
Technische Hochschule Ingolstadt

Faculty of Computer Science
Computer Vision for Intelligent Mobility Systems
Study Program Computer Science - Security and Safety

Bachelor Thesis
for the degree of
Bachelor of Science (B. Sc.)

AI-based Classification of American Football Plays Combining Computer Vision and Historical Play-by-Play Data

Name and surname: Linus Teklenburg

Issued on: November 22, 2023

Submitted on: April 21, 2024

First Examiner: Prof. Dr. Torsten Schön

Second Examiner: Prof. Dr. Marc Aubreville

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, that I have not presented it elsewhere for examination purposes, and that I have explicitly indicated all material which has been quoted either literally or by consent from the sources used. I have marked verbatim and indirect quotations as such.

Ingolstadt, 15.04.2024

A handwritten signature in black ink, appearing to read 'Linus Teklenburg', with a stylized, cursive script.

Linus Teklenburg

Acknowledgments

First up, I want to thank my supervisor for this project, Prof. Dr. Torsten Schön of the Technische Hochschule Ingolstadt (THI), for giving his guidance and inspiration throughout the entire project. Torsten contributed to this approach with helpful information and valuable tips for major components of the pipeline. Secondly, I would like to thank the Faculty of Computer Vision and Computer Vision for Intelligent Mobility Systems of the THI for the opportunity to write my thesis about a topic I am highly passionate about as an American football GFL player myself. Next, I want to thank the contributors of NFLVerse and Roboflow for providing machine learning datasets used for this approach. Finally, I am very grateful for the players of the 2023 Ingolstadt Dukes for participating in my survey.

Linus Teklenburg
Ingolstadt, Germany
April 15 2024

Abstract

This bachelor's thesis explores the integration of visual and text-based models for predicting American football plays based on National Football League (NFL) games. The study focuses on automatically extracting visual features from pre-snap images of NFL plays and combining them with a text-based model trained on historical play-by-play data. The visual features are extracted using computer vision techniques and YOLOv8 model architecture, while the text-based model utilizes an XGBoost model to analyze historical play-by-play data.

The research methodology involves preprocessing and analyzing a dataset of NFL pre-snap images and historical play-by-play data. Computer vision algorithms, such as Optical Character Recognition or Line Extraction are employed to extract relevant visual features, such as player positions, formations, and field dynamics, from the images. Simultaneously, an XGBoost model is trained on historical play-by-play descriptions to capture textual patterns associated with different play outcomes, such as pass or run.

The extracted visual features and text-based predictions are then integrated into a unified prediction pipeline. The effectiveness of the integrated model is evaluated through extensive experimentation and performance analysis. Various metrics, including accuracy, precision, recall, and F1 score, are used to assess the predictive capabilities of the model.

The results demonstrate the potential of combining visual and text-based information for accurate prediction of NFL plays. The integrated model achieves promising performance in predicting pass or run plays with a test accuracy of 74.13% and validation accuracy of 73.78%. The findings of this research contribute to the advancement of predictive analytics in sports and provide valuable insights for coaches, analysts, and the football and data science community.

Nomenclature

σ	Sigmoid function
e	Euler's number
w	Neural network weights

Acronyms

CNN Convolutional Neural Network.

CV Computer Vision.

DDP Distributed Data Parallel Training.

FN False Negative.

FP False Positive.

FPR False Positive Rate.

MCC Matthews Correlation Coefficient.

NFL National Football League.

OCR Optical Character Recognition.

PRC Precision-Recall Curve.

R-CNN Region-based Convolutional Neural Network.

SGD Stochastic Gradient Descent.

SVM Support Vector Machine.

TN True Negative.

TP True Positive.

TPR True Positive Rate.

YOLO You only look once.

List of figures

1	Response times of American football players in three different scenarios.	1
2	Official American football field in the NFL (1).	3
3	Representation of "the Box" on the football field (2).	6
4	Survey question on a Cincinnati Bengals Play. (3)	7
5	Survey question on a Kansas City Chiefs Play (4)	8
6	Survey question on most important features to predict a play	9
7	Survey question on play prediction by football players (optional)	9
8	Survey question on opinions about predicting plays as a football player and human .	10
9	The YOLO approach to object detection (5).	13
10	Comparison of recent YOLO architectures (6).	14
11	Evaluation metrics: The confusion matrix (7).	18
12	Project Architecture of the approach.	21
13	Example of a training image for the scoreboard detector.	22
14	Scoreboard detection training metrics.	23
15	Scoreboard detection model testing with the test dataset.	24
16	Extracted features from scoreboard.	24
17	Example of a training image for the player detector.	25
18	Player Detection YOLO model training results over 350 epochs.	26
19	Example of player detection model on image.	27
20	Example of line detection algorithm on image.	28
21	Detected players and lines combined.	28
22	Homography transformation example on image.	30
23	Transformed player positions.	31
24	Visualization of the visual feature extraction algorithm.	31
25	Comparison between Filter, Wrapper and Embedded Methods (8)	37
26	Small cutout of decision tree of J48 algorithm.	40
27	Relationship between Play Type and Defenders in Box.	42
28	Relationship between Play Type and Offensive Formation.	43
29	Relationship between Play Type and Shotgun.	44
30	Relationship between Play Type and Yards to go.	45
31	Results: 1. Scoreboard detection.	52
32	Results: 2. Line detection.	53
33	Results: 3. Player detection.	53
34	Results: 4. Yard Lines and players combined.	54
35	Results: 5. Homography Transformed image.	54
36	Results: 6. Visual feature extraction.	55
37	Class distribution of player detection dataset.	56

List of tables

1	Metric formulas for evaluating model performance	19
2	Transforming team names in dataset.	34
3	Text dataset after cleaning.	37
4	Information Gain Ranking of remaining features.	38
5	PSO Search Algorithm after 20 Generations.	39
6	Classification report of J48 algorithm	41
7	Trained Text based Models.	46
8	Text models performance metrics on test dataset using 10-fold-cross validation. . . .	47
9	Testing Metrics of XGBoost Model.	49
10	Validation Metrics of XGBoost Model.	50
11	Actual vs. Predicted Labels for Each Image	51

Table of contents

Affidavit	I
Acknowledgments	II
Abstract	III
Nomenclature	IV
Acronyms	V
List of figures	VI
List of tables	VII
1 Introduction	1
1.1 The sport of American Football: Terminology and game concept	2
1.2 Play Type Analysis: A Survey	6
1.3 Objective of the study & Research Questions	10
2 Theoretical Background	12
2.1 Computer Vision (CV)	12
2.1.1 Introduction to Computer Vision	12
2.1.2 Object Detection with YOLO (You Only Look Once)	12
2.1.3 Optical Character Recognition	14
2.1.4 Line Detection using Probabilistic Hough Line Transform	15
2.1.5 Homography Transformation for Perspective Correction	15
2.2 Machine Learning Frameworks and Techniques	15
2.2.1 Introduction to Machine Learning	16
2.2.2 Data Parallel Training	17
2.2.3 Evaluation	18
2.2.4 Optimization	19
2.2.5 Honorable Mentions	20
3 Methodology	21
3.1 Architecture Overview	21
3.2 Computer Vision-Based Section	22
3.2.1 Scoreboard Detection using YoloV8	22
3.2.2 Optical Character Recognition (OCR)	24
3.2.3 Player Detection using YOLOv8	25
3.2.4 Line Detection using HoughLinesP Algorithm	27
3.2.5 Homography Transformation	28
3.2.6 Visual Feature Extraction	30
3.3 Text-Based Section	33
3.3.1 Data Acquisition and Collection Methods	33
3.3.2 Data Preprocessing and Cleaning	34
3.3.3 Data Analysis and Visualization	37
3.3.4 Model Selection	46
3.3.5 Hyperparameter Selection	47

3.3.6	Training	48
3.3.7	Testing & Validation	49
4	Results	51
5	Discussion	56
5.1	Influencing Factors and Limitations	56
5.2	Potential Future Improvements	56
5.3	Related Work	57
6	Conclusion	58
6.1	Is it possible to artificially predict football plays?	58
6.2	Artificial intelligence in Sports: A Look into the Future	58
	Literature references	VI

1 Introduction

American football is a highly competitive sport that demands quick reactions and agility, especially at the highest levels of competition. This is why a few hundred milliseconds are often enough to change the outcome of a single play in this game.

In an article in the "International Journal of Industrial Ergonomics" (9) from 2020 the authors examine the reaction times of American football players. An experimental setup is described in Figure 1, in which the reaction times are measured in three different scenarios.

Experiment	Parameters	Defensive Coach ...	Reaction time
1 (Baseline)	Uncertain	Signs "be ready for anything"	368 ms
2	Reprogramming response trial	Calls a running play, but it is a passing play	462 ms
1 & 2 (Average)	Certain	Guesses offensive play correctly	284 ms

Figure 1: Response times of American football players in three different scenarios.

In the first scenario, the defender did not know whether the offense would pass or run the ball. The reaction time was 368 ms. Then the defensive coordinator announced to his players that a running play was coming up, even though it was a pass play. The reaction time decreased by 25.5%. However, when the defensive player knew the opposing team's next play, an average reaction time of 284 ms was measured. The reaction time improved by 84 ms compared to the scenario in which the player was uncertain. This is an improvement of more than 20%. Anyone who has watched or played a game of American football knows how short a play can be. The average play in a three-hour game is four seconds long (10). So having almost a 100ms head start on each play by knowing the play type of the next play can make a huge difference. Of course, the National Football League (NFL) established strict rules about using technology to gain an advantage over the other team, but the question still remains: can one predict the offense's play before the center snaps the ball and the play starts? Is artificial intelligence able to do so? And even further: can artificial intelligence beat the human brain in this sport?

This bachelor thesis attempts to examine these questions in greater detail and explore possible solutions.

Many approaches already exist, working on this topic. In a 2009 article (11), three baseline classifiers were used in a two level architecture to first classify a run or a pass, and then to determine which exact play type was executed in the pass or the run. Using Spatio-Temporal Pyramid Matching (STPM) multiple frames were processed to achieve an action recognition of the players in a temporal context. With this method, a peak accuracy of 87.1% for predicting passes and 91.7% for predicting runs was achieved. These two accuracies were measured after the 150th frame (frame rate not specified) into the play.

However, there is a major difference when comparing this approach to the one addressed in this thesis. In American football, when a play is carried out, a human can detect the play type (pass or run) within a few hundred milliseconds. So trying to artificially determine the play type while or after the play is carried out adds no real advantage to the players. This

thesis presents a solution for predicting a play before it even happens. So, from an objective standpoint, this approach is essentially a form of predicting a future event.

1.1 The sport of American Football: Terminology and game concept

When speaking of "football", people around the world refer to two different sports. What most countries call "football" is often referred to as "soccer" in others.

The sport of American football actually has its roots in soccer. The first American football game was played in 1869 between Princeton and Rutgers University using the rules of traditional soccer. The objective was to kick as many goals as possible, with each goal ending one of ten games. The team that won the most games out of ten would be the final champion (12).

Obviously this game is far from the American football, as we know it today. A lot has changed in over 150 years. To comprehend the basic idea of modern day football and the intention of predicting plays, following terms should be known.

Objective: The object of the game is to score as many points as possible, while preventing the opposing team from scoring. Points can be scored by scoring a touchdown (6pts) followed by either a two-point conversion (2pts) or a 1-point conversion (1pt). By taking down an offensive player in his own end zone two points will be added to the defenders team. In addition, three points can be scored by kicking a field goal at all times.

Field: The playing field in the NFL has strict specifications, as seen in Figure 2. This ensures equal opportunities for every team, regardless of which field they play on. A standard field in the NFL measures exactly 100 yards in length and 160 feet in width. Every ten and five yards are marked with parallel lines. Adding to the 100 yards of actual field length there is an extending 10 yards of space on each side called the end zone. Located at each end of the end zones are the goalposts, which consist of one vertical crossbar and two horizontal uprights. The field is marked with yard lines throughout the field perpendicular to the sideline. This makes it easier for locating players and the ball on the field. The two sidelines enclosing the space, where the ball is life. If a player catches or carries the ball outside of those lines, the ball is out-of-bounds and therefore dead.

Line of scrimmage: This imaginary line marks the location where the next play occurs. The line of scrimmage is a horizontal line perpendicular to the sidelines and parallel to the five and ten yard lines. There are strict rules about how players are allowed to position themselves on or near the line of scrimmage. The line of scrimmage separates the offense from the defense.

Offense: The offense is the team with possession of the ball at the snap. Their main objective is to score points, either by advancing the ball into the opponent's end zone or by kicking a field goal. Key players are the quarterback, running backs, wide receivers, tight ends, and offensive linemen.

Quarterback: The Quarterback is the leader of the offense. He is responsible for passing the ball, handing the ball to running backs, running the ball himself, and oftentimes even calling plays himself. This position is likely to be the most crucial to the team's success.

Offensive Line: The offensive line is a group of typically five players positioned in front of the quarterback on the line of scrimmage. They are tasked with either protecting the quarterback and making time to throw or blocking and paving a way for a run. Defenders can detect if the play is a pass or run by reacting to the forward or backward motion of the o-line after the

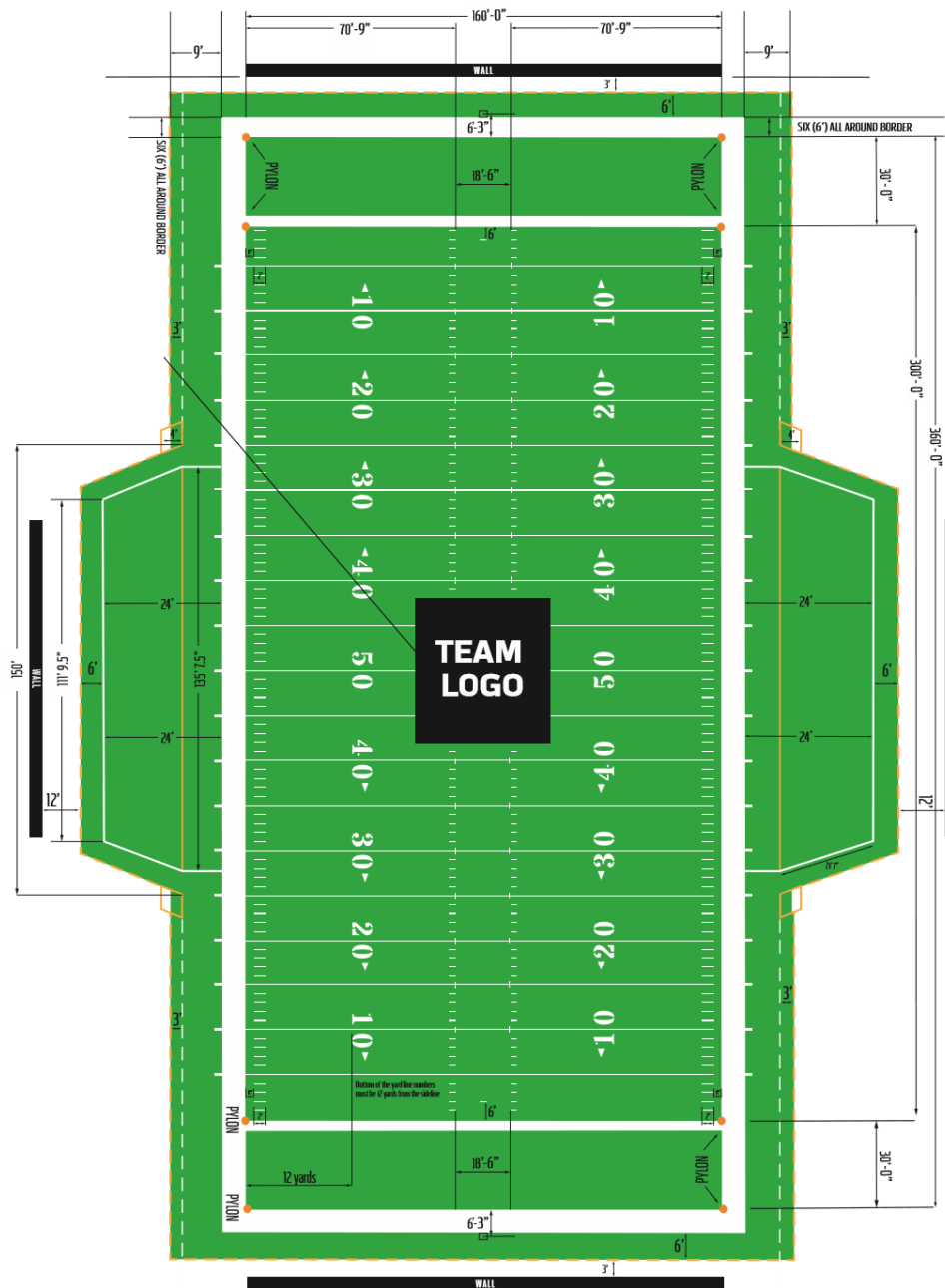


Figure 2: Official American football field in the NFL (1).

snap. If the O-line moves backwards after the snap, the play is a pass. If the O-line moves forward after the snap, the play is a run.

Center: The center is a key player and, as the name suggests, occupies the middle position on the offensive line. He initiates each play by snapping the ball to the quarterback. Additionally to that the center communicates with his o-line to coordinate blocking assignments and adjustments. The center lines up on the line of scrimmage.

Wide Receiver: A wide receiver is responsible for catching passes and gaining yards down

the field. They typically line up on or near the line of scrimmage and run routes to get open for passes. Up to four wide receivers can line up on the line of scrimmage at once.

Tight end: The Tight end is a versatile position. He lines up next to the offensive line, either on the left or right side. Tight ends are similar to wide receivers, but additionally have the task of helping the offensive line to block. They can be described as a hybrid between a wide receiver and an offensive lineman.

Running Back: This position primarily carries the ball and advances to the opponents end zone. With the ball in hand they aim to gain yards or score touchdowns. They are positioned close to the quarterback and receive handoffs when running or help to block defenders when passing. Running backs are known for their speed, agility, and ability to evade defenders, making them effective in both rushing and receiving situations. Additionally, they play a crucial role in pass protection, blocking opposing defenders to give the quarterback time to pass. Running backs are often considered one of the most dynamic and impactful positions in football.

Defense: The defense aims to stop the opposing offense from scoring points by disrupting plays. The players are the defensive linemen, linebackers and defensive backs.

Defensive Line: The defensive line is a group of players positioned opposite the offensive line. Typically, a formation consists of three or four defensive linemen. They aim to penetrate the offensive line, tackle ball carriers, and stop runs or passes behind the line of scrimmage.

Linebacker: Linebackers are often called the quarterback of the defense. They coordinate their team and play central roles in both run defense and pass coverage. They are traditionally positioned about five yards behind the defensive line. Tasks may vary for each play, such as tackling ball carriers, blitzing the quarterback, covering tight ends and running backs and providing support for other players.

Defensive Back: These players are positioned next to or behind the linebackers. They are primarily responsible for covering receivers in pass coverage and blocking or intercepting passes. Defensive back is a collective term for the positions cornerbacks and safeties.

