Faculty of Fundamental Problems of Technology
Wrocław University of Science and Technology

Machine Learning-based methods in predicting e-sports' results

Kacper Ciucias
Index number: 250199

MSc thesis written
under the supervision
of Dr inż. Piotr Syga

Wrocław
University
of Science
and Technology

Wrocław 2023

Contents

Introduction

From the start of the XXI century, a constant upwards trend in the popularity of electronic sports can be observed. Various video games have gathered a massive following; therefore, global events organized for those games attract millions of viewers each year. However controversial, it can be argued that esports is on the brink of becoming just as mainstream as all of the classic sports today [1].

With this interest generated annually, it should not be surprising that a massive industry allowing an average viewer to place a bet on their favourite team came into existence. This, in turn, means that gaining even the smallest amount of competitive edge in predicting a match result beforehand can make an enormous difference. This is why the problem of predicting the outcome of an esports match was deemed relevant and worth taking a closer look at in this day and age.

Alongside the esports rise to fame, the use of machine-learning models, and within them neural networks, is also becoming more and more frequent. Various machine-learning algorithms have found their way into our everyday life, more often than not improving its quality in many different ways. Since those algorithms are, in many cases, designed to make predictions [2, 3, 4], their use for the prediction problem at hand seemed just right.

The main goal of this thesis is to propose and examine different neural network architectures and their performance when dealing with the problem outlined above. The secondary goal is to investigate the impact of auxiliary information introduced to the data used in the process of training those models.

The second, third, and fourth chapters of this thesis are devoted to fully understanding the problem, the algorithms providing the foundation for the later proposed models, and the dynamics of the esport selected for analysis - Counter-Strike: Global Offensive.

The fifth chapter showcased the process of selecting, processing and engineering the data that was later used in experiments. It also contains a detailed explanation of what auxiliary information was considered.

The sixth chapter contains the details of all the neural network architectures examined for the purpose of this thesis as well as the more specific algorithms used within them.

The final two chapters showcase all the results gathered, and the summary and conclusions derived from them can be found there.

Machine learning algorithms

In this chapter, the fundamental ideas and algorithms from the field of machine learning are described. Different extensions to the basic algorithms are introduced alongside their respective mathematical notations. The most general concepts, such as optimization, loss, activation and regularization, are explained in depth, since, at a later stage, those parts are used as building blocks for the proposed models.

2.1 Stochastic gradient descent

Stochastic gradient descent (SGD) is an iterative optimization algorithm used extensively in machine learning for minimizing the relative error of the model's predictions. The function being minimized during gradient descent is the loss function. The gradient descent algorithm works by trying to find the global minimum of a given function based only on the local information. In this case, the local information is the direction in which the gradient of our cost function is negative. By shifting the loss function in that direction, it should, in theory, tend to the global minimum following the formula:

$$x_{n+1} = x_n - \eta \nabla f(x_n) \tag{2.1}$$

where f is the optimized loss function, x_n is the current state and x_{n+1} is the new state.

The crucial parameter, denoted in the above equation as η, is called the learning rate of the algorithm. It describes how large of a step is taken in the corresponding direction. If the learning rate is too large, the algorithm may be unable to reach a global minimum, instead oscillating around it and even diverging from the optimal result altogether. However, if the learning rate is too small, the optimization process may get stuck at a local minimum and never reach the best solution. The size of the learning rate also impacts the speed and resources needed, as more or less time may be required to reach a stable solution. All the above things make fine-tuning of η an invaluable step in achieving the best results.

2.2 Other optimization algorithms

Even though the Stochastic Gradient Descent algorithm provides an effective way of training a neural network, it is not without flaws. Two of its most noticeable weaknesses are the high dependency of the outcome on the selected learning rate and relatively slow convergence [5]. However, the latter is possible to overcome by adding momentum to the SGD algorithm. The momentum works simply by accelerating the gradient descent in the correct direction by taking into account the previous updates [5]:

$$x_{n+1} = x_n - \eta \nabla f(x_n) + \alpha * \Delta x_n \tag{2.2}$$

Where the decay factor α determines the influence of previous updates.

Another variation on the SGD algorithm, one that is currently the most widely spread optimization algorithm, is the so-called Adam optimizer [5]. It combines the concept of momentum with an adaptive learning rate. This mitigates the influence of the initially chosen learning rate on the learning process.

2.3 Back-propagation algorithm

The Back-propagation algorithm is used in machine learning to estimate the gradient in a feed-forward neural network [5]. It allows the model to determine the direction in which the gradient drops the fastest, using the known weights assigned to specific neurons. The algorithm can be generally defined in the following steps [5, 6]:

1. Complete the forward pass of the network to obtain an output.
2. Starting from the output layer, calculate the gradient of the error with respect to the output of that layer.
3. Back-propagate error through the network to calculate the gradients of each hidden layer:

$$\delta_j = f'_j * \sum_i w_{ji} * \delta_i \tag{2.3}$$

   Where δ_i is the relative error at the i-th neuron, f is the neurons' activation function, and $w_j i$ is the weight between neurons j and i.

4. Compute the gradients of the weights across the network.

   Gradients computed this way are then used by the optimization algorithms such as SGD to train the neural network.

2.4 Loss functions

While the neural network can initially create a set of predictions based on some input, some of those predictions will most likely be incorrect. Hence an algorithm is needed to allow the network to recognize the incorrect results and improve its structure based on that. This is where the need arises for a function, which, when minimized during model training, leads to better results. Such a function in machine learning is called a loss function.

There are many loss functions currently known and used in neural network training. When it comes to classification and especially binary classification, the most commonly used loss function is Binary Cross Entropy (BCE), also known as the Logarithmic Loss function. The BCE loss function for a single output is given by the formula:

$$\mathrm{BCE} = y_n * \log(x_n) + (1 - y_n) * \log(1 - x_n) \tag{2.4}$$

where y_n is the actual class of the n-th observation, and x_n is the predicted probability that the observation belongs to the positive class. The calculated loss function is then normalized over the entire batch of samples.

The BCE loss can be further modified by applying weights to classes when their prevalence in the dataset differs. This is commonly the case in real-world scenarios where while performing binary classification, we may expect one of the classes to be slightly less common or harder to predict. The weighted log-loss function is expressed by the following formula:

$$\text{Weighted\_BCE} = w_n * y_n * \log(x_n) + (1 - y_n) * \log(1 - x_n) \tag{2.5}$$

where the w_n term is the weight of the observation assigned accordingly to its class.

Another modification to the BCE loss function is the so-called Focal Loss [7, 8]. While it was initially used to help with object detection, it has also been shown to help in classification tasks [7]. The idea behind it is that applying a certain scaling factor to the BCE loss allows the model to focus on training hard-to-classify inputs. The focal loss can be calculated as follows:

$$\text{Focal\_loss} = BCE * (1 - p_t)^{\gamma} \tag{2.6}$$

Where γ is the weighing parameter for hard-to-classify examples. If γ is chosen to be positive, the loss is affected more by the problematic inputs. The term p_t is calculated in the following way:

$$p_t = y_n * x_n + (1 - y_n)(1 - x_n) \tag{2.7}$$

The focal loss can also be further developed into weighted focal loss [8], similarly to the weighted BCE, by applying some balancing factor α to specific classes:

$$\text{Weighted\_focal\_loss} = \alpha_t * Focal\_loss \tag{2.8}$$

Where:

$$\alpha_t = \alpha * y_n + (1 - \alpha)(1 - y_n) \tag{2.9}$$

Besides binary cross-entropy loss and its derivatives, many loss functions can be used for different machine-learning tasks. One such function is the Huber Loss, mainly used for regression tasks, with its variants being successfully used in classification [9]. Huber loss is a function combining the mean-squared error for small values of error and the absolute error otherwise. It can be expressed as:

$$\text{Huber\_Loss} = \begin{cases} 0.5(x_n - y_n)^2, & \text{if } |x_n - y_n| < \delta. \\ \delta * (|x_n - y_n| - 0.5 * delta), & \text{otherwise.} \end{cases} \tag{2.10}$$

The δ parameter is used to determine the transition point between the two error metrics. The Huber Loss is used to minimize the impact of outliers on the prediction, which in the case of binary classification means reducing the impact of predictions that are located far from the decision boundary.

2.5 Activation functions

Activation function ϕ in a neural network is a type of hidden unit that allows for introducing non-linearities while using a small number of nodes [5]. The activation function works by applying some transformation $\phi(x)$ to the node input x, which results in the node being activated or not. The choice of an appropriate function can drastically influence the network's performance. Hence there exists a wide range of possible functions, while new ones are actively being developed.

The most widely used activation function is the Rectified Linear Unit or **ReLU** for short. Its popularity can be attributed to the short time needed for computation and its applicability to a broad spectrum of different tasks [5, 10]. The ReLU function in its purest form can be expressed as:

$$ReLU(x) = max(0,x) \tag{2.11}$$

and so it behaves as an identity function for any positive input, while otherwise, it yields 0.

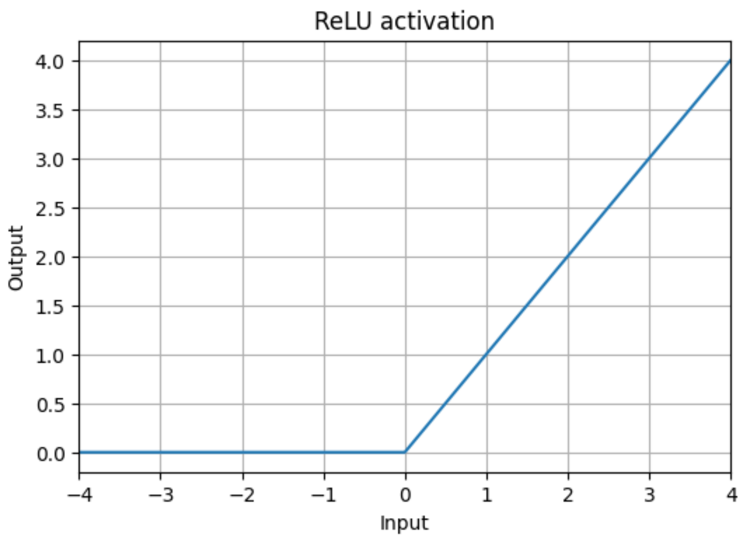


Figure 2.1: Plot of the ReLU activation function.

Another commonly used activation function is a variation of the ReLU called Leaky ReLU [11]. It introduces the addition of a small slope for the negative input values so that:

$$LeakyReLU(x) = \begin{cases} x, & \text{if } x > 0. \\ \alpha * x, & \text{otherwise.} \end{cases} \tag{2.12}$$

While more computationally expensive, this alternative to the standard ReLU allows for surpassing the "vanishing gradient problem", where some of the neurons can become stuck in an "off" state forever. In addition, a leaky ReLU activation function can be further improved by making the parameter α trainable. This type of activation function is called Parametric ReLU or PReLU for short [12].

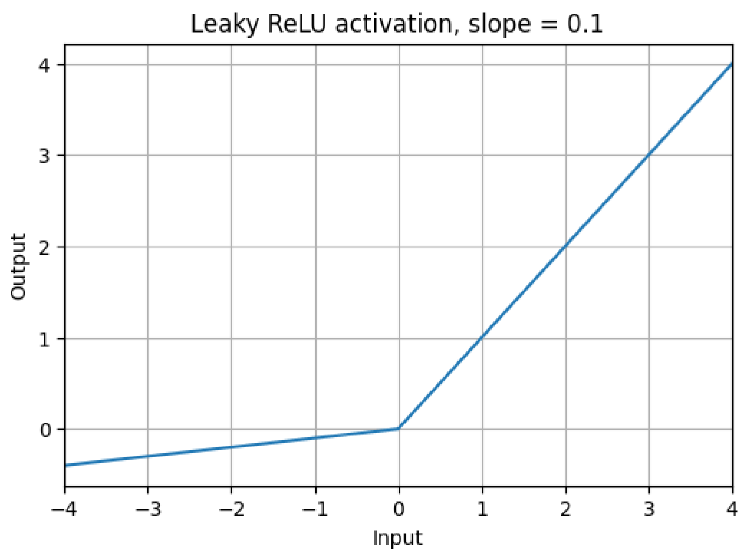


Figure 2.2: Plot of the Leaky ReLU activation function.

Besides linear activation functions, there are also their non-linear counterparts. One such function is the **SiLU** function, which stands for Sigmoid Linear Unit. The function can be expressed as:

$$SiLU(x) = x * sigmoid(x) = \frac{x}{1 + e^{-x}} \tag{2.13}$$

which allows for introducing non-monotonous behaviour to the system. The SiLU function was proposed in 2016 [13] as a way to combat the "vanishing gradient problem" in the back-propagation algorithm.

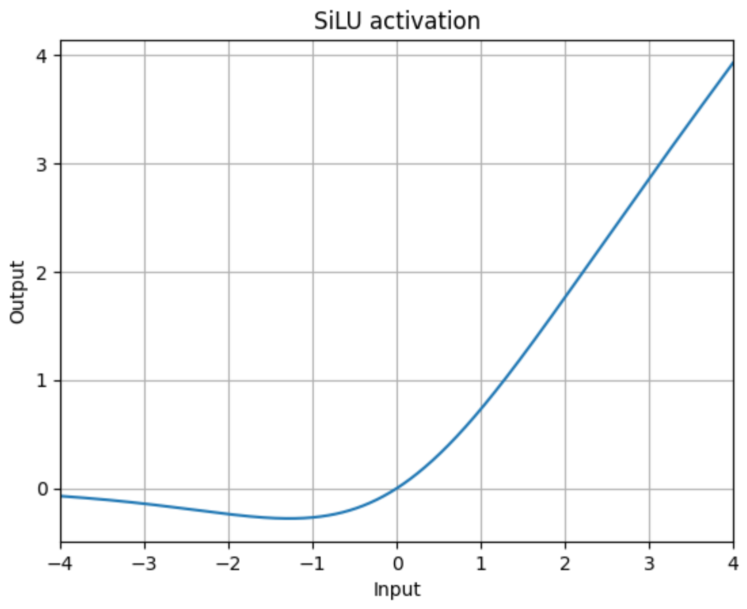


Figure 2.3: Plot of the SiLU activation function.

The introduction of the SiLU activation function sparked the creation of the Mish function [14]. It combines the hyperbolic tangent function and a softplus function and is theorized to be a better choice than SiLU for some problems:

$$Mish(x) = x * tanh(softplus(x)) \tag{2.14}$$

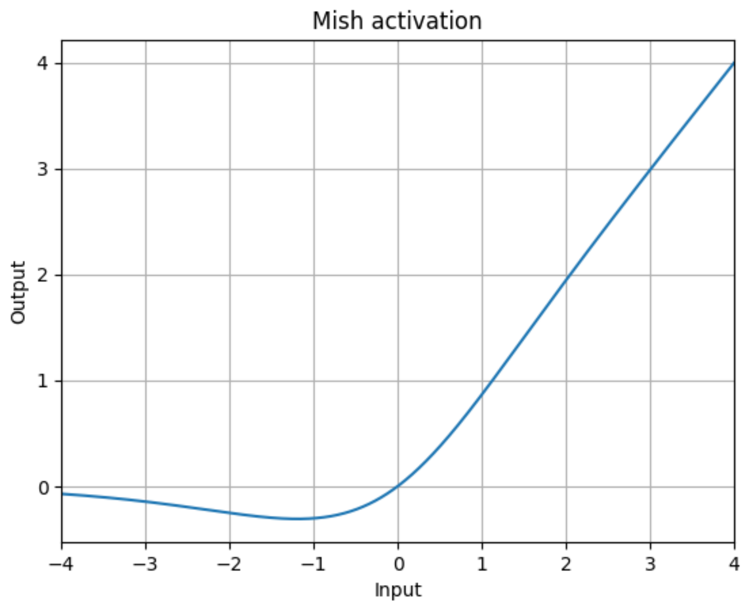


Figure 2.4: Plot of the Mish activation function.

All of the above-mentioned activation functions are in some way variations of the most common ReLU function, which is why they were chosen for comparison in later chapters of this thesis.

2.6 Regularization techniques

A well-designed neural network is a model that generalizes well to unseen data. More often than not, however, the models tend to learn patterns that are specific to the training data that cannot be easily generalized. This phenomenon is known as model overfitting [5]. To fight overfitting, a multitude of techniques were developed, the most relevant of which are discussed below.

One of the most common regularization techniques is the L2 regularization. It works by applying some norm penalty function to weights at each network layer while leaving the biases untouched. This is why this type of regularization is often called weight decay. The regularization term can be expressed as:

$$\Omega(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||_2^2 \tag{2.15}$$

Which is then subsequently added to the objective function [5].

Another type of regularization commonly used due to its fast computation time for large networks is dropout regularization. It works by randomly deactivating specific neurons in layers by setting their value to 0 without affecting their weights. This method allows the network to learn patterns that are not solely dependent on any specific neuron but more on the general properties of the model. The dropout regularization can be applied both to the visible and hidden layers to mimic the behaviour one would achieve when training an ensemble of similar networks with varying node combinations [5].

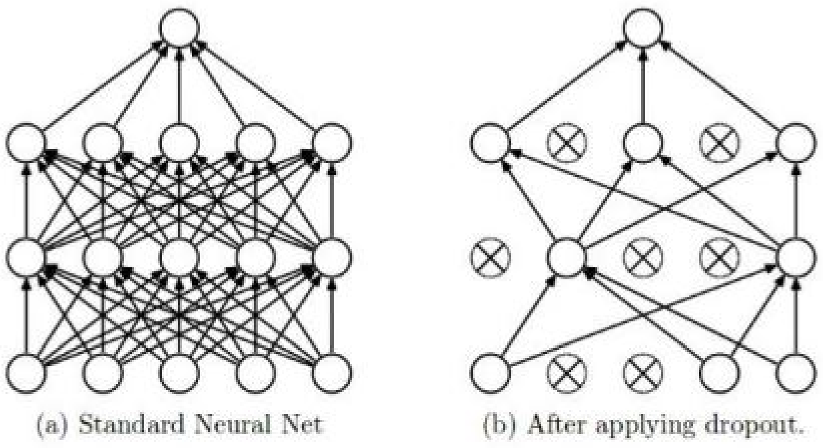


Figure 2.5: Result of applying a dropout to a standard ANN architecture [15].

When training a sufficiently sizeable neural network, most often than not, the experimental training error steadily decreases over time. However, long enough training time may lead to a steady loss in the network's ability to generalize as the validation error increases after an initial decrease, as shown in 2.6.

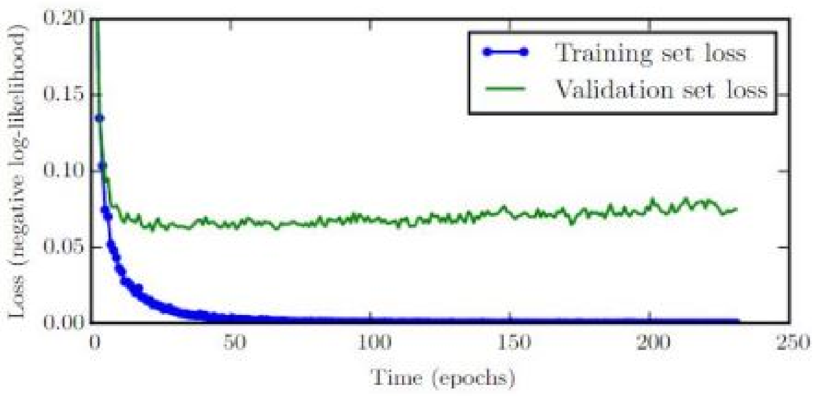


Figure 2.6: The training and validation loss for a deep learning model trained on the MNIST dataset. As can be seen, after some epochs, the validation stabilizes and even starts to rise [5].

To stop this gap in training and validation loss from growing, a technique called early stopping can be utilized. The idea behind it is that by saving the model state when the error on the validation set is the lowest, we should also obtain the lowest test error when evaluating the model on unseen data. The early stopping is probably one of the most common cases of regularization in machine learning, mainly due to its simplicity. The technique can be improved by further tuning the patience parameter, which indicates how long we should wait before stopping the training process. This makes the method robust to momentary oscillations in the validation error[5].

