Configuration Manual

MSc Research Project
Data Analytics

Hashir Sayeed
Student ID: X21214611

School of Computing
National College of Ireland

Supervisor: Paul Stynes, Eugene McLaughlin, William Clifford

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Hashir Sayeed
Student ID:	X21214611
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Paul Stynes, Eugene McLaughlin, William Clifford
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	1675
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	13th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Hashir Sayeed
X21214611

1 Introduction

This configuration manual gives a brief guide on how to execute the code/module used for the following research project. The process includes the specification of the desktop the code was executed, the installation of the required software to execute the code seamlessly and all the procedure of how the code was executed which includes data gathering, model building and model training and evaluation. To make the instruction more user friendly, the manual also consists of several code snippets.

2 System Configuration

2.1 Software Requirements

The code was executed on an open source environment IDE known as the "Jupyter Notebook" which is available on the "Anaconda Software". The environments can be created with respect to any programming languages. For the current research the environment runs on a python module.
Tensorflow was the package used to build and implement the machine learning models. The package can be installed using the python module. For "Tensorflow" to detect and use the graphic card of the device, several software are to be installed accordingly. One is the CuDNN and other one is CUDA which are available on NVIDIA website. Detailed description of installation are give further in the manual.

2.2 Hardware Specification

- Processor: 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz 2.69 GHz
- Graphic Card: NVIDIA GeForce RTX 3050 Laptop GPU
- RAM: 16.0 GB (15.7 GB usable)
- System type: 64-bit operating system, x64-based processor
- Storage: 500 GB SSD
- Operating System: Windows 11(64 bit)

3 Installation and Environment Setup

- Python: Python programming language was used to build the research project. There are many advantages of using this languages as it supports Deep learning and Machine Learning models with its built-in modules. To install the python package, based on your operating system the installer packages can be downloaded through their website [1] through any type of browser. The figure 1 shows the snippet of the website. After installing the python, one can check whether the python

Figure 1: Python Website

was installed properly by either typing python in the start menu which will show some results related to the python or by typing "python -version" in the command prompt which will show the version of python currently installed in the system or by typing "python" in command prompt which will open a python environment which is shown in the figure 2 below.

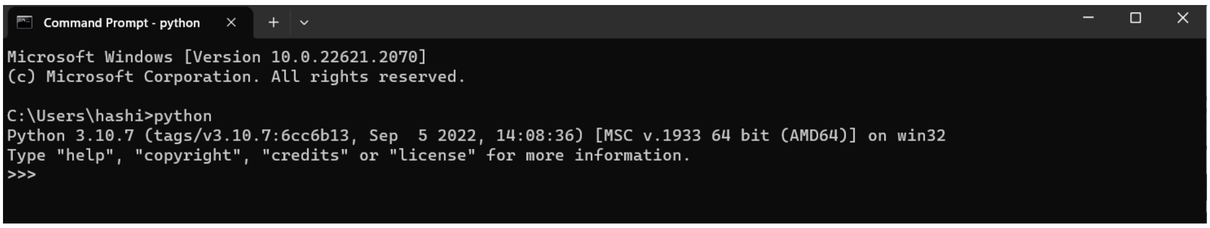


Figure 2: Command Prompt

- Anaconda: Anaconda have many modules and packages which are related to product development, code development, IDE etc. For this research we require the "Jupyter Notebook" IDE which is available through anaconda. To install Anaconda, the installer can be downloaded from their website [2]. After installing the anaconda, one

[1] https://www.python.org/downloads/
[2] https://www.anaconda.com/products/individual

can see the navigator in their start menu through which one can find the "Jupyter Notebook" IDE which is shown in the figure 3 below.

