Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Master's Thesis

Football Score Prediction Using Deep Learning Methods

Arjun Chettiyattil Pankaj

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

ARJUN CHETTIYATTIL PANKAJ

Systems Engineering and Informatics
Informatics

Thesis title

Football Score prediction using Deep learning methods

Objectives of thesis

The aim of the work is to develop a methodology for solving a regression problem related to professional football. Scope of this work is to predict the score of the football matches by implementing deep learning algorithms using the historical data of football matches in the major European football leagues. Prediction of score with the maximum accuracy is the ultimate objective of this work.

Methodology

The student describes the current state of research in the field of deep learning. It gives an overview of the areas in which deep learning methods have been successfully used and describes some typical applications. It also describes the open-source software designed for their implementation, especially the frameworks TensorFlow and Keras.

The task is to predict the score of football matches which could a closer accuracy as a prediction from an expert in the football field using the available dataset. The data set used is taken from the data repository www.kaggle.com.

Student will use CNN algorithm to find the solution of this problem with the help of Python language and Deep learning frameworks are used for calculations. The result of implementation will be verified using stand methodologies in use and the conclusion of the findings is drawn.

The proposed extent of the thesis

60 pages

Keywords

Python, Deep learning, Neural network

Recommended information sources

GÉRON, A. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems.* Beijing ; Boston ; Farnham ; Sevastopol ; Tokyo: O'Reilly, 2019. ISBN 978-1-492-03264-9.

CHOLLET, F. – ALLAIRE, J J. *Deep learning with R.* Shelter Island, NY: Manning Publications Co., 2018. ISBN 9781617295546.

Expected date of thesis defence

2021/22 WS – FEM

The Diploma Thesis Supervisor

doc. Ing. Arnošt Veselý, CSc.

Supervising department

Department of Information Engineering

Electronic approval: 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 22. 03. 2022

Declaration

I declare that I have worked on my master's thesis titled "Football Score Prediction Using Deep Learning Methods" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master's thesis, I declare that the thesis does not break any copyrights.

In Prague on 30/03/2022

Acknowledgement

I would like to thank doc. Ing. Arnošt Veselý, Ing. Martin Čejka, Mgr. Nikola Bučková, friends and my family for their advice and support during my work on this thesis.

Football Score prediction using Deep learning methods

Abstract

The major aim of the work is to use convolutional neural network to predict the outcome of the football match result of Spanish first division football league using the historical dataset available from the season 2013-14 till 2017-18. The Possible outcomes are classified into win, draw or lose. The work has been done using Convolutional Neural Network algorithm with the help of several opensource libraries to implement deep learning framework. The programming language used to perform the tasks is Python programming language. The ultimate goal of the work is to predict the result with an accuracy as close as a prediction from a football domain expert. The result of the work is verified and conclusion drawn.

Keywords: Deep learning, Classification, CNN, RNN, Keras, Convolution layer, football result prediction, sport results, artificial intelligence

Fotbalové výsledky a hluboké učení

Abstrakt

Hlavním cílem práce je predikovat výsledky fotbalových utkání první divize španělské fotbalové ligy za využití konvoluční neuronové sítě a sady dat předešlých zápasů v sezónách od 2013-14 do 2017-18. Predikované výstupy jsou klasifikovány do kategorií výhra, remíza, prohra. Konvoluční síť byla sestavena za využití několika knihoven s rozhraním pro hluboké učení, které disponují otevřeným zdrojovým kódem. Programovacím jazykem byl Python. Cílem v kontextu je pak predikovat výsledky s přesností blížící se přesnosti predikcí fotbalových expertů. Výsledky práce jsou ověřeny, práce je zakončena představením závěru a diskuzí.

Klíčová slova: predikce fotbalových výsledků, sportovní predikce, umělá inteligence, konvoluční neuronová síť, hluboké učení, rekurentní neuronové sítě

Table of Contents

1. Introduction

Football being one of the world's most popular sports has a lot of craze in everyone's mind. In sports prediction, a large number of features can be collected including the historical performance of the team, results of the matches, and data on players to help different stakeholders understand the odds of winning or losing the forthcoming matches. Predicting which team is likely to win is important because of the financial assets involved in the betting process; thus bookmakers, fans, and potential bidders are all interested in approximating the odds of a game in advance.

This research aims to summarize the research done so far in solving the presented task as well as provide a thorough conclusion for the provided proposals.

Deep Neural Networks (DNNs) are usually used for pattern recognition and to solve non-linear relationships such as Stock Exchange and Prediction and Image Compression. Recurrent Neural Networks (RNNs) are usually used to solve sequence-based tasks; tasks involving text, speech or video streams. Convolutional Neural Networks (CNN) are used to solve image-based tasks. In this thesis, we are going to present and compare various approaches to how Neural Networks are used to solve the task of predicting the results of football matches.

2. Objectives and Methodology

2.1. Objective

The aim of the work is to develop a methodology to solve one of the classification problems in professional football, i.e. predicting the outcome of a professional football match. The scope of the work is to predict the final result of football matches in the Spanish first division professional football league using the available historical data. The possible outcome of the matches can be win, draw, or lose. Predicting this outcome with maximum possible accuracy is the ultimate objective of the work.

2.2. Methodology

The methodology of this work involves reviewing the current state of research in the field of deep learning. It also involves reviewing literary works and journals relevant to the topic in order to understand the general approach of the similar work in the same domain. This also involves giving an overview about the areas in which deep learning methods have been successfully implemented in the past and provides abstract idea about its typical applications. Open source software such as Keras, TensorFlow, NumPy, etc are explained briefly since they have been designed to implement deep learning framework.

The task is to predict the outcome of the football match with an accuracy close to that of an expert in the domain, using the available historical dataset. The dataset is created by choosing relevant attribute from the football statistics of Spanish la Liga. The data contains information from the seasons 2013-14 and 2017-18. The dataset is created by collecting data mainly from www.whoscored.com & www.transfermarkt.com.

The tasks are performed using CNN algorithm to find the solution of the problem using Python programming language and other deep learning frameworks for calculations. The result of the implementation will be verified using standard methodologies like accuracy value and based on them the conclusion of the findings is drawn.

3. Literature Review

3.1. Terminology

One of the advantages of Neural Networks is their versatility, meaning that multiple networks can be used to solve various tasks. In this section, we are going to present a quick overview of Machine Learning, Deep Learning, as well as Convolutional and Recurrent Neural Networks, how they work, the differences between them, and their applications.

3.1.1. Machine Learning

Definition: An application of artificial intelligence that includes algorithms that parse data, learn from that data, and then apply what they've learned to make informed decisions (Grieve, 2020).

Machine learning is a subfield of artificial intelligence which has increased in popularity over the last few years, in both research and industries. In contrast to the traditional rule-based artificial intelligence where an algorithm is more or less a list of predefined static rules, machine learning tries to use data to learn to make predictions or decisions.

An example of a Machine Learning algorithm is Spotify's music recommendation. Spotify sees the user's current taste in music and the type of songs they listen to the most and uses that information to recommend new songs for the user. Same as how Amazon recommends new products for its users. This technique is called a *Recommendation System* and it is one of the most popular Machine Learning algorithms (SystemDesign, 2021).

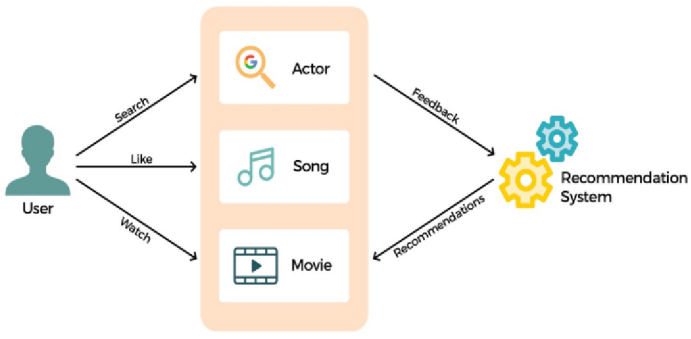

Figure 1. A simple chart showing how recommendation systems work

Supervised learning is a machine learning method in which models are trained using labeled data. In supervised learning, models need to find the mapping function to map the input with the output. Examples: Classification and Regression.

Unsupervised learning is another machine learning method in which patterns are inferred from the unlabeled input data. The goal of unsupervised learning is to find the structure and patterns from the input data. Examples: Segmentation and Clustering.

3.1.2. Deep Learning

Definition: A subfield of machine learning that structures algorithms in layers to create an "artificial neural network" that can learn and make intelligent decisions on its own (Grieve, 2020).

Deep Learning models analyze the data and their structure like a human would – they extract useful features from the data and use these features to learn certain patterns in the data.

Some types of Deep Learning algorithms are: Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). We will discuss each of them in simple detail in the next subsections.

Artificial Neural Networks

Artificial Neural Networks are inspired by the biological network of neurons in the human brain. The human brain is composed of nerve cells called neurons that are connected by axons. ANNs are composed of multiple nodes, which imitate the biological neurons of a human brain. The neurons are connected by links which imitate the biological axons, and they interact with each other. Each node takes input data, performs a simple operation, and passes the result to other nodes. Like a biological brain, an ANN is self-learning and can therefore excel in areas where the solution is difficult to express through a traditional programming approach (Aravindpai, 2020).

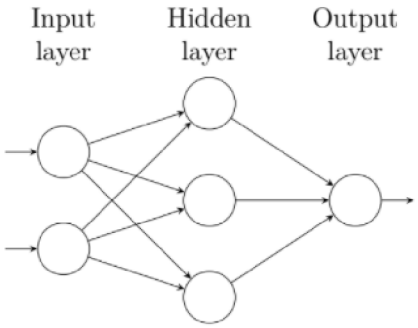

Figure 2. Artificial Neural Networks

A neural network with at least one hidden layer with a finite number of neurons in that layer can approximate any continuous function. This is known as the universal approximation theorem, and is the reason why one could believe that neural networks can be used for general artificial intelligence. It seems likely that being able to approximate functions is a very good property to possess when trying to learn how to behave.

Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are prevalent in image and video processing projects. The building blocks of CNN are filters or "kernels". Kernels are used to extract the relevant features from the input using the convolution operation.

How CNN Works?

CNN learns the filters implicitly, which is one of the main features of the Deep Learning models. These filters help in extracting the right and relevant features from the input data. CNN captures the spatial features from an image. Features help us in accurately identifying the objects present in images or videos, locating the said objects and their relations with other objects.

For example, we – as humans – can identify human faces by looking at specific features like eyes, nose, mouth and so on. We can also see how these specific features are arranged in an image. That's exactly what CNN is capable of capturing (Aravindpai, 2020).