Overfitting can often be caused by insufficient data used for the model training. With a small dataset, the model can find patterns that are representative of only the small sample it was provided with, thus losing the ability to generalize to the underlying problem. While in practice, the real-world data available to us is limited, it is usually relatively simple to generate new synthetic data to feed into our model. For example, when it comes to a classification task, where the output is simply a combination of the input features x, altering those features numerically allows for introducing new data. The addition of random noise to the existing data can also be used as a form of data augmentation, further increasing the size of the dataset [5].

Related work

The problem of predicting a match outcome has been approached multiple times in recent years. Starting with traditional sports such as football [16, 17, 3, 4] and tennis [18, 19] the opportunities in using machine learning for this purpose became apparent.

As has been shown, even the simpler models allowed people to gain an advantage when picking the winner of a match. In work [17], it was shown that a simple SVM classification model with fine-tuned parameters could achieve an accuracy of over 50% when predicting a football match outcome. Again in 2019, [16] an accuracy of over 80% was achieved using an ensemble method called the Random Forest. Similar results were accomplished for different sports. For example, in 2021 [19], an accuracy of over 83% when using the random forest algorithm. Simple neural network algorithms were also used to predict horse racing contest winners [20], with an average accuracy of 77%.

When it comes to electronic sports, or esports for short, there have not yet been many studies conducted on the topic of match-winner prediction. This may be because the complexity of esports games is often much higher than that of a regular sport. There are simply many more features that can influence the outcome of a match, so a more robust and advanced machine-learning method is needed. The other reason this problem is not yet as thoroughly examined may be that esports is still on the rise and are just now gaining interest comparable to that of regular sports [21].

Having said that, some groundwork has been laid out for match prediction in esports. When it comes to "Counter Strike Global Offensive" ("CS:GO" for short) specifically, there are works published in recent years that aim to predict the result of a single match by using different machine learning methods [2, 22, 23, 24]. The methods used in the above publications span a wide variety of algorithms, from simple ones like logistic regression [2], decision trees [24], and ensemble methods [23] to more complex artificial and convolution neural networks [2, 22].

In 2018 [24] decision tree classification model was used for predicting CS:GO match outcome. The model was trained on data from parsed match demo files obtained from the hltv.com website. The analyzed data comprised 162 matches played between October 2016 and January 2017. The paper states that the highest accuracy achieved using this method was 62%. It is unknown, however, what features were taken into account as well as what were the used model's exact parameters.

The same year another publication that touched on the subject in question was made available [22]. This time the authors decided to analyze the performance of a thousand top-ranked players according to the Faceit.com European rankings. Around 6000 match demo files were parsed to extract the relevant features. Based on those features, the players were assigned to clusters reflecting their play style using the K-means algorithm. Those clusters were then used to train artificial neural network models. In this way, the authors achieved an accuracy of over 65% while the selected benchmark was at 59% for this particular problem.

Next, in the year 2022, two publications of similar nature were released. The first one [23] focused on predicting round winners and, based on that, the match winners. The data used to make those predictions again came from match demos parsed to extract meaningful features; however, only 45 matches from between September and December of 2021 were obtained.

The performance of three selected machine learning algorithms was studied. Random forest achieved an accuracy of around 59%, multi-layer perceptron of around 52.5%, and the best-performing model was the XGBoost, with an accuracy of over 62

The second publication from the year 2022 [2] may be the most informative one to date. It focuses on predicting CS:GO match outcomes based on historical player performance data collected from the hltv.com website. The initial dataset consisted of around 30000 matches played between 2015 and 2020. The machine learning models examined in this paper consisted of logistic regression, SVM, random forest, artificial neural network, convolution neural network, and an embedding neural network. As a result, it was possible to determine which algorithm shows the most promise when attempting to solve the match prediction problem. It was concluded that the best model for this task is once again the random forest with an accuracy of 63% followed by the convolution neural network at 59.8%.

It is also worth mentioning that similar results were obtained when predicting the outcome of other esports games such as "League of Legends" [25] where the best performance was attributed to decision tree models with 55% accuracy or "DOTA 2" where the neural network was shown to achieve accuracy between 55% and 60% based on the features used in training [26].

Esport selected for prediction

For research contained in this thesis, a video game called "Counter-Strike Global Offensive", or CS:GO for short, was selected. CS:GO is the 4th game in the Counter-Strike series and is still being actively maintained by the developer. Ever since its release, it has kept the status of one of the most watched esports in the world, alongside games like League of Legends and DOTA2, with a peak viewership of around 3 million concurrent viewers. The game's popularity is often attributed to its relative simplicity in terms of rules.

While having only a few simple rules, the game is regarded as having one of the highest skill ceilings of all the esports games. This means that while it is relatively easy to get started, it often takes thousands of hours and unmatched abilities to master. This, alongside the fast-paced gameplay and a multitude of regularly held tournaments, is the reason why CS:GO is such a popular esport.

Counter-Strike:Global Offensive is a team-based first-person shooter. A first-person shooter is a game where a player sees the environment from the character's perspective, while the main mechanic of the game consists of aiming and shooting at other players' characters to eliminate them.

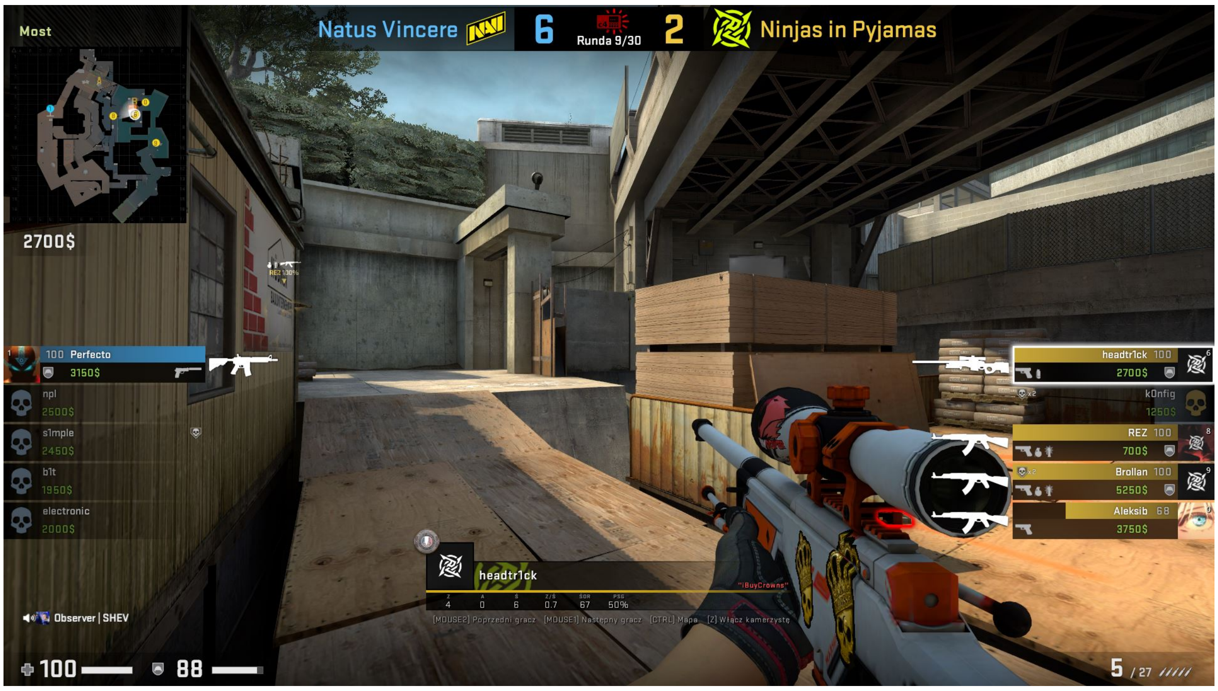


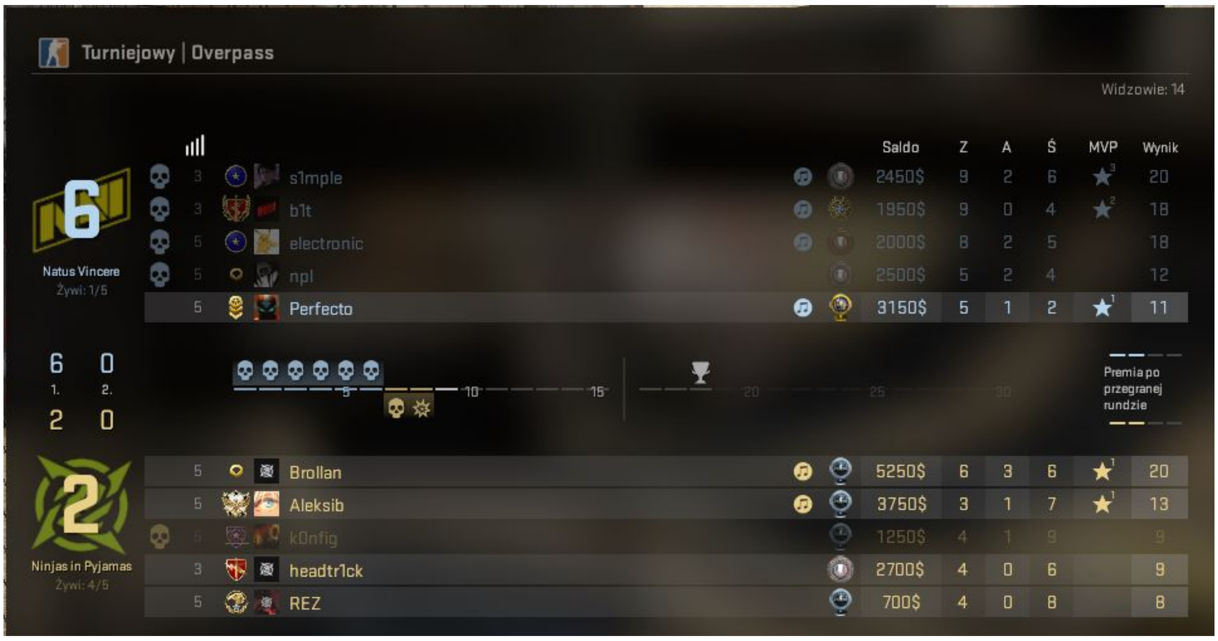
Figure 4.1: An image from an ongoing game. Representation of the player's perspective.

There are two teams taking part in a CS:GO match, each consisting of five players. The teams are assigned to one of the two possible fractions, Terrorists or Counter-Terrorists. For the players on the former side, the objective is to plant a bomb on one of the two available sites while the opposing side tries to defend them. Eliminating all of the opposing team's players is also considered a win condition. Completing an objective is considered a round win, while a regular match is played in a best-of-30 system. This means that the first team to score 16 rounds wins the game. Depending on the tournament format, in a 15-15 score, the game ends in a draw or overtime is played.

The game is divided into two halves of 15 rounds each. At halftime, the players switch sides, resetting their economy to its original state. This is done to remove the bias one side may have over the other.

After each round, players gain a set amount of currency, which can be used to purchase weaponry or other items for their advantage. This amount is based on a round's outcome, so winners are usually awarded more. This system creates a need for the thought-out management of one's economy.

On the individual level, the players are awarded points based on their performance, namely on the number of eliminations and assists they get during the game. This score is not strictly connected to the game's outcome but is helpful when measuring the player's impact on the result.

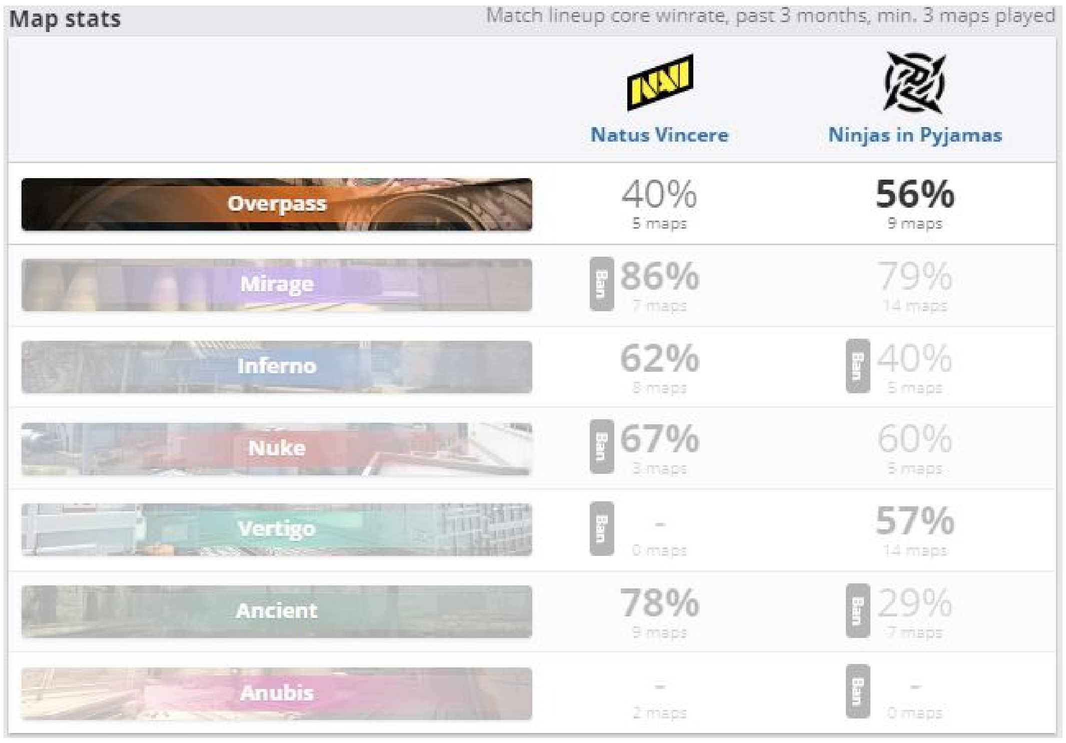

Turniejowy | Overpass

Widzowie: 14

			Saldo	Z	A	Ś	MVP	Wynik
Natus Vincere Żywi: 1/5	3	s1mple	2450$	9	2	6	★3	20
	3	b1t	1950$	9	0	4	★2	18
	5	electronic	2000$	8	2	5		18
	5	npl	2500$	5	2	4		12
	5	Perfecto	3150$	5	1	2	★1	11
Ninjas in Pyjamas Żywi: 4/5	5	Brollan	5250$	6	3	6	★1	20
	5	Aleksib	3750$	3	1	7	★1	13
	5	k0nfig	1250$	4	1	9		9
	3	headtr1ck	2700$	4	0	6		9
	5	REZ	700$	4	0	8		8

Figure 4.2: An example of a scoreboard in an ongoing match. Different player statistics, as well as the overall match score, can be seen.

A CS:GO match can be played on one of several maps that are available at a given time. The choice of a specific map usually comes down to a vote, where the teams alternate removing maps from the current pool until one or more remain, depending on the number of games being played. In addition, the set map pool can change between events based on the game being updated; new maps may be introduced while others are removed.



Map stats — Match lineup core winrate, past 3 months, min. 3 maps played

	Natus Vincere	Ninjas in Pyjamas
Overpass	40% 5 maps	56% 9 maps
Mirage	Ban 86% 7 maps	79% 14 maps
Inferno	62% 8 maps	Ban 40% 5 maps
Nuke	Ban 67% 3 maps	60% 5 maps
Vertigo	Ban - 0 maps	57% 14 maps
Ancient	78% 9 maps	Ban 29% 7 maps
Anubis	- 2 maps	Ban - 0 maps

Figure 4.3: A list of maps available in a map pool for a given tournament. The percentages represent win ratios for each of the competing teams on each of the maps.

A more detailed explanation of how the player scores and economy work and how the maps play into the match result can be found in the next chapter in the "Available features" and "Feature engineering" sections.

The Dataset and Exploratory Analysis

For every machine learning problem, choosing a suitable data sample is crucial. The selected dataset should be sufficiently large to allow the network to learn complex patterns without overfitting. It should preferably contain diverse data, for example, spanning an extended period, and be biased as little as possible. Finally, it would be beneficial if the selected dataset was detailed and disaggregated to allow for the creation of new, more complex features.

When obtaining data from professional CS:GO matches, one can usually take two approaches. The first is to parse the recorded match demos to extract in-game events. This approach provides great flexibility in selecting data because information about everything happening during a game can be obtained, not only the player's statistics but also the player's movement and positioning. However, the drawback to this approach is that parsing a large amount of video files is both time and memory expensive, to the point where it is often not feasible to gather a large enough dataset.

The second approach is collecting post-match data from one of the available websites. When it comes to CS:GO, the leading esports information website is hltv.com. At the time of writing this thesis, it contains records of over 75 thousand matches played between August 2012 and now. For every recorded game, detailed player and map statistics are available and for the more recent matches, economy and even positional heat map data for each player.

The main drawback to collecting data this way is that the hltv.com website does not provide an easy way to collect data automatically. This means that custom-built software is required for such tasks. Although this can be done if the need for information about the most recent matches arises, for the purpose of the research conducted in this thesis, a dataset of already collected match data was obtained from the kaggle.com website [27].

5.1 The dataset's structure

The selected dataset contains matches played between 07.10.2015 and 26.02.2020 that were available on the hltv.com website. The data is divided into four files, each of a different category:

1. "players.csv" file - the main data category used for predictions. Contains information about each player's performance for every game in the dataset. There are 383318 unique entries in this category.

2. "results.csv" file contains the winner of any given match, the final score, and scores for each team in both halves separately. There are 45774 unique entries in this category.

3. "economy.csv" file - contains the information about the team's equipment value for each of the match rounds. Only available for matches played after 01-01-2017. There are 43235 unique entries in this category.

4. "picks.csv" file - contains information about the map selection process for each match.

The dataset contains information from over 43 thousand matches, which can be considered sufficiently large for the purpose of this classification problem. Each of those files, however, contains missing or corrupted data that must be imputed somehow. In a related work [2], it was shown that while working on a similar dataset, there was no noticeable difference when imputing the data with mean instead of simply removing it. This is why the problematic entries were also removed in this case. This process reduced the original dataset to a little under 41 thousand viable matches.

5.2 Available features

The "players" data file contains 101 columns of data, with the first nine being used for identification purposes. Each player is assigned an id, his and the opposing teams' event, and match information. Next is the information about the format in which the event was played. This field has four possible values: 1, 2, 3, and 5. This value refers to the number of individual games played between the two teams at a time. Due to the way the CS:GO tournaments are usually designed, the majority of teams face each other in either the best of 1 or best of 3 formats, with the former accounting for over 48% of the data while the latter for over 46.5%.

Next is the information about the maps on which the games were played. More often than not, the map can significantly affect the team's performance simply because the players may be more used to playing on one map than on the others. This is why all the summary statistics for this research were calculated after dividing the entire dataset on the map level.