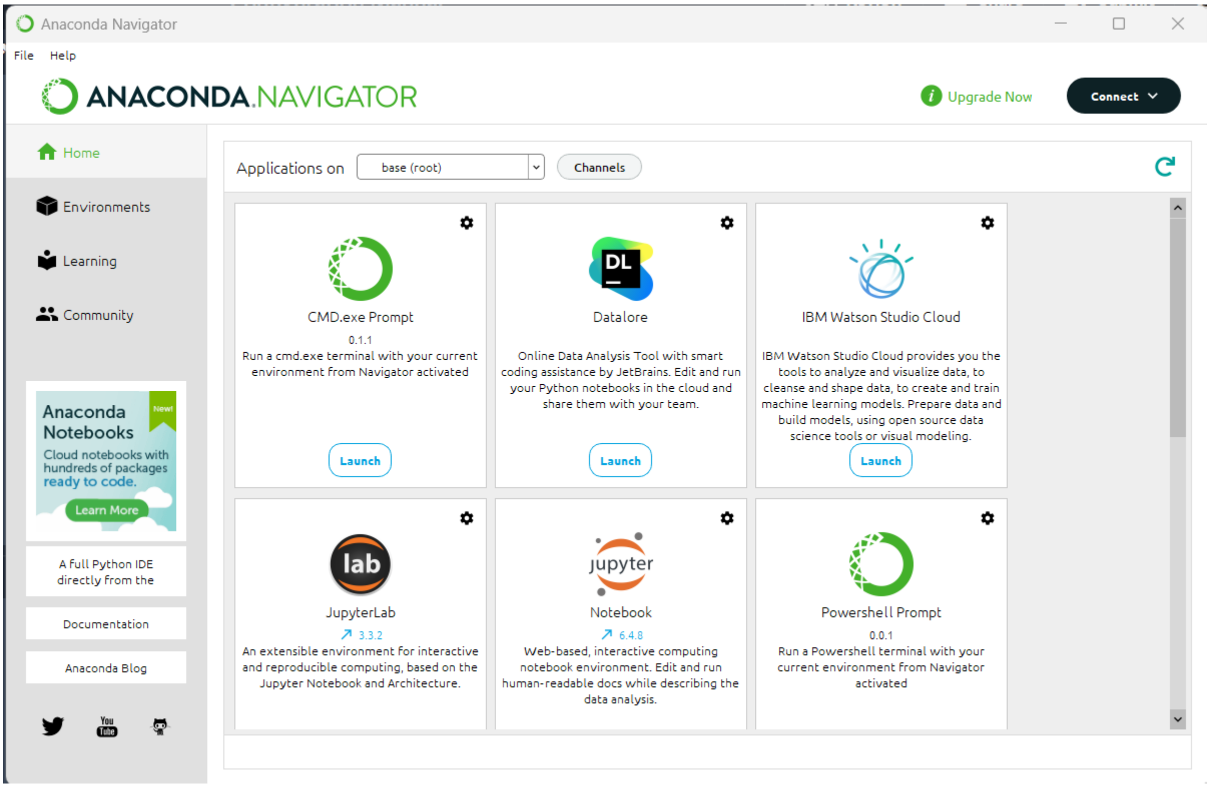


Figure 3: Anaconda Navigator

- Jupyter Notebook: Juyter notebook can installed using Anaconda. After installing jupyter notebook, one can access the notebook by typing "jupyter notebook" command in the command prompt. The command open the IDE on the default browser. Any required libraries can be installed in the environment using "pip install package name" command.

- Tensorflow: For the tensorflow which supports the GPU in the training can be tricky as it requires other supporting softwares to be installed in the system beforehand. Before installing "Tensorflow", download and install the "CuDNN" and "CUDA" from NVIDA websites [3] and [4]. The snippet of the site is shown below in the figure 4. The version have to inaccordance with the graphic card installed in the system, thus make sure the graphic card name is known before downloading the siftwares. Make sure the version of CuDNN and CUDA are supporting as all the version have different support with repect to each other. As the current system, CUDA version 11.2 and CuDNN version 8.1 is used which are said to stable for tensorflow 2.10. After installing cuda and cudnn one can verify the installation by typing "nvidia-smi" which will show the result similar to the result shown in the figure 5 below. Also by typing "nvcc -V" command one can check the version of CUDA installed which is shown in figure 6 below. If the command doesn't work, then the whole

[3] https://developer.nvidia.com/cudnn
[4] https://developer.nvidia.com/cuda-downloads

