Ministry of Higher Education and
Scientific Research

University of Tunis El Manar, Tunisia

National Engineering School of Tunis

Industrial Engineering Department

Engineering Internship

Text Classification

Author :

MASMOUDI Imen

2nd year Industrial Engineering

Project Supervisor : Mr. Nabil Badri

Internship conducted in :

مركز الحساب الخوارزمي
Centre de Calcul El-Khawarizmi

www.cck.rnu.tn

Academic year: 2021/2022

Acknowledgements

With these few words, I would like to pay tribute to all the people who have contributed to the good progress of this project and its accomplishment in the best conditions.
This work would not have been possible without the presence of my supervisor Mr. Nabil Badri, who was always motivating, helpful and responsive. his valuable advice had spared no effort to transmit to me information and put me on the right track.
I also want to thank BambooGeeks and Google's Developer Ecosystem, namingly Mr. Salim Abid, Developer Ecosystem MENA Region Team Lead at Google, Aya Sayed, Google Developer Student Clubs' MENA Region Program Manager and last but not least Nour Bouayadi, Women Techmakers and Google Developer Groups MENA Region Program Manager. I thank them for giving me the opportunity to grow my technical skills and prepare for the TensorFlow Developer Certificate and for all the opportunities that they gave me.
I would also like to thank the members of the jury for having devoted their precious time to judge this work.

Abstract

Bullying is one of the verbal violence that can be considered as a crime. Therefore, text classification is a crucial part to everyone in our lives to provide safety.
Despite the development of Artificial intelligence, this domain of NLP in Tunisia is still somewhat underdeveloped!
The words that we use have an impact on our mental healthcare, that is why using text classification is very important to know if what we are going to read is healthy or not, harmful, or not! and if we should read it or let it slip!
In our work we started with a research about the classical AI methods used for Classification, and then we started exploring how to deal with text and how to mine it.
Afterwards, we went to the modelling part using a Deep Learning approach, where we prepared the data and the models and tuned the hyper parameters and then started the training process.
After that we tested our models and approaches on different dataset.

Key words: Artificial Intelligence: AI, Machine Learning: ML, Deep Learning: DL, Natural Language Processing: NLP, Neural Network Method

Résumé

L'intimidation est l'une des violences verbales qui peut être considérée comme un crime. Par conséquent, la classification des textes est un élément crucial pour la sécurité de chacun dans nos vies.

Malgré le développement de l'intelligence artificielle, ce domaine de la PNL en Tunisie est encore quelque peu sous-développé.

Les mots que nous utilisons ont un impact sur notre santé mentale, c'est pourquoi l'utilisation de la classification de texte est très importante pour savoir si ce que nous allons lire est sain ou non, nuisible ou non ! et si nous devons le lire ou le laisser passer !

Dans notre travail, nous avons commencé par une recherche sur les méthodes classiques d'IA utilisées pour la classification, puis nous avons commencé à explorer comment traiter le texte et comment l'exploiter. Ensuite, nous sommes passés à la partie modélisation en utilisant une approche Deep Learning, où nous avons préparé les données et les modèles et réglé les hyper paramètres, puis nous avons commencé le processus d'entraînement.
Après cela, nous avons testé nos modèles et nos approches sur différents jeux de données.

Mots clés : Intelligence artificielle : IA, Apprentissage de la machine: ML, apprentissage profond : DL, Traitement du langage naturel : NLP, méthode des réseaux neuronaux

Contents

List of Figures

List of Abbreviations

ML : Machine Learning
DL : Deep Learning
NLP : Natural Language Processing
ANN: Artificial Neural Network
LSTM: Long Short Term Memory
CNN : Convolutional Neural Network
Colab : Google Colaboratory
GPU : Graphics Processing Unit
EDA : Exploratory Data Analysis

General Introduction

The way we provide constructive feedback is very specific, it falls into three portions:

1. What you liked: should do more of.
2. What you didn't like or appreciate and how to improve it: ways to improve and suggestions to do that.
3. What to Start doing: Things to include and to add. What things can you do for continues growth.