Down: The time sequence, in which the ball is live. An offense has four downs to gain ten yards. If they don't gain ten yards in four downs, the ball is turned over to the opposing team.

Pass: A player throws the ball to a player of the same team.

Run: A player carrying and advancing the ball by foot.

Punt: When an offense decides to punt the ball, the ball is snapped to a kicker behind the line of scrimmage. The kicker then kicks the ball as far as possible in front of the opposing end zone. The location where the ball is picked up becomes the next line of scrimmage. The ball is turned over.

Turnover: The offense loses the ball to the opposing team. This happens after the fourth down, after a punt, fumble or interception.

Drive: A drive describes an attempt by the offense to collect points. The offense has four attempts to gain ten yards. If they cannot gain said yards the ball is "turned over" and the opposing offense starts their drive at the last line of scrimmage. Often times, the offense punts

the ball on the fourth down, to move the line of scrimmage farther in the opposing direction, therefore making it more difficult for the opposing team to score.

Yards to go: The term "yards to go" in American football refers to the yards the offense still has to gain to achieve another first down. This measurement provides a clear indication of the offensive team's progress toward this objective on any given play. As the offense gains yardage on each play, the number of yards to go decreases, bringing them closer to achieving their goal of sustaining their drive down the field.

Formations: The formation or "offense formation" refers to the formation in which the offense is set up on the field before the start of a play. While there are numerous possible formations, some of the most common ones include the I-Formation, Pistol formation, Shotgun formation, Singleback, Empty and Strong formation. These formations vary in the placement of players relative to the line of scrimmage and can influence the types of plays that can be executed effectively. It is also worth mentioning that the formations are not disjunctive. For example, the Shotgun formation can be intersected with the Empty formation.

Shotgun: The Shotgun formation is only characterized by one feature. In this formation, the quarterback lines up several yards behind the center. Shotgun is commonly used in passing situations, as it provides the quarterback with more time to throw the ball. This formation can often be combined with others such as the Pistol formation or the Singleback formation.

I-Formation: This formation is characterized by its own name. The quarterback and running back(s) align in the shape of an "I" behind the center of the O-line. The I-Formation is often used for running plays, as it allows for a strong push from the offensive line. The uniqueness of the I-Formation is, that the quarterback is standing right behind the center, therefore this formation is traditional under-center formation.

Pistol: The Pistol formation is characterized by the quarterback lining up in a Shotgun formation (a few yards away from the center), with the running back positioned right behind him. This formation allows for quick running plays or pass options. Compared to the I-Formation, it allows the quarterback to have a better view of the playing field.

Singleback: The Singleback formation is described by one characteristic. This is that only one running back is located in the backfield at the time of the snap. This formation enables versatile playcalling, as the outcome can be a run by the lone running back or a pass play with the running back acting as a blocking shield to the quarterback.

Empty: This formation is an offensive alignment where there are no running backs positioned in the backfield at the time of the snap. Instead, only the quarterback is located there. Most of the time, this formation is combined with the shotgun formation. Its goal is to stretch the defense horizontally across the line of scrimmage to create miscoverages and quick passes. The Empty formation is mostly used for passing plays.

Jumbo: The Jumbo formation is very special. When an offense lines up in this formation all players are located in the box. The Jumbo formation is very narrow. It is very indicative of run plays.

Snap: The snap refers to the action by the center where the ball is thrown or handed backwards between his own legs. A snap can be executed by throwing the ball (shotgun snap) or handing it directly between the legs into the hands of the quarterback.

The Box: The term "Box" or "the Box" is used to refer to the area near the line of scrimmage. This virtual box can vary in size depending on the lineup. Typically, the box is as wide as one side of the O-line to the other. It extends about seven to ten yards in front of the O-line and about five yards behind the O-line as seen in Figure 3. The box contains players such as the O-line, running backs, tight ends, the defensive line, and linebackers. The box plays a significant role in running schemes. Often, offensive coaches will call run plays when there are few defensive players in the box because of light resistance (2).

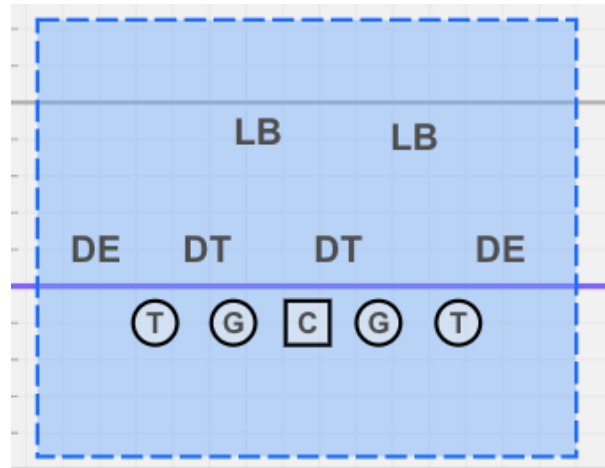


Figure 3: Representation of "the Box" on the football field (2).

Personnel: The term personnel refers to the composition of the players on the field. For example, the Offense Personnel describes the quantity and type of each player on the offensive side.

Seconds remaining: This term refers to the time remaining in a specific interval, such as the quarter or half. In this sport, terms such as "seconds remaining in quarter" are often used to indicate the playing time left.

Play type: The type of play can be "Punt", "Pass", "Run", or "Kickoff." Kickoffs occur at the beginning of each game half or after a touchdown. Punts typically occur on the fourth down. In this thesis, only Pass and Run plays will be examined, as they are the most common. Please keep in mind that these terms and descriptions are not entirely complete. However, they are sufficient for this thesis to understand the artificial approach to distinguish between pass and run plays.

1.2 Play Type Analysis: A Survey

What does it take to predict an offense play? Is it even possible? As an experienced American football player of more than 5 years, I asked my teammates of the Ingolstadt Dukes in Bavaria, Germany to answer the following questions for me.

Predicting a play by the Cincinnati Bengals: In the survey 48 experienced football players tried to predict the play type of the play illustrated in Figure 4. They were only provided with the image. The actual outcome of this play was an inside run. Only 27% of players predicted the inside run correctly, but 45% of players predicted the overall run correctly. In contrast

to this about 55% predicted a pass, which was incorrect. Although the prediction is false, a pattern can be seen. Every player logically excluded a long pass and a punt because there are only three yards to go. Most players also excluded a pass option (PO) because it would take too much time in this situation. A large number of players were drawn to a short pass or outside run because the box is tightly stacked with defense players, which would make an inside run harder to complete.

What type of play will be executed by the offense of the Cincinnati Bengals after the snap?

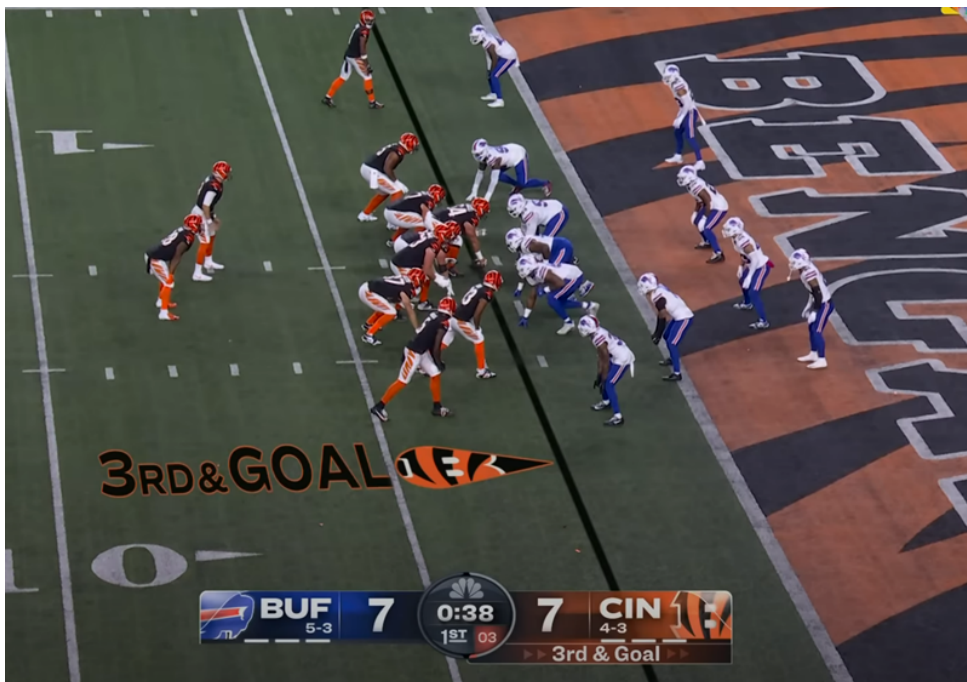
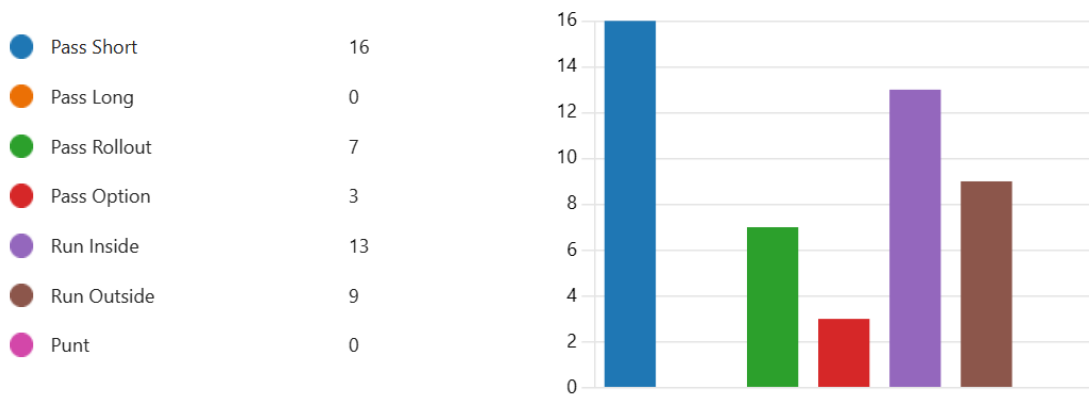


Figure 4: Survey question on a Cincinnati Bengals Play. (3)

Predicting a play by the Kansas City Chiefs: This question has the same architecture as the one before. The players needed to predict the play type and were only presented with the pre-snap image of the play as illustrated in Figure 5. The actual outcome of the play was a long pass. Twelve of the 48 players predicted the long pass correctly, and 75% of players predicted an overall pass correctly. When looking at the image, a few features can be extracted. First of

all, it is the first down, and there are still ten yards to go, which means there is no pressure to gain quick yardage. Also, the offense lined up in a Singleback formation, which can result in a run or a pass with the running back acting as a blocking shield. Additional to that, there are still 12 minutes and 43 seconds, or 763 seconds, to play in this quarter. There are two tight ends on the right side of the O-line, and two wide receivers on the left side of the field. The defense is lined up in a 4-2-5 formation and is spread evenly. There are six defensive players in the box. These features all have a certain impact on the type of play that will occur.

What type of play will be executed by the offense of the Kansas City Chiefs after the snap?

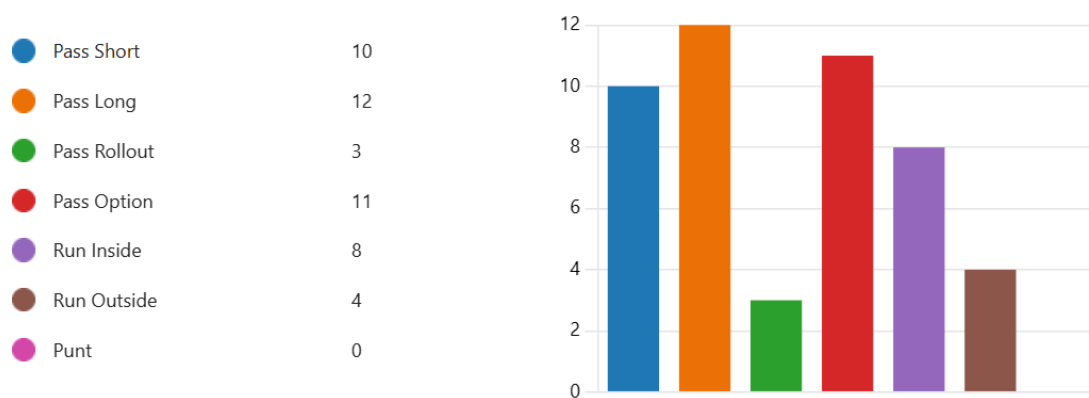


Figure 5: Survey question on a Kansas City Chiefs Play (4)

Feature Importance: When the offensive coordinator or head coach calls a play for the offense, he chooses an action that is basically a reaction to the current state of the game. The question the play caller asks himself is what type of play will result in the best possible outcome. The play caller often has only a few seconds to decide which play will best fit the current situation and produce the best result. The current state of the game is characterized by a lot of features including the current down, yards to go, or the time left in the quarter. I asked the players of the Ingolstadt Dukes which factors have the most influence when predicting an

offensive play in their opinion. The ranked results are shown in Figure 6. The players predicted that the yards to go to the next first down is the most important feature when predicting a play. The second important feature is the offense formation and personnel, followed by the current down. These are the most important features for defensive players when preparing for the next play. When you are on the field as a defensive player, you are expected to adapt to the offense's formation and even shift when motion occurs. If the offense has three or fewer yards to go, as a defensive player you will prepare for a run play. On the other hand, if there are five or more yards to go, a pass play would be more likely. It is very interesting to see how these factors are ranked by top league American football players themselves because they follow absolutely logical decision-making.

Which factors have the most influence when trying to predict an offensive play?
Sort from most important to least important.



Figure 6: Survey question on most important features to predict a play

Human ability to predict plays: This next question asked specifically the defense players of the team if they ever predicted a play correctly themselves. 40 players answered this question and only one has never recognized a play just by looking at the offense. This is an interesting question to find out if a trained player can predict the next outcome just by visually recognizing important features. The outcome suggests that players can identify plays before the snap and that probabilistic play prediction may be quite achievable.

(Defensive players only) Have you ever known what play the opposing team is going to run next in the game?

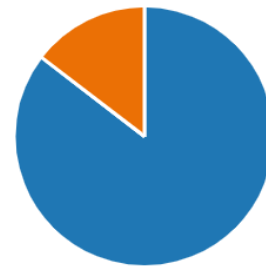


Figure 7: Survey question on play prediction by football players (optional)

Possibility of predicting plays: These last two questions, shown in Figure 8, asked players for their opinion on the possibility of predicting plays before the snap. The first question focused on the ability of the players themselves to make predictions. 41 of 48 (85,4%) saw themselves confident in predicting offense plays before the snap. On the other hand only 32 out of 41 players (78%) thought plays could be predicted artificially. Seven players did not know if plays could be predicted artificially.

Do you think it is possible to predict offense plays pre-snap as a football player?

● Yes	41
● No	7
● Don't know	0



Do you think it is possible to predict offense plays pre-snap as a computer?

● Yes	32
● No	9
● Don't know	7

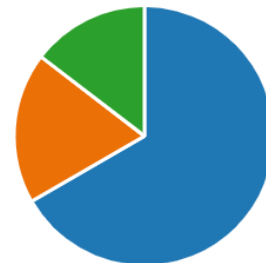


Figure 8: Survey question on opinions about predicting plays as a football player and human

1.3 Objective of the study & Research Questions

The main objective of this bachelor's thesis is to distinguish between a run and a pass play by processing a single image taken before the snap. Specifically this prediction should be made autonomously by an architecture of neural networks and algorithms for NFL image data. Typically, when working with image data, a single model, such as a Convolutional Neural Network (CNN), is trained to predict specific outcomes. However, this single model is only effective when large amounts of training data are available. In the case of American football plays, it can be quite challenging to manually gather the necessary amount of visual data required for effective training. Secondly the prediction is very complex and takes many visual features into account. Out of those two reasons, training multiple models and breaking down the problem into smaller parts may be necessary to generate a stable solution. This approach is divided into two parts. The first part, which is computer vision-based, aims

to preprocess the image, extract the most important features, and transform them into text or boolean-based features. The second part of the approach uses historical play-by-play data to recognize reoccurring patterns to finally predict play types.

This thesis will explore the following research questions:

1. What visual features can be extracted from football play images?
2. What image transformation is necessary for accurate feature extraction?
3. Which features are most important for play prediction?
4. Which neural network model performs best for text-based game prediction?
5. How can hyperparameters be optimized?
6. How are multiple neural networks and algorithms be chained together in a pipeline?
7. What level of accuracy can be achieved in play prediction using this approach?

These research questions aim to further explore the field of computer vision and machine learning in sports analytics, particularly in the context of play recognition in American football. The ultimate goal is to develop an autonomous, robust fusion of neural networks and algorithms that can classify play types solely based on visual features and historical data.

2 Theoretical Background

This section aims to provide a theoretical background of the approach. It is divided into two parts. The first part focuses on the field of computer vision, while the second part addresses machine learning frameworks and techniques.

2.1 Computer Vision (CV)

Computer vision is a widespread field in machine learning and a significant part of this thesis. The following subsection explains the components of computer vision necessary this approach of predicting American football plays.

2.1.1 Introduction to Computer Vision

In computer vision, one tries to "describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions" (13). Today, CV is used in a lot of real-world applications, such as optical character recognition (OCR), medical imaging, 3D model building, self-driving vehicles, surveillance, and many more (13). The field of CV can be subdivided into four categories: Classification, Classification & Localization, Object Detection, and Image Segmentation. While the first two categories focus on single objects, the second two categories deal with multiple objects (14). This thesis uses Classification & Localization to classify and localize the scoreboard of an image, and Object Detection to detect football players in images.

CV goes as far back as 1970 starting with Digital image processing to try to mimic human intelligence. In recent history, especially with growing computing power, CV has paved the way to human tracking and modeling and deep learning (13).

2.1.2 Object Detection with YOLO (You Only Look Once)

To compare with some previously developed models, the Deformable Parts Model (DPM) uses filters that represent complete objects and parts of objects to create a detection confidence map using a sliding window technique. The idea is to ascertain if an object is likely to be present if all parts of the filter are present (15). Additionally, Region-based CNNs (R-CNN) also employ the same sliding window technique. At first, around 2,000 region proposals are extracted from one image. After that, a Convolutional Neural Network (CNN) extracts a feature vector, and a prediction is made with a Support Vector Machine (SVM) for each class. Every class has its own SVM (16).

You Only Look Once (YOLO) presents a completely new approach to object detection. YOLO streamlines the whole process, predicting bounding boxes and class probabilities for the entire image all at once in one single network.

This network uses features from the entire image to predict the bounding boxes for all objects across all classes at the same time. To achieve this, the input image is overlaid with an $S \times S$ grid as shown in Figure 9. Each grid cell then produces a set of bounding boxes centered on a point inside the grid cell with the associated confidence scores and a class probability

map. The class probability map points out which class is most probable for the cell. At last the Bounding boxes and class probability map are combined to determine the actual bounding boxes (5).