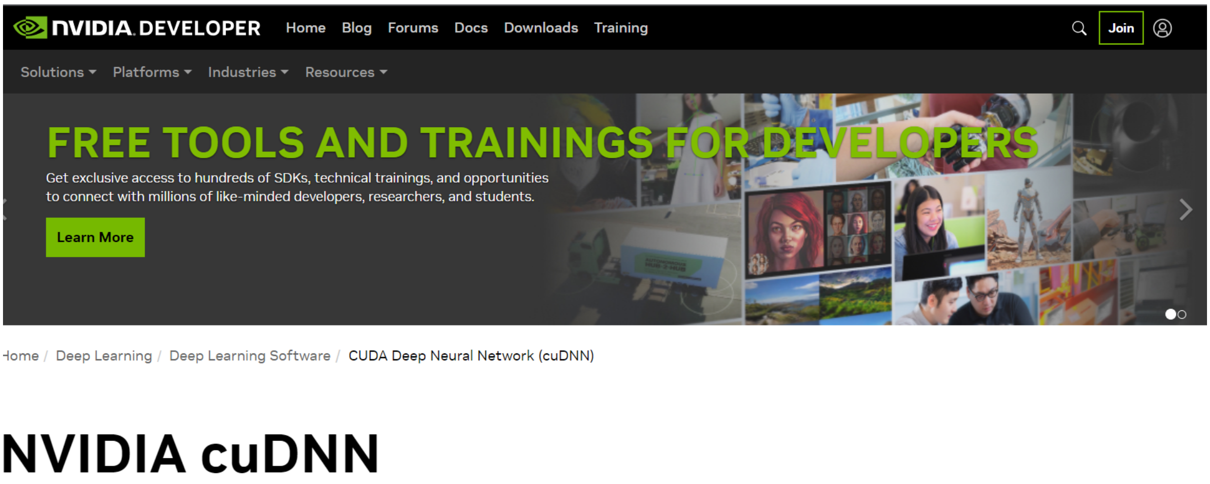
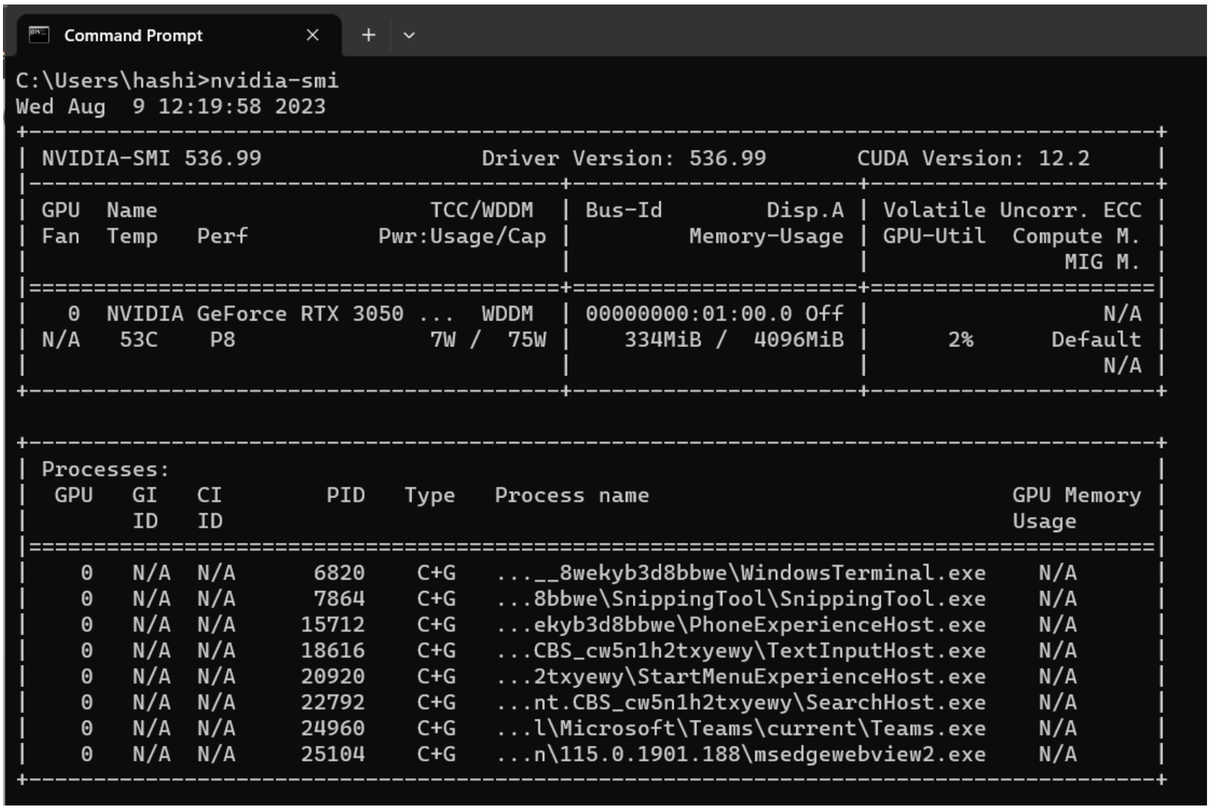