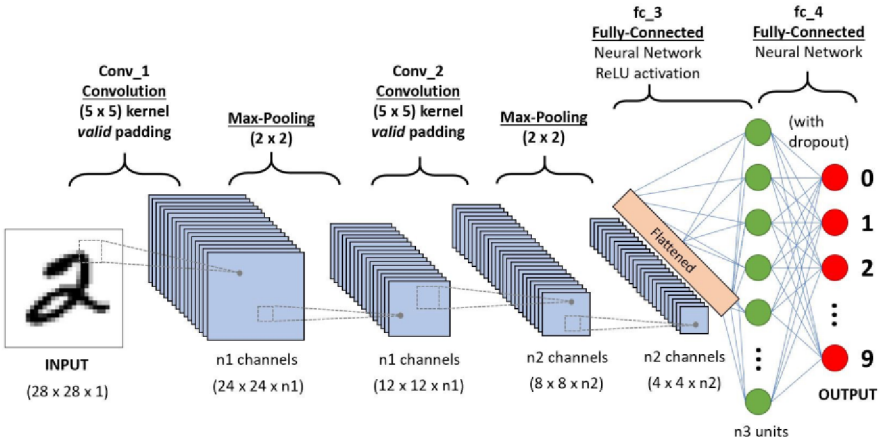

Figure 3. Simple architecture of CNN

Convolutional Neural Networks Layers

Convolutional Layer:

The convolutional layer is the central piece of the CNN where it derives its name from. The layer performs a *convolution* operation between the image and kernels/filters in order to extract meaningful features from the image. Kernels are two-dimensional sets of weights that are smaller than the input image, usually 3x3, 5x5 or 9x9 (Chollet, 2018).

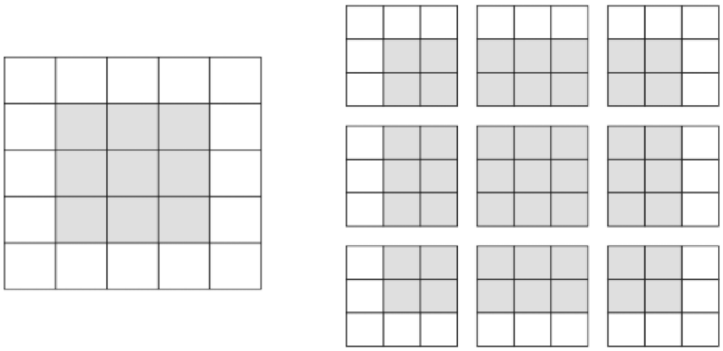

Figure 4. Convolution operation between the input image and a 3x3 kernel

The output of this convolution operation is called a “feature map”. Feature maps contain relevant features from the image that were extracted from the convolution layers.

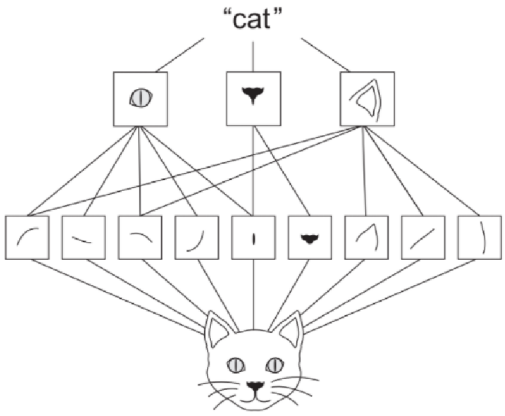

Figure 5. Visual representation of the feature map for a cat classification task

Pooling Layer:

The primary aim of the Pooling layer is to reduce the sizes of the convolved feature maps in order to reduce computational costs. There are several types of pooling: Max Pooling and Average Pooling. Max Pooling is by far the most used type. Pooling layers are quite simple, for example, Max Pooling works by calculating the largest element taken from a window of the feature maps (Chollet, 2018).

Pooling layers usually act as a bridge between Convolution layers and Fully Connected layers.

Dense (Fully-Connected) Layer:

Dense or Fully-Connected layers consist of the weights and biases along with the neurons. As the name suggests, the neurons in this layer are *fully-connected* to the ones in the layer before them.

It is worth noting that the final Dense layer always has the same output shape as the number of classes the data contains. For example, if we want to classify cats and dogs, we have two classes – cats and dogs – therefore, the final Dense layer should have an output shape of 2.

Dropout Layer:

Usually when using Fully-Connected layers, it can lead to overfitting. Overfitting happens when the model learns a feature *too well* or *memorizes* the feature. This leads to the model achieving very high accuracy on the training set but low accuracy on the testing set. i.e. the model does not generalize well to new data. To combat over-fitting, the concept of dropout is introduced. Drop-out means dropping random neurons in order to force all

neurons to learn new features in every iteration. Usually, a good dropout percentage is around 25-50%.

Flatten Layer:

Flatten layers simply convert 2D output to 1D, so it *flattens* the output of the previous layer. It is usually followed by one or more Dense layers as the flatten layer facilitate a one-dimensional input to the dense layer.

Activation Layer:

Finally, Activation Functions or Activation Layers learn and approximate continuous and complex relations between variables of the network. They basically decide which information should be fed forward through the network and subsequently fire the following neurons and which information should be discarded.

There are several types of activation functions: ReLU, Softmax, tanH, and Sigmoid. Each of these have their own specific usage. Generally, between layers we use ReLU activation and for classification tasks Sigmoid or Softmax activations are used.

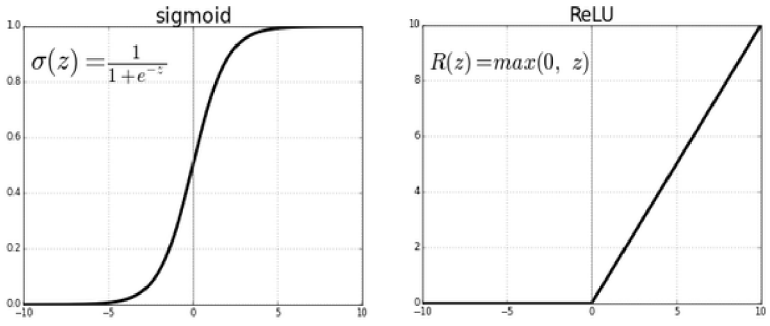

Figure 6. Graphs of Sigmoid and ReLU activation functions

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
maxpooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
maxpooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0

Figure 7. Simple CNN architecture used for digits classification

Applications of CNN

There are a myriad of Convolutional Neural Networks applications that we use in our everyday life, like Face Recognition - used in security systems as well as Facebook face tagging, Optical Character Recognition - used in extracting text and information from documents, Image Search, Object Detection, Object Recognition, and many others.

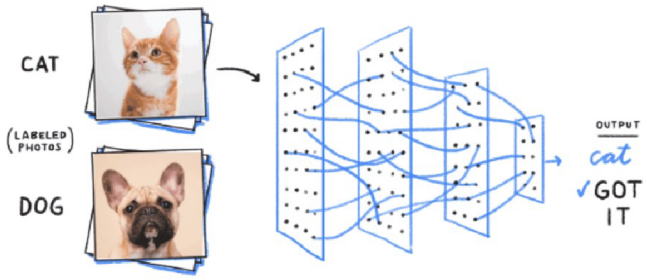

Figure 8. Object classification - an example of CNN

Recurrent Neural Networks (RNN)

Recurrent Neural Networks save the output of processing nodes and feed the result back into the model. Each node in the RNN model acts as a memory cell, continuing the computation and implementation of operations. If the network's prediction is incorrect, then the system self-learns and continues working towards the correct prediction during backpropagation.

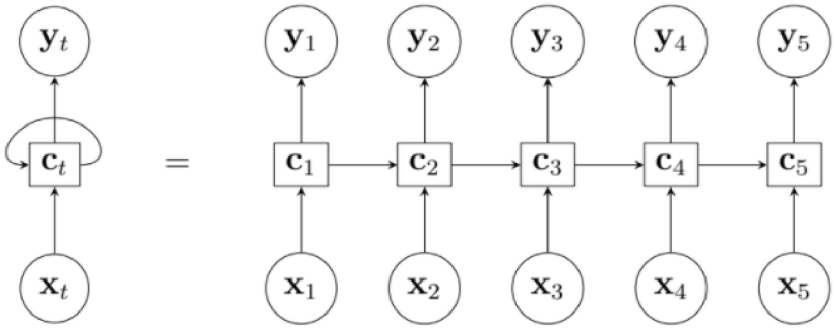

Figure 9. Recurrent Neural Networks architecture

One limitation of a normal neural network is that the input and output is of fixed length. If you input images they all need to have the same size. RNN introduces loops between events or steps allowing information to be used in a later stage, just like a memory.

One can see a loop as a copy of the network with the same parameters that just sends the state to the next step. This enables the model to be used on sequences of input and output which can take previous information into account. It has been shown that this works very well for a number of situations like natural language processing, video classification, image classification, etc (Aravindpai, 2020).

Some examples of the applications of Recurrent Neural Networks include: Sentiment Classification - classifying whether a sentence carries a positive or negative connotation; Image Captioning - describing the objects and events happening in a given image; Language Translation - automatically learn how to translate from one language to another, including knowing how the vocabulary, semantics and grammar of each language work.

Figure 10. Image captioning – an application of RNN

Long Short-Term Memory

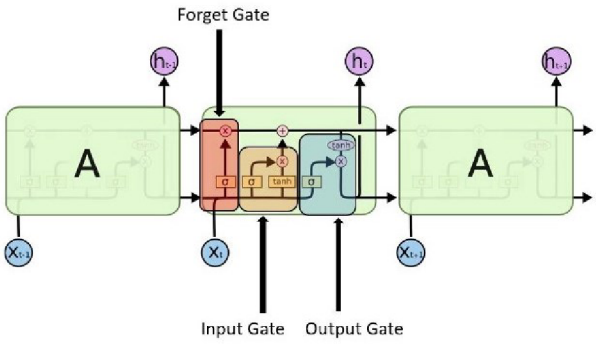

Figure 11. LSTM architecture

RNNs remember the information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.

Long Short-Term Memory (LSTM) unit is a recurrent network unit that is designed to remember values for either a long or a short duration of time. For example, if the LSTM unit detects an important feature from an early input sequence, it carries this information over a long distance. This is significant for many applications, such as speech processing,

music composition and time series prediction (Text Classification Algorithms: A Survey, 2019).

The below figure is a quick summary of the differences between ANN, CNN and RNN (Gupta, 2020).

	ANN	*CNN*	*RNN*
Type of Data	Tabular Data, Text Data	Image Data	Sequence data
Parameter Sharing	No	Yes	Yes
Fixed Length input	Yes	Yes	No
Application	Facial recognition and Computer vision.	Facial recognition, text digitization and Natural language processing.	Text-to-speech conversions.
Main advantages	Having fault tolerance, Ability to work with incomplete knowledge.	High accuracy in image recognition problems, Weight sharing.	Remembers each and every information, Time series prediction.
Disadvantages	Hardware dependence, Unexplained behavior of the network.	Large training data needed, don't encode the position and orientation of object.	Gradient vanishing, exploding gradient.