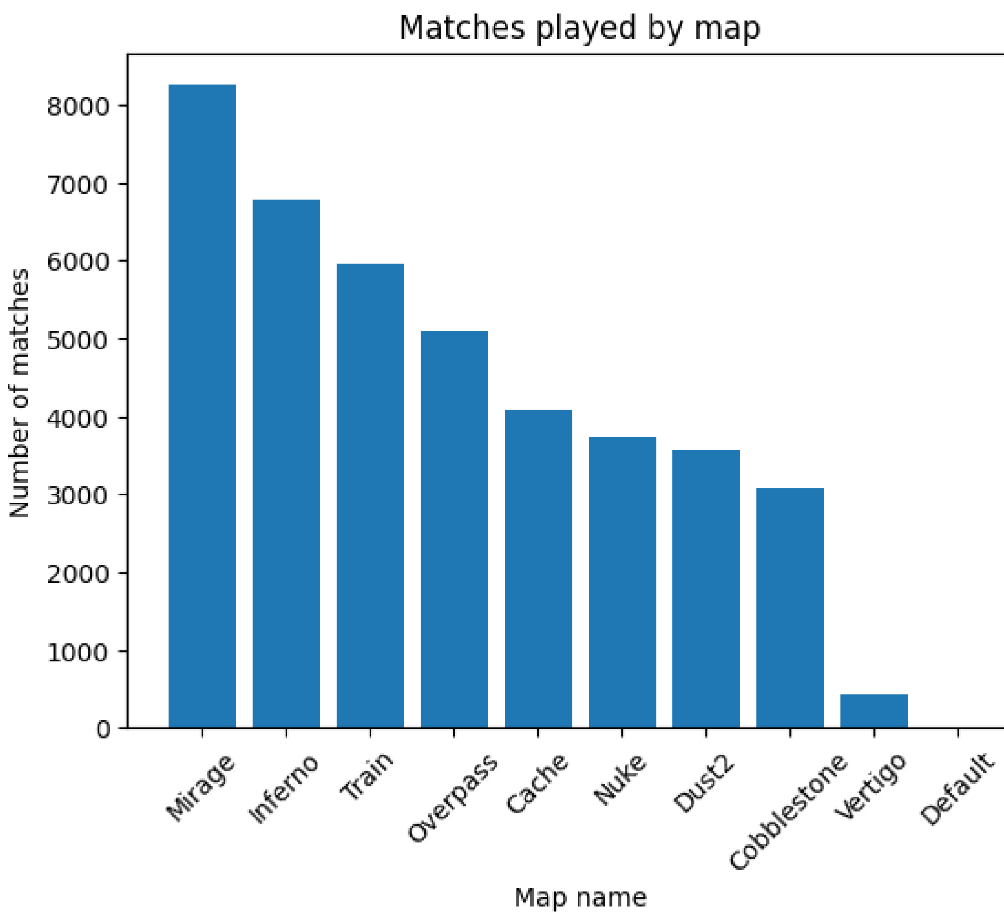


Figure 5.1: The number of games played on each of the maps during the selected time period.

As shown in figure 5.1, Mirage is by far the most popular map, followed by Inferno and Train. This is primarily because out of all the maps, those are the oldest and least frequently changed maps in the pool. On the other side of the spectrum, there is Vertigo, which was only added to the pool in March of 2019, which would explain the low number of matches on this map in the selected dataset. There is also a map called Default appearing in the analyzed data. However, it appears only three times throughout the match data, so it was treated as corrupted data and consequently removed.

The individual player's statistics contained in this file are explained in table 5.1. Those statistics are available for every single map played as well as for each side, Terrorists and Counter-Terrorists, separately.

Kills	Total number of eliminations a player has had during the match.
Assists	Total number of assists a player has had during the match
Deaths	Total number of times a player was eliminated during the match
Hs	Total number of headshots a player has hit during the match
Flash assists	Total number of assists due to a flash grenade being thrown
KAST	A performance metric indicating what percentage of rounds a given player was awarded a kill, assist, has saved or was traded
Kddiff	Difference between player's kills and deaths
Add	Average damage a player has inflicted per round
Fkdiff	Difference between first kills and first deaths in a match
Rating	The HLTV 2.0 rating that a player has been given after a match

Table 5.1: Player-related features available in the 'players.csv' file

In the results data file, there are 19 columns with information about the given match and event, the final and halftime scores for each of the two teams facing each other, as well as the map that was played and the final result indicated as 0 or 1, depending on whether the first or the second team has won.

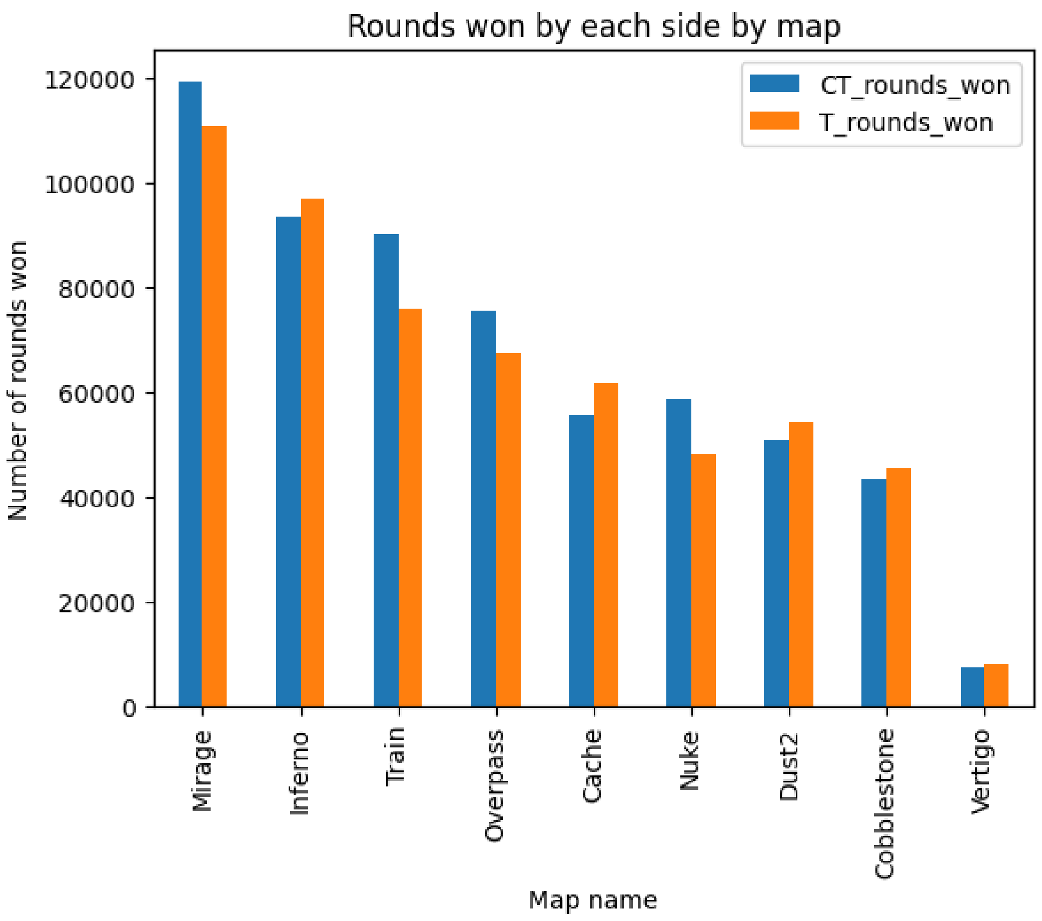


Figure 5.2: Number of rounds won by the Terrorist and Counter-Terrorist sides by map

When analyzing the individual scores at halftime and at the end of a match, it can be noticed that there is always a slight bias towards one of the sides, which is shown in figure 5.2. For Mirage, Train, Overpass, and Nuke, it seems like the maps are biased towards being easier to play on the Counter-Terrorist side, while the other maps are biased towards the Terrorist side. The biases, however, in almost all cases are relatively small, meaning that, for the most part, the maps are fairly balanced.

As for the individual teams that can be found in the dataset, after removing missing information, there are 1509 unique teams left in the dataset. After further inspection, it can be found that out of those unique teams, a significant amount has played only in a handful of matches. This can be a limitation to the later proposed prediction mechanism, as there may simply be too little historical data for some of the teams to make a meaningful prediction for their match. In table 5.2, there are the summary statistics for the number of matches played for each of the teams.

Summary statistics for each team (matches played)					
Count	Mean	Std	Bottom 25%	Bottom 50%	Bottom 75%
1509	57.46	137.27	4	11	39

Table 5.2: Summary statistics for the number of matches played for each team in the dataset.

As can be seen from the data in table 5.2, 25% of the teams played at most 4 matches in their registered history, and so their usefulness for the purpose of this research could be questioned. However, for any team that has played at least two matches, the statistics taken from the first match can be used to try and predict the result of the second match. This is why only the teams that have played a single match were excluded from the calculations. This resulted in 80 teams being removed. Additionally, the high standard deviation can be explained by the fact that there are a few teams that have consistently appeared in the competitive scene, and as a result, have significantly more matches on their record than the mean or median would suggest. The top 35 teams with the highest number of matches played have over six hundred of them in the dataset, with the highest value being 954 matches.

There is also a piece of additional information in this file, which is the global rank of the team at the time the match took place. This information can be used as both a benchmark for predicting the match outcome and as an additional feature in the input to the machine learning models.

The economy file contains information about what was the amount of money available to each of the teams at the beginning of each of the rounds. Additionally, the file contains information about which team won each given round. This allows for understanding how the economy affects the probability of winning a round and, in turn, the entire match.

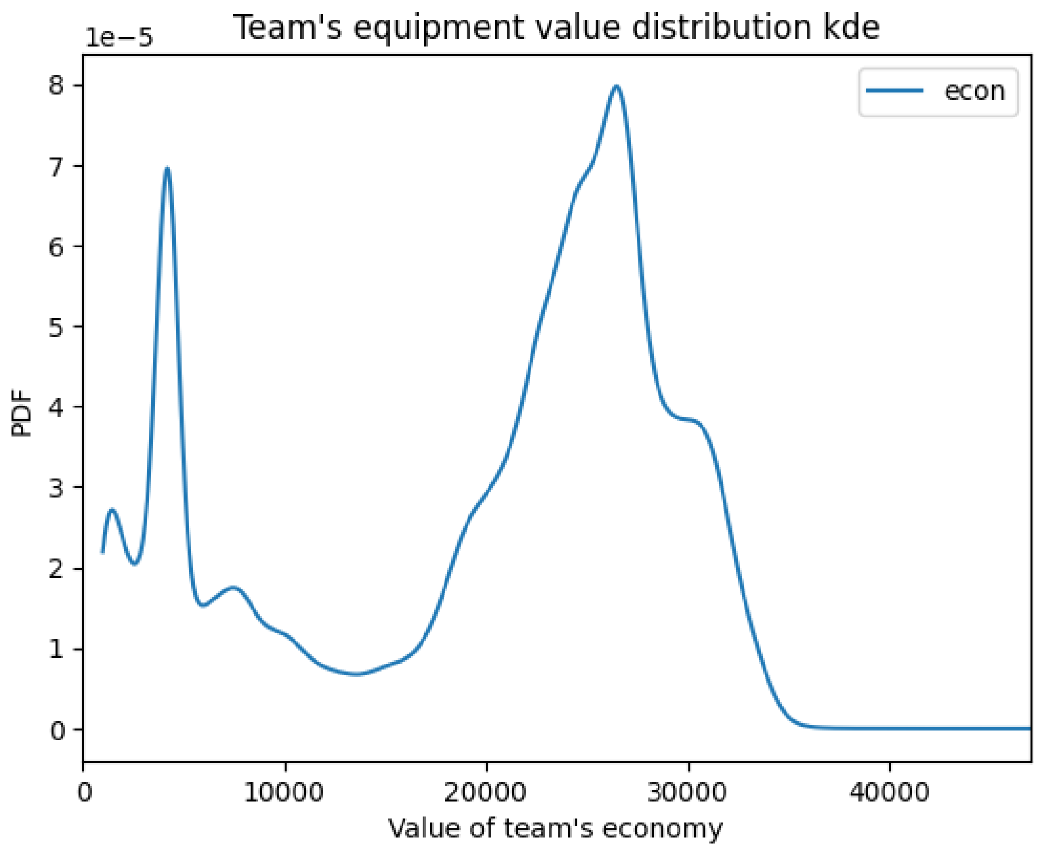


Figure 5.3: A Kde approximation of the distribution of teams' equipment value per round.

Plot 5.3 shows the kernel density estimate, KDE for short, which is the approximate distribution of economy in individual rounds across all recorded matches. The value of the team's equipment seems to form two distinct categories - below 10000 and over 20000 of in-game currency. The reason for this is that for the team to manage its economy correctly, all of the players in the team are either spending their money on equipment or saving it for later rounds. It is a widely accepted convention that if the team is not investing in the current round, it is an eco round, while in the case of each player having full equipment, it is a full buy round. The exact values of the teams' gear may vary slightly depending on the current meta, the team's strategy, and the roles of individual players within a team. This is why in plot 5.3, some smaller peaks may form around those two main categories. It is also worth mentioning that the minimum value of a team's equipment is 1000, while the maximum value is 47000 in-game currency.

The last table in the dataset, the picks table, contains information about how the process of selecting a map played out. In theory, this information can help determine the team's strategy, as a team may be regarded as a favourite on their own map pick. However, it was discovered that this set of data contains a large number of missing entries compared to the other three files. This, combined with the fact that the team's map picks can vary greatly depending on the situation, the data in this file will not be used to avoid overcomplicating the feature vectors.

5.3 Feature engineering

Since almost all of the available features are known only after the given match has ended, the prediction cannot be made on the raw data. This is why for every match in the dataset, an aggregation of some historical statistics was used for predicting the outcome of any future match. For example, if a team's 5th match appeared in the dataset, the data from four previous matches would be aggregated to predict its outcome.

The approach mentioned above assumes that a team can keep up its performance over some time window and that it is, in fact, possible to distinguish teams that perform better at any given time. This approach, however, has two significant limitations. First of all, some of the teams in the dataset have played a relatively small number of matches, so aggregating any summary statistics for them would be difficult, if not impossible (for example, for teams that played only one or two games). The second limitation comes from the fact that aggregating the statistics over too large of a time window would make them converge to some global mean values, in turn leading to a loss in variability and information about the teams' performance. To combat the latter, a selection of sizes for the aggregation window was examined and tested using different machine learning algorithms for classification, and a rolling window of 39 most recent matches was applied.

Most of the player statistics used as features for this problem are presented in the dataset in a per-match form. For example, the Kills value indicates how many eliminations a player had accumulated during the match. This is not an appropriate metric since the length of the match can vary between 16 rounds for a 16-0 scoreline to 30+ rounds if overtime is reached. A better understanding of the player's performance can be achieved when those per-match statistics are converted to their averaged per-round counterparts. This procedure was applied to the first five features listed in table 5.1 and fkdiff and kddiff metrics. Next, those were aggregated using cumulative averages over the matches in the selected rolling time window.

Player statistics such as Average Damage per Round, KAST, Impact, and Rating were aggregated using expanding averages since those do not directly depend on the match length.

Those computed cumulative statistics were the basis for calculating the mean team's statistics for each match by taking the mean over each player in the roster. This was done to study the impact of using aggregated vs disaggregated on the team-level data for prediction purposes.

The last feature added at this stage was the information about each team's placement in the global ranking. However, when examining the dataset, it was discovered that only about 50% of the matches were won by the team with the higher rank. This is why a new feature is proposed, where the rank trend is taken into account over some recent time window. So, for example, if a team was ranked 5th before some major event and after it dropped to 10th place, we denote that change as a decrease in performance. This, in theory, should better capture the state in which the team's form is at any given point in time.

All of the values were normalized before being used as training data for the machine-learning models.

5.4 Auxiliary information

Other than creating new features based on the existing player data, introducing new auxiliary information was tested. There are three areas in which additional information can be added, either using remaining parts of the dataset - for example, the "economy" data which is not strictly related to the players' statistics, or by looking for other sources of information - for example the official game update logs, to see when did a major game change occur that could affect the overall gameplay. Moreover, the information about roster changes can be extracted from the "players" dataset by establishing when a roster has changed one or more players.

A team's economy measures how much a given team has invested in equipment going into a round. Based on this metric, a team can play three main types of rounds: eco rounds, force-buy rounds, and full-buy rounds. Additionally, the first round of each half is the so-called pistol round, in which every player is equipped with only a pistol.

This distinction allows for calculating teams' win ratios in different types of economic rounds, adding new features and further granularity to the data. However, this poses a question of how to divide the rounds into those four available categories.

While the pistol rounds are not hard to classify, it is simply sufficient to consider rounds 1 and 16 for this category; the other three require setting sensible boundaries in the team's equipment value. As a rule of thumb, it can be assumed that an eco round corresponds to a situation where the given team has invested less than 5000 of in-game currency into their equipment. This is enough to purchase some of the lower-tier gear as well as a small amount of utility.

A force-buy round investment can be estimated at around 5000 to 20000 in-game currency. This allows more flexibility in the purchase of gear and corresponds to a situation where some of the equipment was carried over from the previous round. This value range corresponds to the valley in figure 5.3, where a small number of rounds fall. This is because, more often than not, it is better to play one eco round and save some of the funds.

Full-buy rounds are generally the most common, as in mid to late game, both teams have sufficient funds to purchase optimal gear for each team member. The value of equipment for this type of round is therefore set at over 20000, which is enough for a team consisting of five players to purchase a complete set of items.

The importance of winning any given type of round is different; for example, winning a pistol round is an excellent opening to a half for the victorious team, and besting the opponent while playing on a low budget can boost the team's morale. That is why this approach could be extended even further by adding weights to different win rations depending on the round category's importance.

The economy data is aggregated similarly to the teams' data in the previous step. At first, the total percentage of rounds of each type won by each team in a match is calculated, and then an expanding mean is used as a means of aggregation over a rolling time window.

Another way of introducing new information to the data is taking into account whether a team has made any changes to its existing roster in the past. It is a general notion that a team that has shuffled its players recently will need time to adjust, bringing down its overall performance. Since the "players" dataset comes with timestamps and dates, it is possible to extract such information from it.

There are two separate features created with this approach. The first one represents a situation when the team recently had a permanent player change. The second feature represents whether the team is playing the current match with a temporary stand-in. This is motivated by the fact that aside from the usual roster changes, where an organization may sign up a new player, sometimes there is a need for a temporary replacement for a particular player. For example, since CS:GO tournaments are played in various international locations, a given player may be unable to travel to an event, which is a fairly common occurrence.

This type of auxiliary information could be further improved by collecting the most recent news on any last moment changes or difficulties a team or a player may face, for example, from social media platforms, that can introduce unpredictability to the data.

Since CS:GO is constantly being developed, there are many instances where the way the game is played can change in a matter of days. In the past, such updates included the introduction of new maps and weapons or significant changes to them, changes to the core game mechanics, for example, the character movement, while other times being just cosmetic. This allows for collecting information about whether any update was introduced recently and whether this update had a major impact on the game.

Here again, two features were introduced. The first one is, similarly to the changes in the roster, an indication of whether or not an update to the game has happened in some recent time window. The second feature indicates the impact such an update had or could have on how the game is played. The introduction of such auxiliary information is motivated by the fact that after a significant change is made to the game, the players usually need time to adjust to it. For example, when a new map is introduced to the current pool, an old map is removed to keep the number of maps consistent. This might greatly inconvenience some teams if the removed map was their forte.

Possible update classes	
Update class	Description
Class 1	The least significant update, usually regarding some minor change to only one of the maps, or to the gameplay. This type of update, while not purely cosmetic, has historically had a minor impact on the way the game was played. **Example: Changing visibility of certain map areas.**
Class 2	An update of moderate significance. Generally, changes to weapons or maps that make them strategically different. **Example: Changing the price and accuracy of a weapon, making it more or less viable for competitive play.**
Class 3	A major update, that has the ability to single-handedly change the way the game is played. Most such updates introduce a new weapon or a new map to the competitive play. **Example: Introduction of a new map, "Anubis", on 18.11.2022, which replaced one of the previously played maps in the active pool.**

Table 5.3: Additional information about the classes into which the game updates were divided.

In table 5.3 are explained the classes into which the updates were divided. The distribution of the updates with regards to those classes is shown in figure 5.4.

