predicts the joints. Top-down models tend to be slower than their bottom-up counterparts because bottom-up networks do not run single person pose estimation separately on each detected person. However, they tend to be more accurate and are less susceptible to occlusions, in turn [73].

A top-down architecture of particular interest for this thesis is Mask-RCNN, as introduced by Kaiming He et al. in 2018 [78]. Internally, Mask-RCNN uses a ResNet-50-feature pyramid network (FPN) as a feature extractor. Further information on FPN can be obtained from [79].

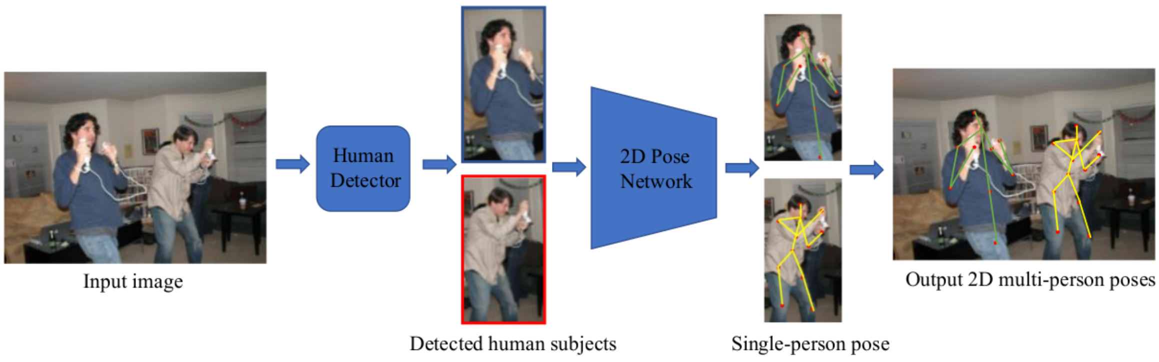


Figure 2.8: Top-down pose estimation (figure adapted from [73]).

3 Pose Estimation and Analysis for American Football Videos

The central contribution of this thesis is the usage of a classification pipeline consisting of five intermediary steps instead of direct end-to-end classification. In particular, this pipeline is a combination of a model-based and learning-based approach. Since specific knowledge can be extracted via minor manual labeling, the classification model has less to learn. The five steps of the pipeline are:

1. Video input
2. Pose estimation
3. Playing field registration via a projective transformation
4. 3D transformation from 2D poses
5. Classification

Before discussing the five steps and their implementation, Section 3.1 describes why this pipeline was developed, highlighting benefits obtained over direct end-to-end learning. Then follow sections on the individual parts of the pipeline and corresponding implementation work. Thus, Section 3.2 discusses how the video input for the first step of the pipeline was obtained. For the next step, Section 3.3 discusses the three pre-trained pose estimation models investigated and an in-depth description of the model of choice. Next, Section 3.4 contains a description of how the video input was projected into a field model. Before the next step of the pipeline – 3D transformation –, follows Section 3.5 on how the data was labeled for the final classification. This section was inserted here because a tool developed for the playing field registration was reused for labeling. Continuing with the pipeline parts, Section 3.6 explains how a 3D model was extracted from the 2D pose key points using the projection from the third step. Finally, a classification of the 3D models into two and four classes is described in Section 3.7.

3.1 Benefits of Additional Pre-Transformations

Before explaining how I implemented the different steps of the processing pipeline, I want to outline the benefits of the intermediary steps. In comparison to end-to-end learning directly from the input images to the final labels, I obtained several benefits, which are explained in the following subsections.

3.1.1 2D Pose Estimation

Employing pose estimation yields two benefits. Firstly, it induces robustness against different weather, lighting, and coloring conditions, since the pose estimation network used in this thesis is already trained to be robust against those. Figure 3.1 depicts consistent detection of the player key points despite drastically different lighting and colors. Secondly, the pose of players can reveal indicators for the type of coverage a defense plays. In zone coverage, for instance, players stay to keep turned towards the quarterback, whereas, in man coverage, a defender turns around to run with the assigned man.

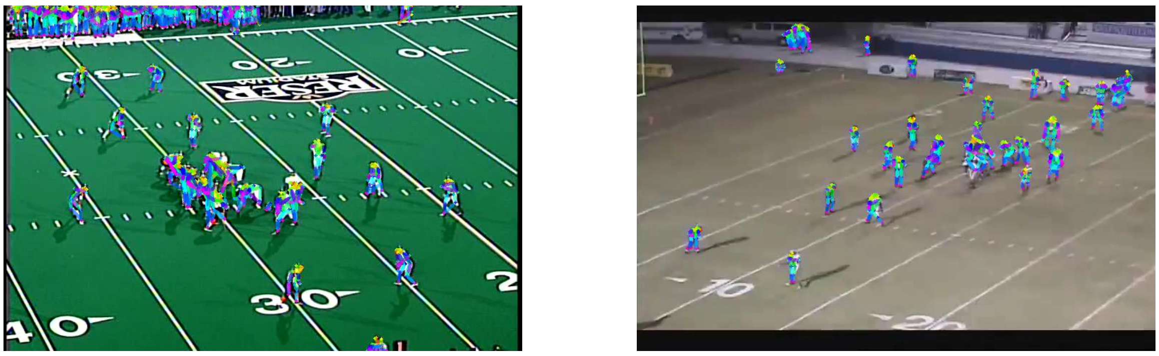

(a) Pose estimation (OSU Dataset).   (b) Pose estimation (Sports-1M Dataset).

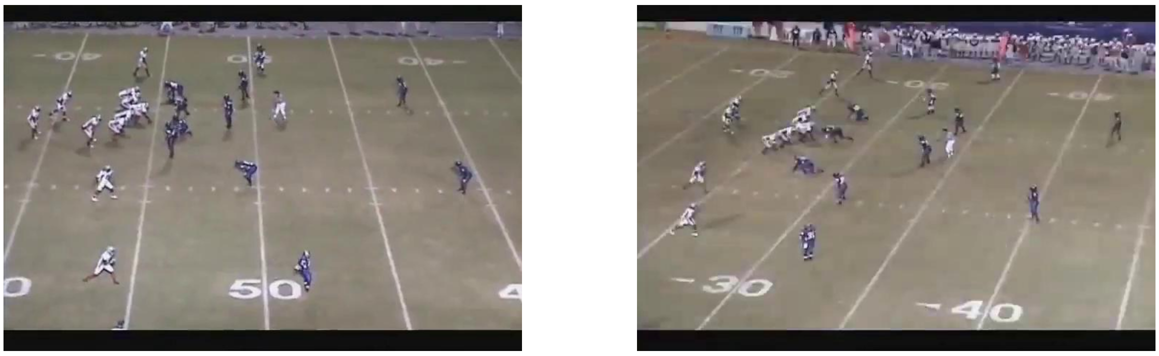
Figure 3.1: Robustness induced by 2d human pose estimation network.

3.1.2 Playing Field Registration

A significant benefit of the playing field registration is that it makes the field model independent of video resolution and aspect ratio. That is because the playing field is projected into a uniform model. Therefore, no adjustments are needed for different video resolutions or aspect ratios. Extracting the playing field and creating a model of the field yields even further benefits. For once, situations could be analyzed both locally, centered around the line of scrimmage, and in the context of the position on the field. Moreover, it is straightforward to exclude non-player persons from the remaining pipeline by looking at whether their feet are located within the playing field or not.

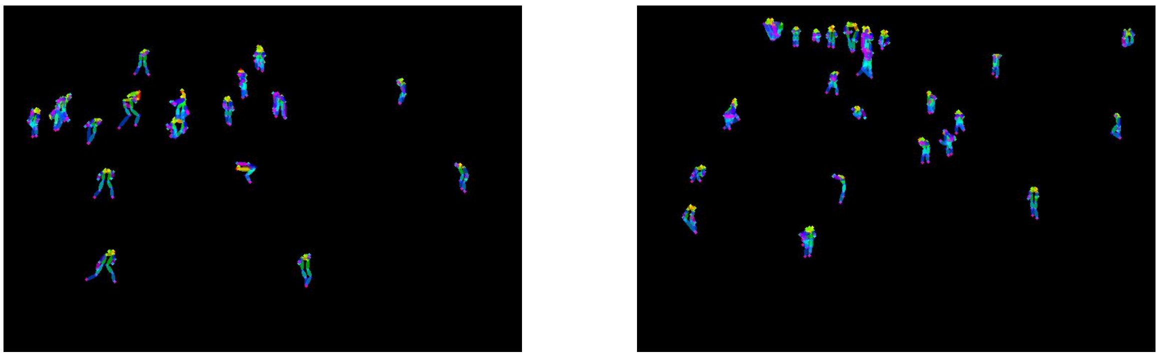
3.1.3 3D Point Extraction

Through the 3D extraction of player joints, not only the field of play but the entire model becomes independent of the video resolution and aspect ratio. The reason for this is that the same projection as in the previous step is applied to the 3D points. As for the playing field registration, no additional adjustments are needed for different video resolutions or aspect ratios. Furthermore, using projected 3D points in comparison to the 2D pose estimation points makes the analysis independent of camera angles. The skeletons (of the relevant players) of two similar situations can look significantly different just due to field position. To the human observer, the two situations in Figure 3.2 appear similar. However, Figure 3.3 depicts the difference in the perspective on the skeletons and their alignment. Finally, the 3D skeletons and their alignments look similar again in the 3D model in Figure 3.4:



(a) Situation 1. (b) Situation 2.

Figure 3.2: Two similar situations with different field position.

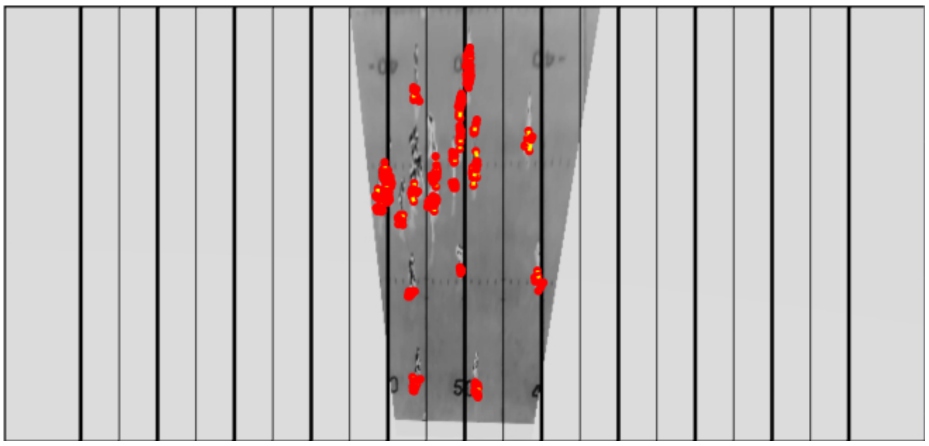


(a) Skeletons of situation 1. (b) Skeletons of situation 2.

Figure 3.3: Skeletons of the two situations in Figure 3.2.

3.1.4 Summary

All of the benefits mentioned above yield another, more general one. My approach introduces a camera-angle- and resolution-independent intermediary model and obtains robustness against weather and lighting conditions from the pose estimation. Therefore, the classification algorithms do not need to be trained or adjusted for variations in any of those parameters. Thus, domain knowledge is used and modeled to reduce learning effort.


(a) 3D skeletons of situation 1.

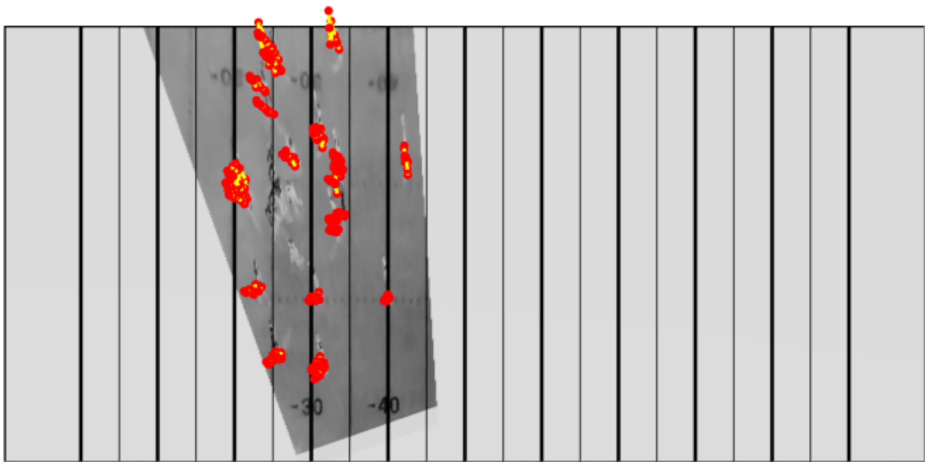

(b) 3D skeletons of situation 2.

Figure 3.4: 3D skeletons of the two situations in Figure 3.2.

3.2 Obtaining Data

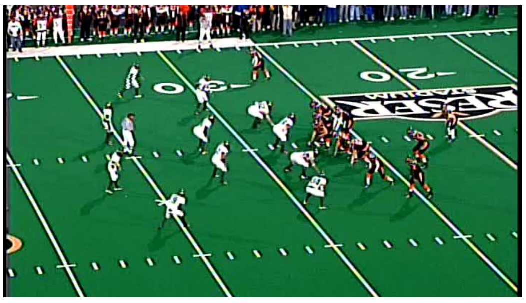
For football analysis, particularly the tactical analysis, professionals and coaches use a unique camera angle called "all 22". As the name suggests, the camera is positioned high above the players filming all 22 of them. In order to analyze and learn from videos, two things are required: Firstly, the raw videos, and, secondly, labels for the videos. The labeling is discussed in Section 3.5. Regarding raw video material, the initial plan was to do pose estimation and analysis on NFL videos in the all 22 camera angle. However, due to legal restrictions, this plan was adapted to use 60 single play all 22 videos from the following two video sources:

3.2.1 OSU Dataset

For once, the OSU researchers Rob Hess and Alan Fern kindly provided their data set from their studies [16, 17, 19, 27]. This data set consisted of 20 cut, single-play videos. Hence, little additional effort was needed to use the videos.

3.2.2 Sports-1M Dataset

Moreover, Andrej Karpathy et al. published a paper and corresponding data set, where they classified one million open-licensed videos from YouTube [80]. Luckily, there were around 2000 American football videos contained in this data set. Hence, I sought through all 2000 videos looking for the proper all 22 camera angle. This process in itself took a considerable amount of time. In contrast to the OSU videos, however, the YouTube videos are not cut into single plays. Therefore, additional work was needed to cut the videos. Thus, I selected a video with 40 plays in the all 22 camera angle and cut out each of the 40 plays.


(a) OSU Dataset.

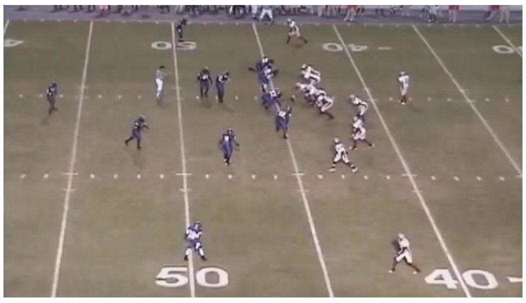

(b) Sports-1M Dataset.

Figure 3.5: Sample frames of the OSU Dataset and the video from the Sports-1M Dataset.