Table 1. Comparison of ANN vs CNN vs RNN

3.2. Current State Overview

In this section, we will present some of the previous work done to predict football matches with RNNs, LSTMs, CNNs or other approaches.

3.2.1. Football Match Prediction using Deep Learning

The first approach (Football Match Prediction Using Deep Learning, 2017) discusses using Recurrent Neural Networks to predict the football match results. The dataset used for this project includes the information for each player in the teams. The dataset includes matches for leagues and tournaments for the 54 countries in the Union of European Football Associations (UEFA), USA, Brazil, Japan and others. It contains features like *Line-ups*, *Position*, *Goals scored*, *Substitutions*, and *Penalty*.

The recorded data is *embedded* before being fed to the model. Afterwards, the embeddings are converted to one-hot encoded vectors which makes the data inputs more expressive and re-scalable.

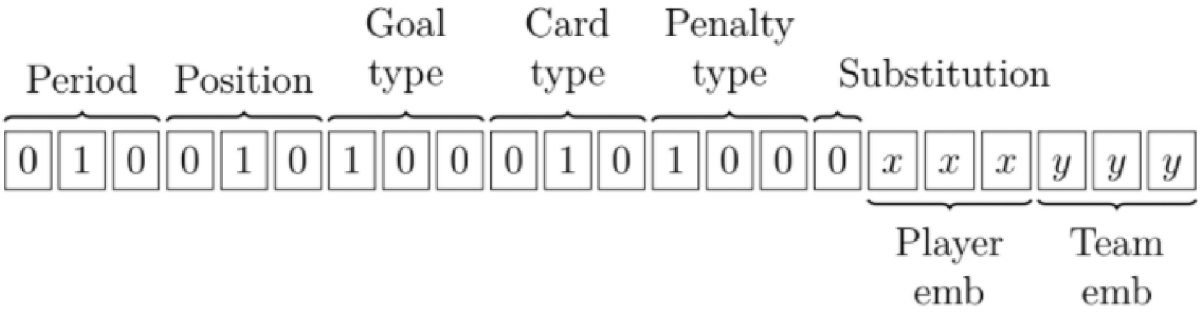

Figure 12. Architecture of the embedding model used in the thesis

The highest model achieved an accuracy of 98%. The authors have noted an exponential increase in accuracy as the match goes on (Football Match Prediction Using Deep Learning. Daniel Pettersson, Robert Nyquist).

Using the proposed LSTM architecture the final test classification accuracy of the outcome was 98.63% for the many-to-one. The more information the networks are fed about a match, i.e. the longer an ongoing match is played, the better the network performance on predicting the outcome.

3.2.2. A Deep Learning Framework for Football Match Prediction

In this research (A deep learning framework for football match prediction, 2020), the authors propose another Recurrent Neural Network-based approach to tackle the problem of predicting the outcome of a football match, more specifically, the scores of each team.

The aim of this study is to focus on the players initially. In this way it takes into account if a player does not play for a team in a specific match, which will have an impact and it will be recorded. A match is played at a specific time, and events occur at a relative time in a game. The order of matches and events matter since they have an impact on the future.

	Date	Home team	Away team	Home score	Away score	Tournament	City	country	Neutral	Rank date home	three_year_ago_ weighted_home
0	1993-08-08	Bolivia	Uruguay	3	1	FIFA World Cup qualification	La Paz	Bolivia	False	1993-08-08	0.0
1	1993-08-08	Brazil	Mexico	1	1	Friendly	Maceio	Brazil	False	1993-08-08	0.0
2	1993-08-08	Ecuador	Venezuela	5	0	FIFA World Cup qualification	Quito	Ecuador	False	1993-08-08	0.0
3	1993-08-08	Guinea	Sierra Leone	1	0	Friendly	Conakry	Guinea	False	1993-08-08	0.0
4	1993-08-08	Paraguay	Argentina	1	3	FIFA World Cup qualification	Asuncion	Paraguay	False	1993-08-08	0.0

Table 2. Sample of the dataset

After 10 epochs of training, the training loss dropped to around 0.594, and around 0.596 for validation. On testing the model with real life data, the model achieved a testing accuracy of 63.3%.

3.2.3. Using CNN to Predict Goal-Scoring Opportunities in Soccer

This research (Using Deep Convolutional Neural Networks to Predict Goal-Scoring Opportunities in Soccer, 2017) focuses on the starting club squad and ball positions to predict the goal-scoring opportunities in a match. Convolutional Neural Networks are used to process and predict the outcome of this data. The desired outcome in this approach is to predict when an attack is said to be a 'goal-scoring opportunity' or not. In other words, this approach focuses more on whether an attack will lead to a goal or not.

Below are samples of the data fed to the network. In these images, players of team A are represented by the blue data points and the keeper is represented by cyan; players of team B are represented by red data points and their keeper is represented by yellow; the ball is represented by a green square and the player who has possession of the ball has green borders around them in order to emphasize their presence.

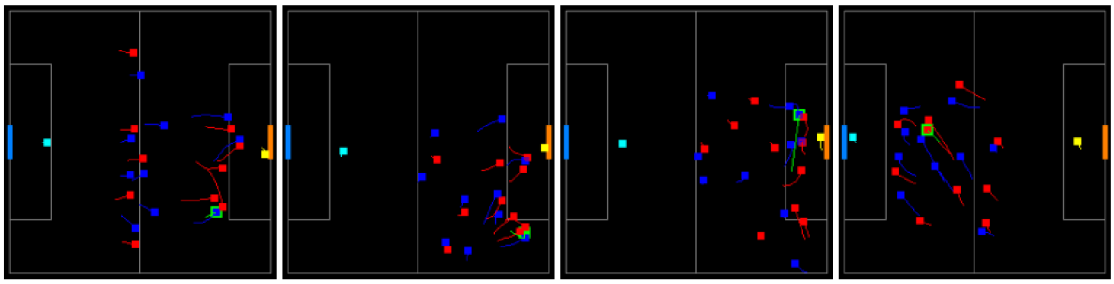

Figure 13. Goal scoring opportunity

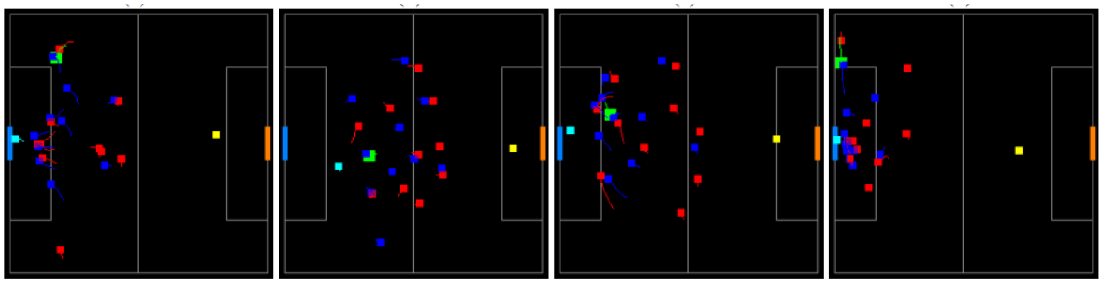

Figure 14. Loss of ball possession

The authors use two types of CNN: Google LeNet (ImageNet Classification with Deep Convolutional Neural Networks, 2012) and a custom 3-layered CNN. The LeNet

model has been trained from scratch and achieved an accuracy of 67%, which is 10% higher than that of the kNN model which achieved an accuracy of 57%. The custom CNN model achieved an accuracy of 63%.

3.2.4. Football Match Result Prediction Using Neural Networks

One approach uses (Football Match Result Prediction Using Neural Networks and Deep Learning, 2020) a Long-Short Term Memory (LSTM) of 512 units to tackle the task of predicting the football match outcome. The data used is from the English Premier League 2010-2018 seasons. The dataset contains data like home and away teams, match results, and winning streaks. This approach achieved a test accuracy of 80%. The input vector is not described, and the model output is whether the home team had won, drawn, or lost the match.

RNN size	32	64	256	512
Accuracy train	0.9731183	0.97043014	0.9919355	0.9892473
Accuracy test	0.8125	0.805	0.805	0.8025

Table 3. Summary of the accuracies achieved (Football Match Result Prediction Using Neural Networks and Deep Learning, 2020) experimenting with different number of units

3.2.5. Neural Networks Football Result Prediction

Another research (Shum, 2020) uses data gathered from both the English Premier League (EPL) and the Spanish La Liga gathered from 8 seasons (2007 to 2016). The data contains the teams' position, points, number of goals, winning streaks, losing streaks and results history. Multiple models are tested in this approach for research purposes. The input to the model consists of some of the selected features by the authors, such as the ones in Table 4.

	FTR	HTGS	ATGS	HTGC	ATGC	HTP	ATP	HM1	HM2	HM3	HM4	HM5	AM1	AM2	AM3	AM4	AM5	HTWinStreak3
2650	2	0.79	0.406250	0.493976	0.531646	0.000000	0.000000	1	0	1	1	3	1	3	1	0	0	0
2651	0	0.52	0.416667	0.457831	0.696203	0.000000	0.078947	1	3	3	3	0	3	0	1	0	0	0
2652	2	0.40	0.385417	0.795181	0.569620	0.000000	0.078947	1	0	3	1	1	3	1	0	1	0	0
2653	2	0.38	0.677083	0.963855	0.468354	0.000000	0.078947	1	1	1	3	1	3	3	1	3	3	0
2654	2	0.95	0.385417	0.385542	0.898734	0.078947	0.026316	3	3	1	3	3	0	1	3	0	1	0
2655	2	0.59	0.354167	0.590361	0.822785	0.026316	0.078947	0	3	3	0	0	3	0	1	0	0	0
2656	1	0.34	0.635417	0.903614	0.443038	0.026316	0.000000	0	1	1	0	1	1	3	3	0	0	0
2657	2	0.82	0.354167	0.337349	0.556962	0.078947	0.078947	3	3	3	0	1	3	0	1	1	0	0
2658	1	0.46	0.750000	0.783133	0.556962	0.000000	0.000000	1	3	1	3	0	1	3	0	1	3	0
2659	2	0.30	0.489583	0.662651	0.683544	0.000000	0.000000	1	0	0	0	1	1	3	3	1	3	0

Feature	Alias
Full Time Result	FTR
Home Team / Away Team Goal Shoton	HTGS,ATGS
Home Team / Away Team Goal Conceded	HTGC,ATGC
Home Team / Away Team Points	HTP,ATP
Home Team previous games result (1-5)	HM1,HM2,HM3,HM4,HM5
Away Team previous games result (1-5)	AM1,AM2,AM3, AM4,AM5
Home Team / Away Team no. of games winning streak (3,5)	HTWinStreak3, HTWinStreak5, ATWinStreak3, ATWinStreak5
Home Team / Away Team no. of games losing streak (3,5)	HTLossStreak3, HTLossStreak5, ATLossStreak3, ATLossStreak5
Home Team / Away Team Goals Difference	HTGD,ATGD
Difference of current league points: Home team's - Away team's	DiffPts
Difference of last 5 games points gained: Home team's - Away team's	DiffFormPts
Difference of league positions: Home team's - Away team's	DiffLP

Table 4. Sample of the input data

Most notably, an odds-based model achieves an average accuracy of 22% for the EPL and 20% for La Liga. Random Forest is then tested, and it achieves an accuracy of 54% across both leagues; a 3-layer ANN achieves an accuracy of 62% for EPL and 54% for la liga. The predicted output is also the match outcome relative to the home team.