Figure 4: NVIDIA Website




```
C:\Users\hashi>nvidia-smi
Wed Aug  9 12:19:58 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 536.99                 Driver Version: 536.99       CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                     TCC/WDDM  | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 3050 ...  WDDM  | 00000000:01:00.0 Off |                  N/A |
| N/A   53C    P8               7W /  75W |    334MiB /  4096MiB |      2%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A      6820    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe      N/A      |
|    0   N/A  N/A      7864    C+G   ...8bbwe\SnippingTool\SnippingTool.exe      N/A      |
|    0   N/A  N/A     15712    C+G   ...ekyb3d8bbwe\PhoneExperienceHost.exe      N/A      |
|    0   N/A  N/A     18616    C+G   ...CBS_cw5n1h2txyewy\TextInputHost.exe      N/A      |
|    0   N/A  N/A     20920    C+G   ...2txyewy\StartMenuExperienceHost.exe      N/A      |
|    0   N/A  N/A     22792    C+G   ...nt.CBS_cw5n1h2txyewy\SearchHost.exe      N/A      |
|    0   N/A  N/A     24960    C+G   ...l\Microsoft\Teams\current\Teams.exe      N/A      |
|    0   N/A  N/A     25104    C+G   ...n\115.0.1901.188\msedgewebview2.exe      N/A      |
+---------------------------------------------------------------------------------------+
```

Figure 5: NVIDIA-SMI command result

procedure have to be re-executed properly and by checking all the version of every software that was listed above.

```
C:\Users\hashi>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Mon_Nov_30_19:15:10_Pacific_Standard_Time_2020
Cuda compilation tools, release 11.2, V11.2.67
Build cuda_11.2.r11.2/compiler.29373293_0

C:\Users\hashi>
```

Figure 6: nvcc command result

4 Data Collection

The dataset used in the project is created by the organization named FIFA. These data set can be downloaded from kaggle which is shown in the link [5] which is from 2019. This includes the information of all the players that were registered to the FIFA with their current statistics and several other informations for example their nationality, current team and so on.

5 Implementation(Base Paper Model)

The first step was to import required libraries which will be used during the process. The following figure 7 depicts the libraries that were used.

```
import pandas as pd
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import seaborn as sns
```

Figure 7: Imported libraries

5.1 Data Pre-Processing

For the initial pre-processing the data was splitted because there were multiple preferred positions for a single player. A new dataframe was created to add new rows including the single and splitted positions of the players. In the new dataframe, all the positions were modified to follow single writing pattern such as "RW" instead of "RW " and so on. Also, the positions were narrowed down to only 9 major positions that were "ST", "WN",

[5]https://www.kaggle.com/datasets/javagarm/fifa-19-complete-player-dataset

"GK", "CM", "CB", "CAM", "MF", "CDM", "DF", "CF". To implement the Random Forest model, the final predictor was factorised and labels were created using the library "Multi Label Binarizer" which is shown in the figure 8. below. After creating labels, the

```
mlb = MultiLabelBinarizer()
```

```
P = d3.iloc[:, 63:64]
print(P)
P_l = P.values.tolist()
# mlb = MultiLabelBinarizer()
mlb.fit(P_l)
P_LABELS = len(mlb.classes_)
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i, label))
lab_p = []
for (i, label) in enumerate(mlb.classes_):
    print("{}. {}".format(i, label))
    lab_p.append(label)
lab_p
```