3.2.3 Additional Cutting

Initially, all 60 videos from both sources were cut from seconds before the snap until the play ended. After the snap, however, the camera often zoomed out substantially. This zooming, combined with the low resolution of the videos, led to low detection rates of the subsequent pose estimation step. Hence, the videos were cut shorter if the camera zoomed out too far.

3.3 Pose Estimation

The next step in the classification pipeline is 2D multi-person pose estimation. For this step, three pre-trained multi-person 2D pose estimation networks from three frameworks were evaluated. The three frameworks are: TensorFlow [81], OpenPose [74, 75, 76, 77], and PyTorch [82]. Each corresponding models is discussed in a separate subsection: the TensorFlow model in Subsection 3.3.1; the OpenPose model in Subsection 3.3.2; and the PyTorch model in Subsection 3.3.3. Finally, Subsection 3.3.4 provides a comparison.

3.3.1 TensorFlow

TensorFlow Hub [83] offers a variety of pre-trained models. For 2D multi-person human body pose estimation, the site offers a model called MoveNet [84]. To extract features, it uses a MobileNetV2 [85] FPN. From the extracted features, joint locations are predicted by CenterNet [86] heads. However, MobileNet was designed to only detect up to six persons. Thus, other options were investigated.

3.3.2 OpenPose

OpenPose is a state-of-the-art, real-time, bottom-up, multi-person 2D pose estimation system. Besides human body key points, it is also capable of detecting hand, facial, and foot key points. Different pre-trained pose estimation networks are available and easy to install via either a traditional website [87] or their GitHub [88]. As a bottom-up approach, it is relatively fast. In particular, OpenPose is even entirely invariant for the number of people in an image [74]. However, OpenPose has difficulties detecting occluded or low-resolution joints [73]. This problem was further confirmed in my experiments. Throughout different videos, the pose estimator did detect just a few of the 22 players with standard configuration. Even by setting a flag that maximizes detected joints (at the cost of much more false positives), all 22 players were not detected consistently. To exacerbate those problems for my application, OpenPose must be run on a GPU to be fast and requires at least – but rather more than – 16GB of VRAM (RAM on the GPU) to achieve optimal results [89]. As the two systems I had access to had only 12GB of VRAM, parameters needed to be reduced, which in turn reduced accuracy drastically. Therefore, OpenPose was run on a CPU and consumed up to 20GB of RAM with the aforementioned optimal configuration. However, in this configuration, the processing time for a single video is two to three hours. Thus, the benefit of OpenPose – fast computation time – was not given anymore. In sum, problems with occlusion, low-resolution videos, and low true positives (or more true positives at the cost of much more false positives) unfortunately rendered OpenPose impractical for this thesis.

For model training in the final step of the pipeline, I carried out some experiments on Google Colab [90], where I discovered that some instances offer 16GB of VRAM. Hence, OpenPose can be run on those with optimal configurations. This reduced processing time significantly to three to four minutes per video. However, the other problems remained.

3.3.3 PyTorch

In contrast to the bottom-up OpenPose network, PyTorch offers a pre-trained, top-down network for 2D multi-person pose estimation [91]. The model is an implementation of the Mask-RCNN with ResNet-50-FPN backbone [78]. This ResNet backbone was pre-trained on ImageNet [92] to serve as a feature extractor. Pose estimation was pre-trained on the dataset from the 2017 COCO Keypoint Detection Task [93]. For this thesis, I used the pre-trained versions of both the feature extractor and the pose estimator. Applying the model was straightforward with just minor pre-processing required, i.e. permuting the channels of the images to the correct order and normalizing the pixel values to values between zero and one. Compared to the two to three hours *per* video of OpenPose (CPU), the PyTorch model was rather fast, requiring under an hour for *all* videos (on average under a minute for each of the 60 videos). However, the increased speed is mainly attributed to the fact that model inference can be run on the GPU. The pose estimation network of PyTorch outputs a tensor of shape $N \times 17 \times 3$ where N is a variable number of detected persons. For each person, 17 key points are detected with x- and y- coordinates and whether the point is visible or not. The 17 key points are listed below and depicted in Figure 3.6.

1. Nose
2. Left eye
3. Right eye
4. Left ear
5. Right ear
6. Left shoulder
7. Right shoulder
8. Left elbow
9. Right elbow
10. Left wrist
11. Right wrist
12. Left hip
13. Right hip
14. Left knee
15. Right knee
16. Left ankle
17. Right ankle

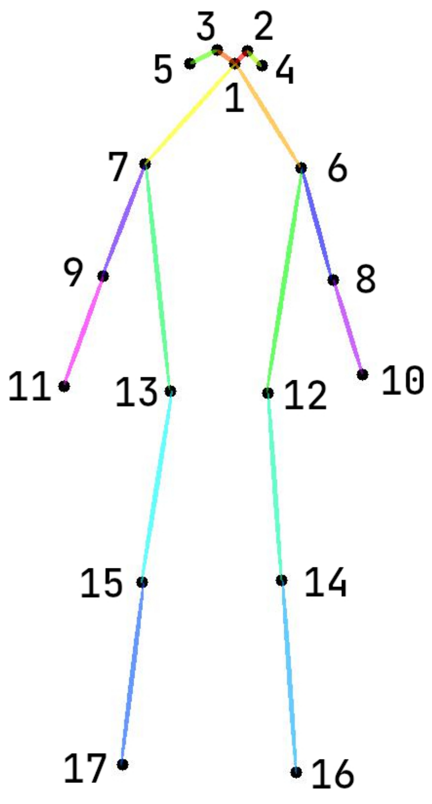
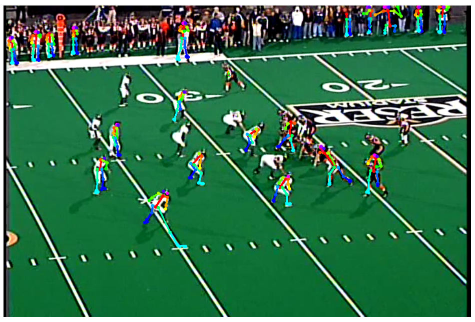
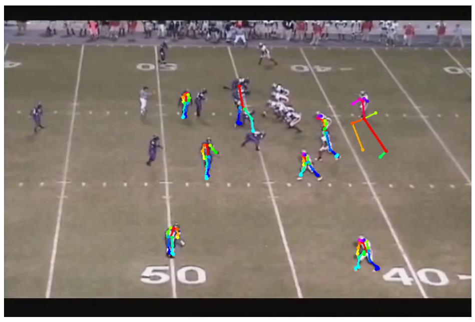


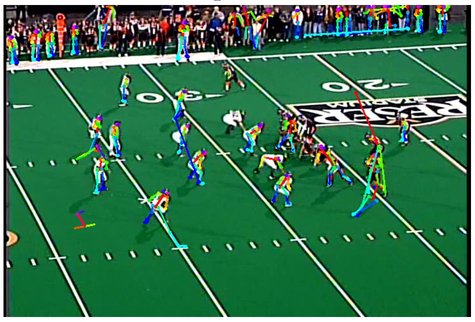
Figure 3.6: The 17 pose key points.

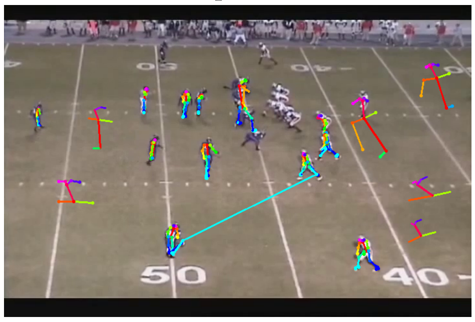
3.3.4 Comparison

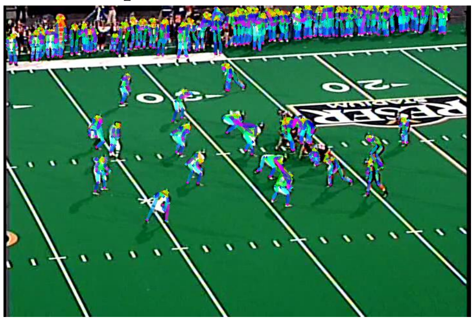
As described in Subsection 3.3.2, OpenPose had problems with detecting the joints of all 22 players consistently. These problems were primarily due to low video resolution. Even by setting a flag that maximizes true positive detections, the system either still detected only a few players or had an unbearable number of false positives. In contrast, the PyTorch model predicted poses well and even was faster than OpenPose. The following figures serve as a sample to depict results from OpenPose (with and without maximized true positives) and from PyTorch:


(a) OpenPose.


(b) OpenPose.


(c) OpenPose (maximize TP).


(d) OpenPose (maximize TP).


(e) PyTorch.

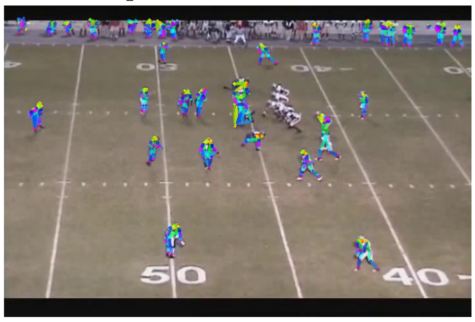

(f) PyTorch.

Figure 3.7: Pose estimation on sample frames.

For the classification in the last step of the pipeline, defensive players are particularly important (in all pictures the players to the left with the same jersey color). Furthermore, offensive pass receivers are relevant. In all pictures, those are the players to the right that are not the offensive line and not the quarterback. As Figure 3.7 indicates, OpenPose has problems detecting all relevant players' joints (top row). Enabling the flag for maximized true positives resulted in the pictures in the middle row. Poses for more players are detected, but with a drastic increase in false positives. The PyTorch network, in contrast, consistently detects all (or most of) the relevant players.

3.4 Playing Field Registration

As mentioned earlier, my approach registers the raw video into a top-down view. Figure 3.8 illustrates an example of that.

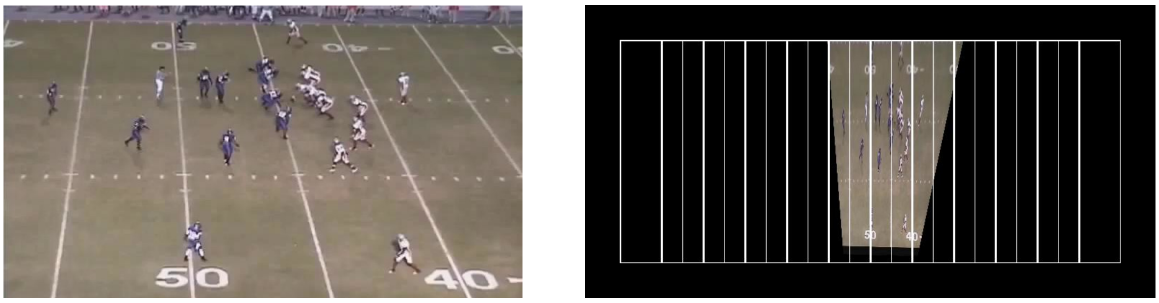

(a) Sample frame. (b) Projected sample frame.

Figure 3.8: A sample frame (left) projected into a field model (right).

As Subsection 2.1.3 indicates, this is a rather common pre-processing technique. In order to compute the projection of a frame, I used two OpenCV [94] functions. The first function (`getPerspectiveTransform`) computes the projective matrix used by the second function (`warpPerspective`) to project the image. To calculate the projective matrix, `getPerspectiveTransform` requires four source points on the image frame and the corresponding destination points on the projection plane. Thus, to create a projection for each frame of a video, the following steps are necessary:

1. Select a video
2. Set the four source points for the projective transformation.
3. Track the points (otherwise, they would have to be set for each frame manually).
4. Enter the location of and the distance between the points on the field.
5. Calculate the projective matrix with the points and parameters set.

6. Show the projection to validate correctness.
7. Reset the source points/set new points and parameters whenever the point tracker diverged too far.
8. Store the projective matrix for later use on the key points.

At first, I did this process manually, but this was tedious and time-consuming. Therefore, I implemented a dashboard to facilitate this process. The dashboard was implemented in Plotly Dash, an open-source Python framework to develop dashboards in pure Python [95]. I selected Plotly Dash over other Python GUI frameworks due to prior experience shortening implementation time. An image of the entire dashboard is depicted in Figure 3.15, whereas the following subsections discuss and illustrate the implementation of each individual part.

3.4.1 Selecting a Video

To do the first step – i.e. select a video – I used a dropdown menu. To be visible in the dropdown, a video must be placed in a specific folder. On the selection of a video, the video frames are loaded into the dashboard, as illustrated in Figure 3.9

3.4.2 Setting the Four Source Points

The second step is selecting the four source points of the projection. This was implemented with a button, which opens a window. In this window, the four points can be set by clicking on the corresponding point on the image. To implement this, I used the Python API of OpenCV to record the pixels where the four points should be. Here, the ordering and alignment of the points are vital. The points must be a rectangle when projected. To facilitate the calculation of the destination points, they must be in the following order: top-left, top-right, bottom-left, bottom-right. When the four points are set, and the enter key is pressed, the four points are stored.

3.4.3 Tracking the Source Point through the Video

Repeating the second step for each image frame is impractical. Therefore, I applied the OpenCV point tracker function `calcOpticalFlowPyrLK`. This function uses the iterative Lucas-Kanade method with pyramids [96]. To track points from one to the following image, `calcOpticalFlowPyrLK` needs the previous image, the current image, and the points from the previous image that should be located in the current image. Furthermore, I experimented with the parameters `winSize`, `maxLevel`, and `criteria` to improve tracking results. `winSize` is the search window size at each pyramid level, and I found the best results with a 31-by-31 window. `maxLevel` specifies the depth of the pyramid, where the default – zero – means no pyramids are used. One means using a

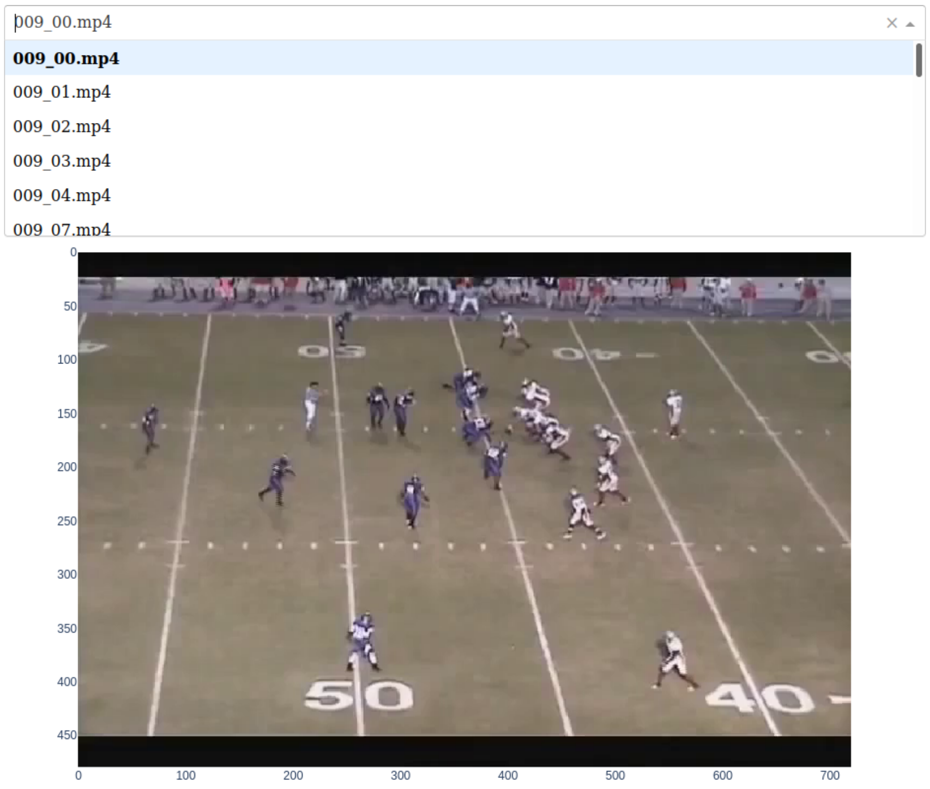


Figure 3.9: Dropdown for video selection (top) and loaded video frames (underneath).

pyramid of two levels, and so on. I achieved the best results with a `maxLevel` of 400. Finally, `criteria` is a three-tuple. First is the termination criteria, where I used the integer number three to incorporate both the maximum iteration count and the epsilon criteria. The second and third are maximum iteration count and epsilon, respectively. The former is self-explanatory, and I found the best results with 10000. The latter is a parameter for adjusting accuracy. Here, the best results were obtained with 5000.