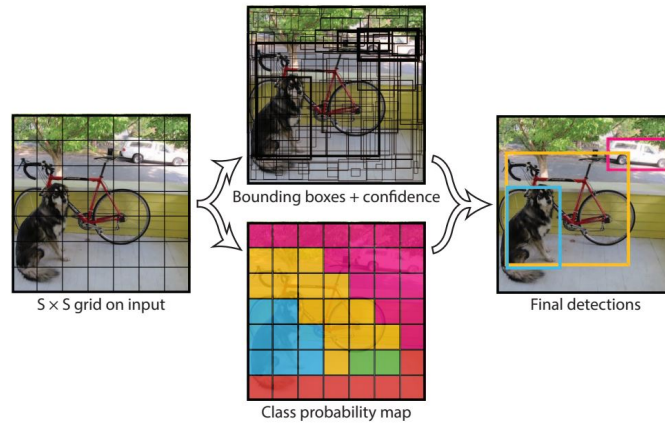


Figure 9: The YOLO approach to object detection (5).

This integrated approach allows YOLO to efficiently detect objects in real time while maintaining accuracy.

YOLO consists of a backbone CNN, typically based on the Darknet architecture, followed by detection layers that predict the bounding boxes and class confidences. It also uses techniques such as batch normalization, anchor boxes and dimension clusters to improve accuracy (14). The first YOLO model (YOLOv1) was produced in 2015, and the most recent (YOLOv8) was developed in 2023. This thesis utilizes specifically the Ultralytics YOLOv8.1.26 and Ultralytics YOLOv8.1.42 versions with either an Adam or Stochastic gradient descent (SGD) optimizer. Ultralytics, the company who also developed YOLOv5, specializes in the development and optimization of the YOLO object detection framework, providing enhancements and pre-trained models to improve accuracy and efficiency in real-time object detection tasks (17).

"YOLOv8 provided five scaled versions: YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large) and YOLOv8x (extra large). [...] Evaluated on MS COCO dataset test-dev 2017, YOLOv8x achieved an AP of 53.9% with an image size of 640 pixels (compared to 50.7% of YOLOv5 on the same input size) with a speed of 280 FPS on an NVIDIA A100 and TensorRT" (18). Figure 10 shows this performance comparison.

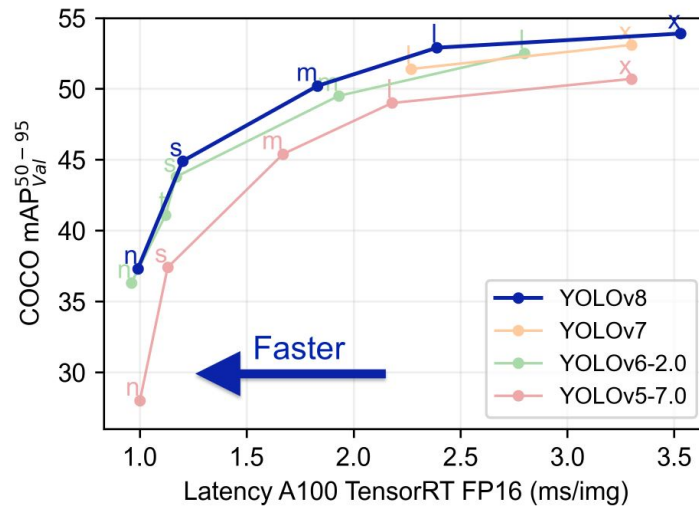


Figure 10: Comparison of recent YOLO architectures (6).

Ultralytics YOLOv8 implements PyTorch as the deep learning framework for model training and computations, and CUDA for GPU-accelerated computations. This integration enables Ultralytics YOLOv8 to utilize the power of GPUs for efficient and effective object detection tasks. Additionally, Ultralytics partners with the computer vision platform Roboflow (19) to enhance tasks such as dataset labeling, training, visualization, and model management.

2.1.3 Optical Character Recognition

The art of Optical Character Recognition (OCR) enables the automatic recognition of single and multiple characters from images. Fundamentally OCR functions as the human ability to read. OCR dates back to the early 20th century, as the first patents were filed in the 1920s and 1930. Especially in recent years it has evolved to read complex documents, including tables, mathematical symbols and handwritten characters. Challenges of OCR technology include recognizing irregular handwriting, poor image quality and similarities between characters, such as "U" and "V" (20).

While there are many OCR modules and APIs on the market today, such as TesseractOCR and EasyOCR, the module that worked best for our purposes was EasyOCR. EasyOCR is a Python module, supporting over 80 languages, designed so simplify text extraction. It consists of three components: feature extraction, sequence labeling and decoding. Feature extraction involves deep learning models such as Residual Networks (ResNet) and VGG to extract the sought features from the image. Sequence labeling uses Long-Short-Term Memory (LSTM) networks to interpret a sequential context. Finally the sequences is decoded into text using Connectionist Temporal Classification (CTC) (21).

EasyOCR is often used to convert documents to text, to recognize number plates or to recognize text in images. For our purposes, EasyOCR is used as a solution to extract important features from the scoreboard in a football image, such as the scores, down, yards to go and the time.

2.1.4 Line Detection using Probabilistic Hough Line Transform

The Hough Lines Transform is a popular method used in computer vision to detect straight lines within an images. First of all a canny edge detector is applied using kernels. This enables extraction of hard edges and contrasts. After that the lines in the image are represented as points in a parameter space, using the polar coordinate system (represented by vectors). For each point a set of parameters (r, θ) is computed using the equation below.

$$r = x \cdot \cos(\theta) + y \cdot \sin(\theta)$$

In this equation, (x, y) are the coordinates, r is the perpendicular distance from the origin to the line, and θ is the angle.

After representing each point in the polar coordinate system, they are accumulated. Only lines with a significant number of edge pixels are detected as lines. The presence of lines can be manually adjusted in the next step by setting the threshold. Finally the detected lines are represented in the original image (22).

OpenCV has two implementations of Hough Line Transform: the Standard Hough Transform (HoughLines) and the Probabilistic Hough Transform (HoughLinesP). The difference is, that HoughLinesP is an optimized version of the Hough Line Transform, aimed to reduce computation time. HoughLinesP randomly selects a subset of points and computes lines using only these points, which makes it more efficient. This thesis uses the optimized version HoughLinesP to detect yard lines. The output of this method is represented as line segments defined by the endpoints (x_1, y_1, x_2, y_2) .

2.1.5 Homography Transformation for Perspective Correction

The term homography transformation refers to a geometric transformation that maps points from one image to another image. It is used to display the original view of an image in another perspective. In our case, homography transformation was used to transform the side view perspective of football images into a bird's eye view. Typically, it is performed by selecting certain points of the original image and specifying corresponding points in the target image. To automate this task, keypoint detectors such as SURF or SIFT are often used. There are very interesting papers on homography transformation of football images, including solutions like projecting the frame on a pre-made field using a neural net or slicing the image, but this thesis uses a rather simple approach, which will be explained later on.

2.2 Machine Learning Frameworks and Techniques

Apart from computer vision, machine learning has experienced significant developments in most recent history. The following subsection outlines the techniques and tools used in this thesis, such as the principles of Supervised Learning or Distributed Data Parallel Training (DDP).

2.2.1 Introduction to Machine Learning

Machine learning is a subset of artificial intelligence (AI) which focuses on developing algorithms and models that enable computers to learn and improve from experience. This allows computers to act in new contexts without being explicitly programmed to do so. The methods of machine learning can be subdivided into four categories: Supervised Learning, Unsupervised Learning, Semi-Supervised Learning and Reinforcement Learning. This thesis only focuses on the field of Supervised Learning. Specifically, it focuses on the classification type of supervised learning. Given a fully labeled training dataset with the following appearance:

$$D = \{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$$

The input data $x^{(i)}$ must be provided with the labeled data $y^{(i)}$. The main objective is to approximate a function $f: X \rightarrow Y$, where $f(x^{(i)}) \approx y^{(i)}$, or in other words, where the loss of the function is as close to zero as possible. To compute the aggregated loss of a model first of all a loss-function is selected. For binary classification, the logistic loss is often used as follows:

$$L(Y, \hat{Y}) = -\frac{1}{N} \sum_{i=1}^N [Y_i \log(\hat{Y}_i) + (1 - Y_i) \log(1 - \hat{Y}_i)]$$

where:

- X is the input feature vector,
- Y is the label,
- N represents the quantity of data points,
- Y_i is the true label for the i -th sample, and
- \hat{Y} is the predicted probability of the first class for the i -th sample.

The class probability \hat{Y} is obtained by the output of the activation function. For example binary classifiers often use the sigmoid function:

$$\hat{Y}_i = \sigma(f(X_i; \theta))$$

The decision function used in this formula is calculated as follows:

$$f(X; \theta) = XW + b$$

where:

- W are weights, and
- b is the bias.

Keeping in mind the idea of reducing a loss to optimize the model, the models for predicting passes or runs in this thesis do not require the use of loss functions or activation functions in the same way as neural networks or machine learning algorithms.

The random forest model is based on decision trees. The architecture doesn't directly optimize a loss function. Instead, multiple decision trees are constructed in feature space and their predictions are aggregated to select the best trees. Each tree is trained to minimize an impurity measure like Gini impurity or entropy. This impurity measure acts similar to the loss-function and measures the relative amount of the target class to tell how impure a leaf of the tree is. Support Vector Machines aim to find a hyperplane that separates different classes in the feature space. Instead of using a loss-function SVMs optimize a margin-based objective function that tries to maximize the margin between decision boundaries and the data points. In summary SVMs want to separate the two target classes as clean as possible with a hyperplane.

XGBoost is an ensemble learning method based on decision trees, similar to Random Forests. The difference between random forest and XGBoost is, that XGBoost uses gradient boosting, which actually involves optimizing a loss function. XGBoost minimizes a loss function, often the logistic loss, as previously mentioned, for binary classification. It also uses gradient descent techniques to minimize this loss function, which is similar to what happens in neural networks during training. XGBoost does not have an activation function in traditional sense.

Each model has its advantages and disadvantages. They all perform differently in different environments. SVMs are effective for high-dimensional spaces. They have a good generalization ability especially with rather small datasets and can handle non-linear data with the kernel trick. The kernel trick projects the feature space in a higher-dimensional space where it becomes linearly separable. Disadvantages of SVMs are that they are computationally intensive, especially with large datasets. Often they are not suitable for large datasets. In comparison random forest models are more suitable for large datasets with high dimensional. Random forest is robust to overfitting due to ensemble learning, but may not perform well on imbalanced datasets. XGBoost on the other hand implements parallel processing and is therefore more efficient with large scale datasets. Also XGBoost often outperforms Random Forest in prediction accuracy and can capture complex relationships in data (23).

2.2.2 Data Parallel Training

When working with large datasets processing power is often a concern. Especially images hold a lot of information to be processed in the training process. To counterweight this problem more and more models have been trained on multiple GPUs in recent years. There are multiple ways to train a model on more than one GPU. Depending on the nature of the data and the project one can use Data Parallelism, Model Parallelism, Pipeline Parallelism or Local Parallelism. Those architectures all offer different advantages and disadvantages. In our case, because of big data quantity in images Data Parallelism makes the most sense. Data parallelism distributes the workload across multiple computational resources or GPUs. To apply data parallelism, several libraries can be used, such as torch.distributed. The training is constructed as follows: First of all the model is replicated for each GPU. Then the data is divided into smaller batches and each batch is assigned to one GPU. Each GPU performs a forward pass to compute predictions and a backward pass to calculate gradients independently. After that the gradients are aggregated to perform a parameter update to optimize the model (24). Gradient aggregation and parameter sharing can be achieved using one of two methods. Either a parameter server is used to collect and distribute parameters or the ring-reduce method

is used, where each GPU sends the gradients to the next GPU and the last GPU sends their gradient to the first, to essentially build a ring. Following formula is applied when computing the gradients in weighted data parallel training:

$$\nabla_w f(w; X) = \frac{1}{B} \sum_{j=1}^B \frac{1}{N} \sum_{i=1}^N \nabla_w \mathcal{L}(w; X_{ij})$$

where:

- B is the total number of batches,
- N is the total number of GPUs,
- $\mathcal{L}(w; X_{ij})$ is the loss function for batch j on GPU i with data X_{ij} , and
- $\nabla_w \mathcal{L}(w; X_{ij})$ is the gradient of the loss function in relation to the model parameters w for batch j on GPU i .

This formula is used to determine the new gradients for the next iteration. In our case, data parallel training was used to accelerate the YOLOv8 training of the scoreboard and player detection.

2.2.3 Evaluation

To evaluate the quality of models different evaluation metrics indicate different aspects of the model. The most significant metric is the accuracy of a model. The accuracy can be calculated using the confusion matrix seen in Figure 11.

		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	TP True Positive	FP False Positive
	negatives	FN False Negative	TN True Negative

Confusion Matrix

Figure 11: Evaluation metrics: The confusion matrix (7).

The confusion metrics shows four cells: True positives (TP), false positives, false negatives and true negatives. TP is the number of instances correctly predicted as positive. FP is the number of instances incorrectly predicted as positive (actually negative, but predicted as positive). TN is the number of instances correctly predicted as negative. Finally, FN is the number of instances incorrectly predicted as negative (actually positive but predicted as

negative). Based on this confusion matrix many evaluation metrics can be calculated as seen in Table 1 below.

Metric	Formula
True Positive Rate (TPR) or Recall (R) or Sensitivity	$TPR = \frac{TP}{TP+FN}$
False Positive Rate (FPR)	$FPR = \frac{FP}{FP+TN}$
Precision	$Precision = \frac{TP}{TP+FP}$
F-Measure	$F - Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall}$
Matthews Correlation Coefficient (MCC)	$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$
Accuracy	$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$

Table 1: Metric formulas for evaluating model performance

The TPR or Recall or Sensitivity measures the proportion of actual positive cases that are correctly classified by the model. A high TPR is desirable. The FPR, on the other hand, represent the relative amount of negative classes that are incorrectly classified. The model is good, when the FPR is low. The precision measures the accuracy of only positive predictions by the model. High precision indicates good performance. The F-measure, also known as F1-Score is the mean of precision and recall. This is very useful for cases where the data is imbalanced. A high F-measure is desirable (25). The Matthews Correlation Coefficient (MCC) is a correlation coefficient of the predicted and true classes. It considers TP, FP, TN and FN. An MCC close to 1 is good, close to 0 is a random classifier, and close to -1 is bad. The accuracy measures the overall correctness of the model's predictions all across all classes. An accuracy close to 1 or 100% is desirable. Two other evaluation metrics used in this thesis are the ROC area and the Precision-Recall Curve (PRC) area. The ROC area, or area under the curve (AUC), indicates the ability of the model to discriminate between positive and negative classes across different threshold values. A higher value close to 1 indicates better performance. The PRC area focuses on precision and recall rather than TPR and FPR like the ROC area. The PRC area is often useful for unbalanced data.

2.2.4 Optimization

This thesis works with various optimizers during training. One of them being the Stochastic Gradient Descent method (SGD). SGD is a variant of the gradient descent algorithm. The gradient descent algorithm updates model parameters based on the gradient of the loss function computed on a subset of training examples, rather than the whole dataset. SGD works as follows. Firstly the weights and biases are initialized randomly. Then an iterative process of batch selection, gradient computation and parameter updating is performed. In batch selection the training dataset is randomly shuffled and divided into mini-batches. For each mini-batch the gradients are computed while only considering the model parameters of the current batch. Then the parameters are updated for this iteration using following formula:

$$\theta := \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta; x_i, y_i)$$

Where:

- θ is the model parameters,
- α is the learning rate,
- $\nabla_{\theta}\mathcal{L}(\theta; x_i, y_i)$ is the gradient of the loss function in relation to the model parameters θ for training example (x_i, y_i) .

The iterative process of batch selection, gradient computation and parameter updating are executed until a convergence criterion is met. After completion the final model parameters represent the optimized values that minimize the loss function on the training data. SGD is efficient and well-suited for large datasets since only a small amount of data is taken into consideration when computing the loss (26). However, the random nature of the updating process can lead to oscillations and noise in the optimization process. To counteract these problems, strategies such as momentum and adaptive learning rates can be introduced. This is what the Adam optimizer, which was used with YOLO for this approach, is designed to do. Adam, or adaptive momentum estimation, introduces adaptive learning rates, momentum, and bias correction. Adaptive learning rates also result in better handling of sparse gradients. In comparison to SGD, Adam often converges faster and more reliably, especially for high-dimensional machine learning problems. Adam typically outperforms SGD and other optimizers, such as RMSProp or AdaGrad (27).

This next and final optimization technique was used to find the best hyperparameters for the XGBoost model. GridSearchCV is a hyperparameter optimization technique used to search for the best set of hyperparameters for a given machine learning model. It does not optimize the loss function during training like traditional optimizers. Instead, GridSearchCV searches a given grid of parameters and evaluates the resulting models using cross-validation to find the best combination of parameters for the model (28).

2.2.5 Honorable Mentions

The following are honorable mentions of machine learning frameworks and libraries that have had a major impact, while working on this approach. To generate the computer vision based models of this thesis, PyTorch was used. PyTorch is one of the leading open-source machine learning libraries. It offers dynamic computation graphs and deep learning for research and production. PyTorch also offers support for GPU acceleration and parallel training (29). To accelerate the task of feature importance measuring and text model selection, the toolkit Weka was used. Weka is an open-source tool for machine learning and data mining written in Java. It offers a wide range of algorithms for data processing, classification, regression, and clustering (30). For preprocessing data and performing hyperparameter optimization, Scikit-learn (sklearn) played a major role. Sklearn is a machine learning library in Python, featuring efficient implementations of popular algorithms for classification, regression, clustering, dimensionality reduction, and ensemble learning (31). For cleaning and preprocessing text data, packages NumPy and Pandas were used. NumPy is a fundamental package for numerical computing in Python. The package offers powerful tools for array manipulation and mathematical operations with arrays (32). Pandas is a data manipulation tool built on top of NumPy. Pandas offers data structures such as DataFrames that simplify data handling and analysis tasks (33). Together, NumPy and Pandas form the backbone of data science workflows in Python.

3 Methodology

After establishing the theoretical knowledge, this section elaborates on the practical procedure of predicting plays from an image. This section begins with an architectural overview, followed by the computer vision-based part, and finally the synthesis of a text-based model that makes the final prediction: pass or run.

3.1 Architecture Overview

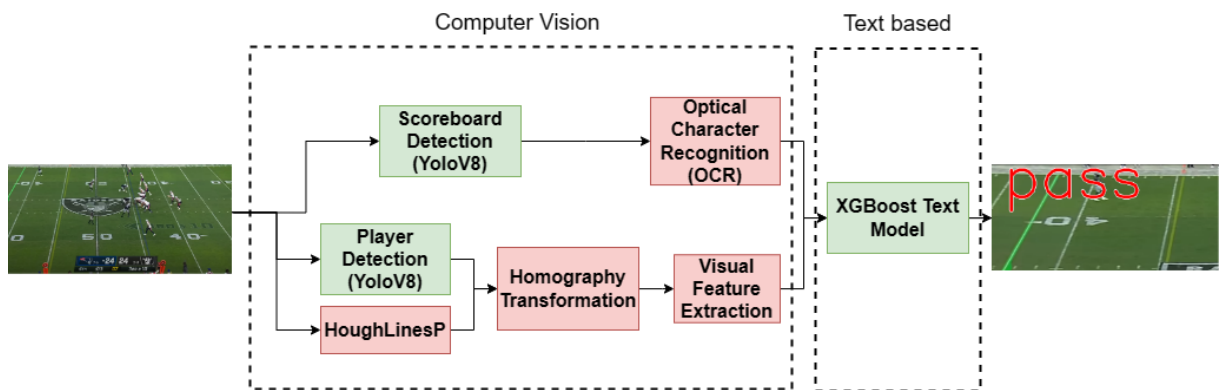


Figure 12: Project Architecture of the approach.