3.2.6. Football Match Results Prediction Using Artificial Neural Networks

This research (Football Match Results Prediction Using Artificial Neural Networks; The Case of Iran Pro League, 2014) uses an ANN to predict the results in the Iranian League. The dataset was gathered by the authors containing match results and table positions of Iranian teams from 12 seasons – 2001 to 2012.

Match-based prediction criteria	Prediction parameters	Symbols
Teams	*The code of Home team*	I_1
	The code of Away team	I_2
Condition of teams in recent weeks	*Average of obtained points in recent 4 matches for Home team*	I_3
	Average of obtained points in recent 4 matches for Away team	I_4
Condition of teams in the league	*Average of obtained points in the league for Home team*	I_5
	Average of obtained points in the league for Away team	I_6
Quality of opponents in the last matches	*Average of obtained points by previous opponents in recent 4 matches for Home team*	I_7
	Average of obtained points by previous opponents in recent 4 matches for Away team	I_8
League of match	*The number of the league*	I_9
Week of match	*The number of the week*	I_{10}
Match results	*The number of goals by Home team*	O_1
	The number of goals by Away team	O_2

Row	*Inputs*										*Outputs*	
	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	O_1	O_2
54	Perspolis	Fajr	2.5	1.0	2.6	1.0	0.8	2.1	6	7	3	3
71	Sepahan	Esteghlal T.	1.3	1.3	2.0	1.0	1.1	1.8	8	7	2	1
570	Foolad	Saba	1.8	1.5	1.8	0.8	1.5	1.6	30	8	2	2
987	Paykan	Naft T.	1.8	0.8	1.0	1.1	1.3	1.8	8	10	1	3
1073	Sanat-mes	Pas	2.5	0.5	1.8	1.0	1.2	1.4	18	10	0	0
1778	Zob-ahan	Damash	0.3	1.0	0.9	1.2	1.6	1.4	28	12	1	0
2068	Tractor S.	Esteghlal K.	0.3	0.8	1.4	1.0	1.7	1.5	29	13	1	0

Table 5. Sample of the encoded data used in this approach

The authors not only predicted the winner of the match, but also the results of the match (the number of goals that each team would score).

Results	The last weeks matches							
	Match 1	Match 2	Match 3	Match 4	Match 5	Match 6	Match 7	Match 8
Prediction Results	2 0	2 0	0 1	2 1	0 2	0 2	2 0	2 0
Actual Results	1 1	2 1	0 1	1 0	0 1	0 1	1 2	0 0

Table 6. Summary of the predicted vs actual match results

3.2.7. Predicting Sports Matches with Neural Models

Finally, an interesting approach is taken in the next research (Pereverzeva, 2021). In this approach, the author uses a dataset gathered from various leagues including but not limited to the Italian, Spanish, German, and English leagues. The data contains information including the league in which the match was played, the participating teams, home and away teams, the scores for each team, and the result of the match - whether it's a win for the home team, a loss, or a draw. The model used in this approach is a Convolutional Graph Neural Network (A Graph Regularized Neural Network for Node Classification, 2020).

This approach achieves an average accuracy of 46% for the football match data.

	1 league soccer	all leagues soccer	chosen leagues soccer	NHL
ANN with embedding	0.4962	0.4553	0.4894	0.4222
GNN	0.5231	0.4640	0.4926	0.4493

Table 7. Summary of the results achieved by the CGNN on various sports

4. NN Implementation

4.1. Introduction

Based on the previous researches, it is apparent that the usage of previous match results, data about the clubs, and their positions in the league is vital to achieving a good accuracy. Therefore, the proposed approach consists of using a Convolutional Neural Network and using data from previous matches.

4.2. Dataset

The data used in this approach contains football match statistics and some team statistics of Spanish football league - Spanish la Liga. It is acquired from scaping the data from several websites including whoscored.com (WhoScored.com), Transfermarket.com (Transfermarkt), etc. assuming that the data provided in the websites is accurate without error.

In detail, the data set contains match statistics like the date of the match, team names, head-to-head match results , previous match results, goals scored by home-team and goals scored by away team etc. Also, it contains some team statistics like the average age of the squad, average height of the squad , transfer market value of the team etc. The acquired data is from the season 2013-14 till the season 2017-18.

Since there are 20 teams participating in the league in one season, each season will have 380 matches in total considering that each team will pay against each other two times in a season. However, other international cup matches and international club competitions haven't been considered for this study since the collection of the data would be more complicated. There could be some effect for some teams due to these extra games on the Season but they are neglected since it will be minor compared to the whole dataset.

	Date	HomeTeam	AwayTeam	H2H win	H2H draw	H2H lose	Home win	Home draw	Home lose	Away win	Away draw	Away lose	Home rest	Away rest	Home avg height	Away avg height	Home avg age	Away avg age	Home squad value	Away squad value	Home result	Away result
0	17/08/13	Real Sociedad	Getafe CF	1.0	3.0	2.0	2	3	1	3	2	1	30.0	30.0	182.5	183	34.3	36.5	€123.00m	€47.00m	2	0.0
1	17/08/13	Valencia CF	Malaga CF	4.0	0.0	2.0	2	2	2	3	1	2	30.0	30.0	178.6	178	35.4	36.1	€150.00m	€62.5m	1	0.0
2	17/08/13	Real Valladolid CF	Athletic Bilbao	0.0	3.0	3.0	1	2	3	3	2	1	30.0	30.0	180.6	181.3	36.5	33.5	€41.2m	€120.70m	1	2.0
3	18/08/13	FC Barcelona	Levante UD	5.0	1.0	0.0	2	3	1	2	2	2	30.0	30.0	176.1	181	35.5	37.1	€582.30m	€38.10m	7	0.0
4	18/08/13	CA Osasuna	Granada CF	1.0	1.0	4.0	1	3	2	2	1	3	30.0	30.0	183.3	178.5	36.6	36.5	€41.00m	€50.38m	1	2.0

Table 8. Sample of the dataset used

4.2.1. Attributes

As in the above example, the raw dataset consists of 22 columns which means 22 attributes.

Date : This attribute is the representation of date of occurrence of the match. The raw dataset contains two types of format for this attribute- MM/DD/YYYY and DD/MM/YYY. Date can be a relevant attribute as it can be correlated with gradual development of strengthening of the team by key players playing together for longer time.

Home Team : The name of the Host team is given by this attribute. The team which owns the ground in which the match is being played. This is a text datatype attribute.

Away Team : This attribute represents the name of the visiting team. This attribute has text datatype as the above one.

H2H win: This attribute represents the number of times the home team won the match when they played against the specific away team, considering the recent 6 head-to-head matches. This attribute can be correlated to the influence of historic match result in the upcoming match, which is mostly considered as psychological effect. This value can be an integer between 0 and 6 since the results of recent 6 matches are considered. This methodology is often applied in professional football match analysis by experts and betting websites. The values can be null when new teams enter the Spanish la Liga from the second division every season based on their performance in the second division. Hence there can be a scenario where some teams have no history of matches played together. This should be cleaned during the preprocessing phase.

H2H draw : The same as the H2H win attribute, this represents the number of times the home team ended up in draw when they played against the specific away team considering the recent 6 head-to-head matches.

H2H lose : This represents the number of times the home team lost when they played against the specific away team considering the recent 6 head-to-head matches.

Home win : This attribute gives an idea of the number of times the home team won in the recent 6 matches. This is irrespective of the away team and it can be an integer value between 0 and 6. This attribute incorporates the concept of *team form* to the dataset. It is highly probable that the team with higher number of winning streaks is in a good form. This will be a key attribute since the form of a team has importance in the football analysis and predictions.

Home draw: Similar to Home win attribute this one conveys the number of time the home team ended up in draw, in the recent 6 matches.

Home lose : The number of loses taken by the home team in the last 6 matches is represented by this attribute in the form of an integer value between 0 and 6.

Away win: This attribute is similar to Home win attribute which is explained earlier but represents the visiting team instead of home team.

Away draw: Similar to Home draw attribute and it gives information about the away team.

Away lose: Similar to Home lose attribute and gives information about the away team.

Home rest: This attribute tells the number of rest days the home team got in between the previous match and the current match. This can be an integer value. Rest days are usually very important as congested match schedule can cause fatigue and drop in performance. It is observed that the team which got better number of rest days shows a competitive advantage over the team without proper rest during important matches.

Away rest: This attribute is similar to Home rest but gives the information about the away team.

Home avg height : The physical aspects of the squad can be a key factor in many games when it comes to the style of the game they are playing or the opponent. Hence this attribute gives the average height of the squad of the home team selected for the game.

Away avg height: This attribute is similar to the Home avg height but represents the away team.

Home avg age: The average age of the squad, similar to average height, can influence the game and this attribute tells the average age of the home team squad. The team with the younger squad usually outperforms in a physically demanding game and the squad with more experience shows some advantage when it comes to games under high stress.

Away avg age : This attribute gives data similar to Home avg height but for the away team.

Home squad value: The talent and consistent performance of professional football players usually reflects on their transfer market value as highly skilled players mostly have high value in the transfer market. Collectively the transfer market value of the squad for a specific season can be a reflection of the talent they have. This attribute tells the transfer market value of the whole squad for a specific season in the denomination of million euros.

Away squad value: This attribute contains similar information like Home squad value but for away team instead of the home team.

Home result: This is the attribute present in the raw dataset which contributes to the result of the match in terms of the number of goals scored by the home team. By comparing this attribute with Away result attribute, it is possible to formulate which team will win the match or what is the final outcome of the match. This can be an integer value from 0.

Away result: Similar to the Home result attribute this one is available in the raw dataset and depicts the number of goals scored by the away team.

Result: This is an attribute derived from the Home result and Away result attributes using logical operation using python dataset and NumPy library. As an outcome this attribute can have 3 values and it conveys the results of the game as below.