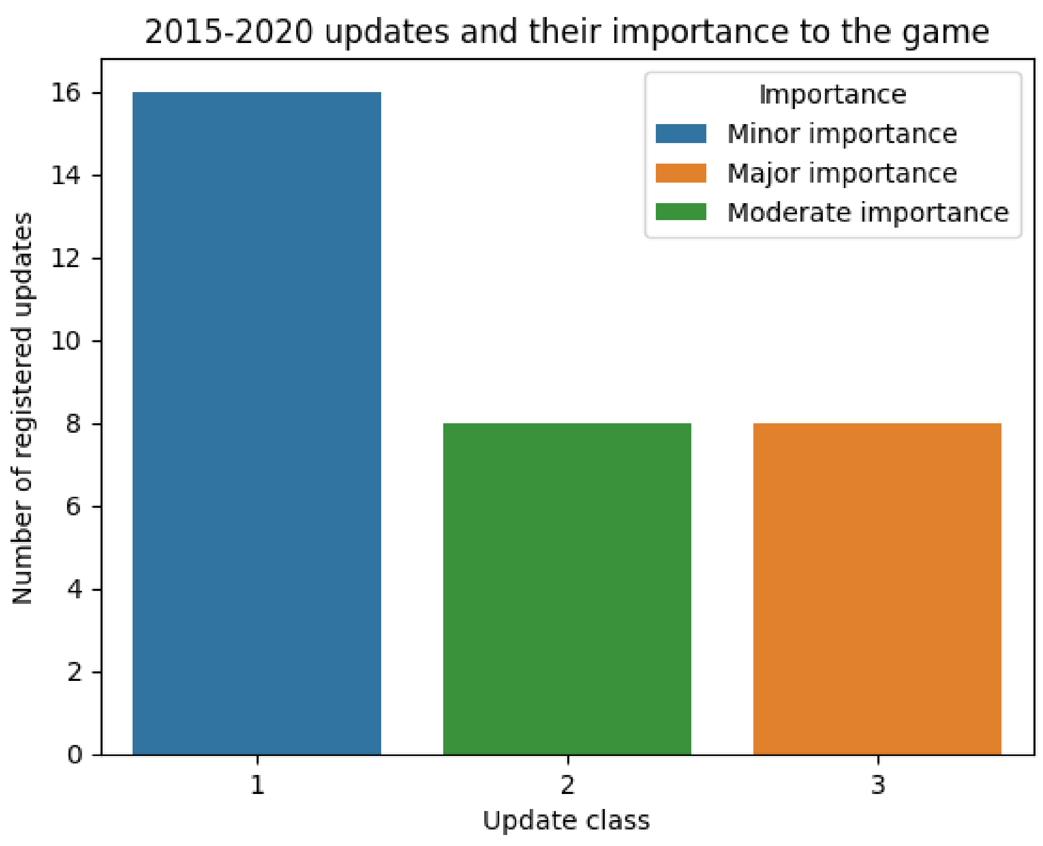


Figure 5.4: The number of updates registered between 01-01-2015 and 01-08-2020 that were deemed relevant to the players' performance.

As shown in figure 5.4, 32 updates to the game affected the maps or overall gameplay in the relevant period. Those updates were divided into three groups depending on their perceived impact on the player's performance. Minor update corresponds to a situation where only some changes were made to a singular map or weapon. Updates of moderate importance occur when more than one aspect of the game is changed or if the change affects the game's core mechanics; for example, a change in the price of the most commonly used weapon would fall in that category. Finally, a major update is considered an update where a new map or weapon is introduced to the game.

The impact of those auxiliary features was studied separately by adding them one by one to the base dataset of player statistics.

Proposed machine learning models

In this chapter, there are described the proposed neural network architectures, used later to generate predictions. The proposed models range from simple artificial neural network architectures to deep convolutional neural networks. First, the baseline models are showcased, and then the extensions made to those models are described, alongside their respective schematic representations. Additionally, the algorithms and processes specific to the CNN models are explained in detail.

6.1 ANN models

Artificial neural networks, often called feed-forward neural networks, are the exemplary architecture in the family of deep learning algorithms. The main goal of such a network is to approximate some function f. In the case of a classification problem, the network creates a mapping

$$y = f(x, \theta) \tag{6.1}$$

where the input vector x is used to determine the class y by learning the values of some parameters θ [5].

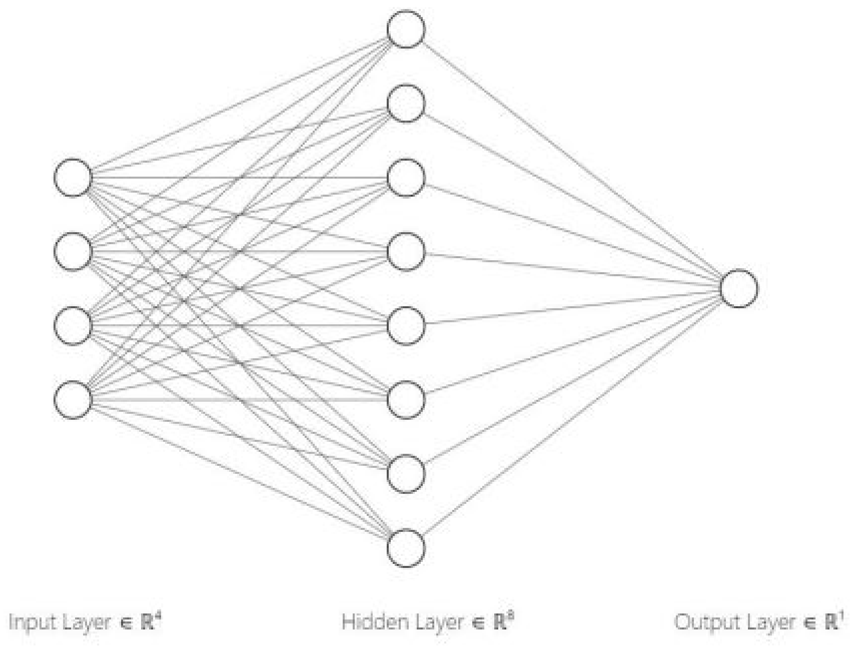


Figure 6.1: Simple ANN architecture with four input nodes, eight nodes in the hidden layer, and one output node for binary classification.

Networks of this kind generally follow a similar pattern; an input layer of neurons is used to feed the input vector x into the algorithm in batches. Then the output of this input layer is fed into the next. Layers of neurons that appear between the input and output layers are called **hidden**. Since each of the layers in an artificial neural network can be expressed as a function, the entire architecture can be expressed as a composition of some arbitrary functions [5]:

$$f_n(f_{n-1}(...f_1(x)...)) \tag{6.2}$$

The baseline ANN model consisted of four interconnected layers, as shown in figure 6.2.

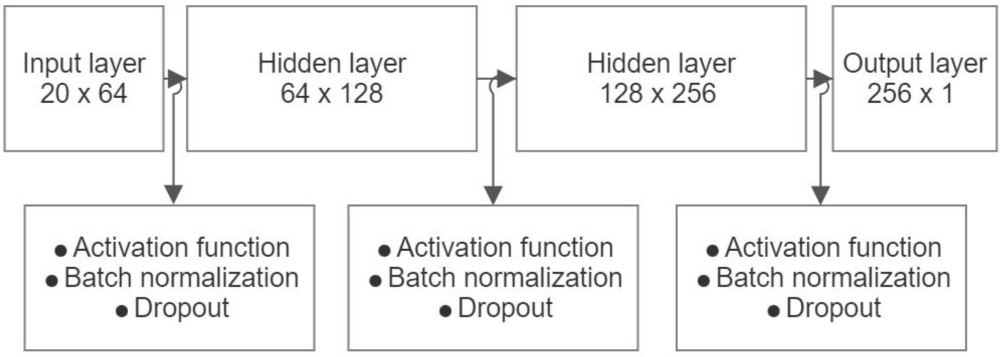


Figure 6.2: Schematic representation of the baseline ANN model used for experiments. After each layer, the output was passed through a block of operations, including activation, normalization and dropout. Then, the information was passed further, to the next layer of neurons.

After each layer of neurons, a selected activation function, batch normalization, and dropout of 50% of neurons were applied to the output. For this thesis, four activation functions were examined. The examined functions consisted of the simple ReLU activation, its modified version, Leaky ReLU, the Swish or SiLU function, and the Mish function. The ReLU activation function was selected as a baseline to which the effect of using different functions was compared.

Three different loss functions were tested for each activation function to add another level of granularity to the obtained results. First, the Binary cross-entropy function served once again as a baseline loss function, the Huber loss function, and the Weighted focal loss with parameters $\alpha = 0.95$ and $\gamma = 2.6$.

Aside from the baseline model, two extensions were made to the ANN architecture. First, the baseline model was made simpler by removing one of the hidden layers, in turn lowering the overall complexity of this method.

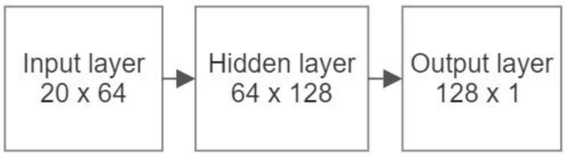


Figure 6.3: Schematic representation of the baseline ANN model after removing hidden layers.

Then the impact of quite the opposite was studied by adding more hidden layers to the baseline model.

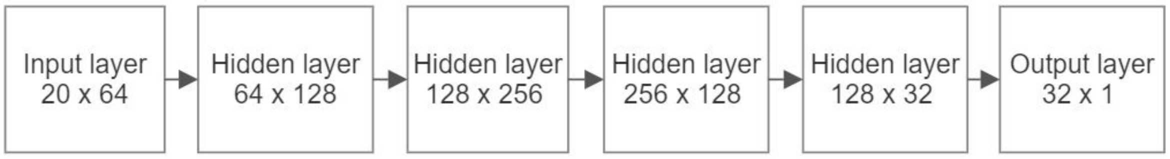


Figure 6.4: Schematic representation of the baseline ANN model after adding more hidden layers.

The resulting model architectures are shown in figures 6.3 and 6.4 and are later called the shallow ANN model and the Deep ANN model, respectively.

The chosen optimization algorithm for the majority of the models was the Adam optimizer; however, the recently proposed RAdam optimizer [28] was studied for selected scenarios to gather more information about the model's performance under different conditions.

6.2 CNN models

Convolutional neural networks, or **CNNs** for short, are a type of neural network used for processing data with a grid-like topology [5]. This means that CNNs are specifically designed to enable the recognition of spatial patterns in data. For example, in the recognition of samples that have a distribution related to some time interval, a 1-dimensional convolution operation can be used. The convolutional approach has also been successful in image recognition tasks where a 2-dimensional grid of pixels can be broken down into meaningful features. A convolutional neural network is a network where at least one of the layers applies a convolution operation instead of a standard matrix multiplication used in ANNs [5].

6.2.1 1D Convolution operation

The convolution operation works on the assumption that the neighbouring inputs in the grid-like data structure allow for the computation of some statistic that can be used as a new, more robust feature by the model. For example, in the case of a noisy signal, this may mean averaging several neighbouring measurements [5]. The one-dimensional convolution operation can be expressed as [5]:

$$(F * K)(i) = \sum_{j=1}^{m} K(j) * F(i - j + \frac{m}{2}) \tag{6.3}$$

F is the one-dimensional input array, and K is the one-dimensional kernel. The kernel is an array of weights sliding over the input creating a new output array that contains values computed according to the above formula. This output array is called a feature map. Based on the used kernel, different feature maps are created that can be used to teach the model new patterns in data. A sample representation of a convolution operation using a sliding kernel of length three is illustrated in the figure below.

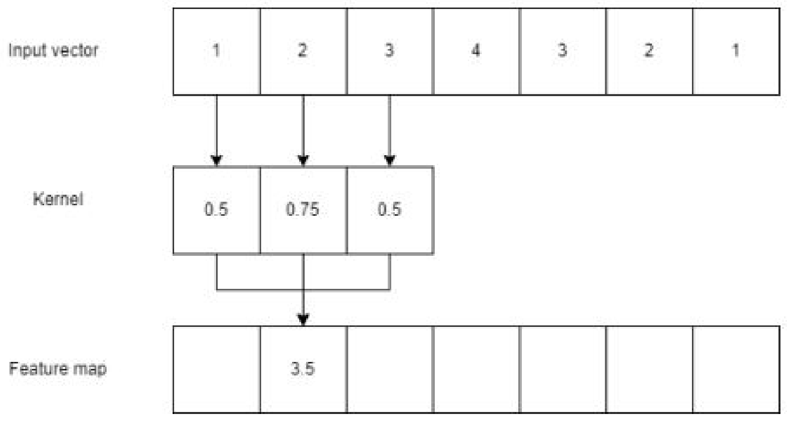


Figure 6.5: Schematic representation of the feature map computation using a sliding kernel of length 3.

Regarding the convolution operation, there are two terms worth mentioning. First is the padding often used as an input parameter to a convolutional layer. The padding helps maintain the shape of the initial input and ensures that the boundary input values are treated equally while computing the convolution[29].

The second term that is connected to the convolutional method is stride. Stride is a hyperparameter that controls how fast, meaning how big of a step is taken by the kernel when computing values for the feature map. The larger the stride, the faster the kernel moves along the input array resulting in a shorter output [29].

6.2.2 Pooling operation

The pooling operation in a convolutional neural network is a way to modify the output of a network layer by computing a deterministic function over some neighbouring values in the output array. The maximum (max-pooling, shown in 6.6) or average functions are most often selected for that role.

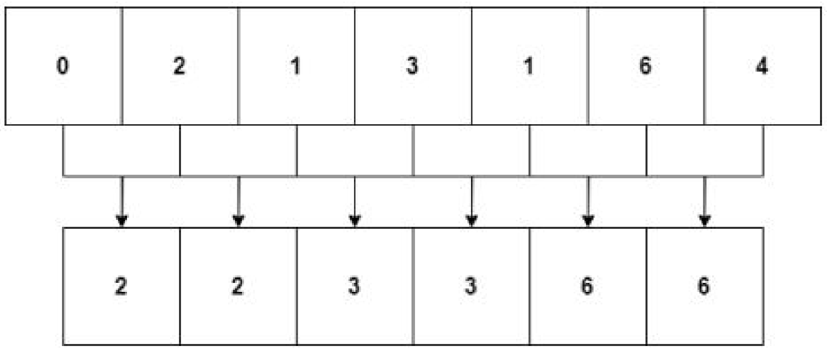


Figure 6.6: Schematic representation of a max-pooling operation for window size two and stride 1.

There are several reasons why the pooling operation is used in a CNN. One of the reasons is that after a pooling operation is applied, the output of a layer becomes invariant to a small translation in the original array. This means that the location of the features in the input is less important, and the emphasis is put on its overall presence [5, 29]. Another reason for having a pooling step in the CNN model architecture is that such a process allows the network to focus more on the most dominant information in the dataset by disregarding the less critical features and parameters. This, in turn, often leads to a faster learning process [29].

6.2.3 Proposed architectures

The baseline CNN model used for the purpose of this thesis consisted of two parts. First, the convolutional part of the model was made up of layers performing the one-dimensional convolution operation. After each of those layers, a max-pooling operation with a kernel of size two was used. The resulting feature map was then sent to the non-convolutional part of the model in order to perform the final classification. This part consisted of 4 fully connected layers interlaced with activation functions, batch normalization, and dropouts, similar to the process used in the case of the ANN architecture. The structure of the baseline CNN model is shown in figure 6.7.

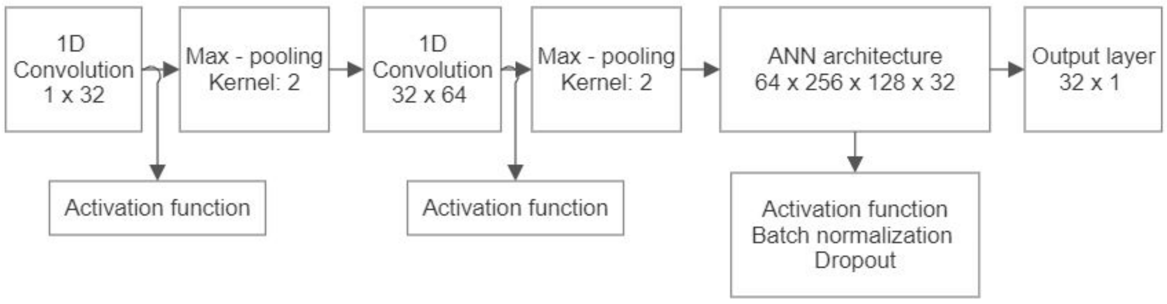


Figure 6.7: Schematic representation of the baseline CNN model's architecture. After each layer of convolution, an activation function was used. The normalization and dropout were only applied in the ANN part of the model.

For the CNN model, the same combinations of activation functions and loss functions were studied as in the case of the ANN architectures mentioned before. Similarly, the Adam and RAdam optimizers and their impact on the model's performance were examined.

To study the effect of changing the model's depth, a VGG convolutional model architecture was adapted to perform the one-dimensional convolution operation. The VGG model is a very deep convolutional network, initially designed as a base model for image recognition tasks [30]. It is, however, the effect of studying how to make the convolutional networks deeper to increase their performance, so an argument could be made that the same principles should apply to other tasks. Furthermore, there have been successful experiments showcasing the use of deep CNN architectures adapted for tasks requiring one-dimensional convolution to be performed [31]. This is why such an approach was considered valid.

The exact architecture of the VGG model used for this thesis was the VGG11 architecture, shown on the schema in figure 6.8.

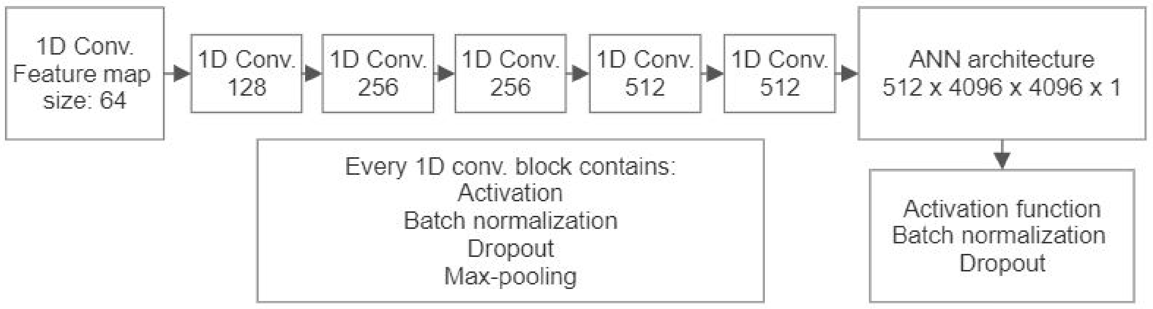


Figure 6.8: Schematic representation of the VGG11 model's architecture. This architecture was constructed according to [30]. After each layer of convolution, the activation, normalization, dropout and pooling operations are applied.

In the proposed VGG11 model's architecture, each convolutional layer has a kernel of size three and padding of size 1. Additionally, the max-pooling operation is performed using a kernel of size 2 with a stride of 2. Finally, an average pooling operation is performed between the convolutional and ANN parts of the model.

Results

The following chapter is devoted to showcasing the results of experiments conducted for the purpose of this thesis. The obtained results are divided into two categories depending on the underlying neural network architecture that was used to obtain them.

The results of a machine learning experiment can be measured in different ways. From simple prediction accuracy to precision, recall, and more advanced methods, selecting an appropriate evaluation metric can significantly affect how the result is presented and, in turn, perceived. For this particular problem, however, a simple accuracy measurement was deemed sufficient. Since the models presented in this thesis only aim at predicting the probability of one of two teams winning a match, neither of the predicted labels has any particular importance assigned to it. Another way of measuring the models' performance was used to gain a more in-depth understanding of the tested models, namely the inference time. Inference time is a measurement of the time it takes for a trained model to make a prediction for a set of unseen examples. For the purpose of this research, the inference time was calculated for each batch, consisting of 64 samples, and then averaged over all the data batches for better results.

The models were evaluated using the cross-validation technique. This means that splitting the available dataset into chunks made it possible to train the model on different parts of the data while using the remaining entries for validation. This process allows for a more robust evaluation, as averaging the results over n folds minimises the randomness and bias some parts of the dataset may introduce when used for training or testing.

The ANN and CNN architectures were trained using a set of selected activation and loss functions. Then, for each architecture, the top three performing configurations were chosen for further tests, which included modifying the initial model's architecture, modifying the data, or adding auxiliary information.

7.1 ANN models

First are the results of training the baseline ANN model for different loss functions. Shown in figure 7.1 are the average accuracy measurements from the 5-fold cross-validation process. The error bars represent the lowest and highest accuracy values achieved.