Figure 12 depicts the entire architecture of the approach. Green boxes mark customized neural networks and red boxes mark python algorithms or scripts. In its entirety the pipeline consists of three neural networks and four python scripts. Technically, the OCR Python script also uses a neural network for text recognition, but it is still only marked red as an algorithm because I did not create this model myself. The pipeline can be divided into two parts. The first parts, which is computer vision based, extracts visual features out of the image, which are necessary for prediction. The computer vision part includes following neural networks as seen in Figure 12: "Scoreboard Detection (YOLOv8)" and "Player Detection (YOLOv8)". It also contains the following algorithms: "Optical Character Recognition (OCR)", "HoughLinesP", "Homography Transformation", and "Visual Feature Extraction". The text-based part only contains one neural network ("XGBoost Text Model") and no algorithms. The pipeline is designed to chain multiple models in series. The outputs of previous models in the pipeline are intended to be the inputs for the XGBoost text model to predict the final output.

This is what a run through the pipeline looks like: First, an image is fed into the architecture. The first model, "Scoreboard Detection (YOLOv8)," determines the location of the scoreboard in the image. With this information, the "Optical Character Recognition (OCR)" algorithm extracts the following features: score differential, time to play in quarter, down, and yards to go. These features are direct input features for the XGBoost model. Next, the initial input image is processed simultaneously in the "Player Detection (YOLOv8)" model and the "HoughLinesP" algorithm. With both pieces of information of the players' locations and all yard lines, the "Homography Transformation" algorithm then determines the line of scrimmage and transforms the side view into a bird's eye view. With the bird's eye view, the "Visual Feature Extraction" algorithms can then extract the remaining visual features. All gathered

features in the computer vision based part are then passed onto the text based part. In the text based part, the features are put into the XGBoost model, and then a prediction in the form of "pass" or "run" is made.

Although the computer vision part is first in the pipeline and will be discussed firstly in this thesis, it was actually put together after the text-based model. The reason for this is that the outputs of the computer vision part had to be fitted to the inputs of the text-based part. So, it was easier to first determine which features were actually relevant to the text model before visually extracting them.

3.2 Computer Vision-Based Section

This subsection elaborates on the computer vision-based part of our pipeline described above. The input of this part is the initial image and the desired output are the features we need for the text-based model.

3.2.1 Scoreboard Detection using YoloV8

Important features when analyzing a football image are within the scoreboard. The scoreboard holds information like the current score of both teams, the seconds left in the quarter, the quarter itself the down and the yards to go. Those features play a major role when predicting the play type. To actually extract the scoreboard from the image so that the OCR doesn't get confused with yard markers or player numbers, a YOLOv8 is generated whose only task is to classify and locate scoreboards.

The images used to train this model were all extracted from one YouTube video (34). Multiple scoreboard layouts from NFL Network, ESPN, CBS-NFL, SNF, and FOX-NFL were used from the video to cover the variety of NFL scoreboards in our model. In total, 146 images were annotated using Roboflow (19) and after augmentation, the dataset consisted of 334 images. This may seem like a small amount of data for a computer vision model, but given the simplicity of the task, the data was more than sufficient for this problem. In Figure 13 an example image from the dataset can be seen.



Figure 13: Example of a training image for the scoreboard detector.

The training itself was done online using the machine learning platform Kaggle (35), with data-parallel training on two NVIDIA T4 GPUs with 15103MiB each. Specifically, the "s" version of the YOLOv8 model was used for smaller models. The images were resized to 640x640 pixels, the batch size was 16 and the training was done over 50 epochs. In addition, the optimizer AdamW was used, which is a derivative of Adam. Several loss functions were used, including box loss, class loss, and DFL loss. The training was completed after 458 seconds or approximately 7minutes and 38 seconds.

The performance of the models was evaluated after each epoch. In Figure 14 you can see the metrics for the model. The first row of the image shows, from left to right: training box loss, training class loss, training dfl loss, precision and recall. The second row of the figure shows from left to right: validation box loss, validation class loss, validation dfl loss, mAP50 (mean average precision at 50% IoU), and mAP50-95 (mean average precision between 50% and 95% IoU). Although the values oscillate a lot during training, the performance after the 50th epoch is still very good.

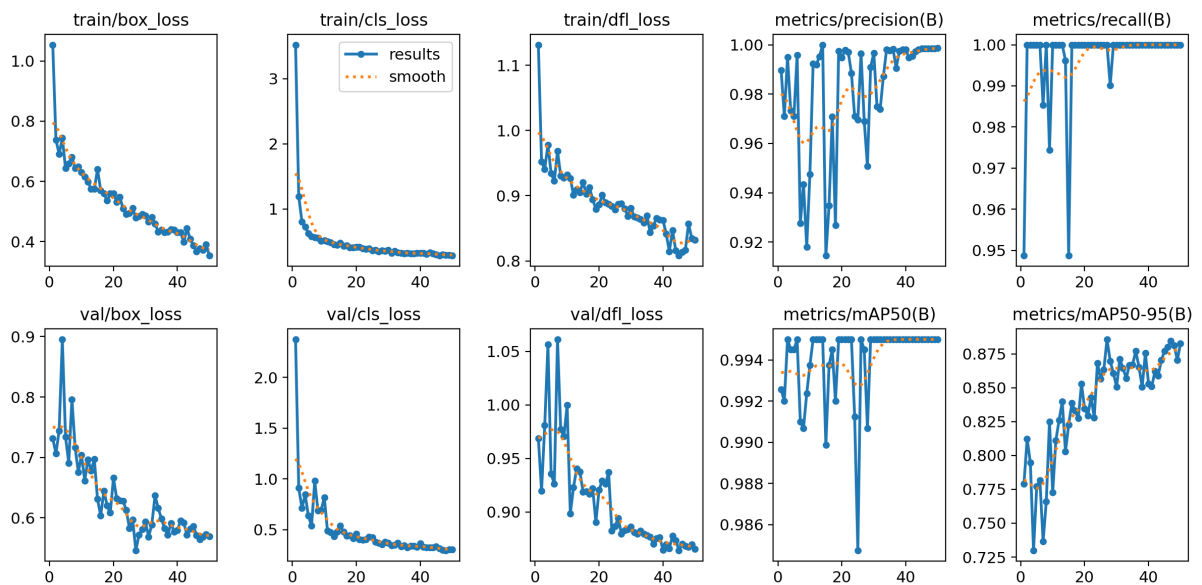


Figure 14: Scoreboard detection training metrics.

Following scores are achieved after the 50th epoch:

- **Train/box_loss:** 0.35285
- **Train/cls_loss:** 0.28429
- **Train/dfl_loss:** 0.83224
- **Metrics/precision(B):** 0.99858
- **Val/box_loss:** 0.56904
- **Val/cls_loss:** 0.30649
- **Val/dfl_loss:** 0.86527
- **Metrics/recall(B):** 1.00000

- **Metrics/mAP50(B):** 0.99500
- **Metrics/mAP50-95 (B):** 0.88272

These metrics indicate a fairly good model, considering that the recall is 1 and the precision is close to 1. Also the mAP50 (B) and mAP50-95 (B) are high, indicating strong performance across different IoU thresholds. However, the validation losses are higher compared to the training losses, which could indicate overfitting or difficulty in generalizing. This could be a consequence of a rather small dataset and many epochs. Nevertheless, the model is quite sufficient for our purposes and detects scoreboards without any problems, as seen in Figure 15.



Figure 15: Scoreboard detection model testing with the test dataset.

3.2.2 Optical Character Recognition (OCR)

After completing the scoreboard recognition, the next step in our pipeline is OCR. Fortunately, because the scoreboard is a digital overlay in every image, the image quality is always good and constant, so there's no noise to factor in. Therefore, the probability of correctly predicting texts is very high. The OCR-algorithms starts with defining custom vocabulary for the reader to use:

```
1 custom_vocabulary = ":-stndrh&0123456789"
```

These letters, numbers and special characters are the only ones possible for the desired features. For example "2nd & 10" contains "nd&012" and so on. Then the threshold is set to 50% to exclude uncertain predictions. Then the algorithm goes through a series of exclusion criteria to determine what, if any, feature the predicted text may be. For example, a text containing ":" is always the time, because there is no other use of the colon in the scoreboard. After running through the entire process the texts are assigned to the corresponding feature as seen in Figure 16.

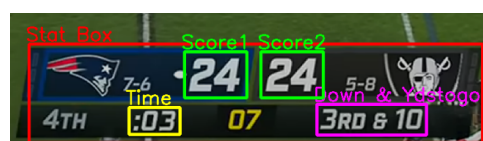


Figure 16: Extracted features from scoreboard.

Assigning the correct text to the score is quite difficult because any number on the scoreboard could be a score. However, when comparing different scoreboard layouts, the scores always have a few things in common: they are one of the largest numbers on the board, they are usually close together, and they are on the same vertical level. So to assign the scores, I extracted all possible combinations of numbers in the image that have a certain height. In Figure 16 the combinations would be: "24-24", "24-7" and "7-24" because only three numbers fit the height of a possible score. Then the euclidean distance is applied to each of the combinations. The vertical distance is also weighted by a factor of 8, since the scores must be on the same vertical level. The combination with the least computed distance had to be the score. This approach worked quite well. As of now the features yards to go, down, score differential and time remaining in quarter were extracted from the image.

3.2.3 Player Detection using YOLOv8

In order to recognize important features such as formation, offensive or defensive personnel, it is necessary to recognize the players in the image.

To achieve this capability, another YOLOv8 model was synthesized. This time, however, the annotation was already done, thanks to a contributor of Roboflow, which provided a finished dataset (36) with the classes: "oline", "qb", "defense", "skill" and "ref" as seen in Figure 17. The class ref for referee was ignored. The other classes were sufficient for later extraction of formations and personnel.



Figure 17: Example of a training image for the player detector.

The dataset consists of 1454 annotated images. The following measures were taken for the augmentation Hue: Between -25° and $+25^\circ$, Saturation: Between -25% and $+25\%$, Brightness: Between -25% and $+25\%$, Exposure: Between -25% and $+25\%$, Blur: Up to 2.5 pixels, Noise: Up to 5% of pixels. After enhancement 2856 images were delivered. The data was divided into 91% training set, 8% validation set and 2% test set. Overall the class "qb" was obviously underrepresented, because only one quarterback is on the field at all times. As far as training goes a SGD optimizer was utilized, with a learning rate of 0.01, weight decay of 0.0005 and a momentum of 0.9. SGD is very efficient on large datasets and was well-suited for this architecture. Again data parallelism was applied using two NVIDIA Tesla T4s (15102MiB) within Kaggle. The training was carried out for 350 epochs, which is seven times as many as for the scoreboard model, because there are more classes, a more complex problem, and a

fundamentally larger dataset. The image was again resized to 640 x 640 pixels. The training was completed after 23.044 seconds or approximately 6 hours and 24 minutes. In Figure 18 the evaluation metrics can be seen over the course of 350 training epochs.

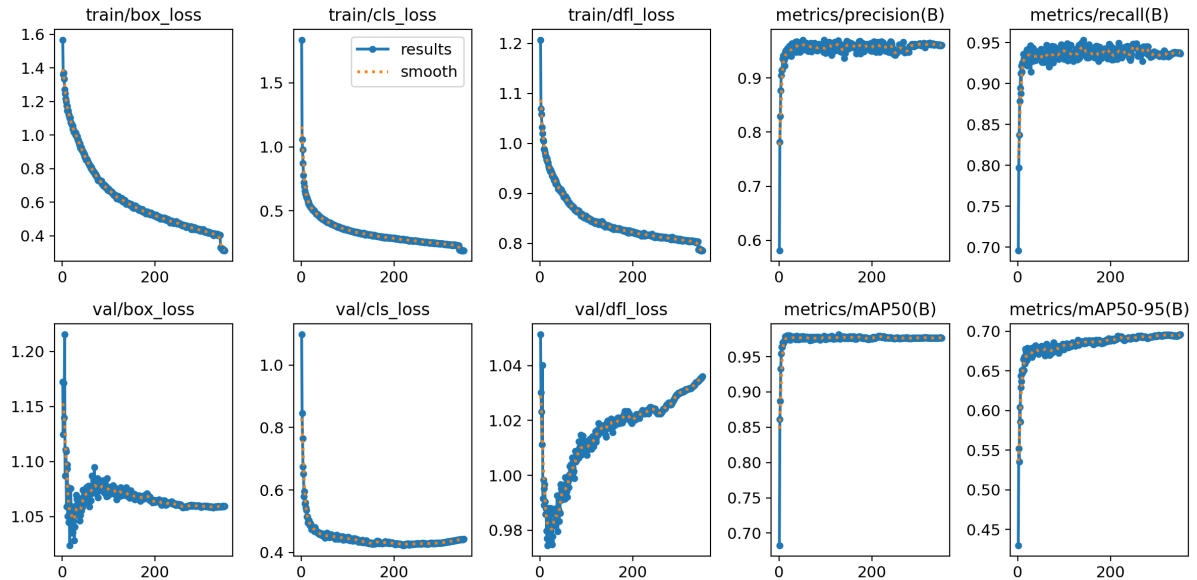


Figure 18: Player Detection YOLO model training results over 350 epochs.

A clear convergence close to zero of the loss functions can be seen throughout the training. Following scores are achieved after the 350th epoch:

- **Train/box_loss:** 0.31094
- **Train/cls_loss:** 0.18995
- **Train/df_l_loss:** 0.78536
- **Metrics/precision(B):** 0.96047
- **Val/box_loss:** 1.0593
- **Val/cls_loss:** 0.44273
- **Val/df_l_loss:** 1.036
- **Metrics/recall(B):** 0.93695
- **Metrics/mAP50(B):** 0.97673
- **Metrics/mAP50-95(B):** 0.69597

The metrics indicate strong performance for detection of players. The high precision and recall scores indicate accurate identification of players while minimizing false positives. The mAP50 score suggests overall effectiveness across various player classes, while the slightly lower mAP50-95 score indicates potential difficulty in detecting players under more challenging conditions, such as partial occlusion or distance.

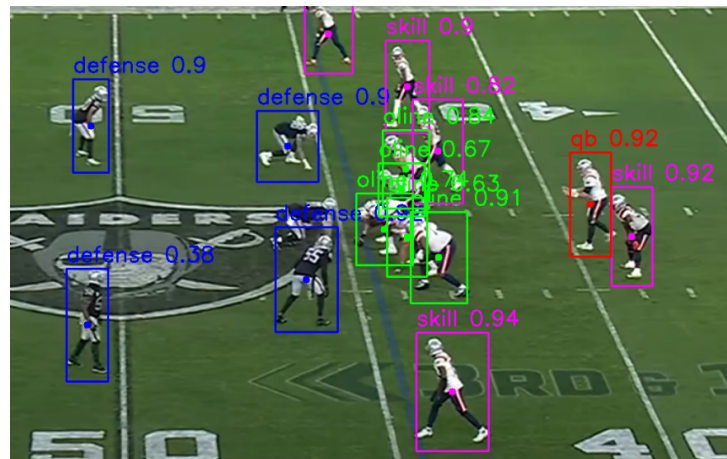


Figure 19: Example of player detection model on image.

Figure 19 is an example of the predicted players, their location in the image and the confidence levels. In summary, all players are correctly predicted with high confidence values. In particular, the O-line prediction has a lower confidence than the rest, indicating problems when players overlap in the image. Overall, the model shows very promising capabilities and fits our approach very well.

3.2.4 Line Detection using HoughLinesP Algorithm

After the players are detected and located on the image, we need to transform the side view of the image to a bird's eye view to be able to extract the final visual features. To achieve this transformation of the image, anchor points on the initial image must be determined to map to the target points on the final transformed image.

Given the fact that the yard lines of the field are always parallel and perfectly vertical in the bird's eye view, it is only logical that we extract the yard markers from the image for later use as anchor points for the homography transformation. The HoughLinesP algorithm in our pipeline does so. The yard lines are extracted in four steps as seen in the python code below: First the image is gray-scaled to simplify processing and a Gaussian blur is applied to reduce noise. Then the canny edge detector uses kernels to detect hard edges in the image. And finally the HoughLinesP algorithm is applied. It takes parameters for edge detection threshold, minimum line length, and maximum gap between line segments. The output is an array of lines represented by their endpoints.

```

1 # First step: Convert image to grayscale
2 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
3 # Second step: Apply Gaussian blur to reduce noise
4 blurred = cv2.GaussianBlur(gray, (5, 5), 0)
5 # Third: Apply Canny edge detection
6 edges = cv2.Canny(blurred, 50, 150)
7 # Fourth step: Apply Hough transform to detect lines
8 lines = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold=50, minLineLength=100,
    maxLineGap=10)

```

To ensure that only vertical lines are extracted, an angle interval from -26.6 degrees to +26.6 degrees is defined within which the lines must remain. Now that the lines have been filtered

using a threshold and an angle interval, they are grouped by their vector. The purpose of vector grouping is to eliminate smaller lines, false positives, and outliers. Only lines that belong to a certain vector group can be detected as yard lines. Also, only groups with a cumulative length of 80% of the height of the image are detected as yard lines to exclude false groups. All these measures result in a nearly flawless detection and localization of line markers as seen in Figure 20.

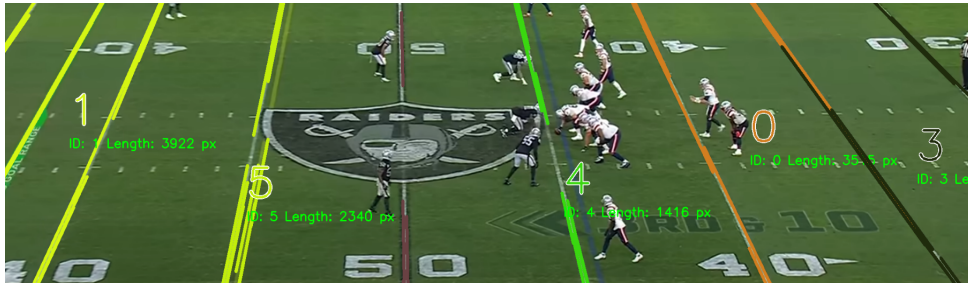


Figure 20: Example of line detection algorithm on image.

A side effect of this approach to yard line detection is that different yard lines with similar vectors are grouped together. This can be seen in group 1 in Figure 20. This effect may be unwanted, but it does not interfere with the development of the project, since we only need to know the location and direction of the yard lines, not which exact line it is. This algorithm saved the position and directions of the group for further processing in the homography algorithm of the pipeline.