Value = 1 : Home team won the match

Value = 0 :Home team lost the match

Value = 2 : The match is draw.

This attribute is considered as the Target attribute throughout the project.

4.3. Data Preprocessing

Data preprocessing includes the steps to transform the raw data with irregularities like noise, unwanted characters, missing values, etc. to a data which can be easily parsed by the machine.

Data preprocessing is important because the raw data with bad quality would lead to bad quality results as the algorithms would fail to identify the patterns effectively (Baheti, 2022).

The following python libraries have been used to perform the preprocessing of data.

Pandas

Pandas is a python library mainly used for working with tabular dataset. This library is popularly used to analyze, clean, explore and manipulate data (w3schools, 1999).

DataFrame

It supports various file formats like CSV, excel , SQL , JSON etc. which makes this library very versatile.

NumPy

NumPy is considered to be one of the fundamental packages for performing scientific computations in Python. It helps with the fast operations on arrays by facilitating multi-dimensional array objects, various derived objects such as matrices, masked arrays etc.

Scikit-Learn

Scikit-Learn is a free machine learning library for Python which features different algorithms such as random forest, KNN etc. Here, it is used mainly for data preprocessing and splitting the data into test and train sets.

Regular expression (re)

Regular expression (re) module is a tiny and highly specific programming language embedded inside Python. They are usually used to match strings of text such as characters, patterns etc. Here, it is used for encoding the date attribute into Year, Month and Day.

4.3.1. Code

Google Colab notebook has been chosen for the coding purpose as it is easy to set up and has inbuilt Python libraries. Google Colab notebooks are the Jupyter notebook IDE (Integrated Development Environment) that runs in cloud and is integrated with Google Drive.

This lets the user to use Google's dedicated GPU (Graphic Processing Unit) and TPU(Tensor Processing Unit).

Initially we need to import all the necessary Python libraries such as Pandas, NumPy, Sklearn, RE, etc.

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
import re
from keras.utils import np_utils

pd.set_option('display.max_columns', None)  # setting the parameter to s
how all columns
```

Then using following code, the raw dataset file named Dataset_1 is uploaded to Colab from the local machine. This is to start working with the data using the cloud notebook.

```
from google.colab import files
uploaded = files.upload()
```

Pandas library is used to read the file and display the dataset with first 5 rows. After that, inspection on the dataset is carried out by displaying dataset info. This will give us an idea about the different datatypes present in the dataset.

```
# Reading Excel file
data = pd.read_excel('Dataset_1.xlsx')
data.head(5)
# Getting general information about data
data.info()
```

By observing the data using the above commands, it is clear that the data contains categorical data (Home team name and Away team name) which need to be converted into appropriate format in order to feed to the neural network. The date column needs to be encoded in such a way that it will be split into year, month, and day columns and extract only Year and Month data.

Apart from the above, there are unwanted characters present in Home squad value and Away squad value columns which need to be removed.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1900 entries, 0 to 1899
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Date              1900 non-null   object
 1   HomeTeam          1900 non-null   object
 2   AwayTeam          1900 non-null   object
 3   H2H win           1881 non-null   float64
 4   H2H draw          1881 non-null   float64
 5   H2H lose          1881 non-null   float64
 6   Home win          1900 non-null   int64
 7   Home draw         1900 non-null   int64
 8   Home lose         1900 non-null   int64
 9   Away win          1900 non-null   int64
 10  Away draw         1900 non-null   int64
 11  Away lose         1900 non-null   int64
 12  Home rest         1896 non-null   float64
 13  Away rest         1891 non-null   float64
 14  Home avg height   1900 non-null   float64
 15  Away avg height   1900 non-null   object
 16  Home avg age      1900 non-null   object
 17  Away avg age      1899 non-null   object
 18  Home squad value  1899 non-null   object
 19  Away squad value  1900 non-null   object
 20  Home result       1900 non-null   int64
 21  Away result       1900 non-null   float64
dtypes: float64(7), int64(7), object(8)
memory usage: 326.7+ KB
```

Figure 15. Dataset with null values

Also, H2H win, H2H draw, and H2H lose columns contain null values. We need to remove them from the dataset by eliminating the corresponding row since the number of null values are less. Below code is used to perform the elimination of null values.

```
# Dropping the rows having null values
data.dropna(inplace=True)
```

As mentioned earlier in the attributes, the raw dataset contains two columns Home result and Away result which collectively give information about the final result of the match by comparing the values of above mentioned columns.

We formulate a single target variable called "Result" using the below formula.

IF Home result > Away result THEN Result =1 (means home team WON the game)
IF Home result < Away result THEN Result =0 (means home team LOST the game)
IF Home result = Away result THEN Result =2 (game is DRAW)

Below code is used to generate new target attribute and remove the Home result and Away result attributes from the dataset.

```
# Building up the Result Column
data['Result'] = np.where(
    data['Home result'] > data['Away result'], 1, np.where(
    data['Home result'] < data['Away result'], 0, 2))

# Drop unnecessary columns
data.drop(['Home result', 'Away result'], axis=1, inplace=True)
```

As explained previously, the Date column needs to be encoded to feed to the neural network. Extracting year and month while omitting the day data from the date column is the logic of the encoding done using Regular Expression and Pandas library.

Firstly, eliminate the extra space in the Date column. Convert it into Pandas datetime format. Then extract Day, Month and Year from it as new columns. Moved these newly created columns to the front side of the dataset and eliminate the Date and Day column from the dataset. Below code performs above explained activity.

```
data['Date'] = data['Date'].replace({' ': ''}, regex=True)
data['Date'] = pd.to_datetime(data['Date'])      # Convert date to pandas
 datetime
data['year'] = data['Date'].dt.year              # Extract year from date
data['month'] = data['Date'].dt.month            # Extract month form dat
e
data['day'] = data['Date'].dt.day                # Extract day from date

# Shifting the year, month and day column from last to front
```

```
col = data.pop("day")
data.insert(0, col.name, col)
col = data.pop("month")
data.insert(0, col.name, col)
col = data.pop("year")
data.insert(0, col.name, col)

data.drop('Date', axis=1, inplace=True)        # Drops Date columns
```

The extracted Year and Month data need to be One-hot encoded for better comprehension of the feature by the algorithm. The below code performs this task and omits the Day column from the dataset as we assume it could be less relevant compared to Year and Month.

```
data = pd.get_dummies(data, columns = ['year', 'month'])

data.drop('day', axis=1, inplace=True)
```

Attributes Home team name and Away team name are of datatype object. They need to be one-hot encoded as the above. The unique values in Home team names and Away team names are same. Below line of code checks that condition.

```
# Checking if Away Team names and Home team names are same
np.array_equal(sorted(data['AwayTeam'].unique()), sorted(data['HomeTeam'
].unique()))
```

The following code is used to transform the respective attribute to one-hot encoded format.

```
data = pd.get_dummies(data, columns = ['HomeTeam', 'AwayTeam'])
```

As raw dataset contains special symbols, '€' and 'm' in the Home squad value and Away squad value columns, it needs to eliminated from the dataset. This is performed by creating another function called remove_unwanted_chars() and applying this function to the Test and Train dataset.

```
def remove_unwanted_chars(df):
    df[['Home squad value', 'Away squad value']] = df[['Home squad value
', 'Away squad value']].replace({'€': '',

                                'm': ''}, regex=True)
      return df
train_data = remove_unwanted_chars(train_data)
test_data = remove_unwanted_chars(test_data)
```

Also, the 'Object' datatype present in the dataset needs to be converted to float datatype. This is achieved by creating a function called object_to_numeric(). This function also solves the extra '.' present in some columns. This function is applied to the Test and Train data separately.

```
def object_to_numeric(df):
    for col in list(df.select_dtypes(['object']).columns):

        df[col] = df[col].replace(r'\.{2}', '.', regex=True)

        # Converting to numeric values
        df[col] = pd.to_numeric(df[col])
    return df
train_data = object_to_numeric(train_data)
test_data = object_to_numeric(test_data)
```

It is better to shuffle the data before feeding to the neural network in order to avoid any kind of bias in the dataset. This is achieved by below code.

```
#shuffle the whole data
data = data.sample(frac = 1)
```

Next, split the dataset into Train data and Test data. A Train dataset is used to train the model while the Test dataset is used to validate the model built.

In this project, 80% of the dataset will be Train data and 20% of the dataset will be Test data.

The dataset is split using the below code.

```
# frac is the percentage of data to be taken in the train
frac = 0.8

# converting the frac into number of rows to be taken in the train data
threshold = int(len(data)*frac)

train_data = data[0: threshold]
test_data = data[threshold:].reset_index(drop=True)
```

After performing the above mentioned transformations, the columns are standardized.

Standardization is performed when the features of the input data have a large range of values. This can cause trouble to many machine learning models.

We are using StandardScaler from Sklearn library to perform standardization. Basically, it standardizes by removing the mean of the column and scaling to unit variance.

```
# columns to standardize
cols_to_standardize = ['H2H win', 'H2H draw', 'H2H lose', 'Home win',
                       'Home draw', 'Home lose', 'Away win', 'Away draw'
,
                       'Away lose', 'Home rest ', 'Away rest', 'Home avg
 height',
                       'Away avg height', 'Home avg age', 'Away avg age'
,
                       'Home squad value', 'Away squad value']

# Creating Standard Scaler object
standard_scaler = preprocessing.StandardScaler()

# Fitting and transforming the standard scaer objects with columns
train_data[cols_to_standardize] = standard_scaler.fit_transform(train_da
ta[cols_to_standardize])
test_data[cols_to_standardize] = standard_scaler.transform(test_data[col
s_to_standardize])
```

Output of the above transformation are as below for train dataset and test dataset respectively.