That has to be the order for how we give feedback.
Problem is, most people don't take that into consideration, they use harmful words to deliver harsh feedback that can be turned into criticism and even bullying!
I believe that if the feedback doesn't push you forward and motivate you to improve, it's not healthy feedback! and nowadays, it is very hard to find healthy environments to grow and learn, receiving healthy and constructive feedback.
I see that this subject is very important and I am very happy to have had the opportunity to work on it!
In my project, I produced models using AI and I worked on three levels, the first one was through implementing three basic models to have them as a benchmark and then I developed another approach following an article that is auto-encoder classifier and then using the Ensembling Method to classify comments to three classes: normal, hateful, or abusive.
I worked on a multi-dialect dataset having Tunisian, Lebanon and Egyptian Dialects. I have trained the models with the multi-dialect dataset and then tested it on four data, the Tunnel: Multi-dialect, the Tunisian, the Lebanon and the Egyptian datasets.

Chapter 1

Enterprise Introduction

1.1 Introduction

Figure 1.1: Centre de Calcul El-Khawarizmi

The CCK was created in October 1976 following the adoption of computer science as a university specialty at the Faculty of Science of Tunis.
It provided at that time computing services to institutions, agencies and administrations of the Ministries of National Education and Higher Education.

1.2 CCK's Services

With over 45 years of service to higher education and scientific research, the CCK provides with the following services:

- National university network management
- Hibernation of national applications and ERP websites
- Management and monitoring of the RNU network security
- RNU/OFFICE 365 Drive CCK and SMS management

- Training, digital production and technology watch

1.3 CCK's Mission

As its name indicates, the Computing Center el-Khawarizmi (CCK) has the mission to ensure, organize, promote, and encourage the use of digital technologies in the academic and scientific community in general.
The CCK is also responsible for research in this field to improve the use of digital information technology in the university environment and for the benefit of teachers and students.
The main purpose of the center is to promote scientific and technical academic activity by putting online pedagogical courses, scientific and technical information sources and administrative applications for students and researchers.

The CCK's mission is to organize, promote, ensure and encourage the use of digital technologies in the university and scientific environment in general.

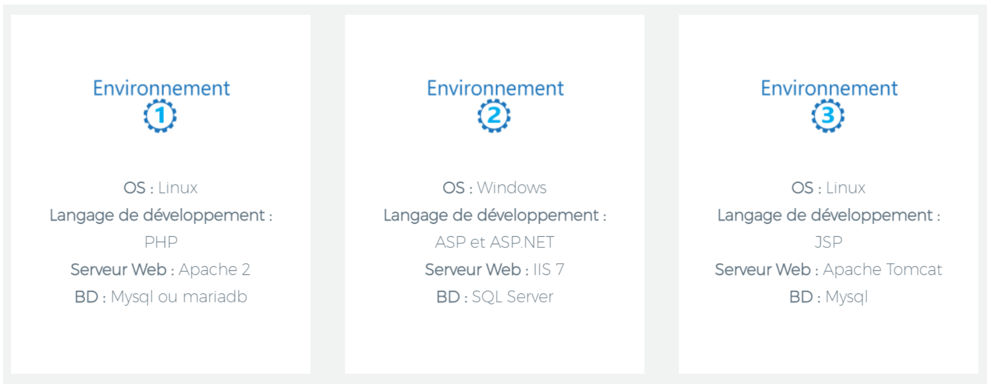
It is also in charge of research in this field in order to improve the use of digital information technology in the university environment and for the benefit of teachers and students.

Rich of its history and benefiting from its location in the university campuses, the CCK is oriented towards the diversification of its activities and the improvement of the quality of its services in the field of ICT.

By being mainly a producer of contents to the service of the Tunisian university establishments and by associating itself with the researchers in ICT to offer an experimental framework to the research in this field and to work on the transfer of the obtained research results.