3.2.5 Homography Transformation

Looking back at the architecture of the project, the homography transformation combines the results of the player detection model and the line detection algorithm. The detected lines act as anchor points for the transformation of the image, and the player's position is the actual information on which further visual features are extracted. Figure 21 shows the input information for the homography transformation.

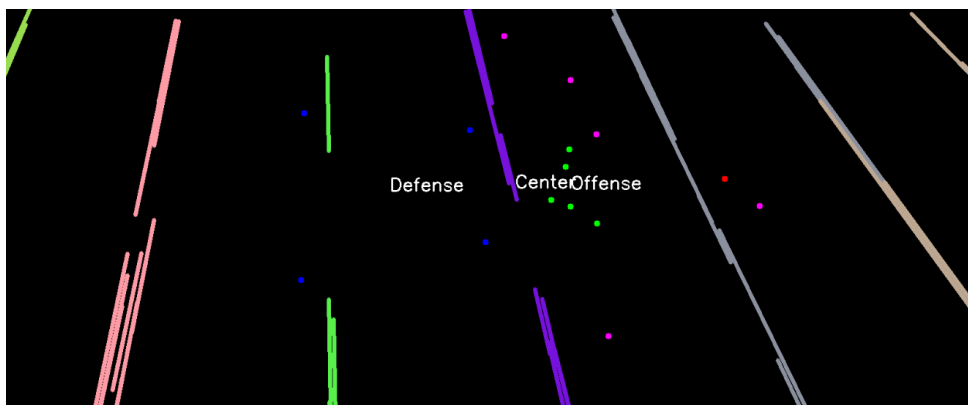


Figure 21: Detected players and lines combined.

Figure 21 shows the extracted lines and the positions of the players. The sides of offense and defense can be concluded by the position of players. Blue dots represent defense players, violet dots are skill players (running backs, wide receivers and tight ends), the red dot is the quarterback and the green dots show offensive linemen. There are five offensive linemen correctly classified. Three skill players on the right side of the field and one skill player on the left side. The correct assignment to Wide Receiver, Tight End and Running Back will be done later as the CV-model is not fit to do so.

So how can a bird's eye view be achieved using this input data? When looking at Figure 21 it can be observed, that the players all surround the line of scrimmage (purple line). Logically speaking, when we shift the lower X-axis of this image so that the line of scrimmage is perfectly vertical, the horizontal alignment of the players should shift to an approximate bird's eye view. This method is very primitive and the results are not perfect, but for our purpose the solution works quite well.

The following function "line_of_scrimmage(...)" takes the position of the center in line 7, which is the central position of the play. Then it finds the line, which is closest to the x position of the center in the for-loop from line 10 to line 22. Knowing that every yard line is extracted from the image and the center always lines up on the line of scrimmage, this function specifically always determines which line is closest to the center, and therefore the line of scrimmage.

```

1 def line_of_scrimmage(lines_map, keypoints):
2     # Initialize nearest x position to line of scrimmage
3     nearest_x = float("inf")
4     # Initialize nearest line group to line of scrimmage
5     nearest_group = None
6     # Extract center coordinates from keypoints
7     center_x, center_y = keypoints["center"][0], keypoints["center"][1]
8
9     # Iterate over extracted lines
10    for group, coordinates in lines_map.items():
11        # Extract top and bottom coordinates of line
12        point1, point2 = coordinates[0], coordinates[1]
13        # Calculate interpolated x-coordinate using linear interpolation
14        x_interp = point1[0] + (center_y - point1[1]) * (point2[0] - point1[0]) / (point2
15        [1] - point1[1])
16        # Check if interpolated x-coordinate falls within line segment
17        if point1[0] <= x_interp <= point2[0] or point2[0] <= x_interp <= point1[0]:
18            # Update nearest_x and nearest_group if closer line is found
19            if abs(x_interp - center_x) < nearest_x:
20                nearest_x = abs(x_interp - center_x) # Update nearest_x
21                nearest_group = group # Update nearest_group
22
23    return nearest_group # Return nearest_group

```

After having determined the line of scrimmage, we can apply the homography transformation using the function "transform_image(...)" below. This function reads the image from the path (line 3) and retrieves the line of scrimmage from the line_of_scrimmage(...) function in line 5. Then, it calculates the difference between the topmost x-axis position of the line of scrimmage and the bottom-most x-axis position of the line of scrimmage (line 10). This x-offset is subtracted from the bottom x-axis of the image to achieve a perfectly vertical line of scrimmage (line 16). The function uses the getPerspectiveTransform() method from OpenCV to determine the initial and target positions for the warping (line 18). Then the function warps the image with the warpPerspective(...) function from OpenCV as predetermined in line 20. Finally the image is saved as "e_transformed.png" in line 22.

```

1 def transform_image(image_path, image_id, lines_map, key_positions, players_list):
2     # Read image
3     img = cv2.imread(image_path)
4     # Get nearest group using line_of_scrimmage() function
5     nearest_group = line_of_scrimmage(lines_map, key_positions)
6     # Extract top and bottom x-coordinates of line of scrimmage
7     ng_top_x = lines_map[nearest_group][0][0]
8     ng_bottom_x = lines_map[nearest_group][1][0]
9     # Calculate difference in x-coordinates for shifting
10    x_diff = ng_top_x - ng_bottom_x
11    # Get image dimensions
12    IMAGE_H = img.shape[0]
13    IMAGE_W = img.shape[1]
14    # Define source and destination points for perspective transformation
15    src = np.float32([(0, 0), (IMAGE_W, 0), (0, IMAGE_H), (IMAGE_W, IMAGE_H)])
16    dst = np.float32([(0, 0), (IMAGE_W, 0), (0 + x_diff, IMAGE_H), (IMAGE_W +
17    x_diff, IMAGE_H)])
18    # Compute perspective transformation matrix
19    M = cv2.getPerspectiveTransform(src, dst)
20    # Perform perspective transformation
21    warped_img = cv2.warpPerspective(img, M, (IMAGE_W, IMAGE_H))
22    # Write transformed image to file
23    cv2.imwrite(f"predict/{image_id}/e_transformed.png", warped_img)

```

The results of the whole algorithm can be seen in Figure 22. As described, the orientation of the line of scrimmage and the players around it has been transformed to be perpendicular to the camera view.

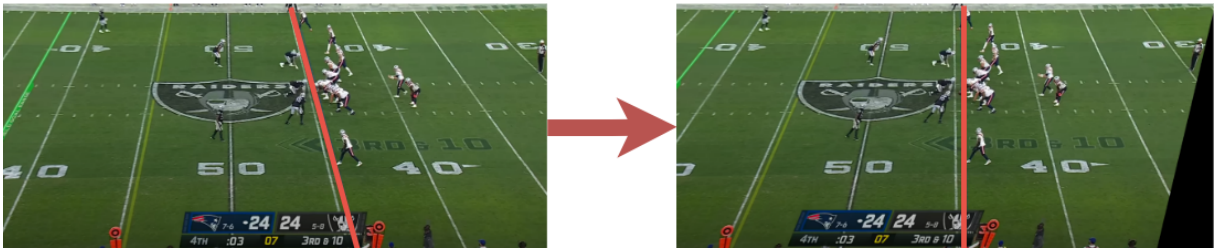


Figure 22: Homography transformation example on image.

Further transformations could be performed, such as vertical rectification, to achieve a true bird's eye view. However, as far as this algorithm goes, it is sufficient to extract the final visual features.

3.2.6 Visual Feature Extraction

This algorithm is the last and final one in the computer vision-based part. Its goal is to determine final visual features. Shown in Figure 23 is the information given to this algorithm by the homography transformation algorithm. The image shows the corrected positions of the players. The offense is placed on the right side of the image and the defense to the left.

be wiser to pass the ball because there are fewer defensive players covering the wide receivers. To generalize the approach of constructing the box, it was calculated relative to the size of the O-line box. To retrieve the feature defenders in box the detected defensive players are counted, which are located in this area. Two defenders in box are counted in Figure 24

The yellow rectangle is called the "DL-Box". When defensive players are located in this area, they are labeled as defensive linemen. Next to the DL-Box is the "LB-Box" in purple. All players located in this area are labeled as Linebackers. All defensive players, that are not in the DL-Box or LB-Box are automatically labeled as defensive backs or "DB". By combining this information, we can determine the feature "defense personnel". In Figure 24 two defensive linemen and two linebackers can be determined. Knowing that there must always be eleven defensive players on the field, we conclude that there are $11-2-2=7$ defensive backs on the defense. Why does this calculation make sense? Often the defensive backs are not all captured in the picture because they are too spread out. However, linebackers and especially defensive linemen are captured most of the time because they are closer to the center of the field. Therefore, assigning the rest of the eleven players to defensive backs is only logical and the best solution if not every player is captured.

As of now we have assigned following features: defenders in box and defense personnel. Still remaining are the features: shotgun snap, offense personnel and offense formation.

Looking at the offensive side (right side) of the image, we can see five offensive linemen, five skill players, and one quarterback. To differentiate between tight ends, wide receivers, and running backs, we look at their location. Skill players relatively close to the O-line are labeled tight ends. Skill players vertically leveled with the O-line (in the green "R-Box"), but not close are assigned wide receivers. Skill players, who are relatively distant in relation to the x-axis (or not in the green "R-Box"), can only be running backs. This method of exclusion criteria works promisingly well when distinguishing between tight ends, wide receivers, and running backs. With this information the feature offense personnel can be determined. In Figure 24 there are three wide receivers, one tight end and one running back.

To identify if a shotgun snap, we measure the relative distance of the quarterback to the O-line. If a threshold is surpassed and the quarterback's distance is too far from the O-line, the snap is classified as a shotgun snap. Finally, we must extract the formation of the offense. There are five possible formations we can extract: I-Formation, Empty, Pistol, Singleback, and Jumbo. All formations were previously explained in Chapter 1.1 "The sport of American Football". "Terminology and game concept". To determine the formation, an exclusion method is applied. First of all, a "Jumbo line" is constructed, as seen in Figure 24. This line has the purpose of constraining the width of the Jumbo formation. If a single skill player exceeds the top or the bottom of the Jumbo line, the Jumbo formation is ruled out. After excluding this formation, the following code below is applied to methodically identify the correct formation.

```

1 if Jumbo==False:
2     if rbs>=2:
3         offense_formation="I_FORM"
4     if rbs==0:
5         offense_formation="EMPTY"
6     if rbs==1:
7         for player in offense_personnel:
8             if player[0]=="RB":
9                 if abs(player[1][1]-qb[1][1])<30:
10                    offense_formation="PISTOL"
11                else:
12                    offense_formation="SINGLEBACK"
```


The code implements following semantic: If more than two running backs are on the field it is an I-Formation. If no running back is on the field it is an Empty formation. If one running back is on the field, the algorithm determines if the running back stands within 30 pixels regarding the x-axis. If so, the formation is Pistol. If the running back does not stand within that range the formation is Singleback. The formation detected in Figure 24 is Singleback, which is correct.

To conclude the computer vision-based section, we have successfully extracted the visual features yards to go, down, seconds remaining in the quarter, score differential, offense formation, offense personnel, defenders in the box, defense personnel, and shotgun. These visual features are the basis for the text-based model to predict a pass or run. We have utilized two neural networks and four algorithms so far. We have applied computer vision techniques to read text, detect vertical lines, transform the perspective of images, and extract visual features based on relative distances.

3.3 Text-Based Section

This is the second part of our pipeline. In the first, computer vision-based part, we extracted visual features. In order to predict game type using these features, a separate neural network must be constructed. This section summarizes the acquisition and collection of play-by-play data, the preprocessing and cleaning of this data, the feature importance in relation to our classification problem, the visualization of feature correlations, and finally the construction of a text-based neural network.

3.3.1 Data Acquisition and Collection Methods

The main objective of this model is to generate a prediction of what type of play is most probable, based solely on features we have extracted before the actual play occurs. Therefore, it is critical to collect data that is quantitatively and qualitatively fit for the best possible outcome.

But which variables are actually important to predict play types in American Football plays? Let's investigate further on this question with a specific example.

Suppose the Miami Dolphins want to score a touchdown ten yards short of the end zone. Miami's quarterback Tua Tagovailoa has an average of 8.5 yards per pass in the 2023 season. Their leading running back De'Von Achane has averaged 12.1 yards per rush/run (37). So intuitively, running the ball seems like the more rewarding decision. Of course, this example is highly simplified. Deciding which play type to run involves many more variables than average yards per run or pass.

So concerning our question, which variables are important to make a prediction, we can only conclude: Whatever influences a logical decision. So we need to gather a dataset with a significant amount of data and features, and then select the most important features for training.

The data used in this model was collected using the official Nflverse Github page. Nflverse is a collection of data to improve data analysis surrounding American football. For our purposes, we used the play-by-play (pbp) and play-by-play participation (pbp_participation) datasets in the Github collection (38).

Both datasets provide essential information. They both use the same primary keys and plays

but different features, so we merged them into one large dataset. This dataset consists of approximately 362,000 plays from the 2016 to the 2023 NFL season.

Each play is characterized by 384 features, including the time to play, yards to go or location on the field, to name some of them.

Of course, many of those 384 variables have little or zero predictive value. Therefore, we have to process them in the next step.

3.3.2 Data Preprocessing and Cleaning

Preprocessing and cleaning the data before training a neural net is critical to the accuracy of the net. Our data consists of a lot of missing, false or unnecessary values.

False Data: Starting off, we need to fix a small but detrimental error in the data. Our data contains 34 team names, although only 32 teams play in the NFL. The names in the dataset are abbreviations of the city names where the NFL teams are located, such as "JAX" for Jacksonville. Because two teams have moved since 2016, we need to unify the data from seemingly four teams in our dataset to the actual two. This affects the Chargers, who moved from San Diego (SD) to Los Angeles (LA) and the Raiders, who moved from Oakland (OAK) to Las Vegas (LV). To solve this problem, we simply convert the team names from the city name to the actual name of the organization. The scheme can be seen in Table 2.

Old name	New name
JAX	Jacksonville
SD	Chargers
LA	Chargers
OAK	Raiders
LV	Raiders
...	...

Table 2: Transforming team names in dataset.

Noisy Data: Initially, all rows where the play type was not "pass" or "run" were removed. Other play types such as "kickoff", "extra point" or "punt" are not important for this research. This step also eliminated a lot of rows with missing or incorrect data, because the permitted values were clearly specified.

The same exclusion method was used with the formation of the offense. The dataset includes following formations: Singleback, Shotgun, Jumbo, I-Formation, Wildcat, Pistol and Empty. Because the formation of the offense is a key feature in our prediction, all rows with other values were excluded. In addition, we concluded that the Wildcat formation could not be extracted visually because in this formation another player is positioned as the quarterback and an image is not enough to tell if another player is lined up as the quarterback. This would require either manual input or identification of player numbers. So the Wildcat formation was also excluded.

Data Bucketing: Data bucketing is the process of aggregating and organizing data points into discrete groups or "buckets" based on specific criteria or attributes. This process reduces

the dimensionality of the data set. The cleaning algorithm for the dataset analyzed the features "yardln", which is the current yard line and the teams side before the play (f.E. JAX_23), and "posteam", which is the possessive team (f.E. JAX). These two features indicate where the play is currently located. To reduce dimensionality, these two features have been combined or "bucketed" into a single feature that has only the values "own_bluezone", "own_redzone", "opp_bluezone", and "opp_redzone". The red zone is the area within 20 yards of the end zone. The code below accomplished this data bucketing.

```

1 def classify_zone(row):
2     team, yardline = row['yardln'].split()
3     if row['posteam'] == team:
4         return "own_redzone" if int(yardline) <= 20 else "own_bluezone"
5     else:
6         return "opp_redzone" if int(yardline) <= 20 else "opp_bluezone"
7
8 df['yardln'] = df.apply(classify_yardline, axis=1)

```

Data Reduction: Our dataset consists of many unnecessary columns, such as "kick_distance". This feature only holds a value if the play type is a kickoff or punt. Since we will not be using this data for our prediction, it unnecessarily increases the dimensionality of our data. All columns, which were of low value for our context were excluded.

Missing Value Imputation: Imputation of data is necessary for further processing. When looking at our data, it seems unwise to use mean imputation or any other statistical imputation. Any statistical imputation would destroy important correlation and covariant variables. So the cleaning algorithm just imputed the missing values with "NaN" or "Not a number".