	H2H win	H2H draw	H2H lose	Home win	Home draw	Home lose	Away win	Away draw	Away lose	Home rest	Away rest	Home avg height	Away avg height	Home avg age	Away avg age	Home squad value	Away squad value	Result	year_2013
0	-0.772554	1.539535	-0.143497	-0.179694	1.646488	-0.733841	0.564637	0.512979	-0.871402	4.078607	4.076334	0.944522	-0.028210	0.090956	0.334380	-0.137425	-0.552710	1	1
1	1.239990	-1.216014	-0.143497	-0.179694	0.528862	-0.156494	0.564637	-0.599896	-0.152027	4.078607	4.076334	-1.320570	-0.038224	0.219749	0.289176	0.008340	-0.469300	1	1
2	-1.443402	1.539535	0.521068	-0.915368	0.528862	0.420853	0.564637	0.512979	-0.871402	4.078607	4.076334	-0.158984	-0.031615	0.348543	-0.004651	-0.579038	-0.156107	0	1
3	1.910839	-0.297498	-1.472629	-0.179694	1.646488	-0.733841	-0.191687	0.512979	-0.152027	4.078607	4.076334	-2.772553	-0.032216	0.231458	0.402186	2.342195	-0.600604	1	1
4	-0.772554	-0.297498	1.185634	-0.915368	1.646488	-0.156494	-0.191687	-0.599896	0.567348	4.078607	4.076334	1.409157	-0.037222	0.360251	0.334380	-0.580118	-0.534521	0	1

Figure 16. Output Train dataset Part 1

year_2014	year_2015	year_2016	year_2017	year_2018	month_1	month_2	month_3	month_4	month_5	month_6	month_7	month_8	month_9	month_10	month_11	month_12	HomeTeam_Athletic Bilbao
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

Figure 17. Output Train dataset Part 2

HomeTeam_Athletico de Madrid	HomeTeam_CA Osasuna	HomeTeam_CD Leganes	HomeTeam_Celta de Vigo	HomeTeam_Cordoba CF	HomeTeam_Deportivo Alaves	HomeTeam_Deportivo La Coruna	HomeTeam_Elche CF	HomeTeam_FC Barcelona	HomeTeam_Getafe CF	HomeTeam_Girona FC
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0

Figure 18. Output Train dataset Part 3

HomeTeam_Granada CF	HomeTeam_Levante UD	HomeTeam_Malaga CF	HomeTeam_RCD Espanyol	HomeTeam_Rayo Vallecano	HomeTeam_Real Betis Balompie	HomeTeam_Real Madrid	HomeTeam_Real Sociedad	HomeTeam_Real Valladolid CF	HomeTeam_SD Eibar	HomeTeam_Sevilla FC	HomeTeam_Sporting Gijon
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Figure 19. Output Train dataset Part 4

HomeTeam_UD Almeria	HomeTeam_UD Las Palmas	HomeTeam_Valencia CF	HomeTeam_Villarreal CF	AwayTeam_Athletic Bilbao	AwayTeam_Athletico de Madrid	AwayTeam_CA Osasuna	AwayTeam_CD Leganes	AwayTeam_Celta de Vigo	AwayTeam_Cordoba CF	AwayTeam_Deportivo Alaves
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Figure 20. Output Train dataset Part 5

AwayTeam_Deportivo La Coruna	AwayTeam_Elche CF	AwayTeam_FC Barcelona	AwayTeam_Getafe CF	AwayTeam_Girona FC	AwayTeam_Granada CF	AwayTeam_Levante UD	AwayTeam_Malaga CF	AwayTeam_RCD Espanyol	AwayTeam_Rayo Vallecano	AwayTeam_Real Betis Balompie
0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0

Figure 21. Output Train dataset Part 6

AwayTeam_Real Madrid	AwayTeam_Real Sociedad	AwayTeam_Real Valladolid CF	AwayTeam_SD Eibar	AwayTeam_Sevilla FC	AwayTeam_Sporting Gijon	AwayTeam_UD Almeria	AwayTeam_UD Las Palmas	AwayTeam_Valencia CF	AwayTeam_Villarreal CF
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Figure 22. Output Train dataset Part 7

	H2H win	H2H draw	H2H lose	Home win	Home draw	Home lose	Away win	Away draw	Away lose	Home rest	Away rest	Home avg height	Away avg height	Home avg age	Away avg age	Home squad value	Away squad value	Result
0	-0.101706	-1.216014	-0.808063	-0.179694	0.528862	-0.156494	-1.704336	-0.599896	2.006097	-0.316958	-0.501873	0.537967	-0.037823	-0.318841	-0.174167	-0.514793	-0.225526	0
1	0.569142	0.621018	-0.808063	1.291655	-0.588765	-0.733841	-1.704336	-1.712772	2.725471	-0.316958	-0.311115	-0.217064	-0.028410	-0.225173	-0.434090	0.095799	-0.473605	1
2	-0.101706	1.539535	-0.808063	0.555980	-0.588765	-0.156494	-0.191687	1.625855	-0.871402	-0.316958	-0.311115	1.002602	-0.033417	-0.072963	-0.049855	-0.012715	-0.549481	1
3	-1.443402	-1.216014	2.514766	-0.915368	0.528862	0.420853	2.833611	-1.712772	-1.590776	-0.316958	-0.311115	1.292998	-0.033217	-0.318841	-0.332381	-0.434353	3.265331	2
4	-0.101706	-1.216014	1.185634	-0.179694	-0.588765	0.420853	0.564637	-0.599896	-0.152027	-0.125847	-0.311115	0.421809	-0.027810	-0.283716	-0.072457	-0.433273	0.297537	1

Figure 23. Output Test dataset Part 1

Test data output follows the same structure as the train data output showed above.

The below code is used to separate target variable from both train dataset and test dataset and store them in different name. Followed by removing the target variable from both test and train data, store them in different name.

```
trainy = train_data['Result']
testy = test_data['Result']
train_data.drop('Result', axis=1, inplace=True)
test_data.drop('Result', axis=1, inplace=True)

trainX = train_data
testX = test_data
```

Here, we use all the columns of the train data as features except the last column which is stored in 'trainy' as target variable of the train set. Similarly Test data is also split and target variables are stored in 'testy'.

One-Hot is the process by which the Categorical value is transformed to a form with which the algorithm can work better. Sklearn's LabelEncoder does a similar job but it comes with a disadvantage. In LabelEncoding, it is assumed that the higher the categorical value the better the category will be. This would lead to errors.

So, to overcome this problem we use one-hot encoding in which we binarize the category and include it as a feature to train (Vasudev, 2017).

This is performed using Keras Library and 'np_utils' function is used for one-hot encoding the categorical variable. Below code converts the target columns in Train data and Test data into 3 different columns which are binarized and each column represents a different class.

```
Trainy = np_utils.to_categorical(trainy, 3)
testy = np_utils.to_categorical(testy, 3)
```

4.4. Model Building

4.4.1. Model with Convolution layer

Importing library

We use Keras deep learning framework for building CNN1D model. Sequential model, Dense layer, Activation layer, Dropout layer, Flatten layer, Conv1D layer, MaxPooling1D layer are imported from Keras library, matplotlib.pyplot as plt using the below code.

```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv1D, Ma
xPooling1D
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

Firstly Creating variables of Train dataset and Test dataset for this model using below code.

```
# Considering all the features of train data (without using any feature
selection algorithm)
trainX3 = train_data
testX3 = test_data
```

Following code is used to define variables and assign them values.

```
# Converting input data into format which convolutional layer takes.
n_timesteps, n_features, n_outputs = trainX3.shape[0], trainX3.shape[1],
 trainy.shape[0]
input_shape = (trainX3.shape[1], 1)
```

where,

'n_ timesteps' represents the number of rows in the training data,

'n_features' represents number of columns or features in the training data,

'n_outputs' represents number of outputs in the training data, and

'input_shape' contains the shape of input data. This format has been used because the CNN layer of neural network takes input in three-dimensional format.

Building model

```
# Creating a sequential model with convolutional and maxpooling layers w
hich automatically selects best features from the input data.

model3 = Sequential()
model3.add(Conv1D(filters=64, kernel_size=5, activation='relu', input_sh
ape=input_shape))
model3.add(MaxPooling1D(pool_size=3,strides=1, padding='valid'))
model3.add(Dropout(0.5))
model3.add(Flatten())
model3.add(Dense(32, activation='relu'))
model3.add(Dense(3, activation='softmax'))
model3.compile(loss='categorical_crossentropy', optimizer='adam', metric
s=['accuracy'])
```

Modeling of the neural network is performed using above lines of codes. In the first line, the Sequential model from the Keras library is defined. Sequential model allows to create the model layer by layer.

'Conv1D ' is a one dimensional convolution layer where one dimensional data is being fed. This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs (Keras). In order to compute convolution, many parameters need to be specified. The first Convolution layer takes four parameters.

- 'filters' - represents the number of output filters in the convolution. Filter detects spatial features from input data and CNN has tendency to learn multiple features in parallel from 32 to 512. In this case, we took 64 as initial value.
- 'kernal_size' – Kernels are filters that are used to detect features from images. In this case, we have used 5 as kernel size.
- 'activation' – Every layer need to be passed through an activation function or Transfer function. It is used to determine the output of neural network like 'Yes' or 'No'. It maps the resulting values are in between 0 to 1 or -1 to 1 etc. based on the type of the function used (Sharma, 2017). Here we use 'ReLU' activation function. It is one of the most used activation function nowadays in deep learning. The output of ReLU activation function is same as input if and only if the input is positive or zero and otherwise the output will be zero (Sharma, 2017).
- input_shape –defines the shape of the input data provided to CNN layer while training.

Max pooling layer is to automatically pick the most important features from the dataset.

Dropout layer randomly sets some input units to 0 with a frequency we input in order to avoid overfitting. In this case, we have used 0.5 ratios of coefficients to be dropped to zero.

Flatten layer is used to make a multidimensional input to one dimensional, commonly used to convert output of the previous layer into one dimensional form in order to feed to the next layer.

Dense layer is the regular deeply connected neural network layer. We are using two Dense layers in this experiment. The first Dense layer has 32 neurons and we came to this by doing trials with 8 ,16, 32 and 64 number of neurons. The first Dense layer with 32 neurons produced the best results. The second Dense layer with 3 neurons produces output in the form of three columns representing each class. Here, 3 represents the number of neurons, meaning the output shape is 3. Also, 'softmax' activation function is used in this layer.

Loss function is used to evaluate how good the algorithm model is. Higher value of loss function represents inaccurate prediction and smaller value of loss function represents more

accurate prediction. The loss function is calculated by calculating the squared difference between the expected value and predicted value. 'categorical_crossentropy' is used as loss function for multiclass classification problems where two or more output categories are available.

An optimizer is a function that modifies the attributes of neural networks such as weight and learning rate to reduce the loss and increase accuracy. We have used 'adam' optimizer here which is a stochastic gradient descent method that is based on adaptive distribution of first-order and second-order moments.

The metric to evaluate the model performance will be 'accuracy'. Higher accuracy means better prediction by the model.

Fitting model

The model needs to be fitted with the training data using the 'fit' function. We fit the model with the training features and their expected target variable represented by 'trainX3' and 'trainy', respectively. Number of epochs and batch size are the other parameters we define while fitting the model. Training the neural network once with all the training data is called an epoch. Here, the number of epochs equals to 100. The batch size is a hyperparameter that defines the number of samples to be processed before updating the internal model parameters. It can be greater than or equal to one and less than the number of samples. We have used 16 as the batch size. Also, we are splitting the train data into train set and validation set using the 'Validation_split' function. As the validation split equals to 0.2 , the initial train dataset will be split into 20 % validation set and 80% train set.

```
history_mode3 = model3.fit(trainX3, trainy, epochs=100, batch_size=16, v
erbose=1, validation_split=0.2, shuffle=True)
```

Plotting Epoch vs Loss

Plotting the loss value versus epoch for the train set and validation set will give us an idea of the model performance in terms of overfitting and optimum number of epoch.