Figure 8: Creating Labels

labels were replaced accordingly in the dataframe. The "Value" of the player and "Wage" of the player column were modified by stripping the alphabets. New classes were created for nationality of the players as well using the same library "Multi Label Binarizer". The changed were applied to the dataframe. Similar thing was done to the "Clubs" column which represents the current club the player is playing in. Also, all the "NaN" were replaced by "Na". The statistics of the players had equation which represented the change in their statistics with respect to their last results. Thus, these were having values like "65+5" or "73-2", in which the first number represented the previous statistic and the later one represented how much it was changed and whether the change was positive or negative. The column had to be changed, thus to add and subtract each row, a function was created which is shown below in figure 9. The labels were also created and the format was saved into another variable called "names" for the names of the players and the column for the names of the players was removed. This was done to later represent the name of the player with respect to the positions that was predicted by the model. The final predicting variable was removed from the main dataframe and added to a new variable named "y" which represented the final predictions. The data was splitted into train and test using the function "train test split" and the ratio was 80 percent train

```
final = 0
for k in cols:
    for i in d3[k]:
        if '+' in i:
            print(i)
            temp = i.split('+')
            for j in temp:
                final += int(j)
            d3.replace({k:{i: str(final)}}, inplace = True)
            final = 0
        elif '-' in i and len(i) > 3:
            print(i)
            temp = i.split('-')
            for j in temp:
                final -= int(j)
            d3.replace({k:{i: str(final)}}, inplace = True)
            final = 0
```

Figure 9: Function

and 20 percent test. Also, to get similar results were time, the random state was set to 42 which allows the split to happen in similar fashion every single time. This makes the results constant.

5.2 Model Building

For the Random forest, the sklearn library have a method named "RandomForestClassifier" which is shown in the figure 10 below which represents the Random Forest model for classification. The model was created with 1000 nestimators that represent that the depth of the random forest was for 1000 layers or trees.

```
model_1 = RandomForestClassifier(n_estimators=1000)
model_1.fit(x_train, y_train)
```

```
RandomForestClassifier(n_estimators=1000)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Figure 10: Random Forest Classifier

5.3 Evaluation

For the evaluation of the random forest, the confusion matrix was created which is in the "sklearn" library. Also, a classification report was created to get the accuracy, precision, recall and F1 score of the model which is shown in the figure 11 below.

```
print(classification_report(y_test, predictions))
              precision    recall  f1-score   support

           0       0.79      0.81      0.80       341
           1       0.90      0.94      0.92       946
           2       0.81      0.72      0.76       352
           3       0.16      0.07      0.09        61
           4       0.89      0.85      0.87       706
           5       0.88      0.92      0.90       888
           6       0.87      0.94      0.90       866
           7       0.90      0.92      0.91       798
           8       0.91      0.69      0.79       274

    accuracy                           0.88      5232
   macro avg       0.79      0.76      0.77      5232
weighted avg       0.87      0.88      0.87      5232
```

Figure 11: Classification Report

6 Implementation(ANN Model)

For the implementation of ANN model, several libraries have to imported which are represented in the figure 12. below.

6.1 Data Pre-Processing

For this model, most of the pre-processing work was already done while implementing the previous model. The only things that had to be changed was the format of the data which was to be used for trainng the ANN model. Firstly, the final predictor was categorised using the same library "Label Binarizer" as shown in the figure 13 below. These categories are in binary as shown in the figure. The category is represented by an array of arrays. The length of one array is equal to the number of classes in the final predictor variable which is in this case, set to be 9. Thus, the length is 9 of each array. The 0 represent that the position was not there and 1 represents the position that was available for that particular row. After creating the category, the data was convert into scaler format as the ANN model can only take scaler format as the input data. The data was then splitted intp test and train with the ratio of 80-20 where 80 percent was for train data and 20 percent was for test data. So as to get constant result while splitting the

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler
from keras.utils.np_utils import to_categorical
# split data into train and test set
from sklearn.model_selection import train_test_split
import numpy as np
```

Figure 12: ANN Model

data, random state was set to 42. This makes the splitting constant where the program is executed thus, making the results constant in every single execution.

```
from sklearn.preprocessing import LabelBinarizer, StandardScaler
```

```
y_ll = d3["Preferred_Positions"]
encoder = LabelBinarizer()
y_cat = encoder.fit_transform(y_ll)
y_cat
```

```
array([[0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       ...,
       [0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0]])
```

Figure 13: Creating Categories

6.2 Model Building

For building the model, Sequential function was used from the tensorflow library which is shown in the figure 14 below. A total of three hidden layers were added with each having number of neuron set to be 512, 1024 and 1024 respectively. All the hidden layers had the "Relu" activation function and each layer was followed by a dropout layer with