3.4.4 Specifying the Parameters Required for the Projection

In order to calculate the projective matrix, the destination points of the four selected and tracked source points are needed. To obtain those, I used four integer number input fields. The first input is for the yard distance of the top-left point to the left goal line. The second input is for the foot distance of the top-left point to the upper sideline. The third input is for the longitudinal foot distance between the left two and the right two points. Finally, the fourth input field is for the lateral distance between the upper two and the lower two points in feet. With this, the destination rectangle is fully defined. Note that I intentionally used yards for the longitudinal position on the field and foot for all others. This was the

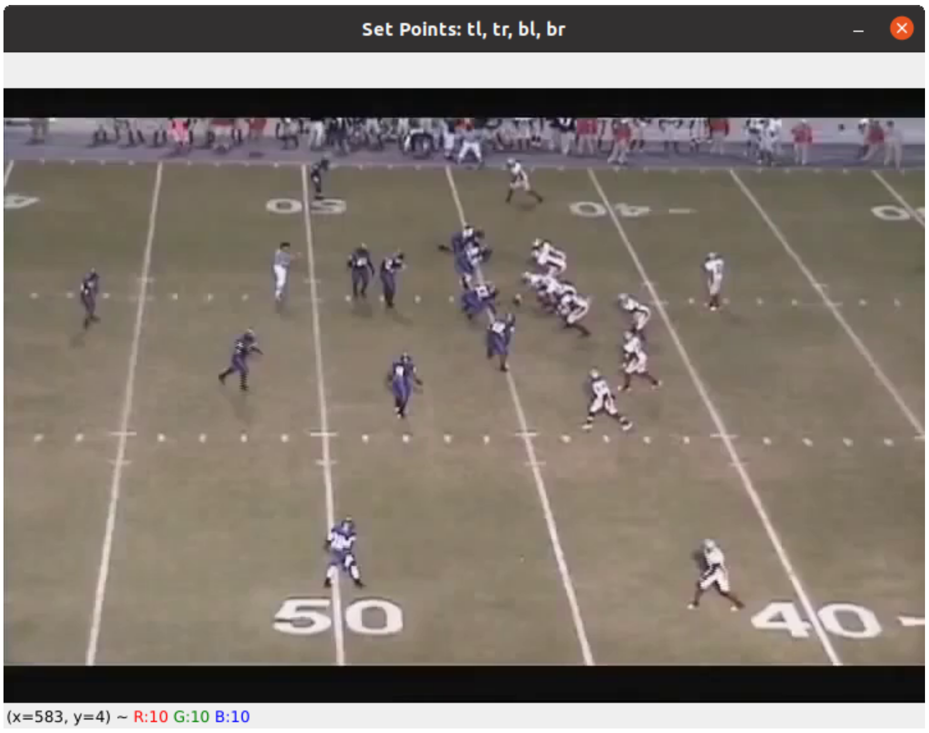


Figure 3.10: OpenCV window to select the four source points.

choice because there are yard lines for the longitudinal position on the field. In contrast, hash marks, which I often used for the lower two points, are distanced an integer number away from the sidelines in feet, but a floating-point number away in yards.

3.4.5 Calculation of the Projective Matrix

With the points set and tracked, and parameters specified in the previous three steps, the OpenCV function `getPerspectiveTransform` has all required inputs to compute the projective matrices. Therefore, the projective matrix for each frame can now be calculated via `getPerspectiveTransform`, and the other OpenCV function `warpPerspective` can calculate the projections.

3.4.6 Resetting of Points

As mentioned in the previous step, projections can be calculated for each frame of a video. However, the point tracker is not perfect. Thus, points can, at times, diverge from where they were set due to blurring, fast camera movement, zooming, occlusion by a player, or the point leaving the image frame. An example of those is depicted in the following figures:

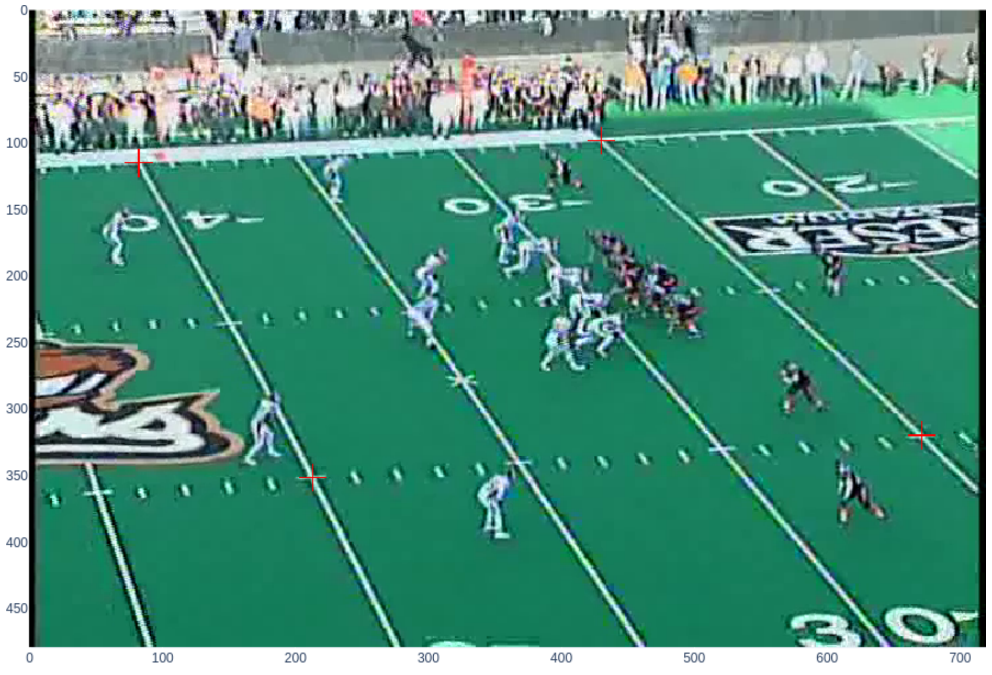


Figure 3.11: Initally set points (red crosses).

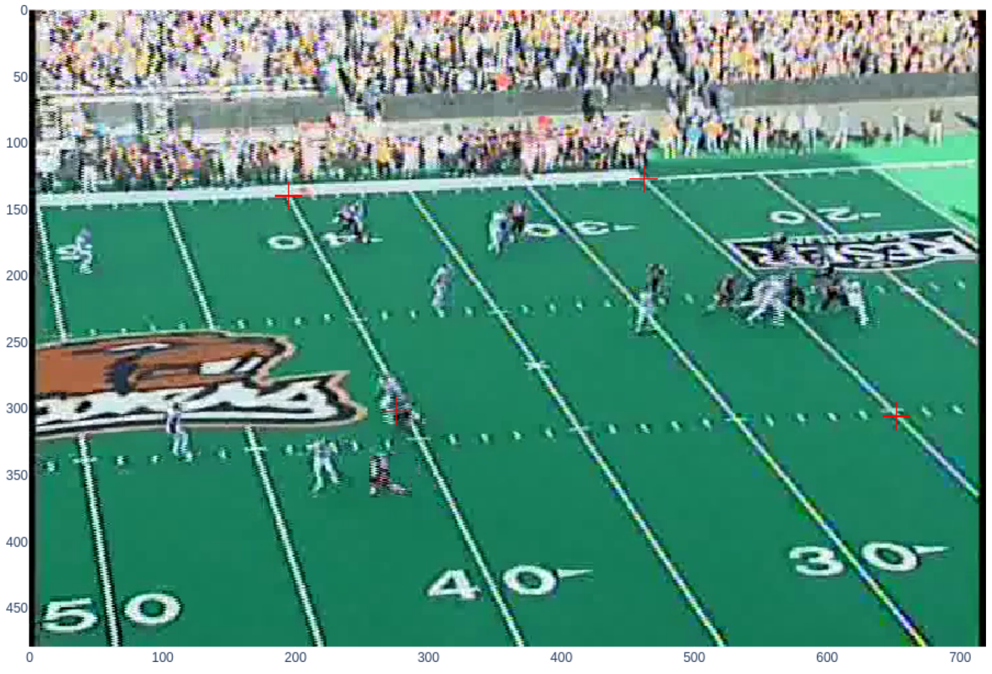


Figure 3.12: Diverging due to occlusion (bottom-left) and motion/zoom (bottom-right).

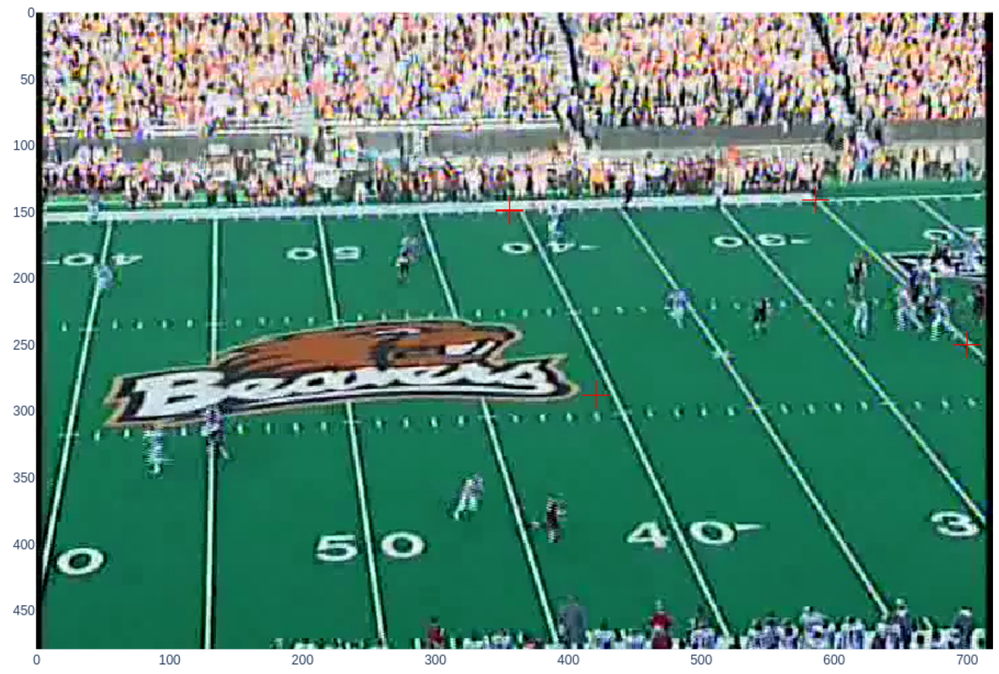


Figure 3.13: Tracked point left the image frame (bottom-right).

Hence, a mechanism for resetting such points was needed. Fortunately, the implementation was straightforward. It included used a slider and two buttons to navigate to the precise frame, where point(s) started to diverge. The two buttons were added to move a single frame forward or backward since moving to just the next frame can be challenging with the slider for longer videos. When I arrived at the desired frame, I used the same mechanism for resetting as I used for the initial setting of the points. Then, the newly set points were tracked through the remainder of the video. To make this step more challenging, there were some cases where I had to select new points instead of resetting the previous ones. This could happen, for example, when a landmark is completely occluded for a few frames, too blurry, or moves out of the image frame, like in Figure 3.13. With new points, however, the location of and distances between the points can change and also have to be reset. Here, I also reused the mechanism I used for the initial setting of those parameters and overwrote the projection parameters for the remaining image frames.

3.4.7 Projection Preview

Furthermore, I needed a possibility to check the projection results. For this, I used another button, which shows the video projected into the field model. To aid validation, I added expected field marks:

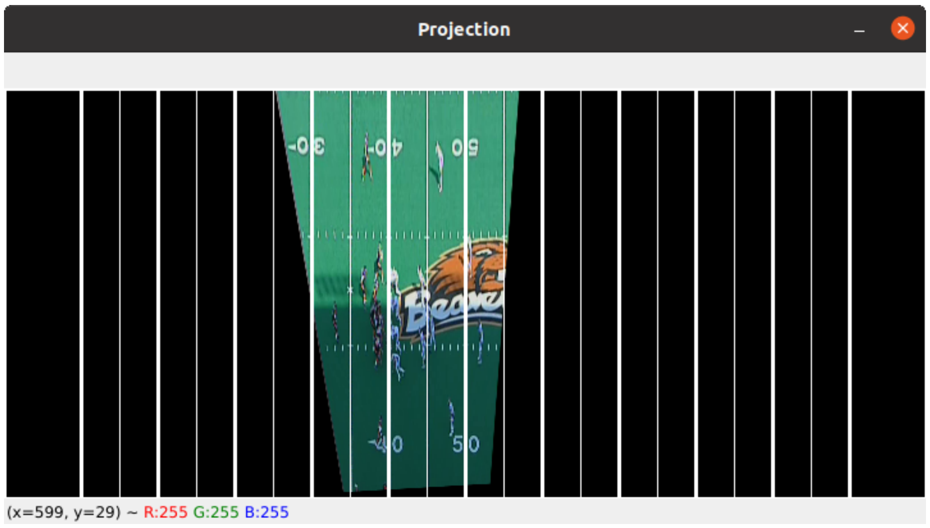


Figure 3.14: Projection preview.

3.4.8 Storing Projective Matrices

Finally, the projective matrices had to be stored, so that I can use them later to project the key points of the pose estimation. This was implemented with NumPy [97], as the projective matrices are NumPy arrays and could be stored easily via the NumPy API.