Feature Engineering: The art of feature engineering involves creating new features or modifying existing ones to improve model performance or interpretability. This method was applied to further expand our dataset. The following code shows the scraping of team specific data from nfl.com/stats. To to remain congruent the data was only extracted from 2016 to 2023 (2024 in python code because the last argument is excluded in the range method). The scraping is performed using BeautifulSoup from the python module bs4 and the module requests. The code finds a specific html class "d3-o-table" of the table (line 7) and extracts the data. After caching the data in a dictionary (line 13-22), the data is accumulated for the year (line 24) and finally saved in a json file (line 28,29).+

```

1 for year in range(2016,2024):
2     url = "https://www.nfl.com/stats/team-stats/offense/rushing/"+str(year)+"/reg/all"
3     response = requests.get(url)
4     if response.status_code == 200:
5         html_content = response.text
6         soup = BeautifulSoup(html_content, "html.parser")
7         table = soup.find("table", {"class": "d3-o-table"})
8         all_data = {}
9         for i, row in enumerate(table.find_all("tr")):
10            cells = row.find_all(["th", "td"])
11            row_data = [cell.text.strip() for cell in cells]
12            if row_data and i != 0:
13                row_dict = {
14                    "run_Att": int(row_data[1]),
15                    "run_Yds": int(row_data[2]),

```

```

16         "run_YPC": float(row_data[3]),
17         "run_TD": int(row_data[4]),
18         "run_20+": int(row_data[5]),
19         "run_40+": int(row_data[6]),
20         "run_to_1st_pct": float(row_data[9]),
21         "run_FUM": int(row_data[10])
22     }
23     all_data[row_data[0].split("\n")[0]]=row_dict
24     rush_json[str(year)]=all_data
25 else:
26     print("Error while scraping: Statuscode:", response.status_code)
27
28 with open("./rushing_stats.json", "w") as json_file:
29     json.dump(rush_json, json_file, indent=2)

```

This Python code only displays the extraction of rushing-specific data. The same scraping method was also applied to passing data. The entire scraped data represents team specific rushing and passing stats over the year. For example, how many rushing touchdowns the team achieved this year. This data could be important for finding preferences in play-calling by specific teams. After scraping the new data, it was assigned correctly to the corresponding play. The following code selected running and passing stats for the specific team in the year of the play for every row in our data set.

```

1 def extract_passing_stats(row):
2     year = str(row['season'])
3     team = row['possession_team']
4     return passing_stats[year][team]
5
6 def extract_running_stats(row):
7     year = str(row['season'])
8     team = row['possession_team']
9     return rushing_stats[year][team]

```

Splits: Before proceeding any further, we will split the dataset into two parts with an 80/20 split. The first part will be processed and visualized. The second part will be our validation data and will remain untouched until the neural net is built and testing/validation begins. Following Python code snippet does so:

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3
4 def split_data():
5     df = pd.read_csv('cleaned_file_all.csv')
6     train_test_df, validation_df = train_test_split(df, test_size=0.2,
7     random_state=42)
8     header = df.columns
9     train_test_df.columns = header
10    validation_df.columns = header
11    train_test_df.to_csv('cleaned_file_train_test.csv', index=False)
12    validation_df.to_csv('cleaned_file_validation.csv', index=False)

```

Using the "train_test_split" function by Sklearn the dataset is split, while also ordering it in a random sequence. The train/test split will be conducted later on.

This step completes our preprocessing.

We have dramatically reduced the dimensionality of our dataset, fixed false, noisy, and missing data, and engineered new features. At this point, the whole dataset consists of 252.360

plays/rows and 37 features/columns excluding the target class. Table 3 gives a glimpse of what the dataset looks like after cleaning.

possession_team	offense_formation	offense_personnel	...	play_type
Broncos	SINGLEBACK	1_RB_1_TE_3_WR	...	pass
Broncos	I_FORM	2_RB_0_TE_2_WR	...	pass
...

Table 3: Text dataset after cleaning.

3.3.3 Data Analysis and Visualization

In the previous step the dataset was collected, repaired, enhanced, but also dramatically reduced. It consists of 37 features. Every feature has a certain impact on the target class. To achieve a comprehensive analysis of our data, distinct feature selection methods are applied: Filter, Wrapper and Embedded methods. Each method has unique advantages to select relevant features.

Filter methods	Wrapper methods	Embedded methods
Generic set of methods which do not incorporate a specific machine learning algorithm .	Evaluates on a specific machine learning algorithm to find optimal features.	Embeds (fix) features during model building process . Feature selection is done by observing each iteration of model training phase.
Much faster compared to Wrapper methods in terms of time complexity	High computation time for a dataset with many features	Sits between Filter methods and Wrapper methods in terms of time complexity
Less prone to over-fitting	High chances of over-fitting because it involves training of machine learning models with different combination of features	Generally used to reduce over-fitting by penalizing the coefficients of a model being too large.
Examples – Correlation, Chi-Square test, ANOVA, Information gain etc.	Examples - Forward Selection, Backward elimination, Stepwise selection etc.	Examples - LASSO, Elastic Net, Ridge Regression etc.

Figure 25: Comparison between Filter, Wrapper and Embedded Methods (8)

Figure 25 describes the difference between Filter, Wrapper and Embedded methods to determine feature importance. Filter methods evaluate features independently of any machine learning algorithm. They are generally faster compared to wrapper methods. Wrapper methods incorporate a specific machine learning algorithm. They involve machine learning models with different combination of features, which leads to high computation time, specifically for big datasets. Embedded methods embed or fix features during the model building process. The feature selection is done by observing each iteration of the training phase. They are typically faster than Wrapper methods, but slower than Filter methods (8). To determine, which features are most important in our dataset one algorithm is applied for each feature selection method. All three algorithms were executed with the help of machine learning platform Weka.

Filter Methods: The algorithm used for filter methods is information gain. Information gain is a measure that quantifies the amount of information obtained about a target variable by

observing a specific feature. Features with higher information gain are considered more relevant for prediction tasks as they provide more valuable information about the target variable. When the information gain algorithm is applied to our dataset, following results can be observed in Table 4. The most important feature is offensive formation, followed by shotgun, which is also a type of formation. The top ten ranked features are also those which would be considered in a logical decision. The statistical features, which were engineered previously for the team's pass and run-specific stats, appear to have little to no impact at all.

Attr. Name	Score	Rank
offense_formation	0.149822	1
shotgun	0.110484	2
defenders_in_box	0.093018	3
defense_personnel	0.063795	4
offense_personnel	0.056627	5
ydstogo	0.043291	6
down	0.035373	7
score_differential	0.014029	8
half_seconds_remaining	0.013516	9
quarter_seconds_remaining	0.007672	10
yardline_100	0.005238	11
run_Att	0.003894	12
run_Yds	0.003446	13
pass_Att	0.003428	14
no_huddle	0.003222	15
possession_team	0.003203	16
qtr	0.00274	17
yrdln	0.002252	18
drive	0.001804	19

Attr. Name	Score	Rank
defteam	0.001451	20
pass_Yds	0.001277	21
run_20+	0.001234	22
run_TD	0.001178	23
run_FUM	0.000855	24
run_to_1st_pct	0.000701	25
pass_20+	0.000632	26
pass_TD	0.00061	27
pass_40+	0.000472	28
run_40+	0.000419	29
run_YPC	0.000353	30
season	0	31
pass_Rate	0	32
pass_Sck	0	33
pass_to_1st_pct	0	34
pass_INT	0	35
pass_Cmp_pct	0	36
pass_Yds_per_Att	0	37

Table 4: Information Gain Ranking of remaining features.

Wrapper Methods: Wrapper methods utilize the Particle Swarm Optimization (PSO) algorithm for feature selection. The implementation of PSO for Weka, a popular machine learning tool, can be obtained from the provided reference (39). The Weka PSO implementation helps evaluate and select feature subsets by optimizing performance metrics. This enhances the efficiency and effectiveness of feature selection within the Wrapper method framework. After running the PSO for 20 generations on our dataset, the results are presented below in Table 5.

Merit	Scaled	Subset	Selected attributes	Nr.
0.10806	0.10129	1 2 5 14 15 22	possession_team	1
0.07977	0.03012	2 5 14 15 24	offense_formation	2
0.12119	0.13432	2 5 9 14 15	offense_personnel	3
0.12545	0.14504	2 5 14 15	defteam	4
0.11648	0.12247	2 5 14 15 36	defenders_in_box	5
0.12545	0.14504	2 5 14 15	defense_personnel	6
0.12545	0.14504	2 5 14 15	yardline_100	7
0.07977	0.03012	2 5 14 15 29	quarter_seconds_remaining	8
0.10594	0.09595	2 5 8 11 14 15 36	half_seconds_remaining	9
0.12545	0.14504	2 5 14 15	drive	10
0.07772	0.02497	2 5 12 14 15 29	qtr	11
0.0678	0	2 4 5 14 15 24 34	down	12
0.12545	0.14504	2 5 14 15	yrdln	13
0.12545	0.14504	2 5 14 15	ydstogo	14
0.12545	0.14504	2 5 14 15	shotgun	15
0.11724	0.12439	2 5 14 15 19	no_huddle	16
0.12202	0.13641	2 3 5 14 15	score_differential	17
0.12545	0.14504	2 5 14 15	season	18
0.12545	0.14504	2 5 14 15	pass_Att	19
0.12545	0.14504	2 5 14 15	pass_Cmp_pct	20
0.12545	0.14504	2 5 14 15	pass_Yds_per_Att	21
0.11724	0.12439	2 5 14 15 19	pass_Yds	22
0.12202	0.13641	2 3 5 14 15	pass_TD	23
0.12545	0.14504	2 5 14 15	pass_INT	24
0.12545	0.14504	2 5 14 15	pass_Rate	25
0.12545	0.14504	2 5 14 15	pass_to_1st_pct	26
0.12545	0.14504	2 5 14 15	pass_20+	27
0.12545	0.14504	2 5 14 15	pass_40+	28
0.12545	0.14504	2 5 14 15	pass_Sck	29
			run_Att	30
			run_Yds	31
			run_YPC	32
			run_TD	33
			run_20+	34
			run_40+	35
			run_to_1st_pct	36
			run_FUM	37

Table 5: PSO Search Algorithm after 20 Generations.

The table shows the merit and the scaled performance and their corresponding subset. Merit typically refers to the evaluation criterion used to assess the quality or importance of a particular feature subset. The scaled column represents the scaled or normalized merit scores of the feature subsets. In summary 20 subsets were selected. The features "offense_formation" and "defenders_in_box", "ydstogo", and "shotgun" were used in every subset. In summary only 17 of 37 features were used. The best outcome was achieved, when using features: 2, 5, 14 and 15.

Embedded Methods: As mentioned earlier, embedded methods include techniques that incorporate feature selection directly into the model training process. A popular example of an embedded method is the J48 algorithm, also known as the C4.5 decision tree algorithm. The J48 algorithm evaluates the importance of each feature during the construction of the decision tree. It selects the best features to split the data based on criteria such as information gain or Gini impurity. Features that lead to the most effective splits, resulting in better classification or regression performance, are considered more important by the algorithm. The following decision tree, shown in Figure 26, was constructed by a J48 implementation of Weka.

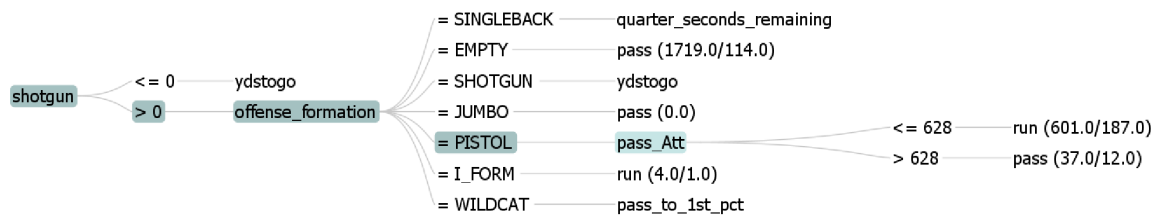


Figure 26: Small cutout of decision tree of J48 algorithm.

For the first data split, the algorithm chooses the shotgun feature, which was also highly ranked by the Wrapper and Filter algorithms before. Then it is divided by offensive formation for shotgun snaps and yards to go for no shotgun snaps. The tree shows similar feature importances as the algorithms before.

This next table illustrated in Table 6 shows the evaluation method corresponding to the above decision tree.

Summary	
Correctly Classified Instances	14355 (71.775%)
Incorrectly Classified Instances	5645 (28.225%)
Kappa statistic	0.416
Mean absolute error	0.3655
Root mean squared error	0.4553
Relative absolute error	75.7158%
Root relative squared error	92.68%
Total Number of Instances	20000

Detailed Accuracy By Class								
Class	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
pass	0.760	0.343	0.763	0.760	0.761	0.416	0.739	0.760
run	0.657	0.240	0.652	0.657	0.655	0.416	0.739	0.605
Weighted Avg.	0.718	0.301	0.718	0.718	0.718	0.416	0.739	0.697

Confusion Matrix		
a	b	Classified as
9007	2850	a = pass
2795	5348	b = run

Table 6: Classification report of J48 algorithm

The first subtable "Summary" of Table 6 presents the overall statistics of the model's performance, such as correctly and incorrectly classified instances, kappa statistic, mean absolute error, root mean squared error, relative absolute error and root relative squared error. With a correct classification rate of 71.775%, the model's performance appears moderately effective. However, the relatively high rate of incorrectly classified instances at 28.225% suggests room for improvement in its accuracy.

The second subtable "Detailed Accuracy By Class" shows detailed evaluation metrics of the classes pass and run and their average. The detailed accuracy by class table shows that the model performs better in classifying passes (TP Rate: 76.0%, F-Measure: 0.761) compared to runs (TP Rate: 65.7%, F-Measure: 0.655), indicating a tendency to better distinguish between passing plays than running plays. However, the weighted average metrics indicate overall moderate performance across both classes, with room for improvement in precision and recall.

The Confusion Matrix, which is the third subtable, shows the total absolute values of TP, TN, FP, and FN.

Now that we have measured the importance of the features in our dataset, we want to explore the correlations between the most important features and the target class. To do this, we will create visualizations using the data visualization tool Tableau (40). The following four graphs show the relationship between the features: Defenders in Box, Offensive Formation, Shotgun, and Yards to Go to Target Class play type. The specific attribute is always plotted on the x-axis and the amount of the specific play type is shown on the y-axis. The data originates from the 2023 season.

Defenders In Box x Play Type

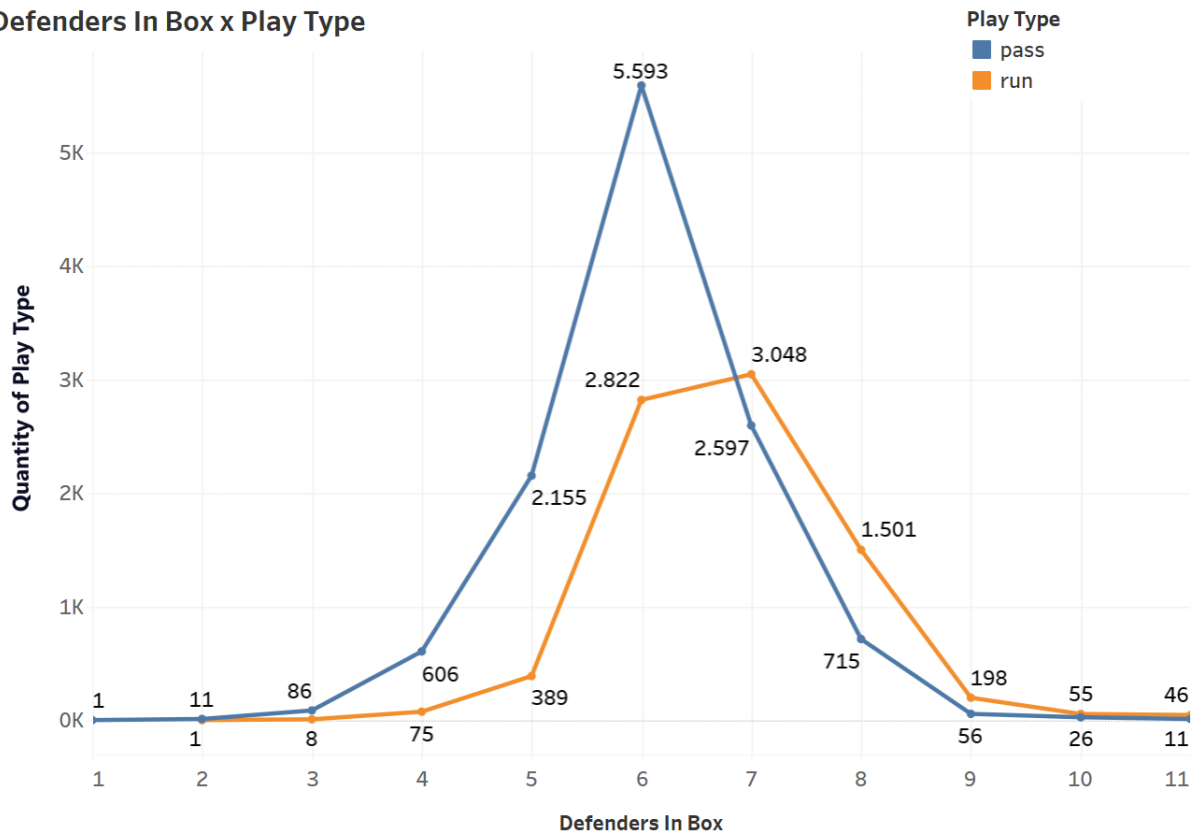


Figure 27: Relationship between Play Type and Defenders in Box.

This first graph (Figure 27) shows the relationship between defenders in box and play type. It can be observed that most most of the time there are four to nine defenders in the box. More passes than runs are carried out. What is interesting about this graph is that the shape of the pass and run lines are very similar, but the course of the run line is shifted slightly to the right, compared to the pass line. This indicates that more defenders will line up in the box before a run play occurs.

Offensive Formation x Play Type

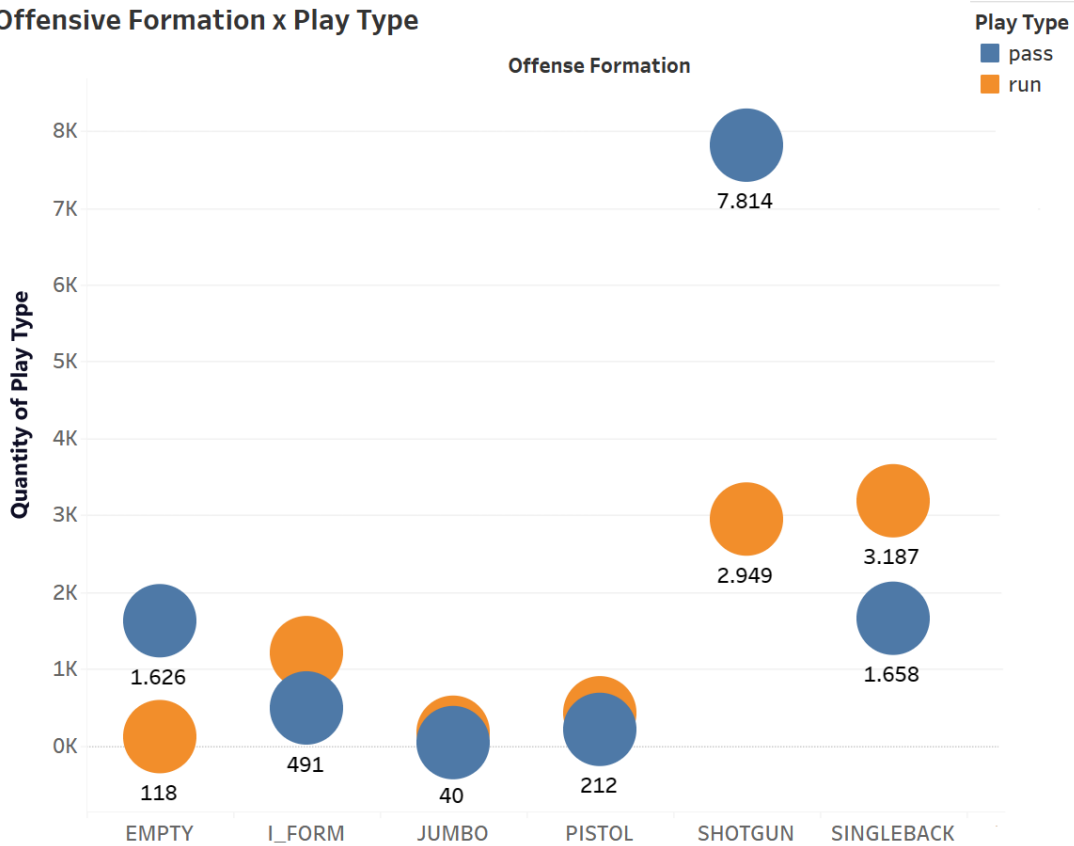


Figure 28: Relationship between Play Type and Offensive Formation.

This next graph depicted in Figure 28 shows the correlation between the offensive formation and the play type. The five formations Empty, I-Formation, Jumbo, Pistol, Shotgun, and Singleback are lined up on the x-axis. What's very interesting about this visualization is that 72.6% of the time a shotgun formation is played, the play turns out to be a pass. On the other hand, 65.7% of Singleback formation plays are run plays. The most remarkable correlation seems to be in the formation Empty, where 93.2% of plays are passes. This seems quite logical because no running back stands in the backfield on an Empty formation.

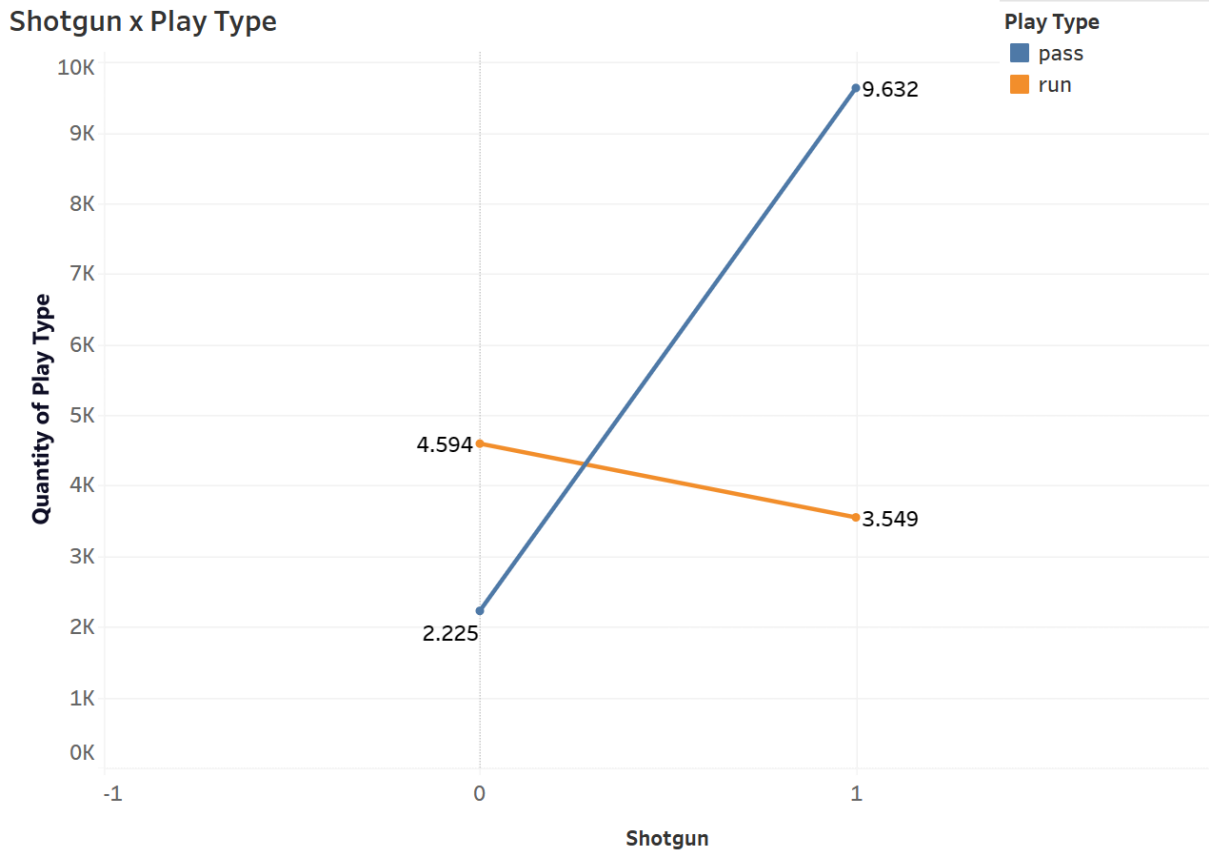


Figure 29: Relationship between Play Type and Shotgun.

Figure 29 illustrates the feature shotgun, which was the first to be selected by the J48 algorithm earlier. The feature shotgun indicates if the ball is directly handed to the quarterback by the center or snapped backwards in a traditional way. When the ball is snapped in the shotgun, the quarterback usually has more time to act and better sight of the field. For passes shotgun snaps are more logical, as the quarterback has more time to throw and a better view of the field. Figure 29 also supports this assumption. When the ball is snapped via shotgun (shotgun=1), 73% of the time a pass is executed, as opposed to only 32.6% when the ball is handed directly to the quarterback (shotgun=0). This shows a clear correlation between the shotgun feature and the play type.

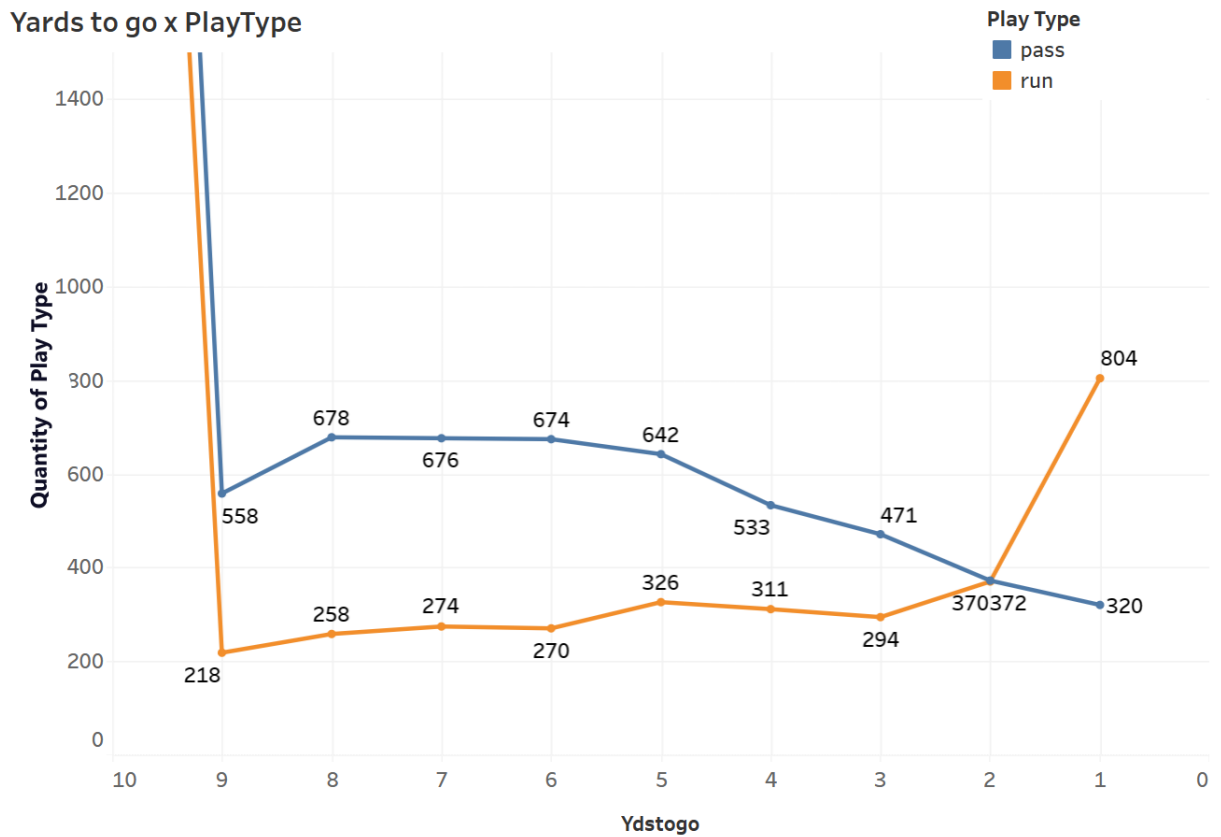


Figure 30: Relationship between Play Type and Yards to go.

Figure 30 visualizes the yards to go in relation to the play type. For clarity, the spike in play count at ten yards to go results from each first down starting at ten yards to go. What stands out is that as the course approaches one yard, the probability of a run increases. This seems very logical as short yardage can easily be achieved with run plays. This graph indicates a correlation between yards to go and the type of play carried out.

3.3.4 Model Selection

Now that the most important features for our prediction have been determined, we need to select a fitting model. The suitability of the following models and feature sets shown in Table 7 was tested.

No.	Classifier	Hyperparameters	Data
1	Random Forest	batch_Size=2048 min_Samples=1 n_trees=100	37 Features 201.888 Instances
2	Random Forest	batch_Size=2048 min_Samples=1 n_trees=100	10 (best) Features 201.888 Instances
3	SVM	batch_size=2048 C=1 kernel=linear $\epsilon = 0.001$	37 Features 201.888 Instances
4	SVM	batch_size=2048 C=1 kernel=linear $\epsilon = 0.001$	10 (best) Features 201.888 Instances
5	XGBoost	batch_size=2048 $\alpha = 0.1$ max_depth=3 n_estimators=100	37 Features 201.888 Instances
6	XGBoost	batch_size=2048 $\alpha = 0.1$ max_depth=3 n_estimators=100	10 (best) Features 201.888 Instances

Table 7: Trained Text based Models.

Table 7 shows six experimental setups. For number one and two the suitability of the Random Forest model was tested with 37 (all) features in No. 1 and only the ten best features in No. 2. Both models had a batch size of 2048, minimum leaf size of 1 and 100 trees. The test also included the Support Vector Machine model with all features (No. 3) and ten best features (No. 4). These two models also had a batch size of 2048, C value of 1, a linear kernel and ϵ of 0.001. Finally the suitability of the XGBoost model with all features (No. 5) and ten best features (No. 6) was tested. The XGBoost models were tested also with a batch size of 2048, learning rate of 0.1, maximum depth of 3 and 100 estimators. Keep in mind that this section only determines the best model architecture or our model and not the best hyperparameters for each model. The following Table 8 shows the results for each model with either 37 (all) features in the first subtable and the 10 best features in the second subtable. All models were trained in the Weka environment for testing with all 252.360 plays/rows of our dataset.

Testing on models with 37 (all) Features			
	SVM	Random Forest	XGBoost
TP-Rate (Recall)	0.731	0.704	0.746
FP-Rate	0.298	0.330	0.273
Precision	0.729	0.701	0.746
F-Measure	0.729	0.701	0.746
MCC	0.439	0.381	0.475
ROC-Area	0.716	0.760	0.822
PRC	0.667	0.744	0.819
ACC	73.12%	70.4%	74.64%

Testing on models with 10 (best) Features			
	SVM	Random Forest	XGBoost
TP-Rate (Recall)	0.728	0.692	0.739
FP-Rate	0.302	0.345	0.283
Precision	0.726	0.688	0.738
F-Measure	0.726	0.689	0.738
MCC	0.433	0.355	0.458
ROC-Area	0.713	0.744	0.809
PRC	0.664	0.720	0.802
ACC	72.84%	69.21%	73.86%

Table 8: Text models performance metrics on test dataset using 10-fold-cross validation.

The results presented in Table 8 indicate that overall the prediction accuracy lies between 69.21% by ten feature Random Forest and 74.64% by all feature XGBoost. These accuracies are by no means perfect, but knowing that predicting a play in the NFL is never certain, these results are actually quite good and by far better than coincidence. Each model does in fact perform a little better with all features in the training process. The best fit model for our dataset is XGBoost. XGBoost models typically exhibit higher prediction accuracy than Random Forest and SVMs. This model architecture is capable of capturing complex relationships within data. Based in the results we will select the XGBoost architecture for the approach. Since we extracted only the top 10 features from the image in the computer vision section earlier, we will only be able to train the XGBoost model with those features.

3.3.5 Hyperparameter Selection

After having selected XGBoost as the best fit model for our dataset, we will optimize the hyperparameters to improve the results of the model. The XGBoost model trained with the 10 best features achieved an accuracy of 73.86% as seen in Table 8. The goal of this hyperparameter optimization is to improve this accuracy and the overall performance of the model. To realize the hyperparameter optimization, the grid search algorithm "GridSearchCV" is

used. This algorithm uses cross-validation, hence the name "GridSearchCV," to systematically explore a pre-defined hyperparameter grid and determine the optimal combination of hyperparameters that produces the best model performance.

Following Python code provides a parameter grid with the hyperparameters maximum depth, quantity of estimators, and the regularization alpha value. Then it initializes an XGBClassifier object with specified settings such as the objective function, booster type, evaluation metric, and learning rate (line 9). After that a GridSearchCV object with the initialized XGBClassifier and a parameter grid is created (line 12). In line 15 the GridSearchCV performs an exhaustive search over the specified hyperparameter grid, using cross-validation to evaluate each combination of hyperparameters. The results are finally printed out in line 19.