CCk has over 352 Hosted websites, 9 National applications, 9 Partners, 14656 Mail accounts
They have established partnerships with: Ooredoo, Orange, Huawei, ASREN, ICANN, AFRINIC, ATI Tunise Internet, Tunisie Telecome, Agance National de la Sécurité Informatique, , ANCE, La Poste Tunisienne and AFNIC.

They have 3 environments available for hosting platforms:



Environnement 1

OS : Linux
Langage de développement :
PHP
Serveur Web : Apache 2
BD : Mysql ou mariadb

Environnement 2

OS : Windows
Langage de développement :
ASP et ASP.NET
Serveur Web : IIS 7
BD : SQL Server

Environnement 3

OS : Linux
Langage de développement :
JSP
Serveur Web : Apache Tomcat
BD : Mysql

Figure 1.2: environments available for hosting platforms

For more information, you can check their website through this link: https://cck.rnu.tn

Chapter 2

Literature overview

2.1 Introduction to AI

Artificial intelligence (AI) is a wide-ranging computer science branch that is concerned with building smart machines that are capable of performing tasks that typically require intervention and human intelligence.
The fields of AI are:

- Computer vision
- Natural Language Processing: NLP
- Time Series Forecasting

In this project we will focus on the NLP filed of AI.
AI is based on Maths, statistics and Algebra, that is why we have fields like data mining and data engineering, it is because we need good numeric data that can provide us with good information.
The definition of Data Mining is the following:
Data mining is the process of extracting and discovering patterns in large data sets involving methods at the intersection of machine learning, statistics, and database systems.
And the definition of Data Engineering is the following:
Data engineering refers to the building of systems to enable the collection and usage of data. This collected data is usually used to enable analysts and data scientists to work on it, which often involves machine learning. The Big data is changing the way we do business and creating a need for data engineers who can collect and manage large quantities of data.
In the figure below, we can find the sub-fields of AI, So let's explore some parts of it:

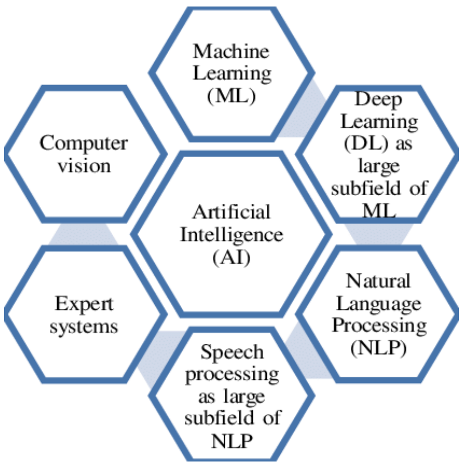

Machine Learning (ML)
Computer vision
Deep Learning (DL) as large subfield of ML
Artificial Intelligence (AI)
Expert systems
Natural Language Processing (NLP)
Speech processing as large subfield of NLP

Figure 2.1: AI Sub Fields
[5]

Let's explore some of the AI sub-fields that we will be using in this project.

2.2 Machine Learning

In its definition, Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.
In the figure below you, can see the most famous ML Tasks Classification:

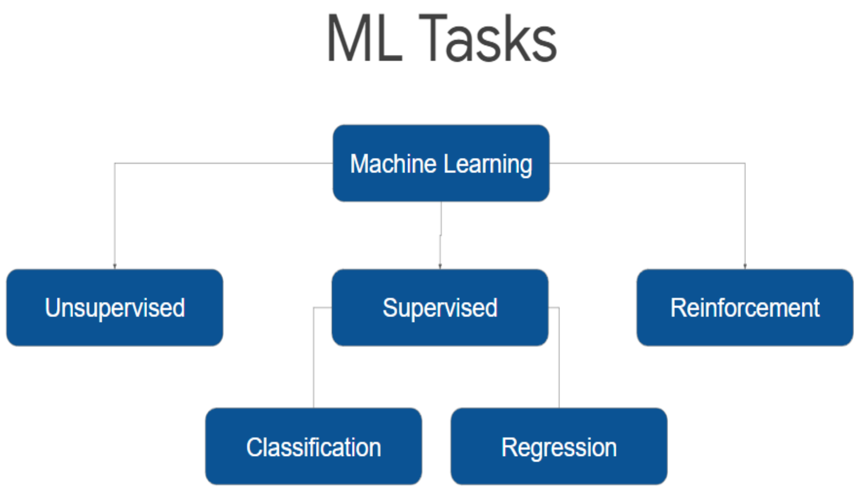


Figure 2.2: ML Tasks

The most famous ML Tasks are the supervised tasks, the unsupervised tasks and the reinforcement tasks, let's explore them a little more:

2.2.1 Reinforcement Learning

In its definition, Reinforcement learning is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward.
In the picture below we have an illustration of the concept of Reinforcement Learning:

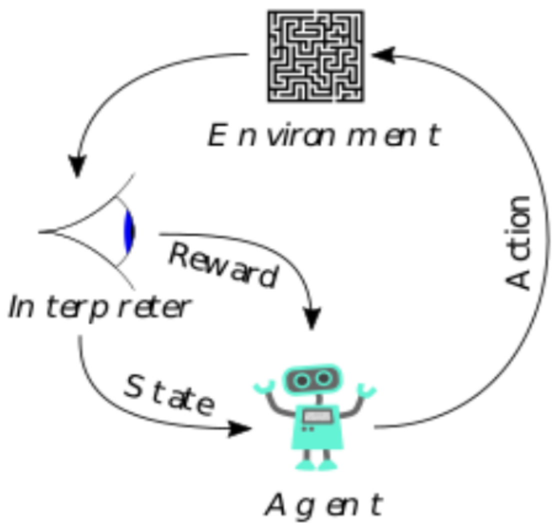


Figure 2.3: Reinforcement Learning

Basically the agent collects the data as it evolves in the learning process. After each action, the model gets tuned.

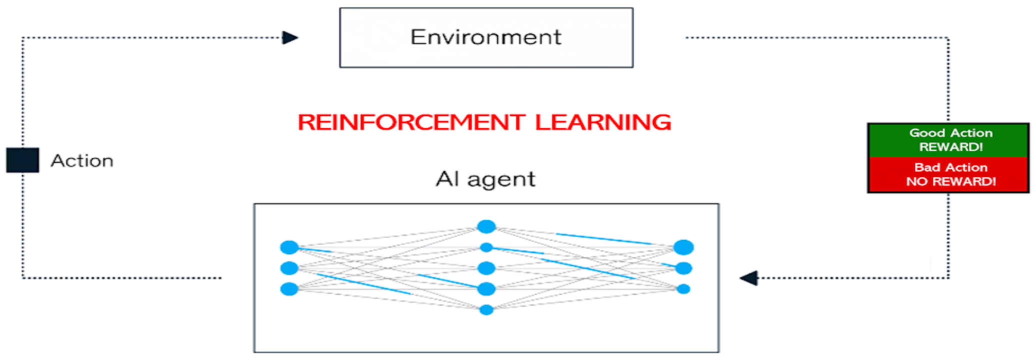


Figure 2.4: Reinforcement Learning

2.2.2 Unsupervised Learning

The unsupervised learning is a type of algorithm that learns pattern in the data, here we can find association problems and clustering problems.
In the clustering problems we can use ML models that are already defined in python like: hierarchical clustering and k-means.

2.2.3 Supervised Learning

The supervised Learning is know for having Xs and Ys, meaning by that having a labeled data, it is composed of two categories.