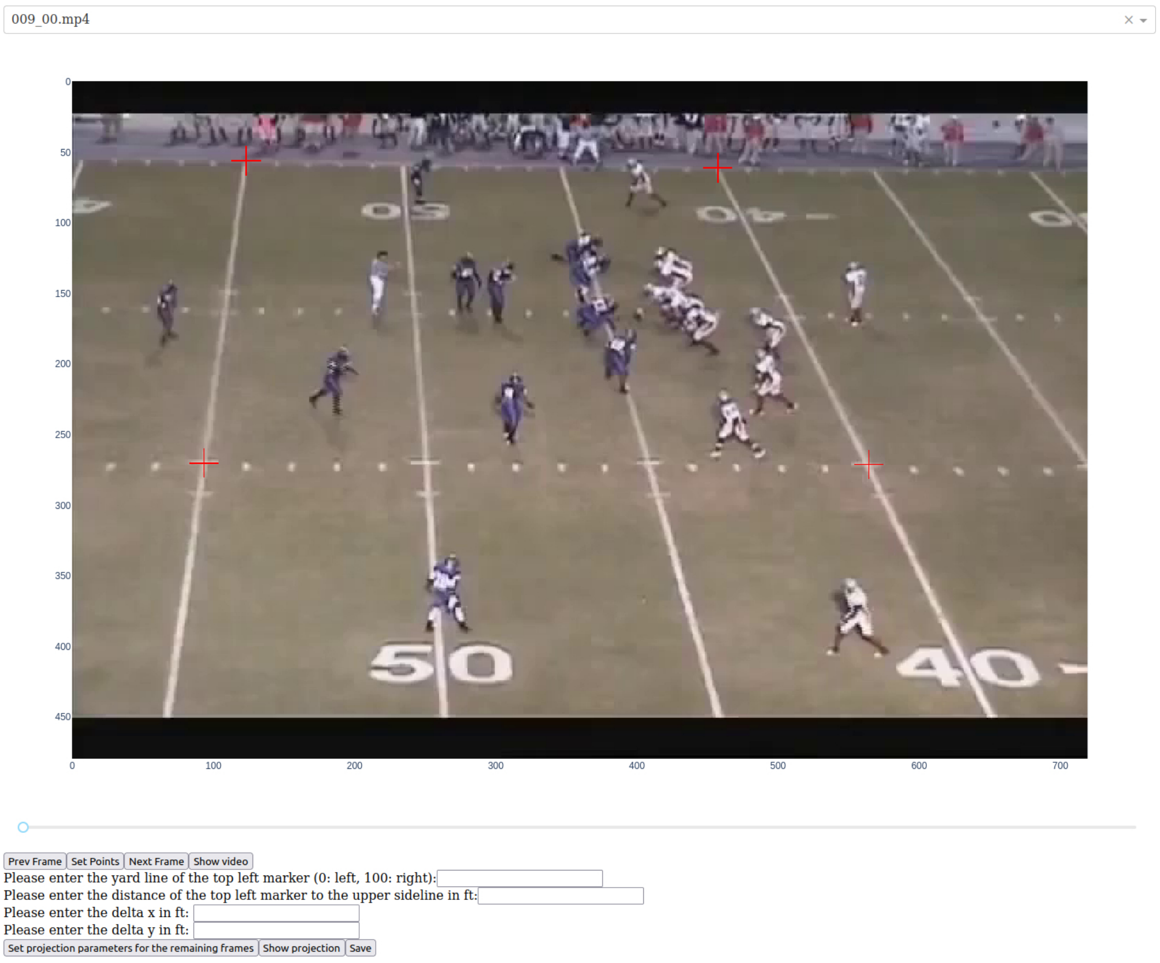
3.4.9 Summary

Overall, this dashboard drastically facilitated and accelerated the calculation of the projective matrices and projections. Furthermore, it provides a platform for the next step – labeling the data. Figure 3.15 illustrates the entire dashboard.

3.5 Labelling

The next step was annotating the videos with labels. For this, I extended the dashboard mentioned above. Thus, two text fields were added – one for man vs. zone coverage and one for the four-class classification. The text input fields of the dashboard are illustrated in Figure 3.16 In order to discuss the labeling process appropriately, the following three

subsubsections contain explanations of defensive personnel, man coverage, and zone coverage with three common variations of the latter two. These coverage classes are then used to label the data appropriately into man vs. zone and the four classes for the classification in the final pipeline step.

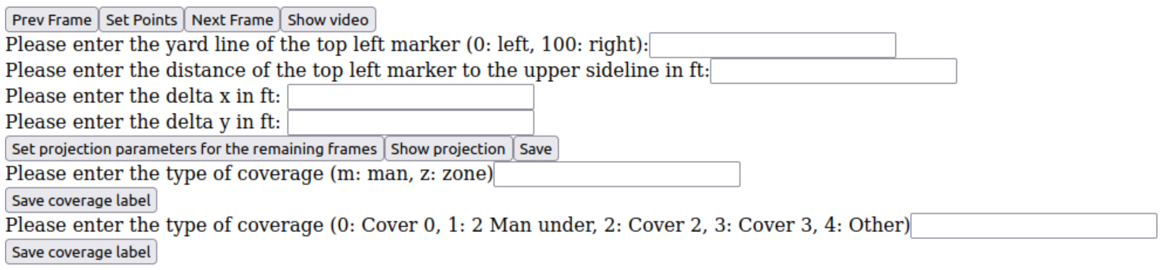


Please enter the yard line of the top left marker (0: left, 100: right):
Please enter the distance of the top left marker to the upper sideline in ft:
Please enter the delta x in ft:
Please enter the delta y in ft:

Figure 3.15: The entire dashboard.

3.5.1 Defensive Personnel

To understand the diagrams in the following depicting man and zone coverage variations, it is crucial to know the defensive positions and their abbreviations. Generally, the defensive personnel is split up into three groups. Firstly, there is the defensive line (DL) consisting of defensive tackle(s) (DTs) and defensive ends (DEs). The defensive line is lined up opposite to the offensive line, close to the line of scrimmage, and close to the ball. The DEs are the two players aligned on the outside – the so-called "edge" – of the defensive line. Commonly, between the DEs play one or two DTs. The two primary responsibilities of the defensive line are (1) rushing to and tackling the passer and (2) providing the first line of defense against run plays. In the following diagrams, there are always two DTs and two DEs, since those illustrate a 4-3 defense, where the four stands for four defensive linemen.



Please enter the yard line of the top left marker (0: left, 100: right):
Please enter the distance of the top left marker to the upper sideline in ft:
Please enter the delta x in ft:
Please enter the delta y in ft:

Please enter the type of coverage (m: man, z: zone)

Please enter the type of coverage (0: Cover 0, 1: 2 Man under, 2: Cover 2, 3: Cover 3, 4: Other)

Figure 3.16: Text input with labelling extension.

The three in 4-3 stands for three linebackers (LBs), which leads to the second positional group. Linebackers are aligned behind or on the side of the defensive line providing a second line of defense and literally "back the line." They can rush the passer, drop into a zone or play in man coverage. Finally and the third group are defensive backs (DBs), which are further split up into cornerbacks/corners (CBs) and safeties – usually strong safety (SS) and free safety (FS). DBs' primary responsibility is pass coverage. Cornerbacks align outside and can drop into zones or play man coverage. Safeties play the same coverages but are aligned more towards the center. The FS most commonly aligns rather deep, whereas the SS can also play closer to the line of scrimmage and help with defending run plays. The abbreviations will be used below for coverage illustrations. Moreover, the offensive players are either boxes or circles. Boxes indicate offensive linemen, which are not allowed to receive a pass. Circles are either the quarterback or offensive players eligible to receive a pass. For all coverage illustrations, defensive linemen rush the passer. LBs and DBs either cover a man or zone or rush the QB. A dashed line indicates man coverage. Zone coverage is indicated by an arrow into a blue (deep) or yellow (underneath) box. An arrow towards the QB indicates a rush to the QB.

3.5.2 Man Coverage

The most basic way to distinguish defensive coverage in American football is man coverage vs. zone coverage. As the name implies, defenders get a man assigned in man coverage. Usually, the quarterback is left unassigned since the pass rush is responsible for him. Offensive linemen are left unassigned as well since they are not allowed to run a route and receive a pass. Hence, only five players remain. Those are covered by a defender each. The remaining six defenders do one of the following things: Rush the passer, drop into a zone, or help a fellow defender cover a man. As this allows for man combinations, the following subsubsections discuss the most common man coverage variations.

Cover 0

"Cover 0" or "Zero Blitz" means that there are zero deep zones. All five offensive receivers are covered by a defender each, and the other six defenders rush the passer. The central goal of this coverage is pressuring the quarterback. It is, however, susceptible to allowing big plays for the opposing offense since there is no player deep acting as a safety net [98].

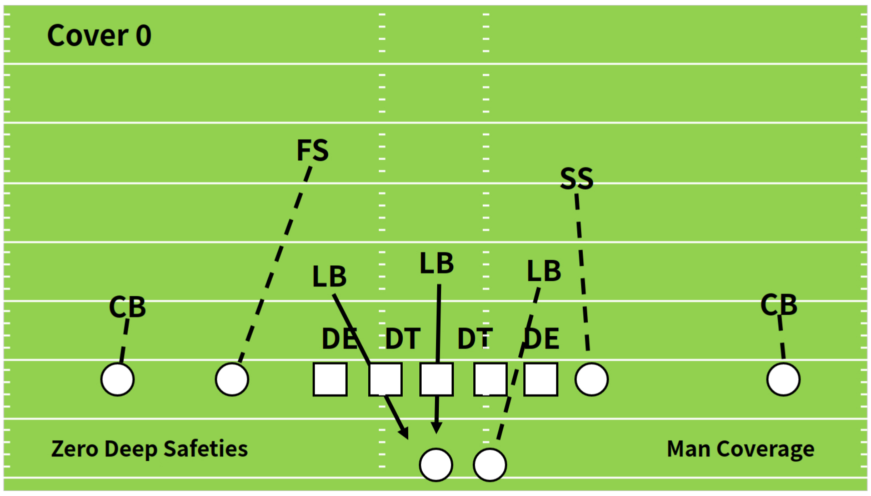


Figure 3.17: Sample Cover 0.

Cover 1

In this defensive coverage, one player plays in a deep central zone. Again, the five offensive players eligible to receive a pass are man-covered. Commonly, four players rush the passer, and the final man either covers a zone underneath the deep middle zone (just "Cover 1"), double-teams a receiver, or also rushes the passer ("Cover 1 Blitz") [99].

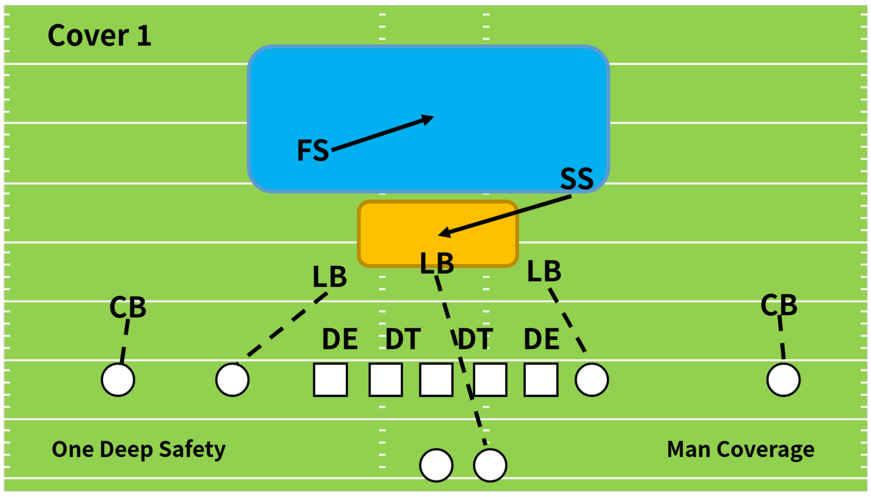


Figure 3.18: Sample Cover 1.

Cover 2 Man

Similar to "Cover 1", the "2" in "Cover 2 Man" indicates that two players play in a deep zone. In particular, each one is responsible for one of the deep halves. Again, there is a four-man pass rush. The remaining five players cover the five corresponding attackers in man coverage [100].

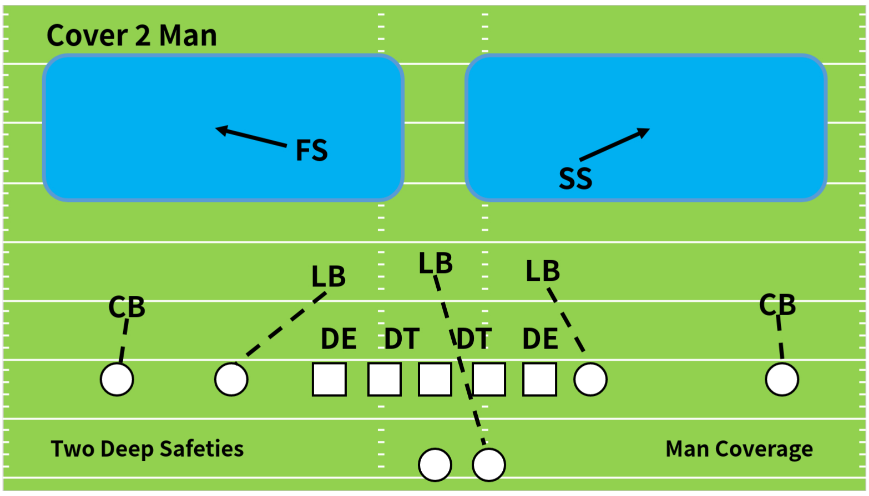


Figure 3.19: Sample Cover 2 Man.

3.5.3 Zone Coverage

For zone coverage, in contrast, every defender is assigned to a zone for which he is responsible. Similar to man coverage, there are many variations, and the following subsubsections discuss the most common ones. Again, a four-man pass rush is most common. However, it is also possible to sacrifice a zone or two for more pressure on the opposing quarterback.

Cover 2

As for man coverage, the number after "Cover" indicated the number of deep zones. Therefore, "Cover 2" has the two safeties playing one deep half each. The two corners play in so-called flat zones on the outside. In between are the three LBs playing in three short zones in the middle and underneath the safeties [101].

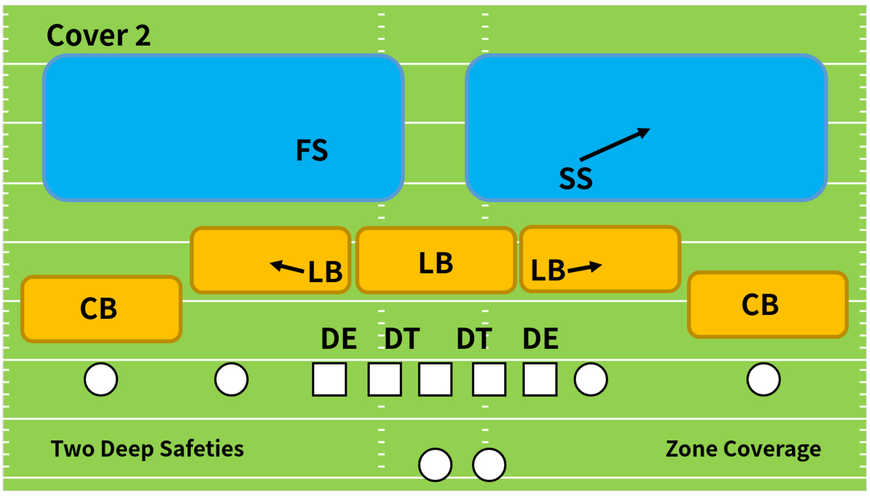


Figure 3.20: Sample Cover 2.

Cover 3

In this coverage scheme, three players play in deep thirds. Typically, these are the FS and the two CBs, as shown in Figure 3.21. A LB and the SS play in the flat zones, and the remaining two LBs cover the middle [102].