```
# Plotting the graph of epoch vs loss
training_loss3 = history_mode3.history['loss']
validation_loss3 = history_mode3.history['val_loss']

# Create count of the number of epochs
epoch_count3 = range(1, len(training_loss3) + 1)

# Visualize loss history
plt.plot(epoch_count3, training_loss3, 'r--')
plt.plot(epoch_count3, validation_loss3, 'b-')
plt.legend(['Training Loss', 'Validation Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();
```

The result of the above code is given below.

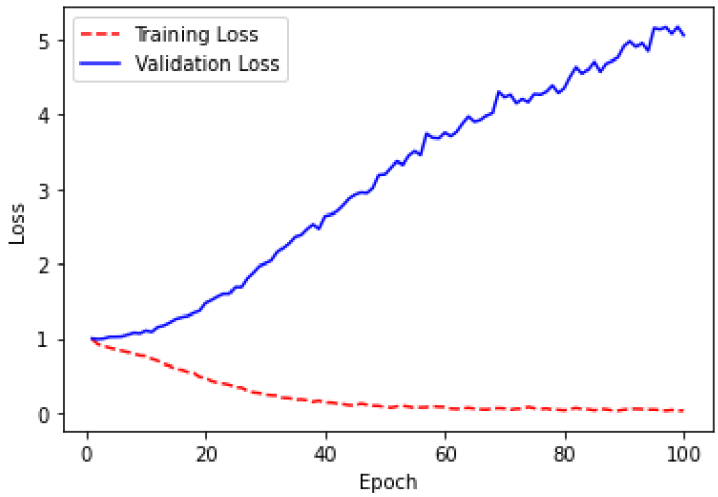

Figure 24. Plot for convolution mode with 32 neurons

From the above result, we find the best epoch using the below code. The accuracy and loss value at this epoch will be considered as the accuracy and loss value of train set and validation set respectively given by this model.

```
df_history3 = pd.DataFrame(history_mode3.history)
df_history3['loss difference'] = np.where(df_history3['val_loss']>=df_hi
story3['loss'], df_history3['val_loss'] - df_history3['loss'], -1)
df_history_factored_3 = df_history3[df_history3['loss difference'] >= 0]
```

```
epoch3 = np.array(df_history_factored_3.sort_values(by='loss difference'
).head(1).index.tolist())+1

train_accuracy3 = df_history3.loc[epoch3]['accuracy'].values
train_loss3 = df_history3.loc[epoch3]['loss'].values
val_accuracy3 = df_history3.loc[epoch3]['val_accuracy'].values
val_loss3 = df_history3.loc[epoch3]['val_loss'].values

print('At epoch: ', epoch3, ' Training accuracy: ', train_accuracy3, ' T
raining loss: ', train_loss3, ' Validation accuracy: ', val_accuracy3, '
 Validation loss: ', val_loss3)
```

The output of the above code shows the optimum epoch for this model is 1.

Training set gives an accuracy of 54.81 % with loss value of 0.923. Similarly, the validation set gives an accuracy of 54.51 % with loss value of 0.992.

Following this, the model is fit on the data with the optimum epoch value we obtained in the above part and accuracy and loss value of the test set are calculated.

```
history_mode3_new = model3_new.fit(trainX3, trainy, epochs=1, batch_size
=16, verbose=1, validation_split=0.2)
loss, accuracy = model3_new.evaluate(testX3, testy, batch_size=32, verbo
se=0)
print('Loss is:, ', loss, ' and accuracy is: ', accuracy)
```

Result

Performance of the model is evaluated by calculating loss and accuracy values.

```
75/75 [==============================] - 2s 13ms/step - loss: 1.0286 - accuracy: 0.5080 - val_loss: 1.0173 - val_accuracy: 0.5251
Loss is:,  0.9999246001243591  and accuracy is:  0.5213903784751892
```

Figure 25. Result of convolution model

	Best epoch	Accuracy (%)	Loss
Train set	1	54.81	0.923

Validation set	1	54.51	0.992
Test set	1	52.13	0.991

Table 9. Accuracy and Loss values

The model has 52.13 % accuracy and 0.991 loss value on unseen test dataset with all the features considered.

4.5. Model Improvement

There are several methods to improve the model performance and we adopt the feature selection. Feature selection means the selection of variables that are more relevant to the prediction and elimination of less relevant variables from the dataset. They can lead to better accuracy by reducing complexity as we use few relevant features.

4.5.1. Chi Square Feature Selection

Below code is used to implement Chi Square feature selection which is suitable for selecting the best features from the input data base on its performance with the target variable. The data

'chi square selector' should be non-negative there for we used MinMax scaler to transform the input data.

'SelectKBest' selects 'K' number of features from the input data based on the chi square test and we have assigned the value of 'K' to 10.

```
#using chi square
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import MinMaxScaler

X_norm = MinMaxScaler().fit_transform(train_data)
chi_selector =SelectKBest(chi2, k=10)
chi_selector.fit(X_norm, trainy)
chi_support = chi_selector.get_support()
```

```
chi_feature = train_data.loc[:,chi_support].columns.tolist()
print(str(len(chi_feature)), 'selected features')
```

The output of the chi square feature selection is printed using below code.

```
#output of selected 10 features by chi square
for i in range(0, len(chi_feature)):

     print(chi_feature[i])
```

Based on the performance chi square function the following are the top 10 relevant features.

- H2H win
- H2H lose
- Home squad value
- Away squad value
- HomeTeam_Athletico de Madrid
- HomeTeam_FC Barcelona
- HomeTeam_Real Madrid
- AwayTeam_Athletico de Madrid
- AwayTeam_FC Barcelona
- AwayTeam_Real Madrid

However, it is clear that some of the selected features are not logical as some team names are used to predict the match result of some other two teams. This shows no logical relevancy.

To continue with the procedure, the new train and test sets are defined based on the chi square feature selection result.

```
# Taking only top 10 features selected by Chi2.
trainX = train_data[chi_feature]
testX = test_data[chi_feature]
```

Model Building

Since the feature selection is already performed using chi2 method, we are adopting simple architecture with two dense layers only. The first Dense layer contains 16 neurons (the number of neurons for the first Dense layer is calculated by trial-and-error and it is found that

the Dense layer with 16 neurons delivers the best results) and the second Dense layer contains 3 neurons.

We split the initial train data into Validation set (20%) and Train set (80%) in order to find the best epoch.

```
# Generating a Dense model architecture for training -Chi square
model1 = Sequential()
model1.add(Dense(16, input_shape=(trainX.shape[1],), activation='relu'))
# model1.add(Dense(32, activation='relu'))
model1.add(Dense(3, activation='softmax'))
model1.compile(loss='categorical_crossentropy', optimizer='adam', metric
s=['accuracy'])
```

Fitting model

The above model with Dense layer architecture is then fitted with the dataset obtained after chi squared feature selection.

```
history_mode1 = model1.fit(trainX, trainy, epochs=100, batch_size=32, ve
rbose=1, validation_split=0.2, shuffle=True)
```

Plotting Epoch vs Loss

The graph is plotted between loss value and epoch for validation set and train set using the below code.

```
# Plotting the graph for epoch vs loss
training_loss = history_mode1.history['loss']
test_loss = history_mode1.history['val_loss']

# Create count of the number of epochs
epoch_count = range(1, len(training_loss) + 1)

# Visualize loss history
plt.plot(epoch_count, training_loss, 'r--')
plt.plot(epoch_count, test_loss, 'b-')
plt.legend(['Training Loss', 'Validation Loss'])
```

```
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show();
```

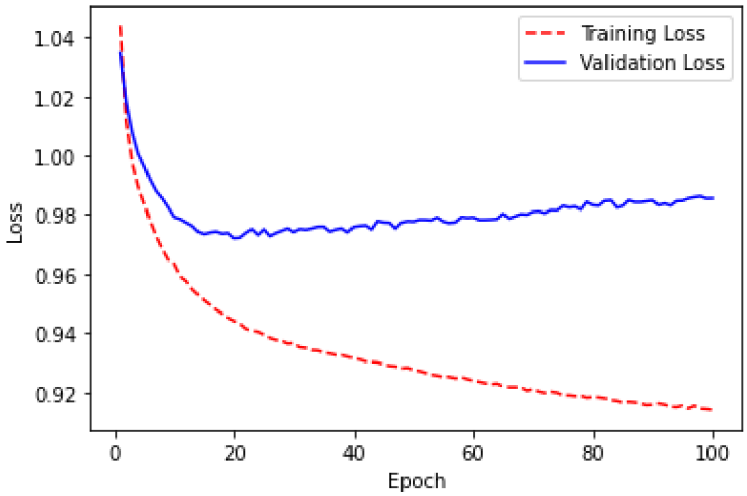

Figure 26. Loss vs epoch for dense layer model-Chi square

The following code calculates the best epoch and it is found that the model overfits when epoch is more than 3. Hence, the best epoch is 3.

The model gives an accuracy of 54.4 % and a loss value of 0.966 on train dataset.

Similarly, it gives accuracy and loss values of 49.49 % and 0.985 respectively on the validation set.

In the next step, this Dense layer model is fitted with the dataset after chi square feature selection with optimum epoch value in order to find the accuracy and loss value on the test set.

```
# Generating a Dense model architecture for training
model1_new = Sequential()
model1_new.add(Dense(16, input_shape=(trainX.shape[1],), activation='rel
u'))
model1_new.add(Dense(3, activation='softmax'))
model1_new.compile(loss='categorical_crossentropy', optimizer='adam', me
trics=['accuracy'])
history_model1_new = model1_new.fit(trainX, trainy, epochs=3, batch_size=
32, verbose=1, validation_split=0.2, shuffle=True)
```

Result

The result of the model is evaluated using accuracy and loss values.

	Best epoch	Accuracy(%)	Loss
Train set	3	54.40	0.966
Validation set	3	49.49	0.985
Test set	3	53.47	0.999

Table 10. Accuracy and Loss values

The model produced an accuracy of 53.47 % on the unseen test dataset.

4.5.2. Selecting From Model Technique

Here, we are use random forest classifier to estimate the best features as it is tree based algorithm and suitable for finding conditions.

Then, the train data has been fed to the classifier. The 'SelectFromModel()' function selects the top 12 features from the classifier.

```
#model improvement using selectfrommodel technique
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

embeded_rf_selector = SelectFromModel(RandomForestClassifier(n_estimator
s=100), max_features=12)
embeded_rf_selector.fit(train_data, trainy)

embeded_rf_support = embeded_rf_selector.get_support()
embeded_rf_feature = train_data.loc[:,embeded_rf_support].columns.tolist
()
print(str(len(embeded_rf_feature)), 'selected features')

embeded_rf_feature
```

The model selected following features as the most relevant regarding the prediction of result.

- H2H win

- H2H lose
- Home lose
- Away win
- Home rest
- Away rest
- Home avg height
- Away avg height
- Home avg age
- Away avg age
- Home squad value
- Away squad value

Based on the findings new test and train dataset are defined using the below code.