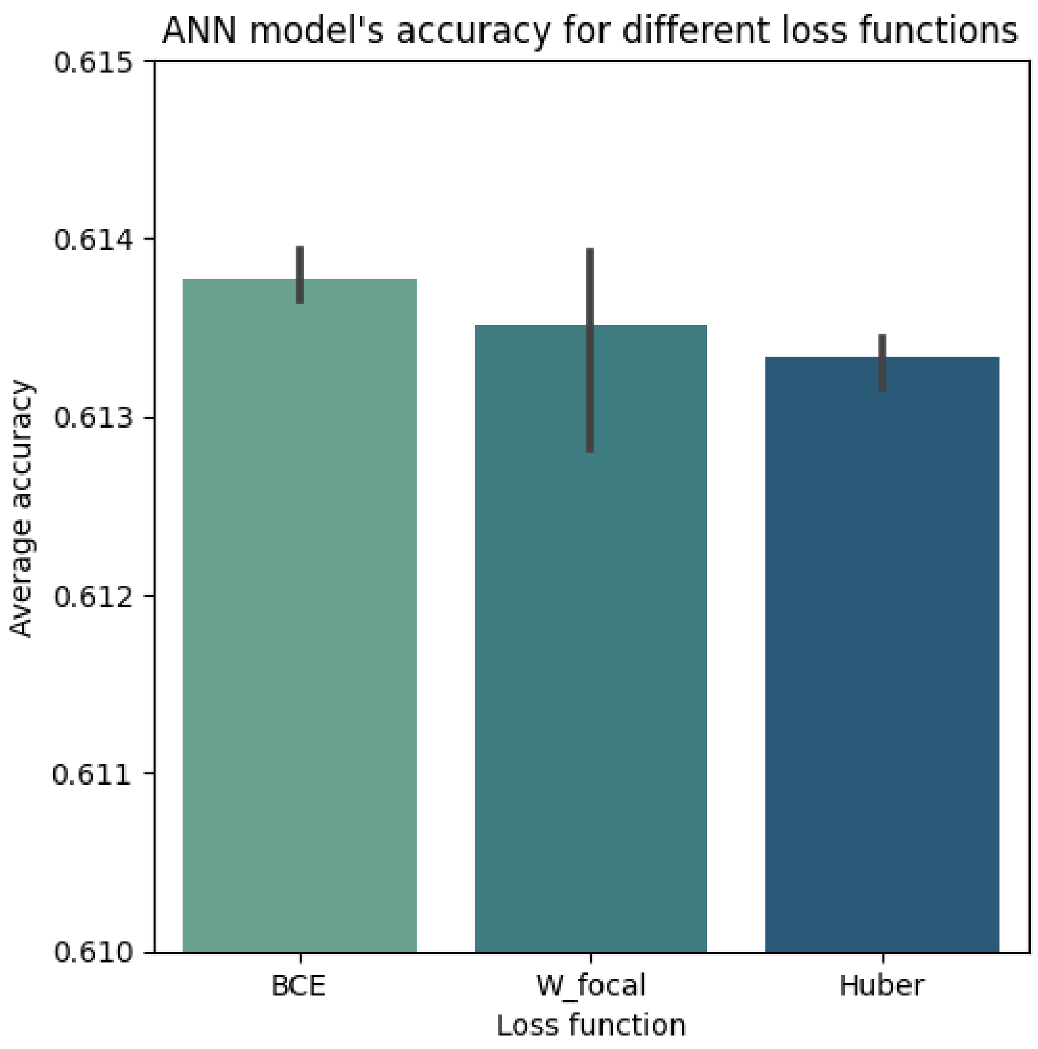


Figure 7.1: Average accuracy of the baseline ANN model for different loss functions.

It can be seen that even though the differences in the average accuracy are not significant, there is a slight advantage in using the BCE loss function. Moreover, the error bar corresponding to the Weighted focal loss function indicates that the accuracy is the highest for some of the tested conditions, namely for a particular activation function.

Figure 7.2 shows the average inference time with respect to different loss functions.

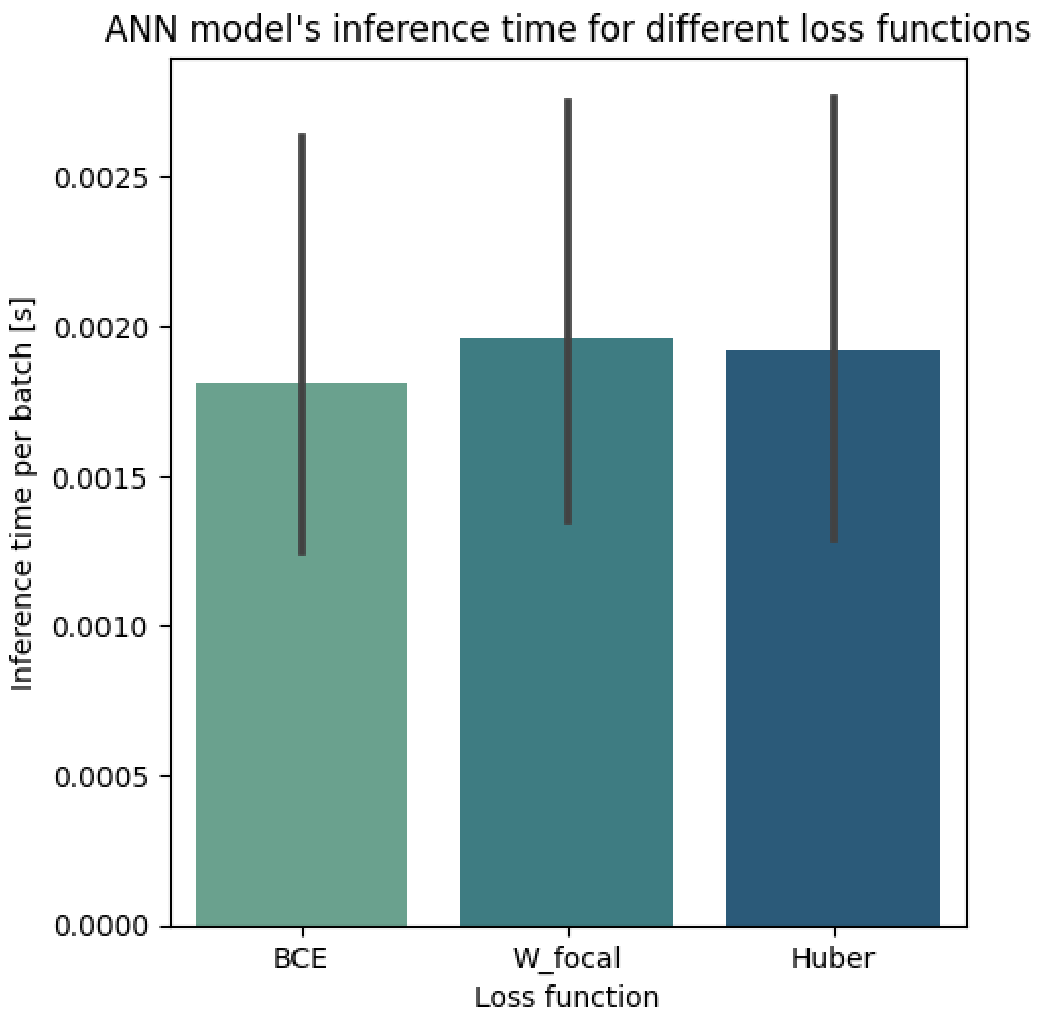


Figure 7.2: Average inference time for the ANN model for different loss functions.

Below are the results of training the same baseline ANN architecture; this time, however, concerning different activation functions used between the fully connected layers.

Figure 7.3 shows the average accuracy measurements for each function averaged over the three loss functions used.

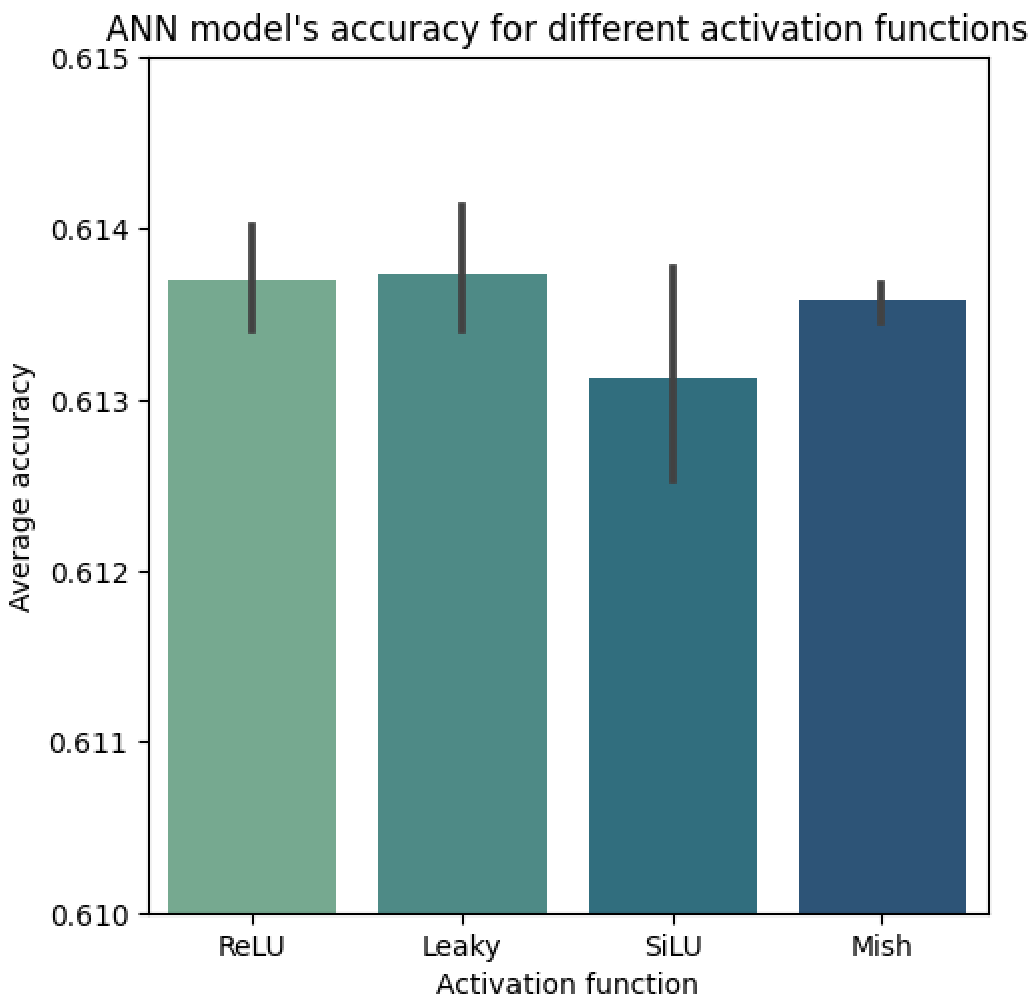


Figure 7.3: Average accuracy of the baseline ANN model for different activation functions.

It can be seen that the ReLU and Leaky ReLU functions performed almost identically in terms of accuracy. In contrast, the SiLU function performed significantly worse, with the exception of one experiment, as indicated by the high error bar. The mish function performed slightly worse than the ReLU variants, but the results obtained using mish were mostly unaffected by the change in the loss function.

Figure 7.4 shows the average inference time with respect to different activation functions.

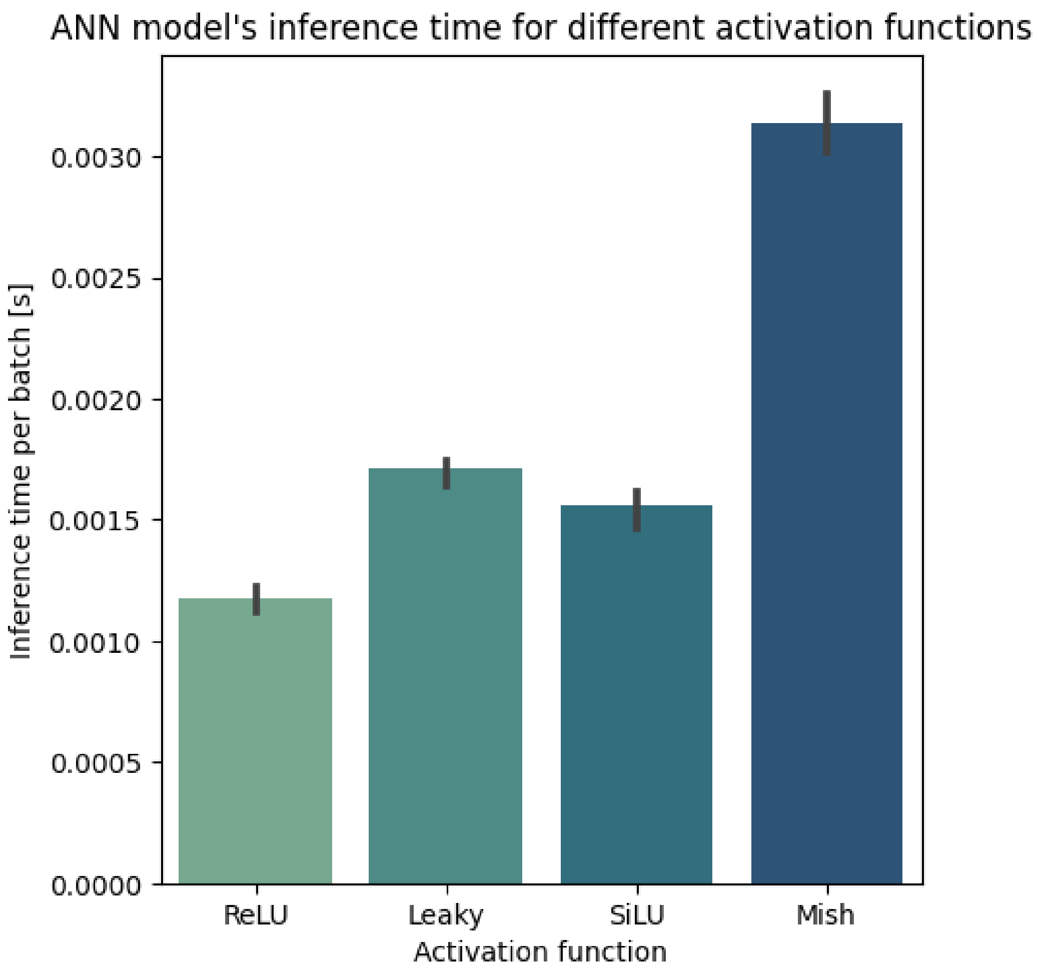


Figure 7.4: Average inference time for the ANN model for different activation functions.

After those initial experiments were carried out, three combinations of loss functions and activation functions were selected, using the average validation accuracy in the cross-validation process as the only criterion in their selection. Table 7.1 lists the three mentioned configurations with their respective average, highest, and lowest recorded prediction accuracy values.

Top 3 performing ANN model configurations					
Loss	Activation	Average Accuracy	Highest Accuracy	Lowest Accuracy	Inference time [s]
W. Focal	Leaky ReLU	61.41%	62.40%	60.66%	0.0017
BCE	ReLU	61.40%	62.32%	60.62%	0.0011
BCE	SiLU	61.38%	62.30%	60.66%	0.0015

Table 7.1: The baseline ANN model's configurations that achieved best results.

Using those configurations, the base model was trained again, each time with a different change to either the dataset (introduction of auxiliary information) or the model's architecture.

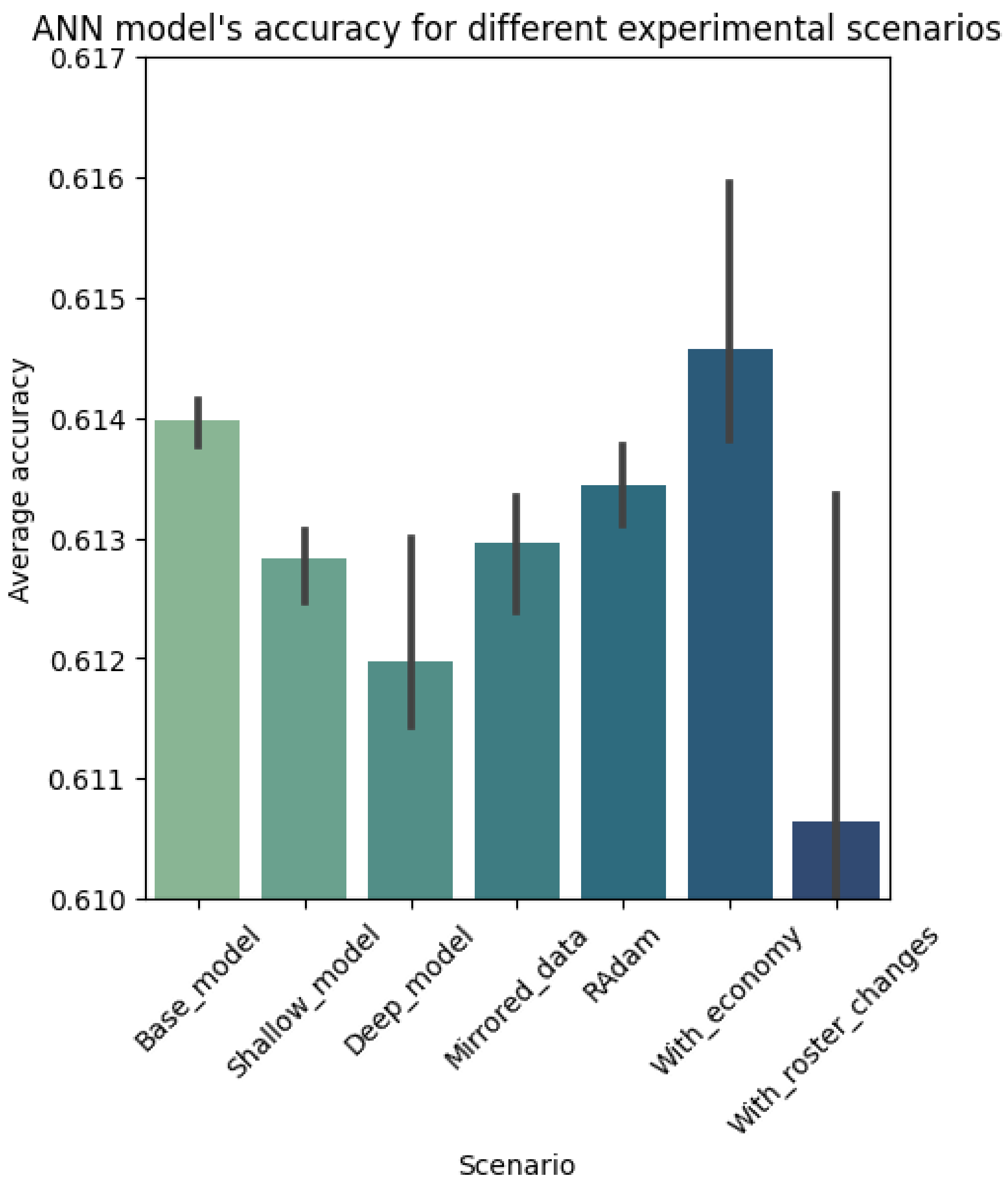


Figure 7.5: Average prediction accuracy for the ANN network under different testing conditions.

And so, the addition of economy data, information about roster changes, and information about game updates were introduced to the training data. Moreover, the effect of the data augmentation, in this case, mirroring of the teams, was examined. The shallow and deep ANN model variations and the impact of selecting the RAdam algorithm as the optimization tool were also tested. In figure 7.5 shows the average prediction accuracy values obtained for each of those extensions.

It can be seen that the only improvement in the accuracy of the model was observed after the economy data was introduced to the dataset. The gain in accuracy at its peak was measured at around 0.3 pp over the average for the baseline model.