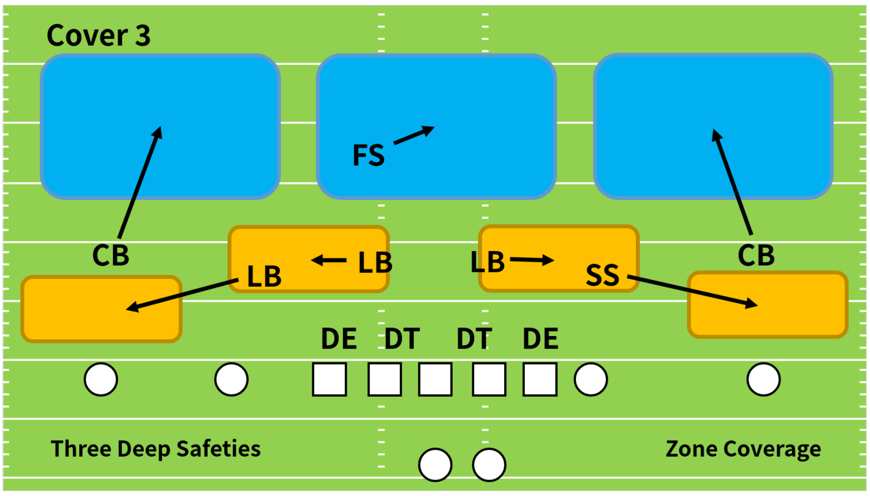


Figure 3.21: Sample Cover 3.

Cover 4

Finally, there is “Cover 4”, which means that the deeper part of the coverage is split into quarters. Thus, this coverage is sometimes called “Quarters.” Those four zones are covered by the FS, the SS, and the two CBs. Underneath, the LBs play in thirds [103].

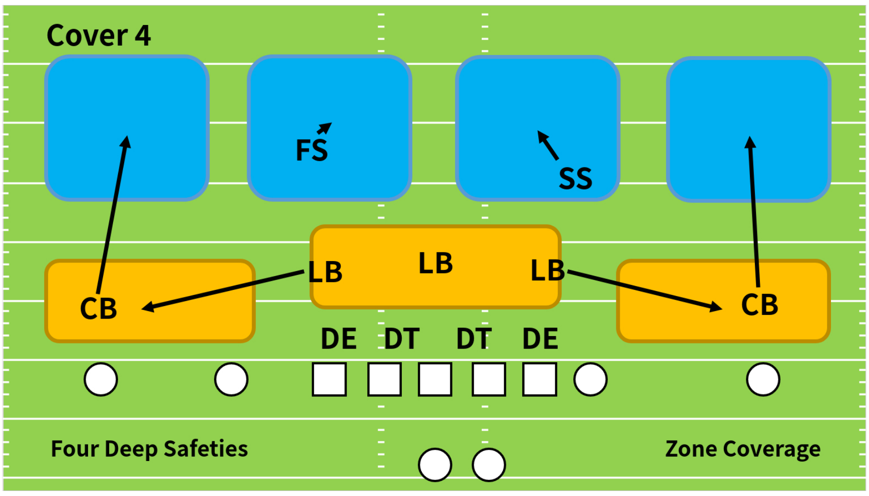


Figure 3.22: Sample Cover 4.

3.5.4 Labelling Man vs. Zone

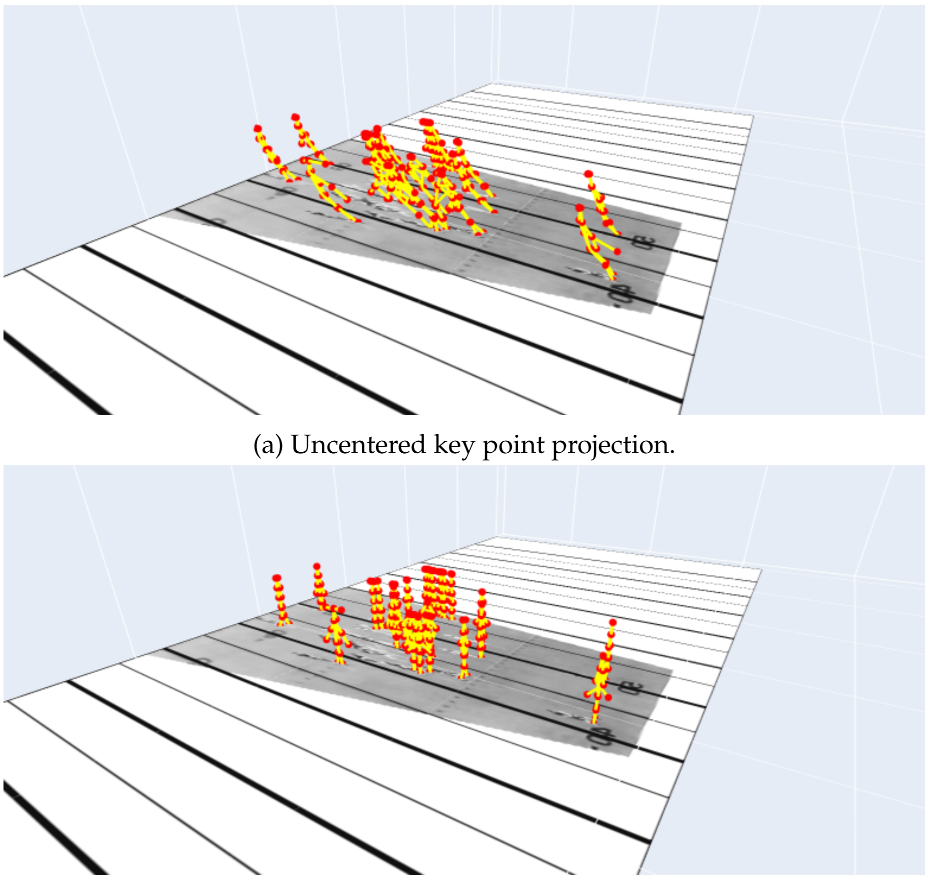
Distinguishing man and zone coverage is straightforward. While watching a video of a play, one only needs to monitor whether defenders follow an assigned man. This indicates man coverage. For zone coverage, defenders do not follow a man but rather drop back into a zone. They also tend to keep their eyes on the quarterback or route combinations in front of them, whereas in man coverage, defenders solely focus on their assigned man.

3.5.5 Four Class Labelling

Previously, I discussed six coverage classes: Cover 0, Cover 1, Cover 2 Man, Cover 2, Cover 3, and Cover 4. Those are the six classes initially intended to classify. However, Cover 1 and Cover 4 were drastically underrepresented in my dataset (just one video each). Thus, I excluded them. Still, knowing these six basic coverage variations as outlined above facilitated labeling man vs. zone. The three man coverage variations can be distinguished by looking at deep zones (zero vs. one vs. two). For zone coverages, the same applies (two vs. three vs. four deep zones). However, two man-zone pairs require further investigation. Cover 1 and Cover 3 can appear quite similar because there is one player in the deep middle of the field for both. Thus, the remaining players have to be observed closely on whether they stay inside a zone or follow an assigned man. Even harder to distinguish can be Cover 2 and Cover 2 Man. Again, due to the same amount of deep safeties, but as before, one has to examine other players' behavior.

3.6 3D Player Extraction

As mentioned in Subsection 3.4, I reused the projective matrix to project the key points of the players, as well. In order to obtain a 3D model, I stacked the key points in six layers (from bottom-up): (1) ankles, (2) knees, (3) wrists and hips, (4) elbows, (5) shoulders, and (6) nose, eyes, and ears. However, since upper key points are also higher in the image, they are projected father back, and players do not stand upright (as in Figure 3.23a). Thus, I centered each pair of left and right key points above the center of the left and right foot (as in Figure 3.23b):


(a) Uncentered key point projection.

(b) Centered key point projection.

Figure 3.23: Uncentered and centered projection and layering of keypoints.

To carry out tactical analysis in the next step, the players' joints were put into 3D tensors. As mentioned in the introduction, an American football field is 120 yards long and 53 ⅓ yards wide. In feet, this translates to 360 by 160 feet. Thus, by rounding the x-, y-, and z-coordinates to integer values, the 3D model of a single image frame can be represented by a 360×160×6 tensor. To represent densities of key points, I added one for each key point that fell into a voxel. However, with a projected field size of 360×160, many key points were in the same voxels. Therefore, the field resolution was doubled to adjust for that impreciseness while staying at a size where models still trained in a reasonable amount of time. Hence, one sample has 720×320×6 voxels. By default, NumPy uses 64-bit integers, which results in $720 \cdot 320 \cdot 6 \cdot 64 = 88473600$ bits or about 11MB *per sample*. With about 20000 overall frames, the largest augmented dataset would require 220GB. This is inefficient. For better memory efficiency, tensors were adjusted to contain 8-bit unsigned integers, thus only requiring an eight of the initial 220GB. Secondly, I switched from dense to sparse tensors decreasing memory requirement even further. Overall, this is a compact representation of the 3D data, ready for classification in the next step.

3.7 Tactical Analysis

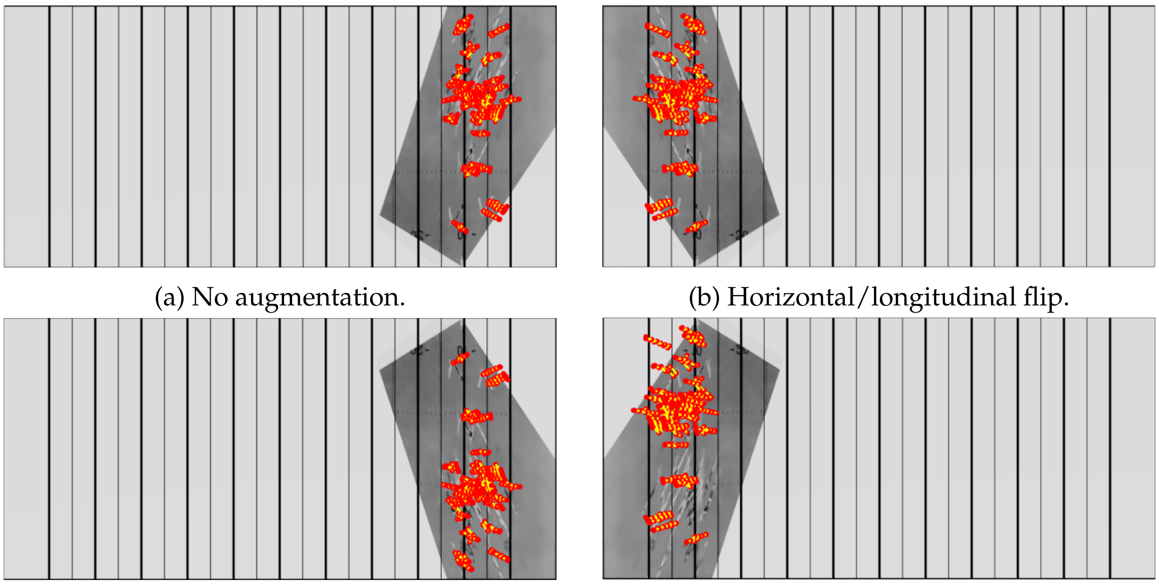
The data format described in the previous section was chosen to have a fixed-size input for classification. Tensors of size 720×320×6 are quite similar to image data in the sense that they can be interpreted as $width \times height \times channels$. Thus, previous research on and experience with convolutional neural networks and residual networks were reapplied for classification.

3.7.1 Dataset

As mentioned previously, the dataset was built from 20 videos from OSU and 40 videos cut from a video from the Sports-1M Dataset. Each frame of each video was used as a single data point totaling 7895 samples. For validation and testing, one full video per label was kept out of training (for each validation and testing) to validate and test on completely unseen data during and after training. Keeping out random frames could result in a meaningless classification accuracy because the network would train on frames closely before and after excluded frames.

3.7.2 Technical Setup

In order to train and evaluate the models, two resources were used: a local desktop computer and Google Colab [90]. The former was used to train both baselines and CNNs, while Google Colab functioned as a second resource to experiment with hyperparameters for the CNNs. Baseline models were trained on a six-core AMD Ryzen 5 3600 with 32GB of RAM, whereas CNNs were trained with the highly optimized GPU version of TensorFlow. The desktop was set up with an Nvidia GeForce RTX 3060 with 12GB of VRAM. Google Colab instances were set up with Nvidia Tesla P100 GPUs with 16GB of VRAM. Thus, both baseline and complex models could be trained efficiently.



(a) No augmentation.

(b) Horizontal/longitudinal flip.

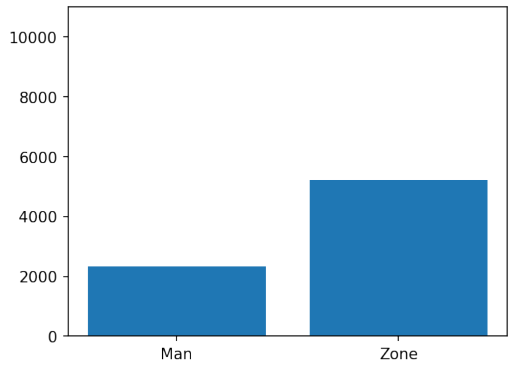
(c) Vertical/lateral flip.

(d) Horizontal and vertical flip.

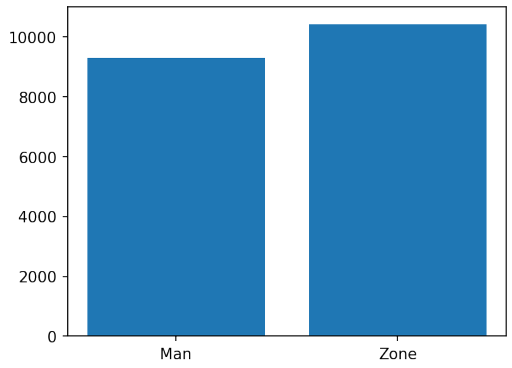
Figure 3.24: 3D data augmentation.

3.7.3 Man vs. Zone Coverage

For the first tactical analysis, I trained models to distinguish between man and zone coverage. As data was limited, it was augmented. Similar to image data augmentation with flipping, 3D point cloud tensors were flipped horizontally – i.e. longitudinal to the field –, vertically – i.e. lateral – and both horizontally and vertically when looking top down. To make this more expressive, see Figure 3.24. For American football, such transformations are sensible. Offensive teams commonly use a lateral flip of their plays, and defenses react to it by flipping coverage laterally. Longitudinal flips are what would happen if the offense played in the opposite direction with a lateral flip. Since a lateral flip is common, and directions are switched each quarter, this is nothing unrealistic either. Finally, a longitudinal and lateral flip just represents the same play in the opposite direction. As described in Subsubsection 3.7.1, two videos per label were kept out of training for validation and testing. Thus, the videos of the train set consisted of 7535 frames (95.44%). The validation and test set videos consisted of 175 frames (2.22%) and 185 frames (2.34%), respectively. Of the 7535 image frames in the training set, 2327 (30.88%) had the label man, while the remaining 5208 (69.12%) were labeled with zone (Figure 3.25a). Hence, labels were imbalanced. Therefore, the minority class – man – was augmented fourfold, while the majority class – zone – was augmented just twofold. This resulted in a larger, much more balanced 9308 (47.18%) man and 10416 (52.81%) zone labeled frames (Figure 3.25b). Both the validation and the test set were already quite balanced. The validation set consists of 96 (54.86%) man and 79 (46.14%) zone samples. There were 98 (52.97%) man and 87 (47.03%) zone data points in the test set. Since the applied augmentations represent meaningful American football plays, both the validation and the test set were augmented, as well, to increase their size and variance while not sacrificing more training samples. This resulted in 700 samples for the validation set and 740 samples for the test set.