```
# Taking only those features which are selected by "SelectFromModel" Met
hod
trainX2 = train_data[embeded_rf_feature]
testX2 = test_data[embeded_rf_feature]
```

Model Building

In this case, the architecture used is the same as that of the chi square feature selection method. The dataset is derived from the 'Select from model' methodology. The first Dense layer is with 16 neurons (calculated by running trials with 64 , 32 ,16, and 8 neurons) and the second Dense layer possesses 3 neurons.

```
# Creating a Dense model architecture
model2 = Sequential()
model2.add(Dense(16, input_shape=(trainX2.shape[1],), activation='relu')
)
model2.add(Dense(3, activation='softmax'))
model2.compile(loss='categorical_crossentropy', optimizer='adam', metric
s=['accuracy'])
```

Fitting model

The above model is fitted with the dataset derived from 'select from model' technique, using below code.

```
history_mode2 = model2.fit(trainX2, trainy, epochs=100, batch_size=32, v
erbose=1, validation_split=0.2, shuffle=True)
```

Plotting Epoch vs Loss

As in the case of the previous models, a graph is plotted between the Loss value and Epoch.

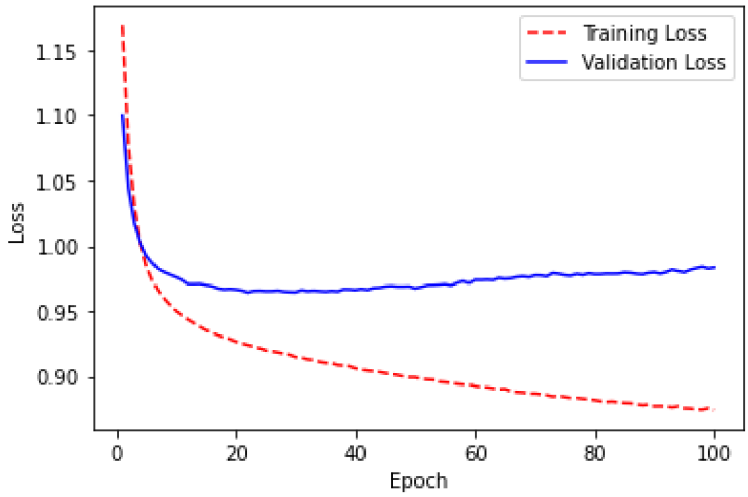

Figure 27. Loss vs epoch for dense layer model-Chi square

The same set of code which was used in earlier, is now reused to calculate the best epoch value.

It is clear that an epoch value of 10 works best for this model as the model overfit after 10 number of epochs.

The accuracy and loss value of this model on the train set are 54.81 % and 0.954 respectively. Similarly, the accuracy and loss value on the validation set at epoch 10 are 52.17 % and 0.958 respectively.

After finalizing on an epoch value of 10, we are fit this Dense layer architecture on the dataset derived from the 'select from model'. This is to calculate the accuracy and loss value on the test set.

```
#Creating a Dense model architecture
model2_new = Sequential()
model2_new.add(Dense(16, input_shape=(trainX2.shape[1],), activation='re
lu'))
# model2_new.add(Dense(64, activation='relu'))
model2_new.add(Dense(3, activation='softmax'))
model2_new.compile(loss='categorical_crossentropy', optimizer='adam', me
trics=['accuracy'])
history_mode2_new = model2_new.fit(trainX2, trainy, epochs=10, batch_siz
e=32, verbose=1, validation_split=0.2, shuffle=True)

loss, accuracy = model2_new.evaluate(testX2, testy, batch_size=32, verbo
se=0)
print('Loss is:, ', loss, ' and accuracy is: ', accuracy)
```

Result

The result of the performance of the model is printed using below code.

```
loss, accuracy = model2_new.evaluate(testX2, testy, batch_size=32, verbo
se=0)
      print('Loss is:, ', loss, ' and accuracy is: ', accuracy)
```

	Best epoch	Accuracy(%)	Loss
Train set	10	54.81	0.954
Validation set	10	52.17	0.958
Test set	10	53.20	1.005

Table 11. Accuracy and Loss values

The model gave 53.20 % accuracy and loss value of 1.005 on the unseen test dataset.

5. Conclusion

Above performed approach to predict the football match results from the dataset using CNN algorithm delivered comparatively low accuracy, which is approximately 53% for all the three models. Initially, the model architecture with convolutional layer delivered an accuracy of 52.13 %. Then, we performed model improvement using feature selection using two different techniques, the chi square feature selection and 'selectFrommodel' method, respectively. The accuracy of the algorithm was close to the previous model with convolutional layer, which was 53.47% after performing chi square feature selection. However, some features selected by the algorithm were not logical. Then, we performed feature selection using 'selectfrommodel' method and the features selected by algorithm were logically convincing from a football analysis point of view. However, the accuracy remained almost same as the previous models. The accuracy value provided by this model was 53.2 %.

Even though an accuracy of 53% is technically low, we also have to consider that it is higher than the probability of a random guess which gives 33.33% probability to be correct. The major reason for low accuracy value in general could be the dataset itself as the dataset is comparatively less complex. Hence, it assumes that the simple architecture would be enough to solve the problem. This could also be the reason why the models overfit after several epochs. This can be understood from the comparatively similar results by model with convolutional layer architecture and models with simple Dense layer architecture.

Also, considering that domain specialists deliver an accuracy of around 72% (A deep learning framework for football match prediction, 2020), the performance by the CNN on the dataset with limited input is not a bad prediction.

The performance of the model could be strongly improved by enriching the dataset. As football is a game which can be influenced by so many factors, it is nearly impossible to generate a perfect dataset. The dataset we have used contains very limited features in the football game aspects. However, increasing the complexity of the dataset by adding more complex features could lead to better fitting of the model. Similarly, the length of the dataset can be improved by

adding more samples. This could also be a significant step towards model performance improvement.

6. Future Scope

There are many areas that can be improved in order to achieve a better performance from the model.

- Fine tuning of model parameters can be performed in order to achieve accuracy.
- Improving dataset by adding more complex features
- Improving dataset by adding more samples to the dataset

Better feature selection techniques can be implemented by research and it could lead to better results.

7. References

A deep learning framework for football match prediction. **Rahman, Md. Ashiqur. 2020.** 08 January 2020, SN Applied Sciences.

A Graph Regularized Neural Network for Node Classification. **S. Dabhi, M. Parmar. 2020.** 2020, Vol. arXiv:2006.09022.

Aravindpai. 2020. ANN vs CNN vs RNN | Types of Neural Networks. *Analytics Vidhya.* [Online] 17 February 2020. https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/.

Baheti, Pragati. 2022. A Simple Guide to Data Preprocessing in Machine Learning. *v7labs.* [Online] 31 January 2022.

Chollet, François. 2018. *Deep Learning with Python.* s.l. : Manning Publications Co., 2018.

Football Match Prediction Using Deep Learning. **Daniel Pettersson, Robert Nyquist. 2017.** s.l. : Department of Electrical Engineering, Chalmers University of Technology, 2017.

Football Match Result Prediction Using Neural Networks and Deep Learning. **Ekansh Tiwari, Prasanjit Sardar, Sarika Jain. 2020.** s.l. : IEEE 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2020, pp. 229-231.

Football Match Results Prediction Using Artificial Neural Networks; The Case of Iran Pro League. **S. Mohammad Arabzad, M. E. T. Araghi, S. Sadi-Nezhad, Nooshin Ghofrani. 2014.** 3, 2014, Journal of Applied Research on Industrial Engineering, Vol. 1.

Grieve, Patrick. 2020. Deep learning vs. machine learning: What's the difference? *Zendesk Blog.* [Online] 23 January 2020. https://www.zendesk.com/blog/machine-learning-and-deep-learning/.

Gupta, Abhishek. 2020. Difference between ANN, CNN and RNN. *GeeksforGeeks.* [Online] 17 July 2020. https://www.geeksforgeeks.org/difference-between-ann-cnn-and-rnn/.

ImageNet Classification with Deep Convolutional Neural Networks. **Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 2012.** s.l. : Curran Associates Inc., 2012, NIPS'12:

Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, pp. 1097–1105.

Keras. Conv1D layer. *Keras.* [Online] https://keras.io/api/layers/convolution_layers/convolution1d/.

NumPy User Guide. **community, NumPy. 2020.** 2020, Vol. 1.18.4.

Pereverzeva, Aleksandra. 2021. *Predicting sports matches with neural models.* Department of Computer Science, Czech Technical University in Prague. Prague : s.n., 2021.

Sharma, Sagar. 2017. Activation Functions in Neural Networks. *Towards Data Science.* [Online] 6 September 2017. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

Shum, Roland. 2020. Neural Networks Football Result Prediction. *Medium.* [Online] 15 June 2020. https://medium.com/@rolandshum.shc/neural-networks-football-result-prediction-d8b0f933118b.

SystemDesign. 2021. System Design Interview: Recommendation System Design. *Medium.com.* [Online] 9 September 2021. https://medium.com/double-pointer/system-design-interview-recommendation-system-design-as-used-by-youtube-netflix-etc-c457aaec3ab.

Text Classification Algorithms: A Survey. **Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, Donald Brown. 2019.** Charlottesville : Department of Systems and Information Engineering, University of Virginia, 2019.

Transfermarkt. LaLiga 13/14. *Transfermarkt.* [Online] https://www.transfermarkt.com/laliga/startseite/wettbewerb/ES1/plus/?saison_id=2013.

Using Deep Convolutional Neural Networks to Predict Goal-Scoring Opportunities in Soccer. **Martijn Wagenaar, Emmanuel Okafor, Wouter Frencken, Marco A. Wiering. 2017.** s.l. : International Conference on Pattern Recognition Applications and Methods (ICPRAM), 2017.

Vasudev, Rakshith. 2017. What is One Hot Encoding? Why and When Do You Have to Use it? *HackerNoon.* [Online] 3 August 2017. https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f.

w3schools. 1999. Pandas Introduction. *w3schools.* [Online] 1999. https://www.w3schools.com/python/pandas/pandas_intro.asp.

WhoScored.com. Primera Division Summary. *WhoScored.com.* [Online] https://www.whoscored.com/Regions/206/Tournaments/4/Seasons/3922/Spain-LaLiga.

8. List of Figures and Tables

8.1. List of Figures

8.2. List of Tables

8.3. Appendix

1. Dataset_1.xlsx
2. Football result prediction using cnn.py