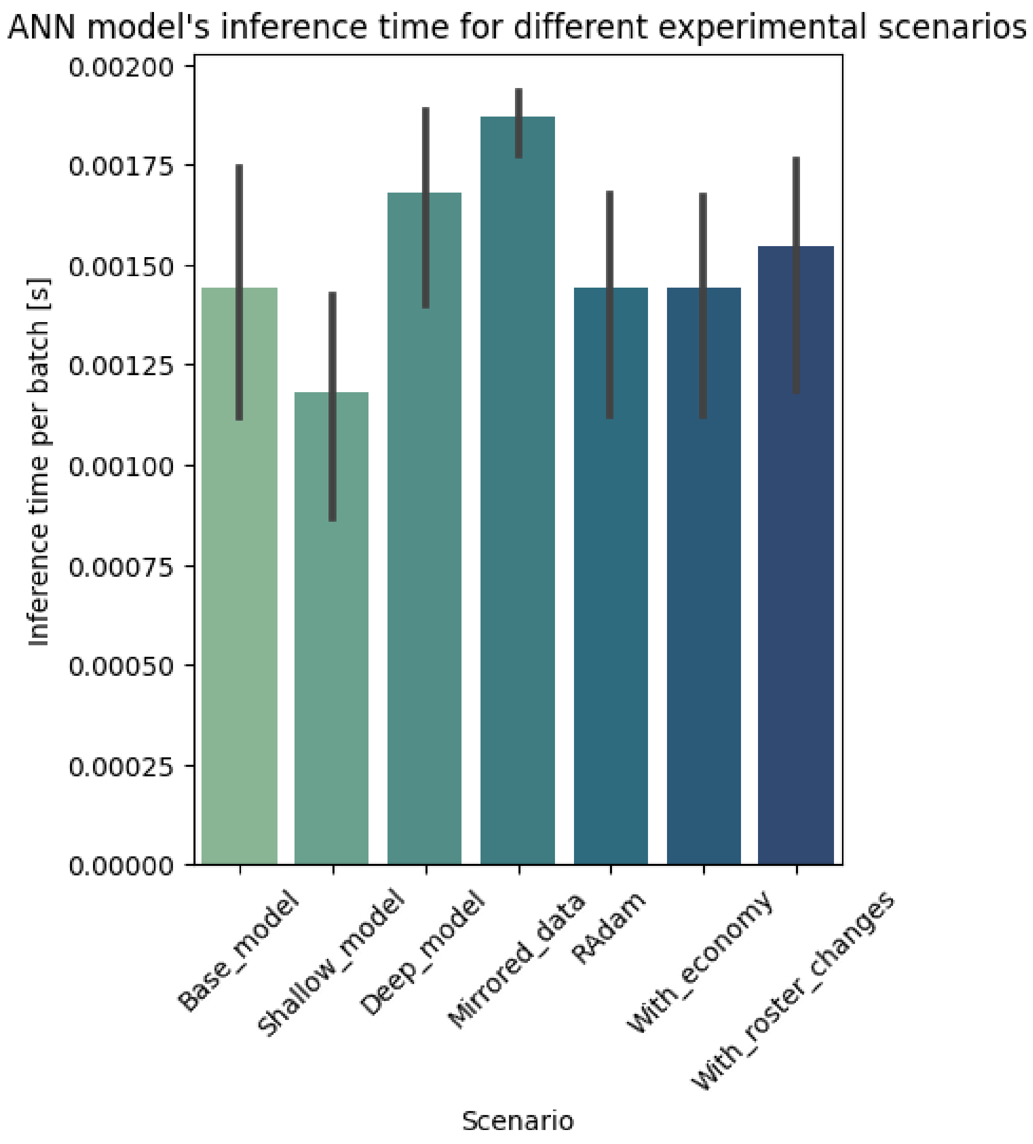


Figure 7.6: Average inference time for the ANN network under different testing conditions.

Figure 7.6 shows the average inference time for the different scenarios. It can be seen that the shallow model and deep model needed less and more time to obtain a prediction, respectively. Regarding the auxiliary information, there was no significant change in the inference time compared to the base model.

Up to this point, all of the mentioned results were obtained using the aggregated team statistics. To see whether this approach results in any noticeable difference over the non-aggregated player statistics, the best-performing ANN architecture was trained on a dataset consisting of each player's statistics separately. The results of this experiment are shown in table 7.2.

Performance of the ANN model on non-aggregated player data					
Model	Loss	Activation	Average Accuracy	Highest Accuracy	Lowest Accuracy
Baseline model	W. Focal	Leaky ReLU	60.13%	60.81%	58.97%
Baseline model	BCE	ReLU	60.24%	61.05%	58.97%
Baseline model	BCE	SiLU	60.19%	60.94%	58.61%
Deep model	W. Focal	Leaky ReLU	60.24%	61.05%	59.01%
Deep model	BCE	ReLU	60.27%	61.28%	59.01%
Deep model	BCE	SiLU	60.25%	61.13%	58.84%

Table 7.2: The baseline and deep ANN models' configurations that achieved the best results used for prediction on the non-aggregated dataset.

It can be seen that the accuracy results obtained when using the non-aggregated data are worse than those obtained before, using averaged teams' statistics, as none of the models' managed to beat the 61% accuracy mark on average. This may be due to the models being overloaded with features since the non-aggregated dataset contained five times as many individual fields. This, in turn, may have caused the models to be unable to find as many meaningful patterns in the data. Since the use of non-aggregated players' statistics did not offer any improvement, it was decided that from this point onward, only the aggregated team statistics will be used for further model training.

7.2 CNN models

The baseline CNN model presented earlier in figure 6.7 was trained similarly to the ANN model, with 12 overall configurations corresponding to the three selected loss functions and four activation functions. The average accuracy for the model during the cross-validation process with respect to the different loss functions is shown in figure 7.7.

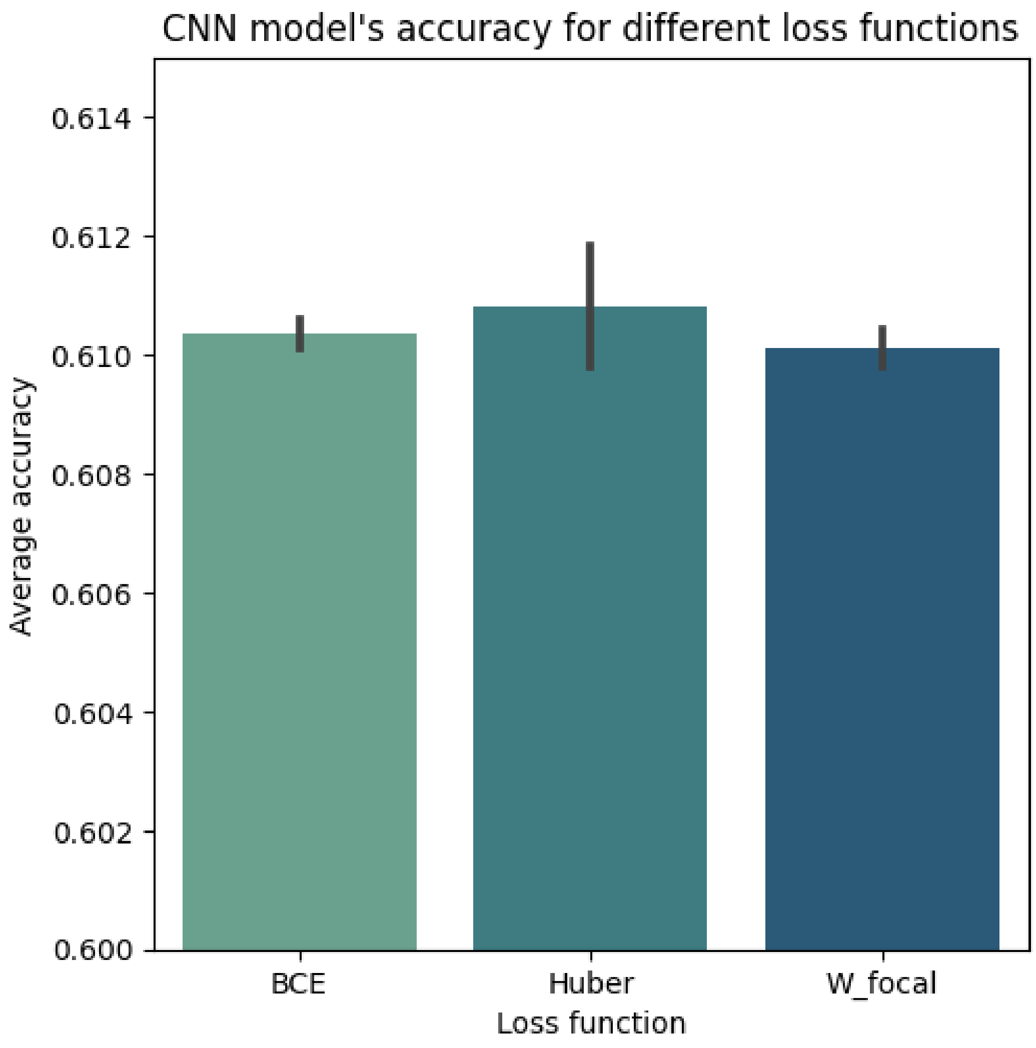


Figure 7.7: Average accuracy of the simple CNN model for different loss functions.

In the above plot, it can be seen that there is not much difference in accuracy across the different loss functions for this particular model. However, the loss function with the highest accuracy registered is the Huber loss, as marked by the highest point of the error bar.

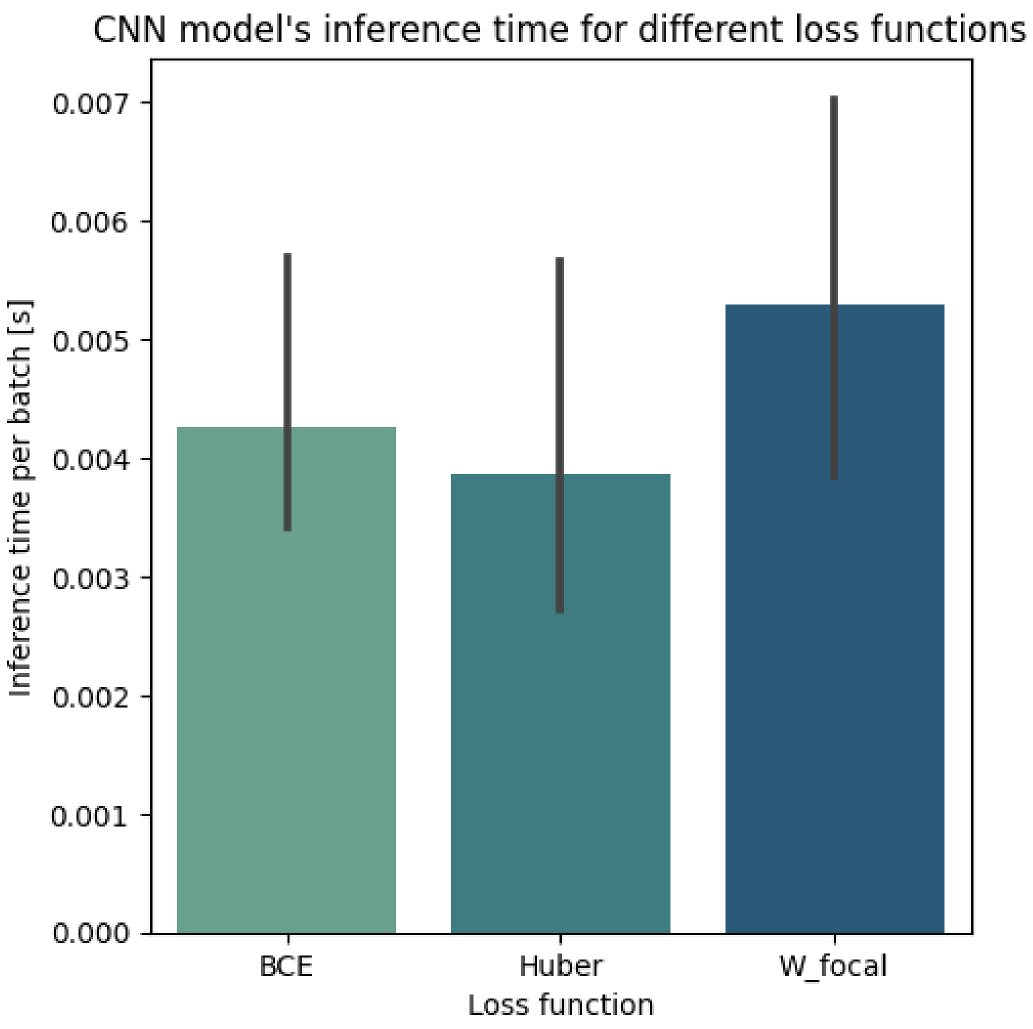


Figure 7.8: Average inference time for the simple CNN model for different loss functions.

Figure 7.8 shows the inference time recorded for this model with respect to different loss functions. It can be seen that the inference time per batch of 64 samples is the lowest for the Huber loss, which also happens to be the best-performing loss function in terms of accuracy.

In figure 7.9, the average accuracy measurements are shown with respect to the different activation functions used. It once again can be seen that there is not much difference in the accuracy between models using different activation functions. However, the slight advantage of the SiLU and Leaky ReLU functions is noticeable and was considered when conducting further experiments.

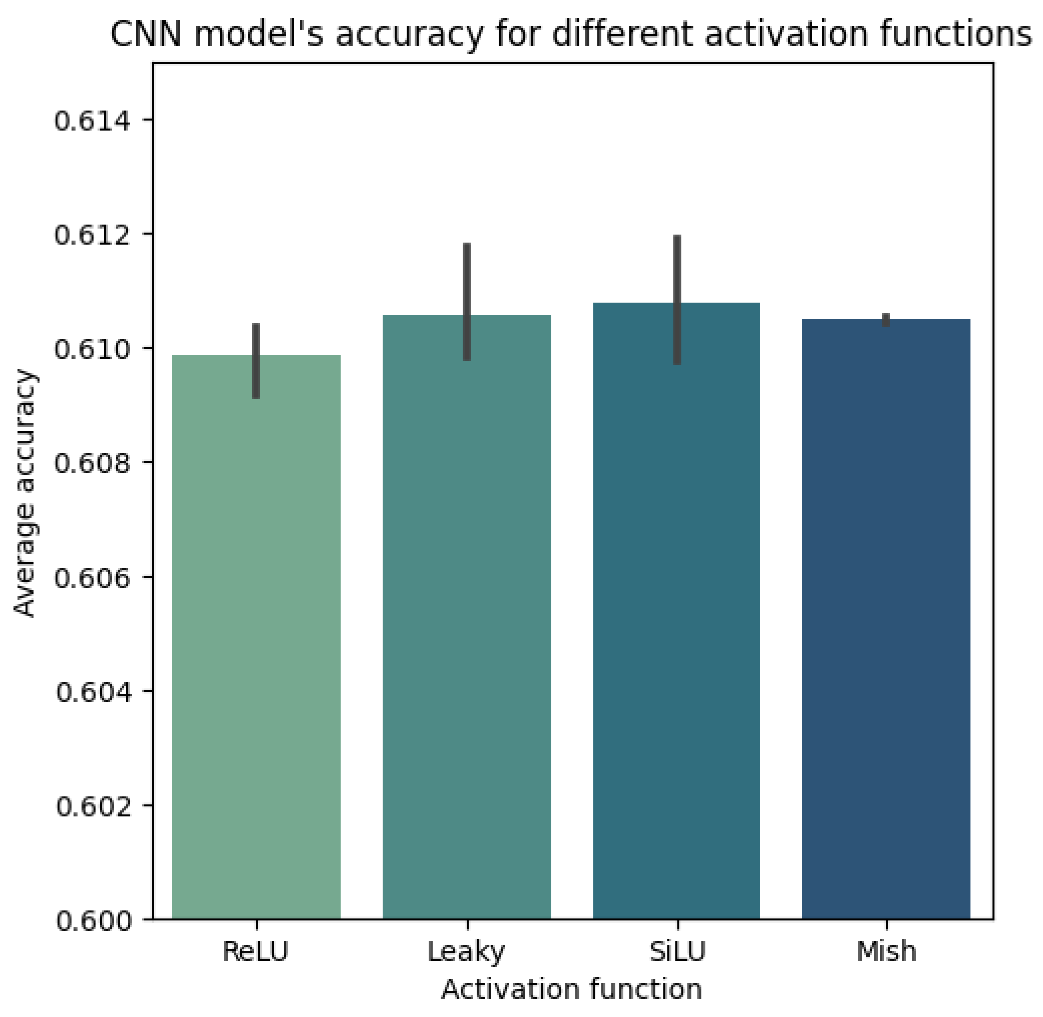


Figure 7.9: Average accuracy of the simple CNN model for different activation functions.

Figure 7.10 shows the average inference time for the proposed simple CNN model with respect to different activation functions. The use of the Mish activation function makes the inference process much longer than for the remaining three activation functions. For this reason, and because the use of the Mish function did not significantly increase accuracy, it was excluded from further experiments.

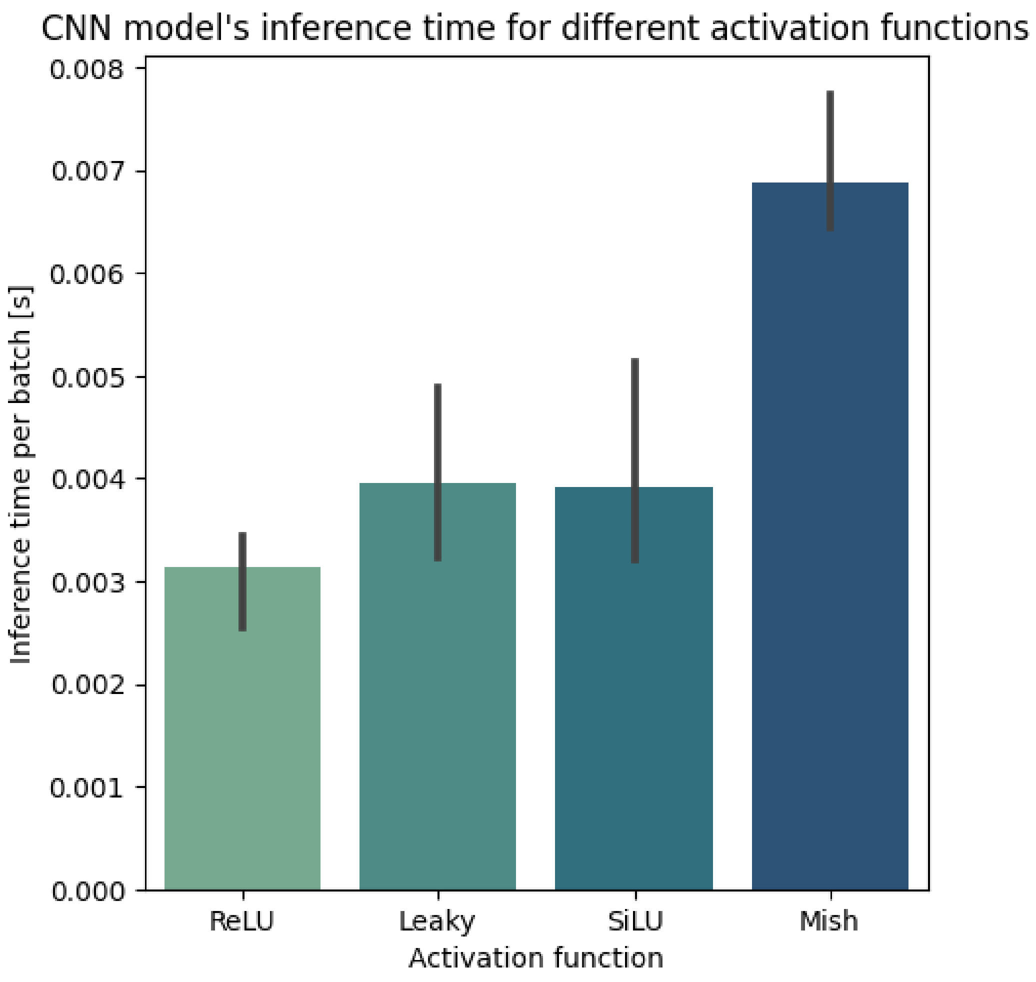


Figure 7.10: Average inference time for the simple CNN model for different activation functions.