(a) Label numbers before augmentation.



(b) Label numbers after augmentation.

Figure 3.25: Label numbers before and after data augmentation for the train set.

Models

For classification into man and zone, I trained several models. Firstly, a linear model was trained as a baseline. Then, two convolutional neural networks were trained with increasing complexity and sophisticated architectures: (1) a CNN with batch normalization, ReLU, and max-pooling and (2) a custom ResNet. If not mentioned otherwise, convolutions always use 3×3 filters with 1×1 stride, a padding of one, and weights are initialized as in [104]. Max-pooling is computed with a 2×2 filter and stride, and batch size was 32.

I also carried out experiments with residual blocks with full pre-activation as in [105]. However, no significant improvements over the standard residual blocks were obtained.

Baseline As a baseline, an `SGDClassifier` from scikit-learn [106] with standard configuration (linear SVM) and a regularization multiplier (alpha) of 0.01 was trained. The data was pre-processed with a `StandardScaler` before training and classification.

The classifier achieved a training, validation, and test accuracy of 99.99%, 60.00%, and 69.73%, respectively. Table 3.1 depicts the confusion matrices for the train (left), validation (middle), and test (right) sets. Precision and recall are concisely represented in Table 3.2 for each of the sets.

Only one zone coverage sample was mispredicted as man for the train set, while all other samples were predicted correctly. For both the validation and the test set, a clear tendency can be obtained. Recall for zone coverage is comparatively good with 76.27% (validation) and 79.89% (test). However, precision is not as good with 54.04% and 64.35%, respectively.

	Man	Zone
Man	9308	0
Zone	1	10415

(a) Train set.

	Man	Zone
Man	179	205
Zone	75	241

(b) Validation set.

	Man	Zone
Man	225	167
Zone	78	270

(c) Test set.

Table 3.1: Confusion matrices for the baseline classifier.

	Precision	Recall
Man	99.99%	100%
Zone	100%	99.99%

(a) Train set.

	Precision	Recall
Man	70.47%	46.41%
Zone	54.04%	76.27%

(b) Validation set.

	Precision	Recall
Man	77.27%	60.71%
Zone	64.35%	79.89%

(c) Test set.

Table 3.2: Precision and recall for the baseline classifier.

For man coverage, the recall was at 46.13% and 57.38%. Precision, in turn, was 70.47% and an even higher 74.26% on the validation and test set, respectively. Thus, the classifier detects zone coverage (comparatively) well but at the cost of a high number of false positives. In contrast, the detection rate of man plays is rather low, while the predictions for the label man are rather precise. The discrepancy between train and validation/test accuracy is an indicator for overfitting. However, increasing the regularization multiplier alpha did not improve the model's generalization on the validation set. Thus, the model was evaluated on the test set with the best alpha value.

CNN As the first more advanced model a CNN was implemented in TensorFlow. Following [58], BN was applied before the activation function directly after convolution. After a convolution block with BN and activation, max-pooling was added, thus sampling down dimensions. After the last convolutional block, global average pooling was used to flatten feature maps into single values. The remaining neurons were connected to a single neuron with a Sigmoid activation function for binary classification. For optimization, the Adam optimizer was used with a learning rate of 0.0003. MAE, MSE, and BCE were tested as loss functions, with MSE yielding the best results.

The final, best-performing model consists of six layers of convolution, batch normalization, ReLU, and max-pooling. Figure 3.26 illustrates the resulting architecture, which contains 1373 parameters – of which 1317 are trainable. The model was trained for 50 epochs. Figure 3.27 illustrates training and validation accuracies (left) and losses (right) over epochs. As the left plot indicates, validation accuracy starts to decrease slightly towards the final epochs. Thus, training for more epochs yields worse results. Moreover, training accuracy flattens around 75%. However, increasing parameters – both via more depth or channels per convolutional layer – resulted in overfitting. Furthermore, validation accuracy surpasses training accuracy for most epochs. This is expected because the regularization via dropout decreases accuracy during training.

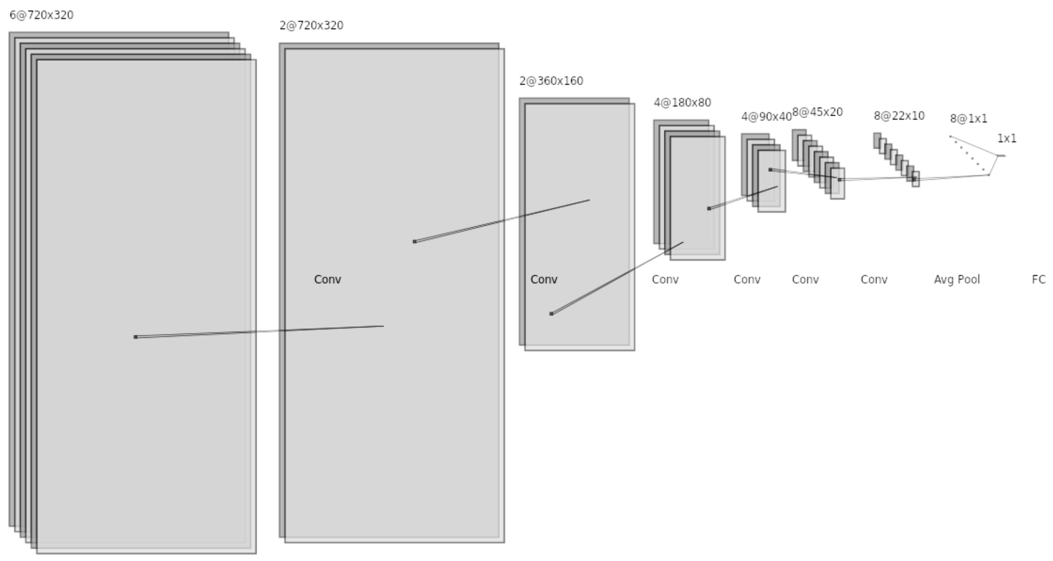


Figure 3.26: CNN architecture (created with [107]).

After the 50 epochs of training, the CNN achieved a final training accuracy of 79.26%. Validation and test accuracies were 85.14% and 74.05%, respectively. Hence, the model overfitted slightly on the validation set. Compared to the baseline, the CNN performed 19.74%-points worse on the training data while surpassing validation accuracy by 25.14%-points and test accuracy by 4.32%-points. In order to compare the results of the CNN to the baseline further, Table 3.3 depicts the confusion matrices. Additionally, precision and recall are illustrated in Table 3.4.

	Man	Zone
Man	6555	2753
Zone	1337	9079

(a) Train set.

	Man	Zone
Man	297	87
Zone	17	299

(b) Validation set.

	Man	Zone
Man	240	152
Zone	40	308

(c) Test set.

Table 3.3: Confusion matrices for the CNN.

	Precision	Recall
Man	83.06%	70.42%
Zone	76.73%	87.16%

(a) Train set.

	Precision	Recall
Man	94.59%	77.34%
Zone	77.46%	94.62%

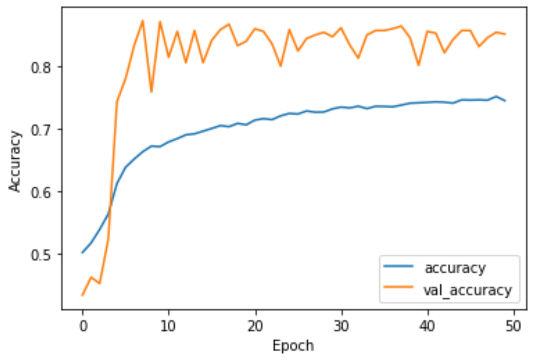
(b) Validation set.

	Precision	Recall
Man	85.71%	61.22%
Zone	66.96%	88.51%

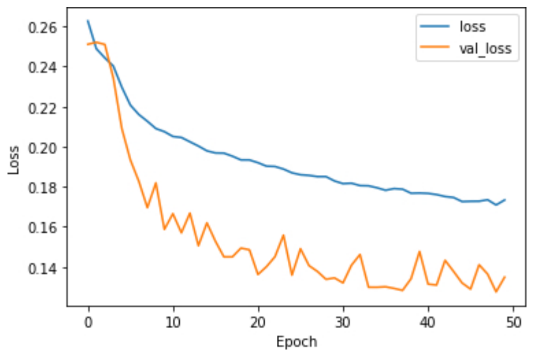
(c) Test set.

Table 3.4: Precision and recall for the CNN.

Regarding precision and recall, the CNN outperformed the baseline (for both man and



(a) Training and validation accuracy.



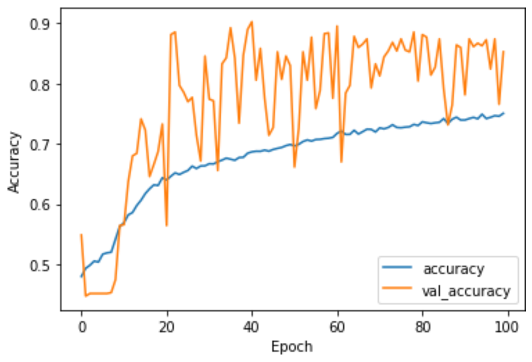
(b) Training and validation loss.

Figure 3.27: Training and validation accuracy and loss over epochs for the CNN.

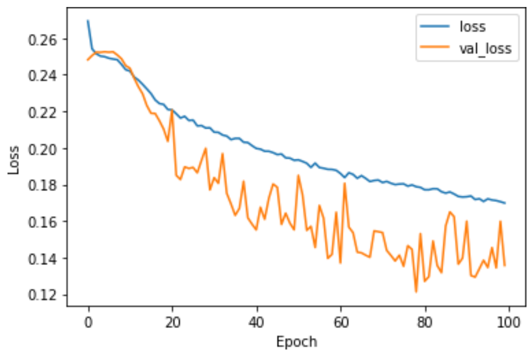
zone) for validation and testing. Similar to the baseline, the CNN detects zone well (higher recall) while being more precise for man coverage (higher precision). Overall, the CNN is superior to the baseline.

ResNet For the second, advanced model, I implemented a custom ResNet. As for the CNN, the ResNet was implemented in TensorFlow. I implemented the standard residual block with and without downsampling (via a stride of two) as described in [71] (or Section 2.2.8 in this document). The network was constructed by stacking residual blocks. The first residual block has two channels and applies downsampling. Then, follow three more residual blocks with two channels before the next downsampling block. This subsequent downsampling block exchanges feature map size for channel depth, increasing the channel number to four. Again, this is followed up by three more standard residual blocks with the same channel number. The last downsampling block halves feature map size again while increasing channel number to eight. After the single following, eight-channel residual blocks follows global average pooling to create eight neurons from the eight final feature maps. Those eight neurons are connected with a 50% dropout to a single neuron with a Sigmoid activation function for binary classification. With a total of ten residual blocks and each residual block containing two convolutions, this network has 20 convolutional layers.

The model was optimized on the MSE by the Adam optimizer with a learning rate of 0.0001. With those settings, the model contains 3987 total parameters and 3799 trainable parameters trained for 100 epochs. Corresponding accuracy and loss plots are illustrated in Figure 3.28. The learning curves exhibit similar behavior to those of the CNN: validation accuracy peaks slightly decrease towards the later epochs, and more training only leads to overfitting.



(a) Training and validation accuracy.



(b) Training and validation loss.

Figure 3.28: Training and validation accuracy and loss over epochs for the ResNet.

The final ResNet predicts the classes in the train, validation, and test set with 79.82%, 85.29%, and 85.81% accuracy, respectively. Hence, it was also outperformed by the baseline on the train set by 20.17%-points. However, the ResNet generalized better than the baseline to the validation and test set by 25.29%-points and 16.08%-points, respectively. Further, the ResNet surpassed the CNN by 0.56%-points, 0.15%-points, and 11.76%-points for train, validation, and test set. Notably, the ResNet generalized much better to the test set than the CNN. Table 3.5 contains the confusion matrices and Table 3.6 precision and recall.

	Man	Zone
Man	7135	2173
Zone	1807	8609

(a) Train set.

	Man	Zone
Man	300	84
Zone	19	297

(b) Validation set.

	Man	Zone
Man	365	27
Zone	78	270

(c) Test set.

Table 3.5: Confusion matrices for the ResNet.

	Precision	Recall
Man	79.79%	76.65%
Zone	79.85%	82.65%

(a) Train set.

	Precision	Recall
Man	94.04%	78.13%
Zone	77.95%	93.99%

(b) Validation set.

	Precision	Recall
Man	82.39%	93.11%
Zone	90.91%	77.59%

(c) Test set.

Table 3.6: Precision and recall for the ResNet.

As for the previous two previous models, recall tends to be higher for zone coverage, whereas precision tends to be higher for the man label. However, the differences are much smaller, and the tendency is even inverted for the test set. Therefore, precision and recall are better. In summary, the ResNet outperforms the CNN – likely due to increased network depth (20 vs. 6 convolutional layers).

Discussion

The baseline was much faster to train (minutes) than the two more complex models (hours). However, increased complexity and training time paid off, and both models outperformed the baseline by a considerable margin (on the validation and test set). The best performing model, the custom ResNet, generalized the best to unseen data with about 85% accuracy (for both the test and the validation set), despite learning on the augmented 19724 frames of only 60 different videos. Moreover, for all three models, inference for a single video runs in under a second. Overall, predictions are quite accurate and can be computed in real-time. Thus, the developed models are not only already helpful for post-game analysis (for players and coaches) but could even serve predictions during a game (for live commentary and fans).

3.7.4 Four Class Coverage Analysis

The task for the second tactical analysis was distinguishing between four coverage classes from Section 3.5. The classes are Cover 0, Cover 2 Man, Cover 2, and Cover 3 and will henceforth be abbreviated with C0, C2M, C2, and C3, respectively. Similar to the previous classification, data was augmented due to a limited number of samples and class imbalances. Data augmentation was carried out with the same flipping technique as before. During training, eight videos were kept out for validation and testing, i.e. two per label. As before, the data was augmented with the flipping technique. This resulted in 13987 samples for the training set, 1512 for the validation set, and 1476 for the test set. The number of labels per class is depicted in Table 3.7 for each set:

C0	3768 (26.94%)
C2M	3508 (25.08%)
C2	3622 (25.97%)
C3	3078 (22.01%)
Total	13987

(a) Train set.

C0	356 (23.54%)
C2M	384 (25.40%)
C2	456 (30.16%)
C3	316 (20.90%)
Total	1512

(b) Validation set.

C0	392 (26.56%)
C2M	392 (26.56%)
C2	344 (23.31%)
C3	348 (23.58%)
Total	1476

(c) Test set.

Table 3.7: Label numbers and distributions.

Models

As for the classification into man and zone, one linear baseline model and two more complex models (CNN and ResNet) were trained. As before, convolutions use 3×3 filters, a 1×1 stride, padding of one, and their weights are initialized as in [104] (if not mentioned otherwise). Batch size was at 32, and max-pooling is computed with a 2×2 filter and stride.

Baseline The baseline for this task was a scikit-learn [106] `SGDClassifier` as well, with the regularization term set to L1-Regularization and the corresponding multiplier (alpha) set to 0.008. Additionally, the data was pre-processed with a `StandardScaler`.