```

1 # Hyperparameter Grid
2 param_grid = {
3     'max_depth': [3, 5, 7],
4     'n_estimators': [100, 200],
5     'reg_alpha': [0, 0.1]
6 }
7
8 # Initialize XGBClassifier
9 xgb_classifier = XGBClassifier(objective='binary:logistic', booster='gbtree',
10     eval_metric='logloss', learning_rate=0.1)
11
12 # Create the GridSearchCV object
13 grid_search = GridSearchCV(estimator=xgb_classifier, param_grid=param_grid, cv
14     =5, scoring='accuracy')
15
16 # Perform Grid Search
17 grid_search.fit(X_train, y_train)
18
19 # Show Results
20 print("Best Hyperparameters:")
21 print(grid_search.best_params_)

```

Following results are logged in the console:

```

1 $ Best Hyperparameters:
2 $ {'max_depth': 7, 'n_estimators': 100, 'reg_alpha': 0.1}

```

Listing 1: Grid-Search Console Output

Resulting from the grid search the best hyperparameters are a max depth of 7, 100 estimators and regularization alpha value of 0.1. These hyperparameter will be used to train the XGBoost model.

3.3.6 Training

The code below shows the use code to train the final XGBoost model, which will be deployed to our pipeline. The code uses pandas to read the CSV-file "cleaned_file_train_test.csv", which is the cleaned and preprocessed dataset for training. The LabelEncoder from sklearn.preprocessing encodes the selected features for training (line 16-20). After selecting the best hyperparameters in line 20 the model is trained in line 31 and saved to our pipeline in line 34.

```

1 import pandas as pd
2 from sklearn.preprocessing import LabelEncoder
3 from xgboost import XGBClassifier

```

```

4
5 # Read File
6 file_path = 'cleaned_file_train_test.csv'
7 data = pd.read_csv(file_path)
8
9 # Select Features
10 selected_features = ["offense_formation", "offense_personnel",
11 "defenders_in_box", "defense_personnel", "quarter_seconds_remaining", "down",
12 "ydstogo", "shotgun", "score_differential"]
13 data = data[selected_features + ['play_type']]
14
15 # Encode categorical features
16 label_encoders = {}
17 for column in data.columns:
18     if data[column].dtype == 'object':
19         label_encoders[column] = LabelEncoder()
20         data[column] = label_encoders[column].fit_transform(data[column])
21
22 # Split data into features (X) and target variable (y)
23 X = data.drop(columns=['play_type'])
24 y = data['play_type']
25
26 # Initialize XGBClassifier with best hyperparameters
27 best_params = {'max_depth': 7, 'n_estimators': 100, 'reg_alpha': 0.1}
28 best_model = XGBClassifier(objective='binary:logistic', booster='gbtree',
29                             eval_metric='logloss', learning_rate=0.1, **best_params)
30
31 # Train Model
32 best_model.fit(X, y)
33
34 # Save Model
35 best_model.save_model('xgboost_best_model.model')

```

3.3.7 Testing & Validation

After having trained the final text based model following results can be achieved.

XGBoost Model Performance (Testing)								
	TP-Rate (Recall)	FP-Rate	Precision	F-Measure	MCC	ROC-Area	PRC	ACC
Test	0.678	0.214	0.691	0.684	0.465	0.732	0.751	74.13%

Table 9: Testing Metrics of XGBoost Model.

Table 9 shows the evaluation metrics measured on the test dataset. In fact, the optimization of the hyperparameters resulted in a better accuracy. The accuracy before hyperparameter tuning for this model was 73.86% and after adjustment it was 74.13%. The XGBoost model performed well overall on the testing dataset, achieving a recall of 67.8%. This indicates its ability to correctly identify positive cases. Additionally, it demonstrated high precision with 69.1% and a strong F-Measure with 68.4%, showing accurate positive predictions and a balance between precision and recall. However, there are areas for improvement. The FP-Rate of 21.4%, suggests a relatively high rate of false positives, and the MCC of 46.5% could be

improved. Overall, the model shows good accuracy in context of predicting plays. Table 10 shows the model's evaluation metrics on the validation dataset. In summary the metrics seem a little lower on the validation dataset, than on the test dataset. This could indicate overfitting on the training and test dataset.

XGBoost Model Performance (Validation)								
	TP-Rate (Recall)	FP-Rate	Precision	F-Measure	MCC	ROC-Area	PRC	ACC
Validation	0.666	0.212	0.686	0.676	0.456	0.727	0.744	73.78%

Table 10: Validation Metrics of XGBoost Model.

The training of this model concludes the text-based section of the approach and is the last element of our pipeline.

4 Results

After completing each component of the pipeline, the algorithms and models were chained together using a main Python file. This file reads images from a folder and executes the prediction process. The scoreboard detection and OCR were applied first, followed by the detection of lines and players. Finally, the image was transformed and the visual features were extracted.

This section summarizes and visualizes the results of the described approach. The cycle time for one image is 15 to 20 seconds, which is not considered real-time detection due to the high overhead of using a Python implementation. Please be advised that the implementation is designed for testing purposes and may be improved over time. However, the pipeline would be capable of running in real time when compiled in a suitable environment given the fast processing times of the YOLOv8 and XGBoost models.

Ten images were processed for play type prediction. All images were randomly extracted from YouTube NFL game highlight tapes. In four of the ten images, the Player Extraction YOLOv8 model failed to detect the quarterback, resulting in no prediction. This failure to recognize the quarterback resulted in no extraction of the offensive formation and shotgun feature, resulting in the failure of the entire attempt. Nevertheless, for the six images where a prediction could be made, the play type in five out of six images were correctly predicted. Following Table 11 shows the results of the six tested images.







No.	Image	Actual	Predicted
1		run	pass
2		run	run
3		pass	pass
4		pass	pass
5		pass	pass
6		pass	pass

Table 11: Actual vs. Predicted Labels for Each Image

Let's look further in the single components of the pipeline using image No.3 as an example.

The images below illustrate the application of the approach to an image from a New England Patriots game against the Miami Dolphins in 2023. The prediction was accurate as it resulted in a pass.

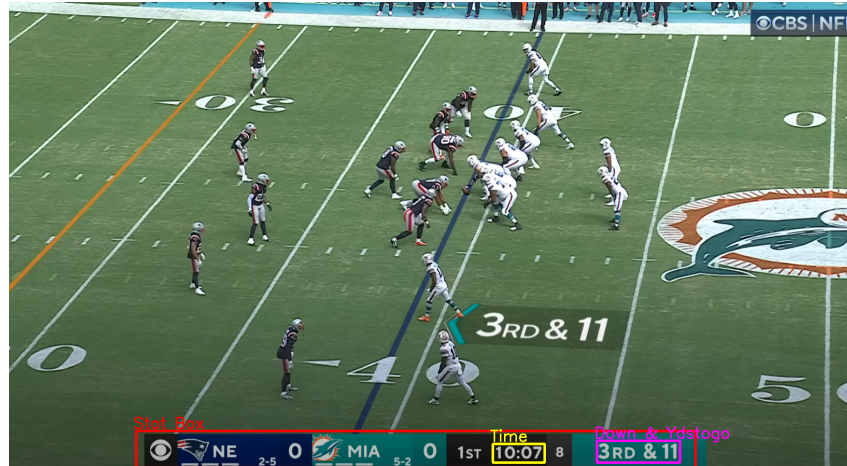


Figure 31: Results: 1. Scoreboard detection.

Figure 31 displays the scoreboard detection results. The YOLOv8 model correctly classified the scoreboard, as well as the time, down, and yards to go. However, the OCR algorithm failed to accurately classify the two scores. The algorithm sets the score differential automatically to zero if no scores could be detected. Following log-messages show the detections:

```

1 Scoreboard detection-----
2 Time: 10:07   Min: 10   Sec: 07
3 Down 3
4 Yards to go: 11
5 Scores: None
6 Scores could not be extracted!
7 Final Results:
8   Score Diff: 0
9   Down: 3
10  Ydstogo: 11
11  Sec Remaining: 607
12  Stats: [0, '3', '11', 607]

```

Final results can be seen in the last line. The score differential is 0, down is 3, yards to go 11 and seconds remaining in the quarter are 607.

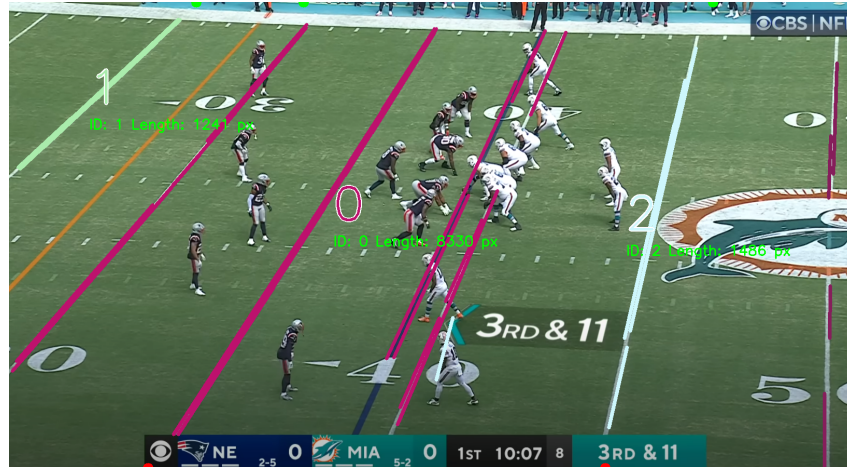


Figure 32: Results: 2. Line detection.

The illustrated lines in Figure 32 mark the detected yard lines of the image. Three groups of lines with similar vectors were detected by the HoughLinesP algorithm. The green line labeled "1", the purple line labeled "2", and the light blue line labeled "3" represent the groups of similar vectors on the image. Based on these lines, the line of scrimmage is detected and the players' positions are transformed.

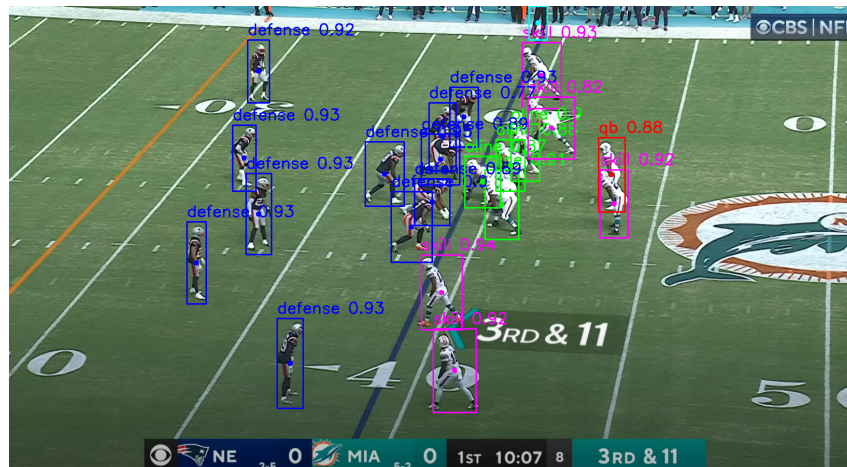


Figure 33: Results: 3. Player detection.

The image depicted in Figure 33 shows the detected players by the YOLOv8 model. The blue boxed players are defenders, pink boxes are skill players, green boxes O-Liners and the red box is the quarterback. Every player in the image was correctly classified. All eleven defensive players were captured in the image and detected.

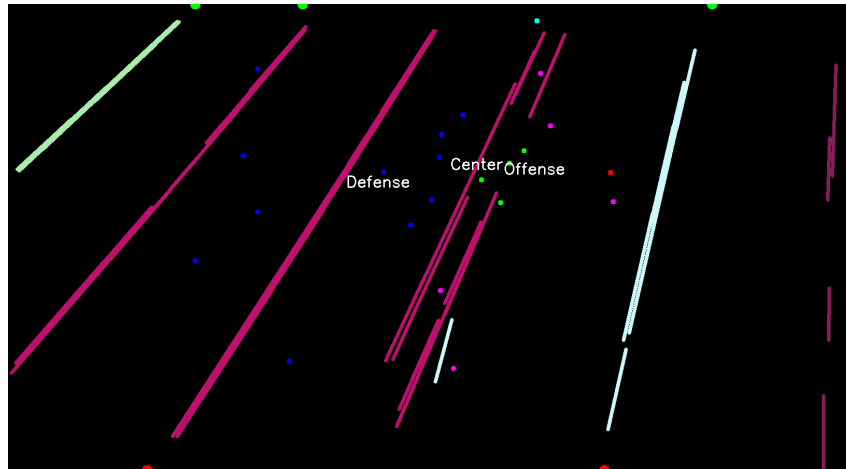


Figure 34: Results: 4. Yard Lines and players combined.

Figure 34 shows the detected players and yard lines combined before image transformation. The colors of the lines and points directly correspond to the images before.

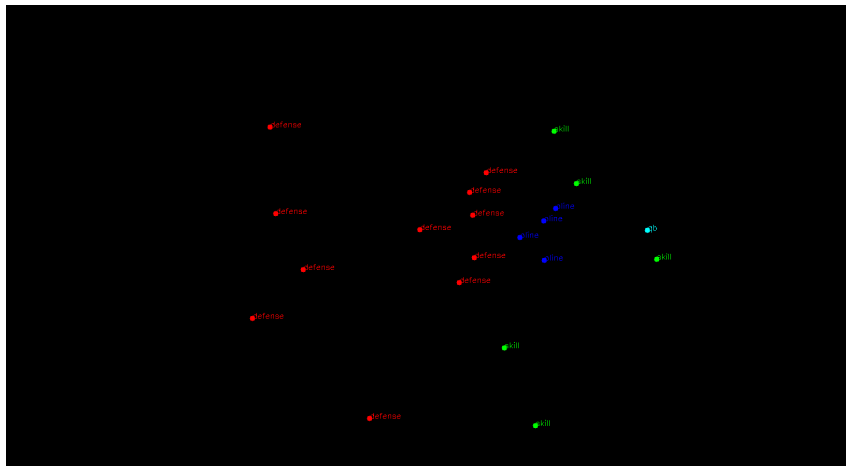


Figure 35: Results: 5. Homography Transformed image.

This next image Figure 35 shows the positions of the players after successful transformation of the players position. The left side of the image shows the defense and the right side the offense.

5 Discussion

The purpose of this section is to look at the results from a critical perspective, suggest future improvements, and review related work.

5.1 Influencing Factors and Limitations

Regarding limitations and obstacles of this approach, there are several issues.

First of all, the quality of the image has a big impact on the result. Often, misclassifications occur because the players are stacked one behind the other and cannot be clearly seen in the image. For example, failing to detect the quarterback will cause the entire prediction to fail. A big risk factor regarding the computer vision based section is the chaining of the components itself. If an element in the pipeline produces an error, the following components will add up to the error. The line detection algorithm is very robust, but if false lines were detected near the center and classified as the line of scrimmage, the homography algorithm would produce a false transformation and the resulting visual feature extraction would be incorrect.

Another common error is unrecognized scores. Since the scores on the scoreboard are just numbers and have the same proportions as other numbers, it is quite difficult to always determine the correct numbers for the score. However, since the score difference has little effect on the play type, this error has little effect on the final result.

Finally, the visual feature extraction algorithm uses very primitive methods. Since the YOLOv8 player detection model is unable to distinguish between defensive positions, such as defensive linemen, linebackers, and defensive backs, the visual feature extraction algorithm must do so. This algorithm approaches player type discrimination by constructing boxes on the field. For example, if linebackers were to "show a blitz", meaning they would line up close to the defensive line, they would also be classified as defensive linemen themselves, which is incorrect. All of these factors affect the outcome of the prediction. Incorrect extraction of visual features would not give the XGBoost model the opportunity to predict the correct play based on the features.

5.2 Potential Future Improvements

Following measures could be taken to improve the approach. Regarding the detection of players, the image dataset is unevenly distributed. The choice of classes is very poor, as the class defense is overrepresented and the class quarterback is underrepresented, illustrated in Figure 37. This, of course, is a result of only one quarterback being on the field at each time and eleven defensive players. Because enlarging the dataset would not solve this problem, the class defense could be subdivided into linebackers, defensive linemen, and defensive backs.

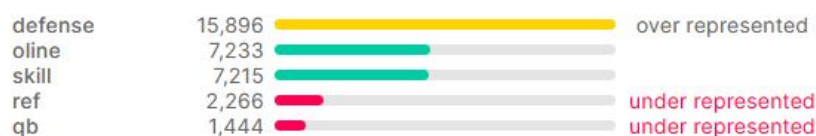


Figure 37: Class distribution of player detection dataset.

To also improve consistency of the approach, the player detection model could be extended to differentiate between tight ends, wide receivers, and running backs within the skill class. This would not only eliminate the error of unilateral distribution of classes but also make the visual feature extraction algorithm, line detection algorithm, and transformation algorithm obsolete, as the visual features could be extracted more easily.

Further improvements could be made to the perspective transformation. The current approach rectifies the horizontal shift of the image so that the line of scrimmage is perpendicular to the top and bottom of the image. This works quite well, but the algorithm does not support depth rectification. Distances closer to the camera will appear larger in the transformed image than distances farther from the camera. To fix this, a constant vertical shift could be applied to all images, pushing the farther distances apart and pulling the closer distances together.

A drastic improvement when extracting features from the scoreboard would be to train the YOLOv8 model itself to detect scores, times, yards, and downs. This would make the OCR algorithm obsolete and minimize errors. However, the model would need way more image data as the task becomes more complex.

Overall, more data could be obtained to improve the prediction accuracy. Especially data with more dimensionality than an image, such as video data, trajectory data of movements before the play, or pose estimation for each player. The more data gathered the better the accuracy of the approach.

Finally, as mentioned earlier, the architecture has the ability to be transformed into a real-time detector. Making detections in real time would also be a great opportunity for processing video data using exactly this approach with multiple frames.

5.3 Related Work

There are only a few scientific papers on predicting plays in American football. But there is a lot of related work, including artificial intelligence in American football. Each approach has different objectives and results.

A master's thesis in this area focused on not predicting a play, rather than analyzing the players and the field in greater detail. Neural networks were trained to differentiate between man and zone coverage, to estimate the pose of the players, and to translate the position of the players from a side view to a bird's eye view (41). This thesis approaches the problem of transforming the perspective of the image by recognizing specific patterns on the field and matching them with a before made template on the bird's eye view. The results are very promising.

In another approach to predict American football plays, similar patterns to the approach of this bachelor's thesis can be seen. For this work, a dataset provided by Kaggle of NFL play-by-play data from 2015 was chosen. The dataset consisted of 63 features and 43,129 plays (42). Three people worked on this project, preprocessed the data, selected a fitting model, and ended up with an accuracy of 74.35% with their best XGBoost model. This approach is very similar to the text-based part of our approach, but no visual features were extracted.

6 Conclusion

6.1 Is it possible to artificially predict football plays?

At the end of this thesis, the question remains. Is it possible to artificially predict American football plays? This thesis proves that it is possible to predict plays with the XGBoost model, which has an accuracy of about 74% with very few features. However, NFL teams prepare 75 to 100 plays for each game, including so-called run-pass option (RPO) plays, where the quarterback makes a split-time decision after the play starts whether or not to throw the ball. Not even the quarterback knows if the occurring play ends up being a pass or a run. So predicting a play before the snap with near 100% accuracy every time seems to be an almost impossible task. Nevertheless, this does not mean that a reasonably accurate prediction is not possible. Especially if you have the right quantity and quality of data. Small indices like the pose of the offensive linemen before the play could indicate a certain kind of outcome. With the right data, it would be realistic to achieve accuracy in the low to mid 80 percent range for NFL games and well into the 90 percent range for the German Football League or youth football teams as they have fewer plays and formations.

6.2 Artificial intelligence in Sports: A Look into the Future

Artificial intelligence is playing an increasingly prominent role over the years. In particular, AI is playing an important role in the analysis of plays in American football. Amazon Prime recently introduced a real-time solution for NFL live streams that determines the likelihood of a linebacker blitzing and alerts the viewer when the likelihood of a linebacker blitzing is high (43). How is this possible? Players are equipped with sensors that track their current position on the field. A machine learning algorithm detects a blitzing pattern and displays it. The fast reaction of the algorithm itself makes it seem like it can predict the future! So the question is not whether artificial intelligence will be utilized in sports, but rather what remarkable capabilities it will be able to achieve.

Literature references

- [1] "2022 OFFICIAL PLAYING RULES OF THE NATIONAL FOOTBALL LEAGUE." <https://nflcommunications.com/Documents/2022%20official%20Playing%20Rules.pdf>. Accessed: 2023-11-14.
- [2] C. Haddad, "What is the box in american football? learn here," Feb 2024.
- [3] NFL, "Buffalo Bills vs. Cincinnati Bengals — 2023 Week 9 Game Highlights." Online Video, November 2023. Accessed on 04.04.2024.
- [4] NFL, "Miami Dolphins vs. Kansas City Chiefs Game Highlights — NFL 2023 Week 9." Online Video, November 2023. Accessed on 04.04.2024.
- [5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [6] "Ultralytics — github." <https://github.com/ultralytics/ultralytics>. Accessed: 2024-04-06.
- [7] "Confusion matrix." <https://medium.com/@awabmohammedomer/confusion-matrix-b504b8f8e1d1>. Accessed: 2024-04-09.
- [8] C.-W. Chen, Y.-H. Tsai, F.-R. Chang, and W.-C. Lin, "Ensemble feature selection in medical datasets: Combining filter, wrapper, and embedded feature selection results," *Expert Systems*, vol. 37, no. 5, p. e12553, 2020.
- [9] B. Reid, K. Schreiber, J. Shawhan, E. Stewart, R. Burch, and W. Reimann, "Reaction time assessment for coaching defensive players in ncaa division 1 american football: A comprehensive literature review," *International Journal of Industrial Ergonomics*, vol. 77, p. 102942, 2020.
- [10] "How much playing really goes on in an nfl game?." <https://medium.com/knowledge-stew/how-much-playing-really-goes-on-in-an-nfl-game-4d1db2731538>. Accessed: 2023-11-08.
- [11] B. Siddiquie, Y. Yacoob, and L. Davis, "Recognizing plays in american football videos," *University of Maryland*, 2009.
- [12] Wikipedia contributors, "1869 princeton vs. rutgers football game — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=1869_Princeton_vs._Rutgers_football_game&oldid=1180464970. Accessed: 2023-11-10.
- [13] R. Szeliski, *Computer Vision*. Springer Nature.
- [14] G. A. Tausif Diwan and J. V. Tembhurne, "Object detection using yolo: challenges, architectural successors, datasets and applications," vol. 82, no. 6, pp. 9243–9275.
- [15] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

-
- [16] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
- [17] "Ultralytics — revolutionizing the world of vision ai." <https://www.ultralytics.com>. Accessed: 2024-04-06.
- [18] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, p. 1680–1716, Nov 2023.
- [19] "Roboflow." <https://roboflow.com>. Accessed: 2024-04-06.
- [20] D. Berchmans and S. S. Kumar, "Optical character recognition: An overview and an insight," *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Jul 2014.
- [21] A. Mahajan, "Easyocr: A comprehensive guide," Oct 2023. Accessed: 2024-04-06.
- [22] "Opencv: Hough line transform." https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html. Accessed: 2024-04-07.
- [23] M. Chen, Q.-Y. Liu, S. Chen, Y. Liu, C. Zhang, and R. Liu, "Xgboost-based algorithm interpretation and application on post-fault transient stability status prediction of power system," *IEEE Access*, vol. PP, pp. 1–1, 01 2019.
- [24] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and et al., "Pytorch distributed," *Proceedings of the VLDB Endowment*, vol. 13, p. 3005–3018, Aug 2020.
- [25] "12 important model evaluation metrics for machine learning." <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/#:~:text=of%20machine%20learning,-,What%20Are%20Evaluation%20Metrics%3F,comparing%20different%20models%20or%20algorithms>. Accessed: 2024-04-09.
- [26] S. ichi Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4, pp. 185–196, 1993.
- [27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.
- [28] "sklearn.model_selection.gridsearchcv." https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Accessed: 2024-04-09.
- [29] "Pytorch." <https://pytorch.org>. Accessed: 2024-04-08.
- [30] "Weka 3: Machine learning software in java." <https://ml.cms.waikato.ac.nz/weka/>. Accessed: 2024-04-08.
- [31] "scikit-learn: machine learning in python." <https://scikit-learn.org>. Accessed: 2024-04-08.
- [32] "Numpy." <https://numpy.org/>. Accessed: 2024-04-08.

-
- [33] “pandas - python data analysis library.” <https://pandas.pydata.org/>. Accessed: 2024-04-08.
- [34] “Top catches of the 2023 regular season — nfl highlights.” <https://www.youtube.com/watch?v=5GAd0ycmRDI>. Accessed: 2024-04-09.
- [35] “Kaggle: Your machine learning and data science community.” <https://www.kaggle.com>. Accessed: 2024-04-09.
- [36] “Football player tracker computer vision project.” <https://universe.roboflow.com/football-tracking/football-player-tracker-tyrjx>. Accessed: 2024-02-27.
- [37] “Espn miami dolphins stats.” https://www.espn.co.uk/nfl/team/stats/_/name/mia/miami-dolphins. Accessed: 2023-11-07.
- [38] “Github nflverse releases: nflverse/nflverse-data.” <https://github.com/nflverse/nflverse-data/releases>. Accessed: 2023-10-21.
- [39] “Pso wrapper method ressource.” <https://github.com/sebastian-luna-valero/PSOsearch/releases/download/1.2.0/PSOsearch.zip>. Accessed: 2023-11-12.
- [40] “Tableau® official site.” <https://www.tableau.com>. Accessed: 2024-04-13.
- [41] L. Dickmanns, “Pose estimation and analysis for american football videos,” Master’s thesis, Technical University of Munich, Nov 2021.
- [42] “Nfl predicitions.” <https://rahuljain28.github.io/NFLPredictions/>. Accessed: 2024-04-15.
- [43] “How did this machine figure out the blitz was coming before the players did?.” <https://www.youtube.com/watch?v=Z7jNFonIecQ>. Accessed: 2024-04-15.