Similarly to the ANN model, the three best-performing in terms of average accuracy combinations of loss and activation functions were selected for further tests. The selected combinations with their respective results are showcased in table 7.3.

Top 3 performing CNN model configurations					
Loss	Activation	Average Accuracy	Highest Accuracy	Lowest Accuracy	Inference time [s]
Huber	SiLU	61.19%	62.27%	60.08%	0.0032
Huber	Leaky ReLU	61.18%	62.27%	60.14%	0.0032
BCE	SiLU	61.07%	61.94%	60.53%	0.0034

Table 7.3: The simple CNN model's configurations that achieved best results.

For those best-performing configurations, several tests were performed, namely introducing the RAdam optimizer and adding auxiliary information to the data. The data augmentation by mirroring data samples approach was dropped, as there was no sign of improvement in accuracy or inference time in the ANN network's results using this method. Figure 7.11 shows the average accuracy measurements taken from the three selected configurations.

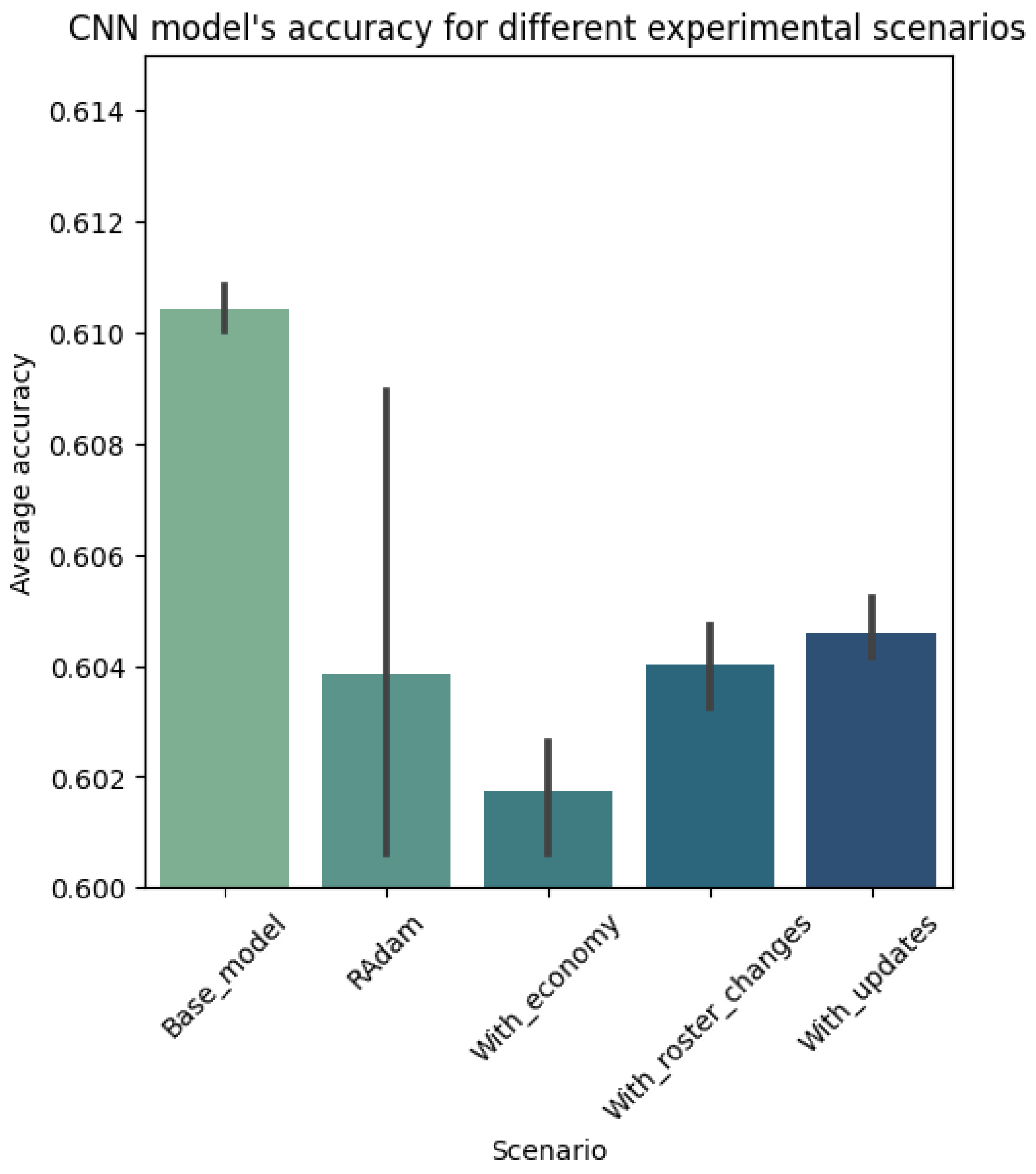


Figure 7.11: Average prediction accuracy for the simple CNN network under different testing conditions.

It turned out that none of the proposed changes provided better accuracy; in fact, they all significantly lowered the score obtained with the best iteration in the simplest of cases. Moreover, using the RAdam algorithm as the optimizer resulted in increased variance in the results for different loss and activation function configurations, as indicated by the extended error bar.

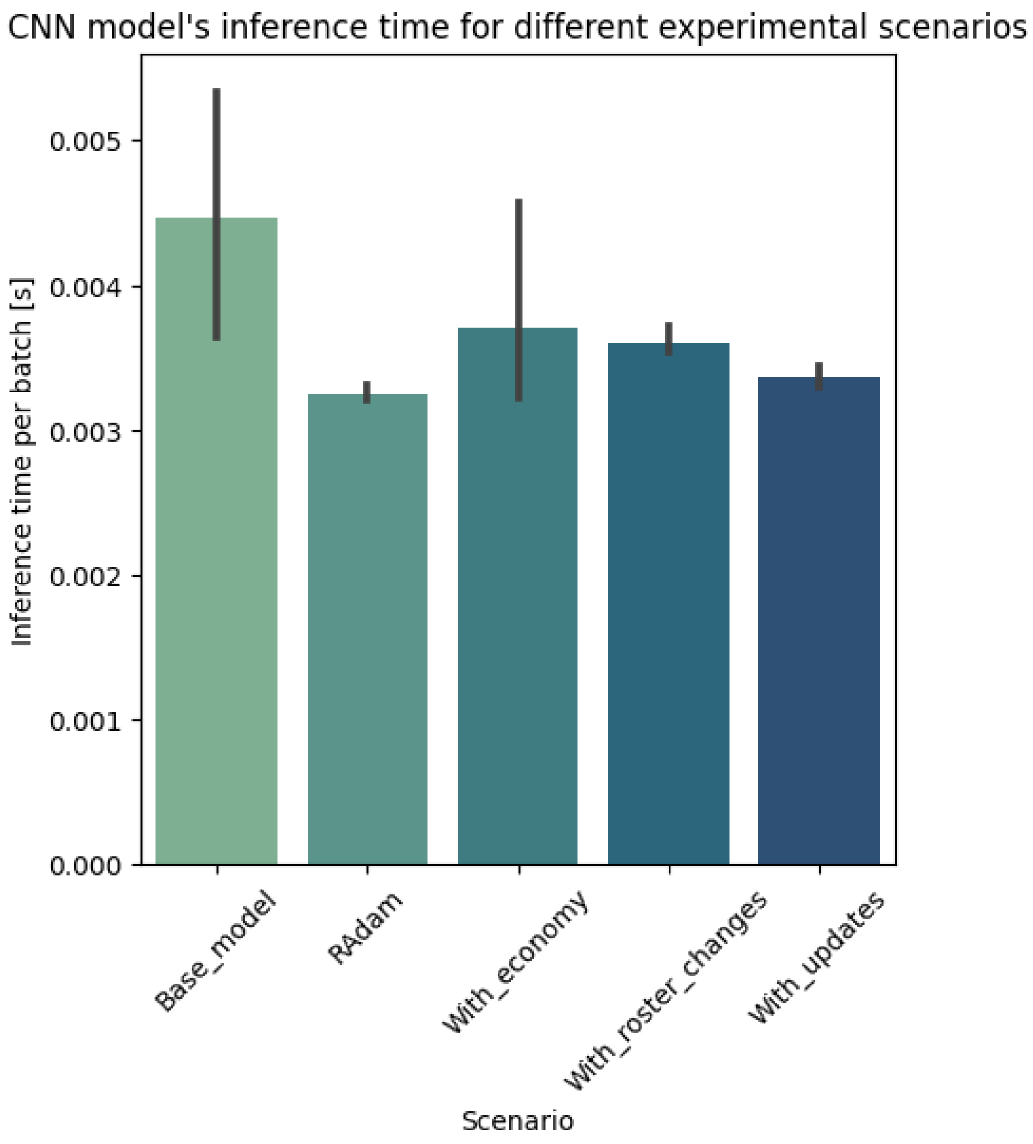


Figure 7.12: Average inference time for the simple CNN network under different testing conditions.

In Figure 7.12, the inference time for each of the testing scenarios is presented. In the case of inference time, all of the proposed changes managed to better the baseline model. Notably, using the RAdam optimizer, the network managed to lower the average inference time by around 30%, depending on the individual case.

Everything learned during the CNN training process was also applied to the selected VGG model. So it was trained using the BCE loss function and ReLU activation as a baseline, and the three best-performing configurations from the simple CNN model were used. In figure 7.13, there are the accuracy measurements taken for each of those configurations individually. The bars show the average accuracy obtained during the cross-validation process.

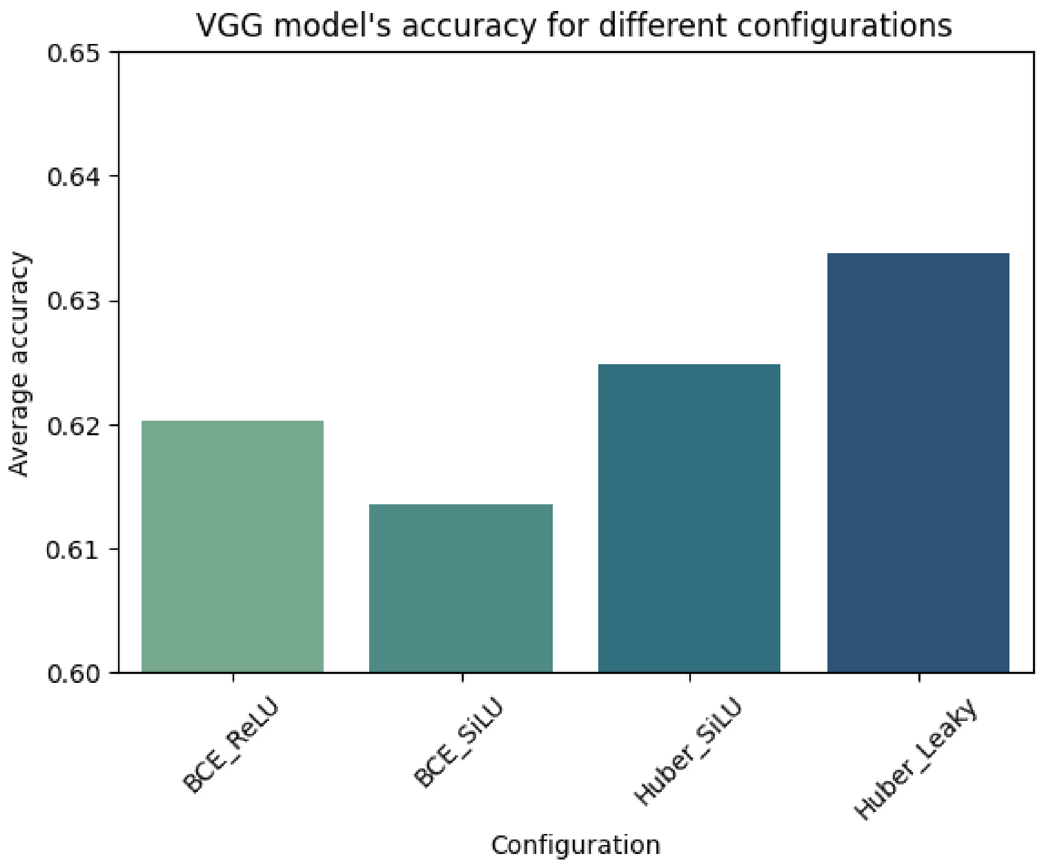


Figure 7.13: Average accuracy for the deep CNN VGG11 network for different configurations of Loss function - Activation function pairings.

It can be seen that two of the three previously best-performing configurations when applied to the deep VGG model, provide better accuracy results than the standard BCE-ReLU pairing. Furthermore, the increase in accuracy when using the Huber loss function is significant, reaching over 1 pp advantage over the standard VGG architecture configuration.

In figure 7.14 is the inference time recorded for each of the proposed variants in relation to the standard model.

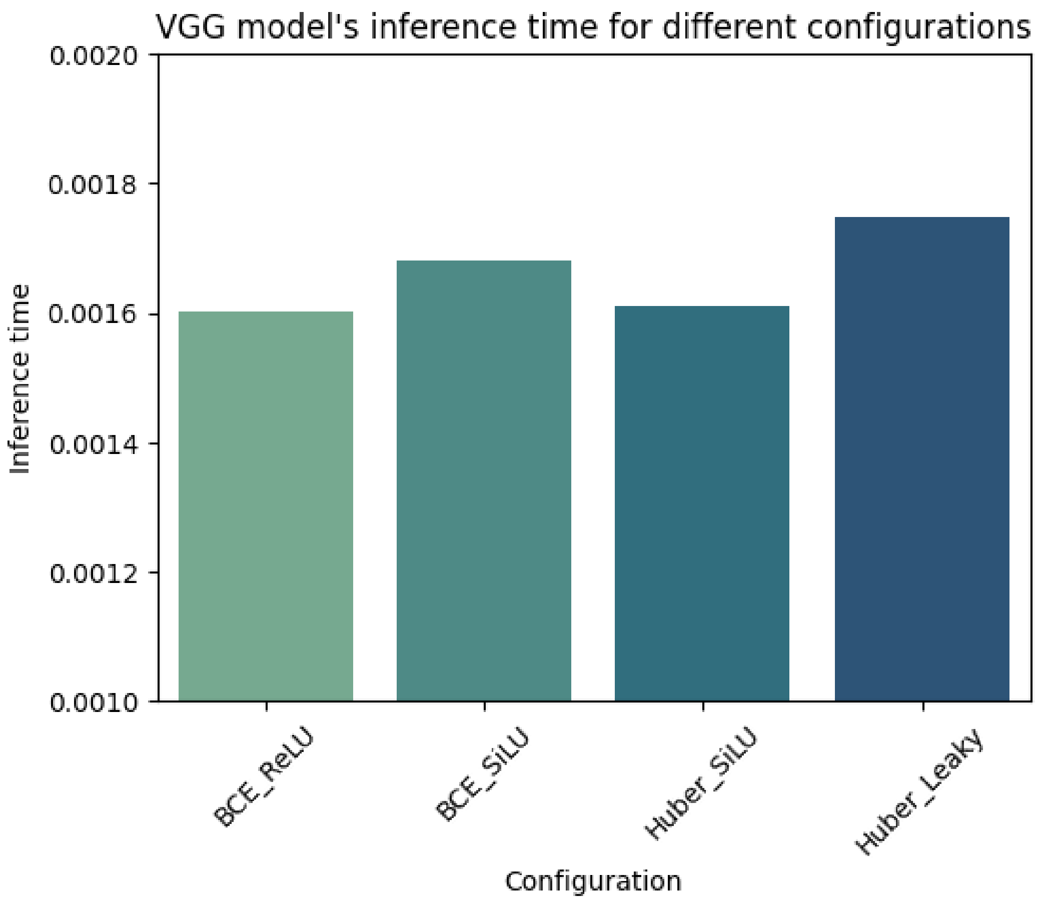


Figure 7.14: Average inference time for the deep CNN VGG11 network for different configurations of Loss function - Activation function pairings.

There is a slight increase in the inference time when using the Huber loss or applying the SiLU and Leaky ReLU functions to the model's layers. However, this minor inconvenience can be overlooked, given the increased accuracy.

Given that the VGG model managed to achieve better results than the previously tested models on the base version of the dataset, it was then trained on its extended versions. For this, the best-performing variant, the Huber loss with Leaky ReLU activation, was used on a dataset containing each of the three auxiliary sets of information mentioned before. In figure 7.15 are the accuracy measurements with respect to the different versions of the dataset used.

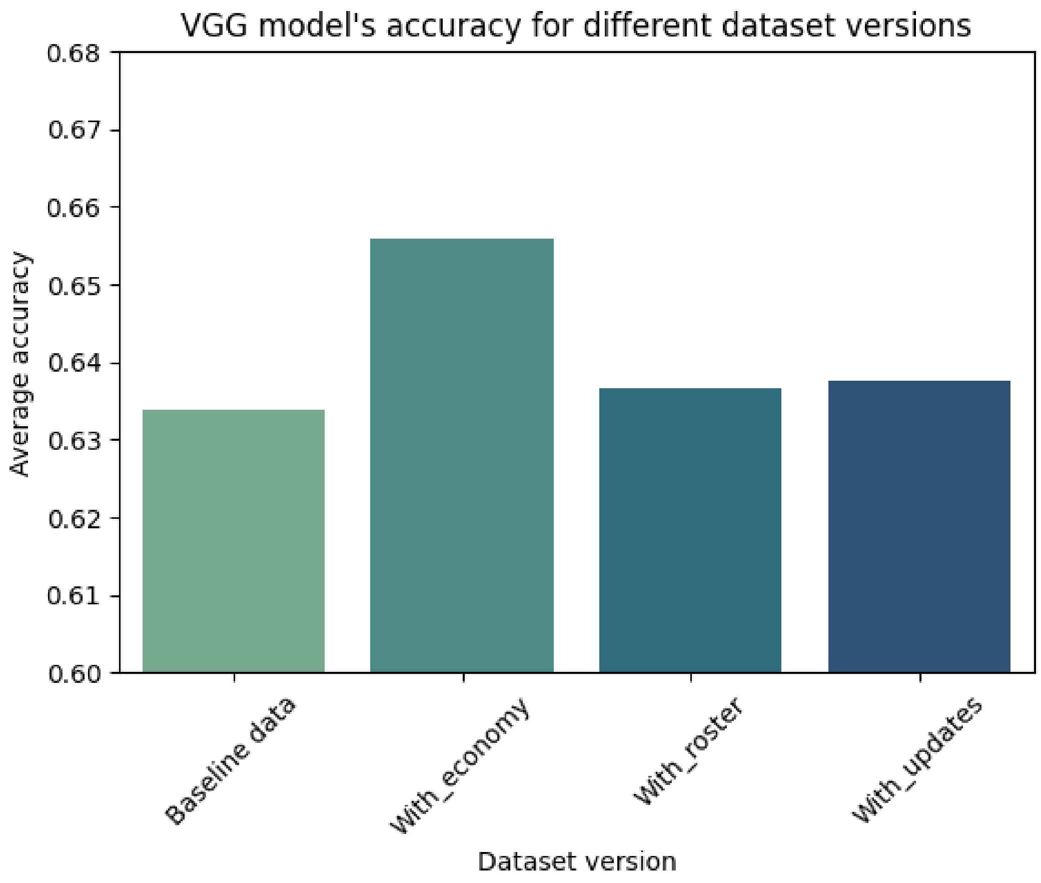


Figure 7.15: Average accuracy for the deep CNN VGG11 network for different dataset versions.

While there seems to be a slight improvement in accuracy when the team roster changes and the game updates are added to the dataset, by far, the best result is achieved when the economy data is introduced. With this addition, the average accuracy of the VGG model increased from 62% to 65.5%, which is a significant improvement.

Below, in figure 7.16 is the inference time measured while performing experiments using different dataset versions on the VGG11 model. The introduced dataset changes did not significantly impact the inference time, with the most notable change corresponding to the case where the information about team roster changes was introduced. Interstingly, the dataset version, which contributed to the most significant improvement in accuracy, also resulted in a slight decrease in inference time, making it superior to the baseline data overall.