With those parameters, the model achieved the highest accuracy on the validation set (50.20%). However, the model seems to be overfitting to the validation set, since test accuracy was just 37.06%. The accuracy on the train set was 88.66%. Similar to the baseline for the first tactical analysis, there is an indicator for overfitting, i.e. the discrepancy between train and validation/test accuracy. However, increasing the multiplier of the regularization term made generalization worse. Confusion matrices, precision, and recall are listed in Table 3.8.

	C0	C2M	C2	C3
C0	3711	4	29	24
C2M	57	2980	262	209
C2	74	285	3098	176
C3	60	234	172	2612

	Precision	Recall
C0	95.11%	98.48%
C2M	85.07%	84.95%
C2	87.00%	85.27%
C3	86.46%	84.86%

(a) Train set.

	C0	C2M	C2	C3
C0	326	0	30	0
C2M	8	214	99	63
C2	10	160	136	150
C3	14	113	106	83

	Precision	Recall
C0	91.06%	91.57%
C2M	43.94%	55.72%
C2	36.66%	29.82%
C3	28.04%	26.27%

(b) Validation set.

	C0	C2M	C2	C3
C0	248	23	85	36
C2M	137	89	124	43
C2	9	79	113	143
C3	49	89	113	97

	Precision	Recall
C0	55.98%	63.27%
C2M	31.79%	22.70%
C2	25.98%	32.85%
C3	30.50%	27.87%

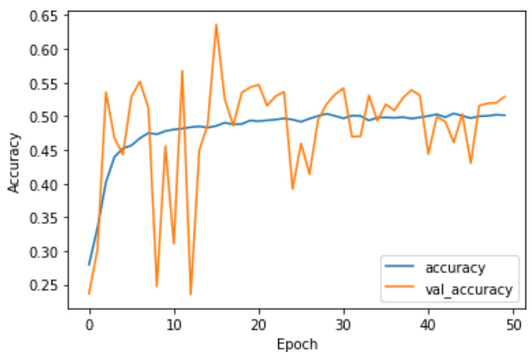
(c) Test set.

Table 3.8: Confusion matrices and precision and recall values for the baseline.

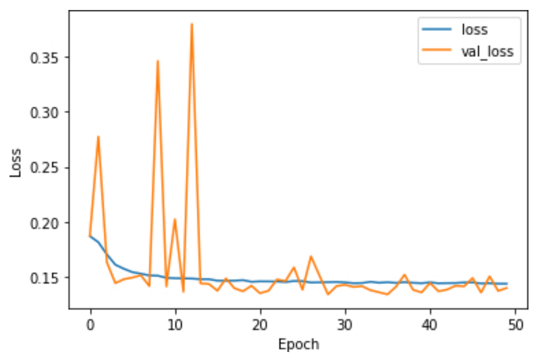
A few notable observations can be obtained from the tables: Firstly, Cover 0 has the highest precision and recall. Thus, it is easy to detect and distinguish for the classifier. Considering the coverage schemes from Section 3.5, this is reasonable. All three other coverage classes have two to three players playing in a deep zone, offering a clear indicator for distinction. Secondly, the confusion matrices depict another tendency that could be expected: Cover 2 Man and Cover 2 are relatively hard to distinguish for the algorithm – likely due to both having two players in deep zones. Somewhat unexpectedly, however, Cover 3 had the worst recall and precision for the train and validation set and the second-worst for both metrics on the test set. Similar to Cover 0, one could expect that Cover 3 is easier to both detect and distinguish because it has a unique number of deep zones among the four classes as well.

CNN The CNN for the second tactical analysis was built similarly to the one in the first tactical analysis. I implemented it in TensorFlow, and BN was applied before activation directly after convolution, as in [58]. In contrast to the first CNN, max-pooling was not used after each activation. Instead, max-pooling was just applied when the channel number was increased. The first six convolution-BN-activation blocks have only a single channel. Then, feature map size is halved while doubling channel depth to two. The same is done for the following two layers quadrupling channel depth to eight after the eighth block. Subsequently, global average pooling is applied and connected to the output layer with a dropout of 50%. The output layer consisted of four neurons with the Softmax activation function. As loss functions, I experimented with MSE and CE. Again, MSE provided better results. Optimization was done with Adam with a learning rate of 0.008.

The model's 613 weights (573 trainable) were trained for 50 epochs. Figure 3.29 depicts the corresponding training and validation accuracies on the left and losses on the right. While the model seems to underfit the training data, increasing parameters only leads to overfitting and obtained no better generalization. Again, validation accuracy surpasses training accuracy during training due to dropout.



(a) Training and validation accuracy.



(b) Training and validation loss.

Figure 3.29: Training and validation accuracy and loss over epochs for the CNN.

After training for 50 epochs, accuracy was at 53.67%, 52.91%, and 44.65% for train, validation, and test set, respectively. Thus, the model performed slightly better on the validation set. The CNN outperformed the baseline in terms of generalization by 2.89%-points and 7.59%-points on validation and test set, respectively. Table 3.9 lists confusion matrices, precision, and recall for the CNN.

	C0	C2M	C2	C3
C0	3211	228	329	0
C2M	72	1592	1844	0
C2	96	837	2700	0
C3	84	1010	1984	0

	Precision	Recall
C0	92.72%	85.22%
C2M	43.41%	45.38%
C2	39.38%	74.32%
C3	-	0.00%

(a) Train set.

	C0	C2M	C2	C3
C0	308	18	30	0
C2M	3	231	150	0
C2	0	195	261	0
C3	0	17	299	0

	Precision	Recall
C0	99.04%	86.53%
C2M	50.11%	60.16%
C2	35.27%	57.24%
C3	-	0.00%

(b) Validation set.

	C0	C2M	C2	C3
C0	100	219	73	0
C2M	47	237	108	0
C2	0	22	322	0
C3	2	13	333	0

	Precision	Recall
C0	67.11%	25.51%
C2M	48.27%	60.46%
C2	38.52%	93.60%
C3	-	0.00%

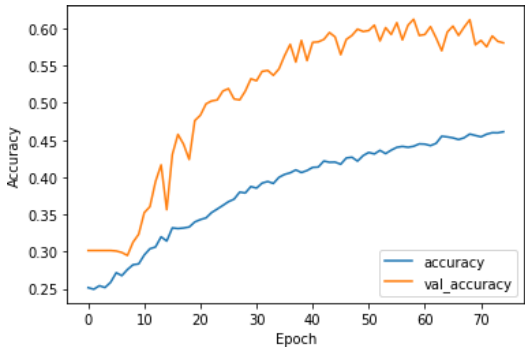
(c) Test set.

Table 3.9: Confusion matrices and precision and recall values for the CNN.

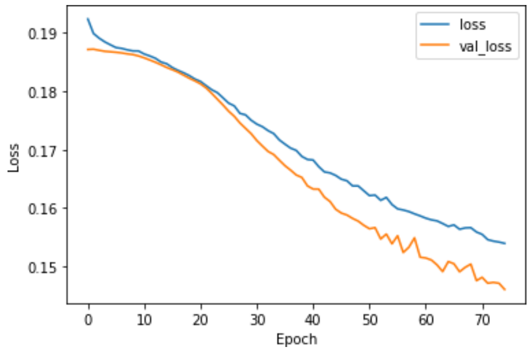
As for the baseline, Cover 0 was easy to detect and distinguish for the model indicated by a comparatively high recall and precision. Further, the model has a higher recall than precision for most labels in all three sets. However, this is partly attributed to the fact that the model could not learn the Cover 3 label. This reflects the findings from the baseline: while Cover 3 should be relatively easy to classify, the model is unexpectedly unable to do so. Despite not being able to classify Cover 3, the CNN still had better accuracy on unseen data than the baseline.

ResNet The ResNet for the second tactical analysis was built in the same manner as the first ResNet, that is, by stacking residual blocks. The first residual block applies downsampling and has four channels. After that, follow three more standard residual blocks with an equal number of channels. Then, feature map size is halved while doubling channels to eight, and another standard residual block with eight channels follows. The head of the network is built from a global average pooling layer connected to four neurons with Softmax activation via a 50% dropout. This architecture contains six residual blocks resulting in twelve convolutional layers.

As before, the best optimization results were obtained by Adam optimizing on the MSE with a learning rate of 0.00005. Overall, the model has 3712 weights, of which 3560 are trainable. The neural network was trained for 75 epochs, and the corresponding plots for accuracies and losses on the train and validation set can be seen in 3.30. While the loss curves indicate that further learning could decrease the loss, the accuracy plots indicate the opposite: validation accuracy decreases towards the later epochs. Thus, training for more epochs resulted in overfitting.



(a) Training and validation accuracy.



(b) Training and validation loss.

Figure 3.30: Training and validation accuracy and loss over epochs for the ResNet.

After optimization, the model predicts the samples in the train set with a 52.43% accuracy. On unseen data, the ResNet achieved 58.07% (validation) and 46.00% (test), thus outperforming the baseline by 7.87%-points and 8.94%-points and the CNN by 5.16%-points and 1.35%-points. Hence, the ResNet outperformed the CNN just slightly. Table 3.10 lists the ResNet's confusion matrices, precision, and recall for train, validation, and test set.

As for the CNN, the ResNet was unable to learn to predict Cover 3. Increasing depth or channels only resulted in overfitting. Cover 3 was misclassified primarily as Cover 2. Cover 2, in turn, was often confused with Cover 2 Man and vice versa. Paralleling the results of the first tactical analysis, the ResNet outperforms the CNN – again, probably due to having four more convolutional layers than the 8 of the CNN. In summary, only three of the four classes, Cover 0, Cover 2 Man, Cover 2, and Cover 3, can be predicted. To do so, the model learned on the 13987 frames of only 58 videos. Two of the 60 videos were excluded from analysis since they were labeled as Cover 1 and Cover 4.

Discussion

In many occurrences, the four-class tactical analysis emitted tendencies similar to those of the two-class tactical analysis. Due to their much more complex architecture, the CNN and the ResNet took much longer (hours) to train than the baseline (minutes). This increase in training time, however, was worth the time because both models generalized better on unseen data. The model generalizing the best was the ResNet with 58.07% and 46.00% accuracies on the validation and test set, respectively. The inference is fast for all three models, running in under a second per video. While there remains room for improvements, predictions run in real-time and thus are close to being usable in many real-world applications such as coaching or live commentary.

	C0	C2M	C2	C3
C0	3755	6	7	0
C2M	275	1163	2070	0
C2	538	679	2416	0
C3	403	896	1779	0

	Precision	Recall
C0	75.54%	99.65%
C2M	42.38%	33.15%
C2	38.58%	66.50%
C3	-	0.00%

(a) Train set.

	C0	C2M	C2	C3
C0	352	3	1	0
C2M	34	228	122	0
C2	23	135	298	0
C3	0	119	197	0

	Precision	Recall
C0	86.06%	98.88%
C2M	47.01%	59.38%
C2	48.22%	65.35%
C3	-	0.00%

(b) Validation set.

	C0	C2M	C2	C3
C0	296	77	19	0
C2M	224	107	61	0
C2	1	67	276	0
C3	4	19	325	0

	Precision	Recall
C0	56.38%	75.51%
C2M	39.63%	27.30%
C2	40.52%	80.23%
C3	-	0.00%

(c) Test set.

Table 3.10: Confusion matrices and precision and recall values for the ResNet.

4 Conclusion

4.1 Summary

The goal of this thesis was to create a classification pipeline, which uses model-based intermediary transformations of the input data. These model-based transformations incorporate knowledge from the application domain – American football. Thus, the learning of labels is facilitated for the final step of the pipeline. In particular, the pipeline uses a pre-trained 2D multi-person pose estimation model on an input image to extract joint locations of the players. Then, the playing field is registered into a top-down model with a projective transformation. This transformation is also applied to the joints of the players. After centering the projected key points above the center of each player's feet, they are stacked in layers resulting in a 3D model. This concludes the model-based steps of the pipeline. Finally, the last step of the pipeline is classifying the 3D point clouds. As sample analyses, the classification pipeline was applied to analyze defensive coverage in American football videos. Firstly, a binary classification between man and zone coverage and secondly, a four-class classification with two variations of man and zone coverage, respectively. For both cases, I trained a baseline, a CNN, and a custom ResNet. In the first case, the baseline achieved a 60.00% accuracy on the validation set and 69.73% accuracy in the final evaluation on the test set. The CNN yielded a higher accuracy for both validation and test set with 85.14% and 74.05%, respectively. Even better were the results of the ResNet with 85.29% and 85.81%, respectively. In the second case, the baseline predicted the validation and test set samples with a 50.20% and a 37.06% accuracy, respectively. Those results were improved by the CNN with 52.91% and 44.65% and slightly more so by the ResNet with 58.07% and 46.00%. While the results of the four-class classification remain improvable, the binary classification task illustrates that the main goal of this work was achieved successfully: Building a robust yet modular pre-processing pipeline, which incorporates – instead of learns – domain knowledge through a model to facilitate learning of labels.

4.2 Outlook

To conclude this thesis, I want to provide an outlook for potential future research and pipeline improvements. One idea is to add the approach for group motion segmentation developed by Ruonan Li and Rama Chellappa [14] as a pipeline step. This step could separate offensive from defensive players enriching the information for the final classification algorithm. For my application case, this could help significantly because knowing if a player is a defender and if he is close to an offender gives a clearer indication of the coverage played. Furthermore, it might be possible to automate the playing field registration using the approach by Rob Hess and Alan Fern from [16]. For comparability to related studies, analyzes could be extended to offensive formations and plays or individual players. Finally, this approach is not limited to American football. The pipeline could be adapted to other field sports such as soccer, basketball, or ice hockey with just minor adjustments to the projection. In addition to those directions, one could further investigate individual steps of the pipeline as well. For instance, pose estimation might be improved further with the recent work of Chen Wang et al. from 2021 [108]. Otherwise, the sample classifications in the final step of the pipeline may be improved using architectures designed explicitly for 3D classification such as PointNet [109], PointNet++ [110], OctNet [111], or O-CNN [112]. However, setting up or even implementing and training another pose estimation network or 3D classifier would have exceeded the frame of this thesis. Thus, these topics provide a second direction for further research. As this section illustrates, the pipeline developed in this thesis provides a platform for various future work.

Bibliography

[1] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, July 1959. Conference Name: IBM Journal of Research and Development.

[2] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The Computational Limits of Deep Learning," *arXiv:2007.05558 [cs, stat]*, July 2020. arXiv: 2007.05558.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," *arXiv:2005.14165 [cs]*, July 2020. arXiv: 2005.14165.

[4] J. Alder, "The Basic Rules of Football," 2019. https://www.liveabout.com/football-101-the-basics-of-football-1333784 (last accessed: 11/14/2021).

[5] J. Alder, "Football Glossary," 2018. https://www.liveabout.com/football-glossary-1335397 (last accessed: 11/14/2021).

[6] H. Elkins, A. Beane, J. Cherkes, W. Clougherty, J. Krohn, H. Sellars, W. T. Scherer, and J. Valeiras, "Implementing data analytics for U.Va. Football," in *2017 Systems and Information Engineering Design Symposium (SIEDS)*, pp. 202–207, Apr. 2017.

[7] J. Corscadden, R. Eastman, R. Echelberger, C. Hagan, C. Kipp, E. Magnusson, G. Muller, S. Adams, J. Valeiras, and W. T. Scherer, "Developing analytical tools to impact U.Va. football performance," in *2018 Systems and Information Engineering Design Symposium (SIEDS)*, pp. 249–254, Apr. 2018.