0.1 percent of dropout rate. The model was compiled using the loss function as the

```
model_2 = Sequential()
model_2.add(Dense(512,activation = 'relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(1024,activation = 'relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(1024,activation = 'relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(9,activation = 'softmax'))
model_2.build((x_train.shape[0], x_train.shape[1]))
```

```
model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (20928, 512)              35328

 dropout (Dropout)           (20928, 512)              0

 dense_1 (Dense)             (20928, 1024)             525312

 dropout_1 (Dropout)         (20928, 1024)             0

 dense_2 (Dense)             (20928, 1024)             1049600

 dropout_2 (Dropout)         (20928, 1024)             0

 dense_3 (Dense)             (20928, 9)                9225

=================================================================
Total params: 1,619,465
Trainable params: 1,619,465
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: ANN model 1

"categorical crossentropy" and optimizer was "ADAM". The batch size for training was set to 20 and the training was set for 500 epochs.

6.3 Evaluation

For the evaluation of the ANN model, the accuracy of the model was taken into consideration as it was a classification problem and this can be added while compiling the model by adding another parameter ”metrics=[”accuracy”]”. This will show the accuracy of each epochs.

7 Implementation ANN Model 2

7.1 Model Building

For improving the model, another hidden layer was added to the previous model having 2048 neurons and activaiton function as 'relu'. For comilation of the model, the optimizer was changed from ”ADAM” to ”SGD”. These changes can be seen in the figure 15 below. The model was trained using same batch size as before and for 500 epochs.

```
model_3 = Sequential()
model_3.add(Dense(512,activation = 'relu'))
model_3.add(Dropout(0.1))
model_3.add(Dense(1024,activation = 'relu'))
model_3.add(Dropout(0.1))
model_3.add(Dense(1024,activation = 'relu'))
model_3.add(Dropout(0.1))
model_3.add(Dense(2048,activation = 'relu'))
model_3.add(Dropout(0.1))
model_3.add(Dense(9,activation = 'softmax'))
model_3.build((x_train.shape[0], x_train.shape[1]))
model_3.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
hist = model_3.fit(x = x_train, y = y_train, batch_size = 20, epochs = 500, verbose = 1)
```

Figure 15: ANN model 2

7.2 Evaluation

As for the evaluation, the accuracy was used similar to the previous model.

8 Implementation(ANN model 3)

For the third model, the model was used to predict the value of the players which was the next stage of the framework. This model was different from previous model as the problem is a regression problem as the final predictor was the value of the player.

8.1 Data Pre-Processing

The final predictor variable was changed from ”Positions” to ”Value”. Similar scaler transform was done to the data as was done for the previous model and data was splitted into test and train with 80-20 ration where the 80 percent was for the training and 20 percent was for testing. The random state was set as same before which was 42 to get constant results.

8.2 Model Building

The model was created by making three hidden layers, having 512, 1024 and 1024 as the neurons for respective layers. The layers had "relu" as the activation function and the final output layer's activation function was changed from "Softmax" to "Linear" and the neurons from 9 to 1. This is due to the dimension of the final predictor. All the changes can be seen in the figure 16 below. The model was ran on the batch size of 20 and for 500 epochs.

```
model_2 = Sequential()
model_2.add(Dense(512,activation = 'relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(1024,activation = 'relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(1024,activation = 'relu'))
model_2.add(Dropout(0.1))
model_2.add(Dense(1,activation = 'linear'))
model_2.build((x_train.shape[0], x_train.shape[1]))
model_2.compile(loss='mean_squared_error', optimizer='Adam', metrics=['accuracy'])
hist = model_2.fit(x = x_train_t, y = y_train_t, batch_size = 20, epochs = 500, verbose = 1)
```

Figure 16: ANN model 3

8.3 Evaluation

As for the evaluation, as the model was a regression model, MSE, MAPE and MAE was calculated. The figure 17 shows the process for MSE. Similarly, other evaluation matrix are defined in the module "sklearn.metrics".

```
from sklearn.metrics import mean_squared_error
rms = mean_squared_error(y_test_t_n, pred_value, squared=False)
rms
```

```
0.6333707760928677
```

Figure 17: Evaluation of Regression Model