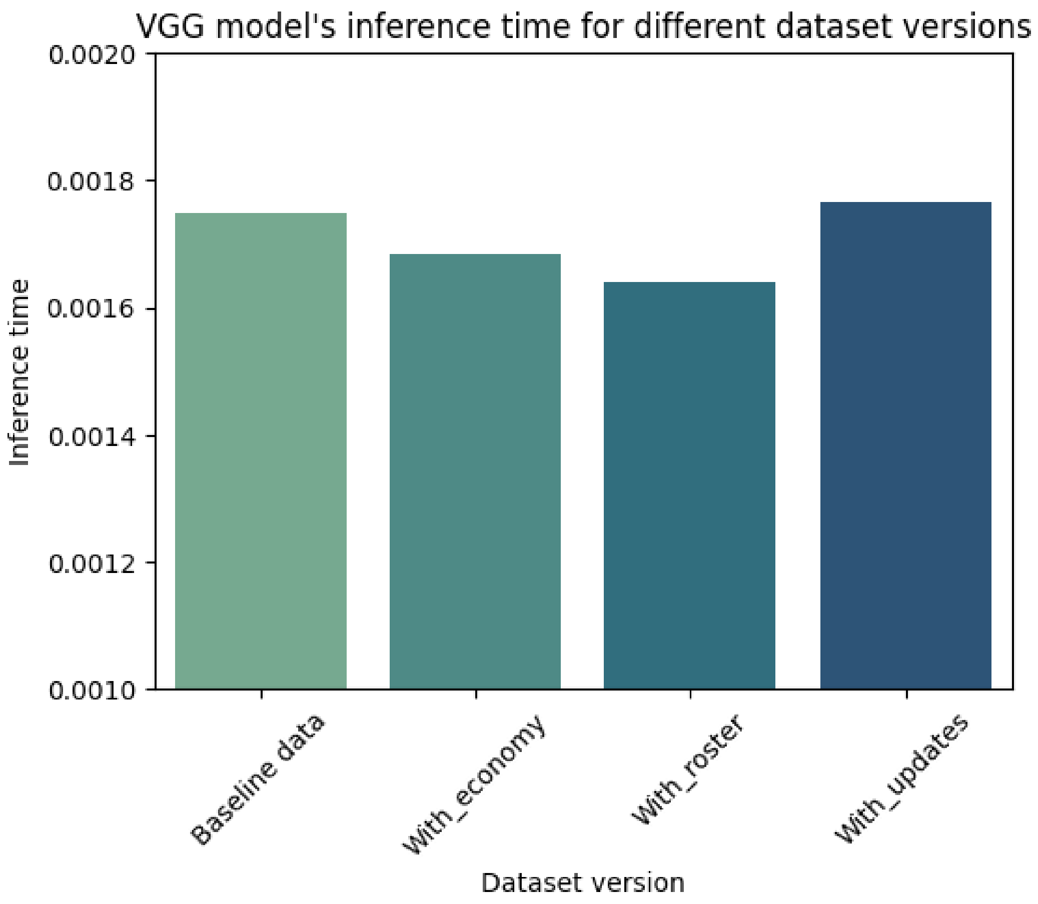


Figure 7.16: Average inference time for the deep CNN VGG11 network for different dataset versions.

In tables 7.4 and 7.5 are the detailed results of the experiments conducted using the VGG11 architecture. It is worth mentioning that the calculations for the VGG architecture were performed with the help of a graphics processing unit since the time to train a deep model like this was substantially increased. Therefore only the relative change concerning the corresponding baseline value should be considered.

Performance of the VGG11 architecture					
Loss	Activation	Average Accuracy	Highest Accuracy	Lowest Accuracy	Inference time [s]
Huber	Leaky ReLU	63.37%	65.38%	60.56%	0.0018
Huber	SiLU	62.49%	64.06%	60.58%	0.0016
BCE	ReLU	62.02%	63.66%	60.26%	0.0016
BCE	SiLU	61.36%	62.62%	60.29%	0.0017

Table 7.4: Performance of the VGG11 architecture for different configurations of the loss function and activation function.

Performance of the VGG11 architecture				
Dataset Version	Average Accuracy	Highest Accuracy	Lowest Accuracy	Inference time [s]
With economy	65.60%	68.93%	59.76%	0.0017
With updates	63.77%	66.20%	60.67%	0.0018
With roster changes	63.67%	66.32%	60.39%	0.0018
Baseline data	63.37%	65.38%	60.56%	0.0018

Table 7.5: Performance of the VGG11 architecture for different dataset versions. The results correspond to the model using Leaky ReLU as activation and Huber loss as loss function.

Discussion and conclusions

To discuss the results presented in the previous chapter, first, a baseline needed to be established to which the results could be compared. The first possible baseline that comes to mind is a simple coin toss, a 50-50 chance for each team to win any match-up. By this logic, any model that offers accuracy greater than 50% would be able to provide an advantage to the potential user. Another approach would be to use a ranking system that is already in place. Considering only the global ranking available on the hltv.com website at the time each game was played, and awarding a win to the higher-ranked team, a prediction accuracy of less than 50% was achieved. This means that the team ranking itself is not enough even to beat the 50% random baseline. Finally, the results obtained previously by other, most often simpler methods can be used as a reference point. For example, in [2], the elo-based statistical model was shown to achieve 64% accuracy, while the random forest and proposed CNN architecture achieved 63% and 59.8% accuracy, respectively.

Having established those reference points, it can be seen that all of the models proposed in this thesis managed to beat the 50% mark by a large margin. Starting with the simple artificial neural network model, which without any modifications to either the model's architecture or the basic dataset, managed to achieve over 61% accuracy. Testing the model further to find the optimal loss function and activation function allowed for a slight increase in accuracy to a point where the average accuracy recorded was 61.41% for the weighted focal loss and leaky ReLU activation. The improvement, however, was only marginal, while the increase in inference time was noticeable. This means that for this model and this particular problem, the combination of BCE loss and ReLU activation, the standard pairing for many machine learning models is sufficient. It must be said, however, that the ANN model was of reasonably low complexity, which in turn may have limited its ability to perform. Regarding the dataset alterations, the only option that allowed the model to achieve better accuracy was the introduction of the auxiliary economy data. This brought up the model's accuracy to an average of 61.6%. Any other additions only lowered the accuracy, which may indicate that a more complex model is needed when introducing more features.

Next, the simple one-dimensional CNN architecture managed to score similarly to the ANN model, oscillating around the 61% accuracy mark for the different loss and activation functions. At the highest point, for the base dataset, the simple CNN model achieved 61.19% accuracy on average, which is a slight downgrade from the, in theory, simpler ANN architecture. When tested for different dataset versions, the model performed significantly worse than the ANN model and the simple CNN model without auxiliary information. This can lead to the belief that the convolutional approach requires a deeper and more robust network.

This is why the VGG architecture was adapted to perform calculations on one-dimensional data. This approach was not discovered in previous works related to this type of problem, namely match prediction, so an attempt was made to see if any improvement occurred. The adapted VGG model was tested using the loss and activation functions that were deemed most beneficial to the model's accuracy during the simple CNN model testing phase. It turned out that for the Huber loss function and leaky ReLU activation, the VGG model managed to achieve 63.37% accuracy, which was the highest score so far, and the first time that all of the benchmarks established earlier were beaten. This means that the deeper, more robust architecture was able to pick up patterns in the data that may have been too complex for the simpler models to handle. The VGG architecture that achieved the best results was later trained using the base dataset with different auxiliary information added. This time, the model was performing better than with just the base data for every new information introduced. This once again proves that for the auxiliary input to be helpful, the model needs to be deeper and recognize more complex relations in the data. The peak accuracy for the VGG11 model was observed when using the dataset with the inclusion of the economy data for each of the teams. In this case, the average accuracy was 65.6%, while in the cross-validation process, the highest validation accuracy was 68.93%.

In the table below are the best results for each of the proposed models, as well as the proposed benchmark values arranged from highest to lowest.

Accuracy measurements vs benchmark values		
Source of value	Accuracy	Comment
VGG11	**65.60%**	**Own model, VGG11 architecture adapted to 1D convolution, result for Huber loss and Leaky ReLU with the addition of economy data**
Elo based rating	64.00%	Statistical prediction method, result taken from [2]
Random forest model	63.00%	Standard machine-learning algorithm, result taken from [2]
Simple ANN model	**61.60%**	**Own model, result for BCE loss and SiLU, with the addition of economy data**
Simple CNN model	**61.19%**	**Own model, result for Huber loss and SiLU**
CNN model	59.80%	Result taken from [2]
Random guessing	50%	The result of assigning the winning team with a coin toss, similar result when using global ranking

Table 8.1: Best accuracy values achieved for each of the proposed models in this thesis (in bold) with the established benchmark values.

The main goal of this thesis was to research the performance of different models when it comes to predicting the outcome of an esports event. The models studied ranged from a simple artificial neural network which is just layers of neurons connected with each other, to a deep and widely used convolutional architecture adapted to the problem's requirements. Each of the proposed architectures, without any tuning or further modifications applied, produced satisfactory results compared to the proposed benchmarks. It was shown that for the problem at hand, the models could be further improved by introducing new machine-learning algorithms and functions, such as the Focal and Huber loss functions and non-linear activation units.

The impact of the size of the dataset with the addition of auxiliary information was examined, and it was shown that the prediction accuracy could benefit significantly from it, given the model is of high enough complexity, as was the case with the proposed convolutional models.

In the end, given the presented results, it is fair to assume that information can be gained from the proposed models and the publicly available data that would otherwise take an experienced individual's expertise in the field of esports and CS:GO itself.

It is worth mentioning that given the results presented in [22], an approach to predicting the outcome of an esports event can be constructed, where in addition to the models presented in this thesis, a clustering method can be applied to increase the prediction accuracy and performance further. This could constitute possible further research of the problem at hand.

Bibliography

[1] Neal Southern. The rise of esports: A new audience model and a new medium. *BA Candidate, Department of Mathematics, California State University Staniaslaus*, 1:65–68, 2017.

[2] Ondřej Švec. Predicting counter-strike game outcomes with machine learning. Bachelor's thesis, Czech Technical University in Prague, 2022.

[3] Gabriel Fialho, Aline Manhães, and João Paulo Teixeira. Predicting sports results with artificial intelligence–a proposal framework for soccer games. *Procedia Computer Science*, 164:131–136, 2019.

[4] S Kevin Andrews, K Lakshmi Narayanan, K Balasubadra, and MS Josephine. Analysis on sports data match result prediction using machine learning libraries. In *Journal of Physics: Conference Series*, volume 1964/4, page 042085. IOP Publishing, 2021.

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

[6] Ria Kulshrestha. Yes, you should listen to andrej karpathy, and understand backpropagation. https://towardsdatascience.com/back-propagation-721bfcc94e34, Oct 2020. Accessed: 2023-05-02.

[7] Chen Wang, Chengyuan Deng, and Suzhen Wang. Imbalance-xgboost: leveraging weighted and focal losses for binary label-imbalanced classification with xgboost. *Pattern Recognition Letters*, 136:190–197, 2020.

[8] Ruoxi Qin, Kai Qiao, Linyuan Wang, Lei Zeng, Jian Chen, and Bin Yan. Weighted focal loss: An effective loss function to overcome unbalance problem of chest x-ray14. In *IOP Conference Series: Materials Science and Engineering*, volume 428, page 012022. IOP Publishing, 2018.

[9] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the twenty-first international conference on Machine learning*, page 116, 2004.

[10] Jason Brownlee. A gentle introduction to the rectified linear unit (relu). https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, 2019. Accessed: 2023-05-02.

[11] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[13] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). arxiv 2016. *arXiv preprint arXiv:1606.08415*, 3, 2016.

[14] Diganta Misra. Mish: A self regularized non-monotonic activation function. *arXiv preprint arXiv:1908.08681*, 2019.

[15] Harsh Yadav. Dropout in neural networks. https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9, Jul 2022. Accessed: 2023-05-02.

[16] Runzuo Yang. Using supervised learning to predict english premier league match results from starting line-up player data. Master's thesis, Technological University Dublin, 2019.

[17] Corentin Herbinet. Predicting football results using machine learning techniques. Master's thesis, IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE, 2018.

[18] Sascha Wilkens. Sports prediction and betting models in the machine learning age: The case of tennis. *Journal of Sports Analytics*, 7(2):99–117, 2021.

[19] Zijian Gao and Amanda Kowalczyk. Random forest model identifies serve strength as a key predictor of tennis match outcome. *Journal of Sports Analytics*, 7/4:255–262, 2021.

[20] Elnaz Davoodi and Ali Reza Khanteymoori. Horse racing prediction using artificial neural networks. *Recent Advances in Neural Networks, Fuzzy Systems & Evolutionary Computing*, 2010:155–160, 2010.

[21] Mariona Rosell Llorens. esport gaming: the rise of a new sports practice. *Sport, Ethics and Philosophy*, 11:464–476, 2017.

[22] Arvid Björklund, William Johansson Visuri, Fredrik Lindevall, and Philip Svensson. Predicting the outcome of cs: Go games using machine learning. Bachelor's thesis, CHALMERS UNIVERSITY OF TECHNOLOGY, 2018.

[23] Allen Rubin. Predicting round and game winners in csgo. *OSF Preprints*, 2022.

[24] Ilya Makarov, Dmitry Savostyanov, Boris Litvyakov, and Dmitry I Ignatov. Predicting winning team and probabilistic ratings in "dota 2" and "counter-strike: Global offensive" video games. In *Analysis of Images, Social Networks and Texts: 6th International Conference, AIST 2017, Moscow, Russia, July 27–29, 2017, Revised Selected Papers 6*, pages 183–196. Springer, 2018.

[25] Tian Wang. Predictive analysis on esports games: A case study on league of legends (lol) esports tournaments. Master's thesis, University of North Carolina at Chapel Hill, 2018.

[26] Petra Grutzik, Joe Higgins, and Long Tran. Predicting outcomes of professional dota 2 matches. *Technical Report. Stanford University, Tech. Rep.*, 2017.

[27] Mateus Dauernheimer Machado. Cs:go professional matches. https://www.kaggle.com/datasets/mateusdmachado/csgo-professional-matches, Mar 2020. Accessed: 2023-05-21.

[28] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020.

[29] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. https://d2l.ai, 2020. Accessed: 21-05-2023.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[31] Álvaro Teixeira Escottá, Wesley Beccaro, and Miguel Arjona Ramírez. Evaluation of 1d and 2d deep convolutional neural networks for driving event recognition. *Sensors*, 22:4226, 2022.

Appendix A - Attached files

The point of this appendix is to list all of the files attached to this thesis, as well as the ways of accessing and using them.

Attached to this thesis is the folder named "MSc_files" in which the following files containing executable python code can be found:

1. "Player_statistics.py" file - A Python script that was used to generate data for model training. The output file contains non-aggregated player statistics without any auxiliary information.
2. "Team_statistics.py" file - A Python script that was used to generate data for model training. The output file contains aggregated player statistics on the team level. The auxiliary information can be added to the resulting dataset by enabling flags set inside the script. A detailed description and documentation can be found inside the file.
3. "ANN_code.ipynb" file - A Jupyter notebook containing all functions and structures used in the process of obtaining results for the proposed ANN networks, alongside the networks themselves. All of the functions are documented and an example use case is presented.
4. "CNN_code.ipynb" file - A Jupyter notebook containing all functions and structures used in the process of obtaining results for the proposed simple CNN network, alongside the network itself. All of the functions are documented and an example use case is presented.
5. "VGG_code.ipynb" file - A Jupyter notebook containing all functions and structures used in the process of obtaining results for the proposed VGG network, alongside the network itself. All of the functions are documented and an example use case is presented.

Alongside those files, there is attached a compressed file named "Data_samples.rar" which needs to be unpacked directly into the parent directory. It contains the following files:

1. "Player_statistics.csv" file - A CSV file containing the baseline non-aggregated player statistics that were used for model training.
2. "Basic_team_statistics.csv" file - A CSV file containing the baseline aggregated team statistics that were used for model training.
3. "Team_statistics_with_economy.csv" file - A CSV file containing the aggregated team statistics with the auxiliary economy data, that was used for model training.
4. "Team_statistics_with_updates.csv" file - A CSV file containing the aggregated team statistics with the auxiliary data about game updates, that was used for model training.
5. "Teams_statistics_with_roster_changes.csv" file - A CSV file containing the aggregated team statistics with the auxiliary data about roster changes, that was used for model training.

After unpacking, those files can be used to train the models, and are needed for the use cases presented in the Jupyter notebooks.

Additionally, a file called "Results.xlsx" can be found in the parent directory. It is a Microsoft Excel file that contains all the final results obtained during the course of writing this thesis.

Appendix B - Summary in Polish

Głównym celem przedstawionej pracy magisterskiej było zaproponowanie i przetestowanie modeli uczenia maszynowego, wykorzystujących sieci neuronowe do rozwiązania problemu przewidywania wyników rozgrywek esportowych.

W pracy przedstawiono podstawowe pojęcia z zakresu uczenia maszynowego, jak również opisano wykorzystywane algorytmy oraz funkcje użyte do budowy poszczególnych modeli.

Za przykład gry esportowej posłużyła gra Counter-Strike:Global Offensive, dla której wykorzystano historyczne dane dotyczące wyników poszczególnych zawodników. Dane te zostały przetworzone w celu uzyskania jak największej liczby znaczących wartości, a następnie poszerzone o informacje dodatkowe, takie jak dane opisujące ekonomię zawodników w trakcie gry, ważne aktualizacje gry oraz zmiany w składach poszczególnych drużyn.

Do zaproponowanych modeli zaliczały się modele o architekturze ANN, czyli jednokierunkowe modele wykorzystujące warstwy połączonych ze sobą neuronów, oraz modele CNN wykorzystujące w swojej architekturze operację konwolucji. Dodatkowo, przedstawiono wyniki uzyskane z wykorzystaniem spejcalnie dostosowanej do problemu architektury VGG11, wykorzystującej konwolucję jednowymiarową.

Stworzone modele zostały poddane eksperymentom w różnych warunkach, dla różnych funkcji aktywacji, funkcji straty oraz dla różnych rozszerzeń samej architektury, oraz wykorzystanego zbioru danych. W badaniach sprawdzono wpływ trzech różnych funkcji straty - BCE, Huber oraz Focal, oraz czterech funckji aktywacji - ReLU, Leaky ReLU, SiLU oraz Mish na dokładność oraz czas predykcji. Następnie warianty modeli radzące sobie najlepiej z podstawową wersją zbioru danych zostały ponownie przebadane, po dodaniu wspomnianych wcześniej rozszerzeń.

Uzyskane wyniki średniej dokładności dla najlepszych z zaproponowanych sieci wyniosły 61.60% dla sieci ANN wykorzystującej funkcję straty BCE oraz aktywację SiLU, 61.19% dla sieci CNN wykorzystującej funckję straty Huber oraz aktywację SiLU, oraz 65.60% dla sieci VGG11 wykorzystującej funkcję straty Huber oraz aktywację Leaky ReLU. W przypadku sieci ANN oraz VGG11, wyniki te zostały uzyskane, gdy podstawowy zbiór danych został rozszerzony o dane opisujące ekonomię zawodników w trakcie gry.

Powyższe wyniki zostały porównane do wartości znalezionych w literaturze, uzyskanych z wykorzystaniem innych metod przewidywania wyników rozgrywek esportowych, takich jak modele "Random Forest" lub modele oprate na punktacji Elo. Porównanie to wykazało, że model VGG11 uzyskał dokładność wyższą, niż największa uzyskana do tej pory w literaturze wartość 64%.

Uzyskane wyniki wskazują na potencjał sieci neuronowych jako narzędzi przewidywania wyników esportowych, który może zostać w przyszłości rozwinięty zarówno poprzez zastosowanie głębszych, bardziej zaawansowanych modeli, lub poprzez zaproponowanie lepszego sposobu przetwarzania danych.