[8] R. Yurko, S. Ventura, and M. Horowitz, "nflWAR: A Reproducible Method for Offensive Player Evaluation in Football," *arXiv:1802.00998 [stat]*, July 2018. arXiv: 1802.00998.

[9] Y. Ding and G. Fan, "Camera View-Based American Football Video Analysis," in *Eighth IEEE International Symposium on Multimedia (ISM'06)*, pp. 317–322, Dec. 2006.

[10] B. Li and M. Ibrahim Sezan, "Event detection and summarization in sports video," in *Proceedings IEEE Workshop on Content-Based Access of Image and Video Libraries (CBAIVL 2001)*, pp. 132–138, Dec. 2001.

[11] T.-Y. Liu, W.-Y. Ma, and H.-J. Zhang, "Effective Feature Extraction for Play Detection in American Football Video," in *11th International Multimedia Modelling Conference*, pp. 164–171, Jan. 2005. ISSN: 1550-5502.

[12] B. Mahasseni, S. Chen, A. Fern, and S. Todorovic, "Detecting the Moment of Snap in Real-World Football Videos," in *IAAI*, 2013.

[13] K. Kim, D. Lee, and I. Essa, "Detecting regions of interest in dynamic scenes with camera motions," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1258–1265, June 2012. ISSN: 1063-6919.

[14] R. Li and R. Chellappa, "Group motion segmentation using a Spatio-Temporal Driving Force Model," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2038–2045, June 2010. ISSN: 1063-6919.

[15] S. Chen, Z. Feng, Q. Lu, B. Mahasseni, T. Fiez, A. Fern, and S. Todorovic, "Play type recognition in real-world football video," in *IEEE Winter Conference on Applications of Computer Vision*, pp. 652–659, Mar. 2014. ISSN: 1550-5790.

[16] R. Hess and A. Fern, "Improved Video Registration using Non-Distinctive Local Image Features," in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, June 2007. ISSN: 1063-6919.

[17] R. Hess, A. Fern, and E. Mortensen, "Mixture-of-Parts Pictorial Structures for Objects with Variable Part Sets," in *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, Oct. 2007. ISSN: 2380-7504.

[18] R. Hess and A. Fern, "Toward Learning Mixture-of-Parts Pictorial Structures," p. 8, 2007.

[19] R. W. Hess, "Toward computer vision for understanding American football in video," 2012. Publisher: Oregon State University.

[20] B. Ghanem, T. Zhang, and N. Ahuja, "Robust Video Registration Applied to Field-Sports Video Analysis," Jan. 2012.

[21] I. Atmosukarto, B. Ghanem, S. Ahuja, K. Muthuswamy, and N. Ahuja, "Automatic Recognition of Offensive Team Formation in American Football Plays," in *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 991–998, June 2013. ISSN: 2160-7516.

[22] I. Atmosukarto, B. Ghanem, M. Saadalla, and N. Ahuja, "Recognizing Team Formation in American Football," *Advances in Computer Vision and Pattern Recognition*, vol. 71, pp. 271–291, Jan. 2014.

[23] B. Siddiquie, Y. Yacoob, and L. Davis, "Recognizing Plays in American Football Videos," Jan. 2009.

[24] D. J. Stracuzzi, A. Fern, K. Ali, R. Hess, J. Pinto, N. Li, T. Konik, and D. G. Shapiro, "An Application of Transfer to American Football: From Observation of Raw Video to Control in a Simulated Environment," *AI Magazine*, vol. 32, pp. 107–125, Mar. 2011. Number: 2.

[25] S. S. Intille, *Tracking using a local closed-world assumption : tracking in the football domain*. Thesis, Massachusetts Institute of Technology, 1994. Accepted: 2005-09-27T20:30:51Z.

[26] S. S. Intille and A. F. Bobick, "Recognizing Planned, Multiperson Action," *Computer Vision and Image Understanding*, vol. 81, pp. 414–445, Mar. 2001.

[27] R. Hess and A. Fern, "Discriminatively trained particle filters for complex multi-object tracking," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 240–247, June 2009. ISSN: 1063-6919.

[28] T. Zhang, B. Ghanem, and N. Ahuja, "Robust multi-object tracking via cross-domain contextual information for sports video analysis," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 985–988, Mar. 2012. ISSN: 2379-190X.

[29] A. Bawaskar, "Interactive player tracking for videos in American football," 2014. Publisher: Oregon State University.

[30] O. Ajmeri and A. Shah, "Using Computer Vision and Machine Learning to Automatically Classify NFL Game Film and Develop a Player Tracking System," 2018.

[31] R. Yurko, F. Matano, L. F. Richardson, N. Granered, T. Pospisil, K. Pelechrinis, and S. L. Ventura, "Going Deep: Models for Continuous-Time Within-Play Valuation of Game Outcomes in American Football with Tracking Data," *arXiv:1906.01760 [stat]*, Nov. 2019. arXiv: 1906.01760.

[32] J. Hochstedler, "Finding the Open Receiver: A Quantitative Geospatial Analysis of Quarterback Decision-Making," 2016.

[33] B. Burke, "DeepQB: Deep Learning with Player Tracking to Quantify Quarterback Decision-Making & Performance," 2019.

[34] N. Sterken, "RouteNet: a convolutional neural network for classifying routes," p. 12, 2019.

[35] M. Lazarescu, S. Venkatesh, G. West, and T. Caelli, "Combining NL processing and video data to query American Football," in *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No.98EX170)*, vol. 2, pp. 1238–1240 vol.2, Aug. 1998. ISSN: 1051-4651.

[36] M. Lazarescu, S. Venkatesh, G. West, and T. Caelli, "On the automated interpretation and indexing of American Football," in *Proceedings IEEE International Conference on Multimedia Computing and Systems*, vol. 1, pp. 802–806 vol.1, June 1999.

[37] M. Lazarescu, S. Venkatesh, and G. West, "Using Natural Language and Video Data to Query and Learn American Football Plays," in *International Conference on Advances in Pattern Recognition* (S. Singh, ed.), (London), pp. 63–72, Springer, 1999.

[38] C. Taylor, "Deep Learning for In-Game NFL Predictions," 2020.

[39] M. Lazarescu and S. Venkatesh, "Using camera motion to identify types of American football plays," in *2003 International Conference on Multimedia and Expo. ICME '03. Proceedings (Cat. No.03TH8698)*, vol. 2, pp. II–181, July 2003.

[40] R. Li, R. Chellappa, and S. K. Zhou, "Learning multi-modal densities on Discriminative Temporal Interaction Manifold for group activity recognition," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2450–2457, June 2009. ISSN: 1063-6919.

[41] R. Li and R. Chellappa, "Recognizing offensive strategies from football videos," in *2010 IEEE International Conference on Image Processing*, pp. 4585–4588, Sept. 2010. ISSN: 2381-8549.

[42] E. Swears and A. Hoogs, "Learning and Recognizing American Football Plays," Jan. 2009.

[43] E. Swears and A. Hoogs, "Learning and recognizing complex multi-agent activities with applications to american football plays," in *2012 IEEE Workshop on the Applications of Computer Vision (WACV)*, pp. 409–416, Jan. 2012. ISSN: 1550-5790.

[44] J. Varadarajan, I. Atmosukarto, S. Ahuja, B. Ghanem, and N. Ahuja, "A Topic Model Approach to Represent and Classify American Football Plays," Jan. 2013.

[45] R. Dutta, R. Yurko, and S. Ventura, "Unsupervised Methods for Identifying Pass Coverage Among Defensive Backs with NFL Player Tracking Data," *arXiv:1906.11373 [stat]*, Apr. 2020. arXiv: 1906.11373.

[46] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, L. Wang, G. Wang, J. Cai, and T. Chen, "Recent Advances in Convolutional Neural Networks," *arXiv:1512.07108 [cs]*, Oct. 2017. arXiv: 1512.07108.

[47] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *arXiv:1811.03378 [cs]*, Nov. 2018. arXiv: 1811.03378.

[48] T. Szandaa, "Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks," *arXiv:2010.09458 [cs]*, vol. 903, 2021. arXiv: 2010.09458.

[49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," tech. rep., CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, Sept. 1985. Section: Technical Reports.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 6088 Primary_atype: Research Publisher: Nature Publishing Group.

[51] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science. Thesis (Ph. D.). Appl. Math. Harvard University*. PhD thesis, Jan. 1974.

[52] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv:1609.04747 [cs]*, June 2017. arXiv: 1609.04747.

[53] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization Methods for Large-Scale Machine Learning," *arXiv:1606.04838 [cs, math, stat]*, Feb. 2018. arXiv: 1606.04838.

[54] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ICLR*, 2015.

[55] J. Kukaka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," *arXiv:1710.10686 [cs, stat]*, Oct. 2017. arXiv: 1710.10686.

[56] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv:1207.0580 [cs]*, July 2012. arXiv: 1207.0580.

[57] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[58] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *arXiv:1502.03167 [cs]*, Mar. 2015. arXiv: 1502.03167.

[59] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *Journal of Big Data*, vol. 6, p. 60, July 2019.

[60] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, pp. 106–154.2, Jan. 1962.

[61] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat," *Journal of Neurophysiology*, vol. 28, pp. 229–289, Mar. 1965. Publisher: American Physiological Society.

[62] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, Apr. 1980.

[63] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, pp. 541–551, Dec. 1989. Conference Name: Neural Computation.

[64] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing Systems*, vol. 2, Morgan-Kaufmann, 1990.

[65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sept. 2014. arXiv: 1409.4842.

[66] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Apr. 2015. arXiv: 1409.1556.

[67] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, Mar. 2010. ISSN: 1938-7228.

[68] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Mller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade: Second Edition* (G. Montavon, G. B. Orr, and K.-R. Mller, eds.), Lecture Notes in Computer Science, pp. 9–48, Berlin, Heidelberg: Springer, 2012.

[69] K. He and J. Sun, "Convolutional Neural Networks at Constrained Time Cost," *arXiv:1412.1710 [cs]*, Dec. 2014. arXiv: 1412.1710.

[70] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," *arXiv:1505.00387 [cs]*, Nov. 2015. arXiv: 1505.00387.

[71] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385.

[72] F. Zhang, X. Zhu, and C. Wang, "Single Person Pose Estimation: A Survey," *arXiv:2109.10056 [cs]*, Sept. 2021. arXiv: 2109.10056.

[73] C. Zheng, W. Wu, T. Yang, S. Zhu, C. Chen, R. Liu, J. Shen, N. Kehtarnavaz, and M. Shah, "Deep Learning-Based Human Pose Estimation: A Survey," *arXiv:2012.13392 [cs]*, Jan. 2021. arXiv: 2012.13392.

[74] Z. Cao, G. H. Martinez, T. Simon, S. Wei, and Y. A. Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[75] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand Keypoint Detection in Single Images using Multiview Bootstrapping," in *CVPR*, 2017.

[76] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields," in *CVPR*, 2017.

[77] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016.

[78] K. He, G. Gkioxari, P. Dollr, and R. Girshick, "Mask R-CNN," *arXiv:1703.06870 [cs]*, Jan. 2018. arXiv: 1703.06870.

[79] T.-Y. Lin, P. Dollr, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," *arXiv:1612.03144 [cs]*, Apr. 2017. arXiv: 1612.03144.

[80] 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*, pp. 1725–1732, June 2014. ISSN: 1063-6919.

[81] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Man, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015.

[82] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An

Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alch-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[83] "TensorFlow Hub." https://tfhub.dev/ (last accessed: 11/12/2021).

[84] "TensorFlow MoveNet." https://tfhub.dev/google/movenet/multipose/lightning/1/ (last accessed: 11/12/2021).

[85] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," Jan. 2018.

[86] X. Zhou, D. Wang, and P. Krhenbhl, "Objects as Points," Apr. 2019.

[87] "OpenPose Web Page." https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/ (last accessed: 11/12/2021).

[88] "OpenPose GitHub." https://github.com/CMU-Perceptual-Computing-Lab/openpose/ (last accessed: 11/12/2021).

[89] "OpenPose Maximum Accuracy Configuration." https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/01_demo.md#maximum-accuracy-configuration/ (last accessed: 11/12/2021).

[90] "Google Colab." https://colab.research.google.com/ (last accessed: 11/14/2021).

[91] "PyTorch Keypoint R-CNN." https://pytorch.org/vision/stable/models.html#keypoint-r-cnn/ (last accessed: 11/12/2021).

[92] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, June 2009. ISSN: 1063-6919.

[93] "COCO 2017 Keypoint Detection Task." https://cocodataset.org/#keypoints-2017/ (last accessed: 11/12/2021).

[94] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[95] P. T. Inc, "Collaborative data science," 2015. Place: Montreal, QC Publisher: Plotly Technologies Inc.

[96] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *In IJCAI81*, pp. 674–679, 1981.

[97] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Ro, M. Wiebe, P. Peterson, P. Grard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[98] "ITP Glossary: Cover 0," Sept. 2015. https://insidethepylon.com/football-101/glossary-football-101/2015/09/23/itp-glossary-cover-0/ (last accessed: 11/14/2021).

[99] "ITP Glossary: Cover 1," Sept. 2015. http://insidethepylon.com/football-101/glossary-football-101/2015/09/30/itp-glossary-cover-1/ (last accessed: 11/14/2021).

[100] "ITP Glossary: Cover 2 Man," Nov. 2015. http://insidethepylon.com/football-101/glossary-football-101/2015/11/02/itp-glossary-2-man/ (last accessed: 11/14/2021).

[101] "ITP Glossary: Cover 2," Nov. 2015. https://insidethepylon.com/football-101/glossary-football-101/2015/11/16/itp-glossary-cover-2/ (last accessed: 11/14/2021).

[102] "ITP Glossary: Cover 3," Dec. 2015. http://insidethepylon.com/football-101/glossary-football-101/2015/12/09/itp-glossary-cover-3/ (last accessed: 11/14/2021).

[103] "ITP Glossary: Cover 4," Jan. 2016. http://insidethepylon.com/football-101/glossary-football-101/2016/01/13/itp-glossary-cover-4/ (last accessed: 11/14/2021).

[104] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv:1502.01852 [cs]*, Feb. 2015. arXiv: 1502.01852.

[105] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks," *arXiv:1603.05027 [cs]*, July 2016. arXiv: 1603.05027.

[106] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[107] "NN SVG." http://alexlenail.me/NN-SVG/ (last accessed: 11/12/2021).

[108] C. Wang, F. Zhang, X. Zhu, and S. S. Ge, “Low-resolution Human Pose Estimation,” *arXiv:2109.09090 [cs]*, Sept. 2021. arXiv: 2109.09090.

[109] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” *arXiv:1612.00593 [cs]*, Apr. 2017. arXiv: 1612.00593.

[110] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” *arXiv:1706.02413 [cs]*, June 2017. arXiv: 1706.02413.

[111] G. Riegler, A. O. Ulusoy, and A. Geiger, “OctNet: Learning Deep 3D Representations at High Resolutions,” *arXiv:1611.05009 [cs]*, Apr. 2017. arXiv: 1611.05009.

[112] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, “O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis,” *ACM Transactions on Graphics*, vol. 36, pp. 1–11, July 2017. arXiv: 1712.01537.

List of Figures