2.2.3.1 Regression Problems

in the regression problems, the Y, which is our label is a continues value and we try to predict a continues value like the price of a stock or the demand for the next month in a product.
Here is an example of a regression problem:

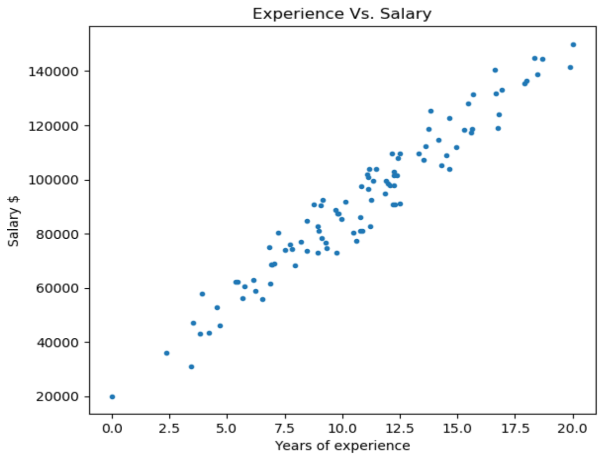


Figure 2.5: Regression Problem

in the regression problems, we have ML models and algorithms that are already defined, like: Linear Regression, K Nearest Neighbours: K-NN, Polynomial Regression Decision tree regression...

2.2.3.2 Classification Problems

In the classification problems, we try to predict a class that our data belongs to or reassembles to the most.
Here is an example of a classification problem:

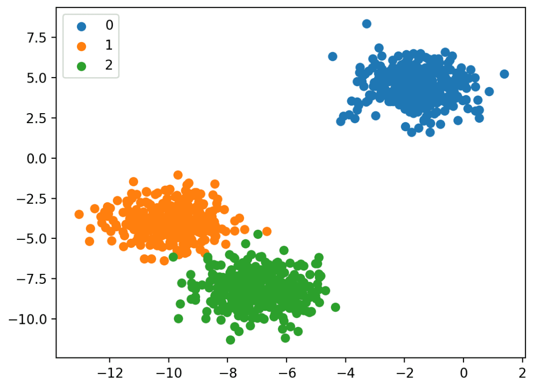


Figure 2.6: Classification Problem

In the classification problems, we have ML models and algorithms that are already defined and ready for us to use, like: Logistic Regression, Naive bayes, K

Nearest Neighbour: K-NN, Decision Tree...

2.3 Deep Learning

In its definition, DL is a subset of ML, which is essentially a neural network with three or more layers.
DL drives many AI applications and services that improve automation, performing analytical and physical tasks without human intervention.
When we talk about DL, the first thing that we should prepare is huge data, because DL is very data driven. The more data you give it, the better it becomes. We can clearly see in the graph below that when the data size increases, DL become much more efficient than ML:

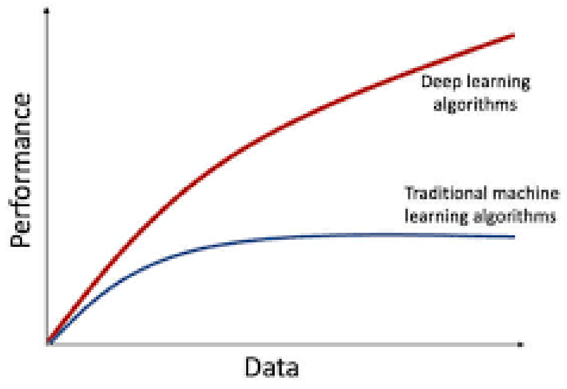


Figure 2.7: ML vs DL
[3]

DL is based on Neural Networks that has more than three Layers, it was first inspired from the neuroscience and the human brains neurons and the way that they are connected, and the way that they transform message to each other and how there is an activation point where we start to have an output from the neuron. All of this have inspired the AI founders to build this domain.
One of the building blocks of DL is the perceptron, which is the simulation of the breain neurons.
In the graph below you can find a perceptron model that takes input and multiplies it by weights:

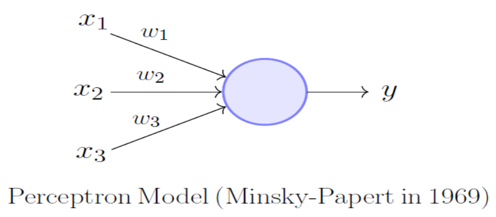


Figure 2.8: The percepton Model
[8]

Every neuron in the neural network takes input variables, they get multiplied by weights and summed and an activation function is then applied to get the output of that neuron.
In the graph below is a more detailed explanation of how we do this:

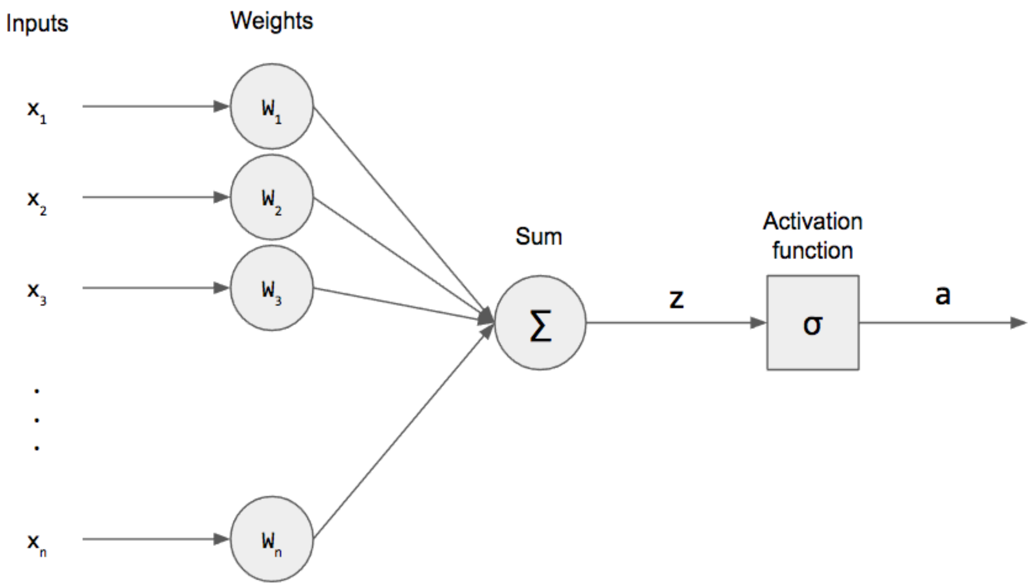


Figure 2.9: The percepton Model detailed

The activation function that can be used are so many, amongst them we can find the RELU activation function, The Sigmoid activation function, and the SoftMax activation function, every one of them has a formula and there are so many other. but let's explore the ones that are the most basic and what they do:

1. The RELU activation function: This function takes a value and returns a zero if the input is negative and returns the number if positive.

2. the sigmoid activation function: This function takes an input and gives it's Euler's number, which means it applies this formula on it:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Figure 2.10: The Sigmoid Activation function formula

3. The SoftMax activation function: This function gives us a probability at the end as it applies the exponential to the input and then divides by the sum of all the other neuron's input's raised to the exponential, here is the formula used:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 2.11: The SoftMax Activation function formula

And these were the activation functions that I uses in my work.

2.4 Natural Language Processing: NLP

In its definition, Natural Language Processing is the practice of teaching machines to understand and interpret conversational inputs from humans.
As much as it is easy to explore numbers with AI, we can clearly see that the first challenge here is to transform the text into numbers that the models can work with. Once the text becomes numbers, The pattern exploration can begin.
In the beginning of exploring the NLP Field, it is very important to know where we are standing after seeing what AI, ML and DL mean.
In the graph below we can explore where the NLP Stands in the middle of all the fields explored:

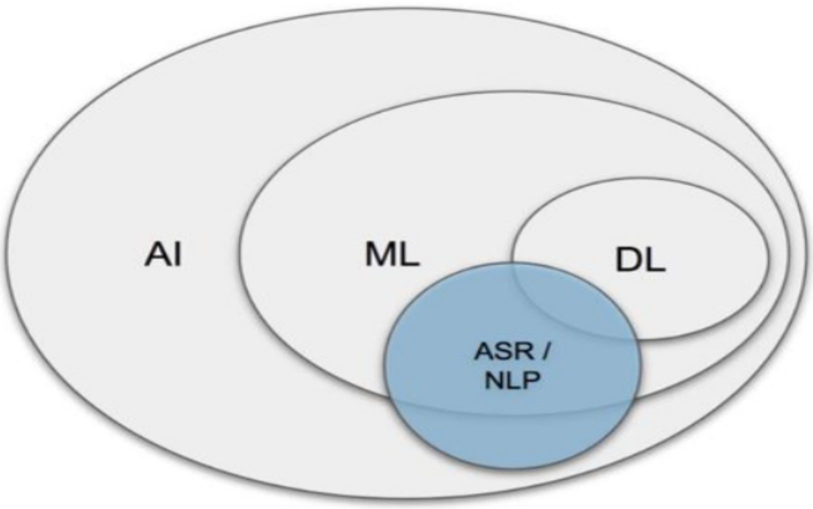


Figure 2.12: NLP In AI

So the NLP field stands somewhere in AI and it belongs to both ML and DL as we use the text transformations tecniques and use neural networks to simulate the human brain.

2.4.1 NLP Building blocks

Through varying different neural networks, advanced NLP algorithms can be created. Machine learning-based NLP algorithms, and most AI, are Directed by Acyclic Graphs networks, DAG, which comprise artificial neurons that are arranged in inter-connected layers that are propagating into the last layer, the output layer. These models represent how neuroscientists think that our brains work.
Here we will explore some of the building block of the NLP:

2.4.1.1 Recurrent Neural Networks: RNNs

The Recurrent Neural Networks, RNN, takes in consideration the past data. In the RNN neurons, the output is re-fed to the neuron so that the algorithm can see the past data and process it.

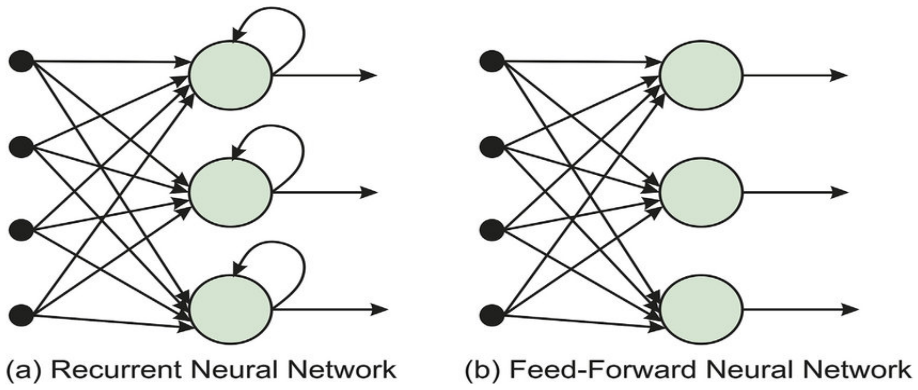


Figure 2.13: RNN VS ANN
[6]

2.4.2 Long Short Term Memory: LSTM

The LSTM mainly has three components, they are called input, output and forget gates. This type of neurons is classified under the umbrella of Recurrent Neural Networks as its output loops over and is being fed to itself. The trick that is used in this method is that the cell, as showing the figure below, “can remember data for a random duration of time and the three main components that act as regulators of information into this cell and out of this cell.”[6]
Thanks to this uniquely structured system, the network won’t have the disappearing problem and essentially wont forget the parameters. This is why LSTM is the best RNN algorithm for application to time series sequential data and NLP.

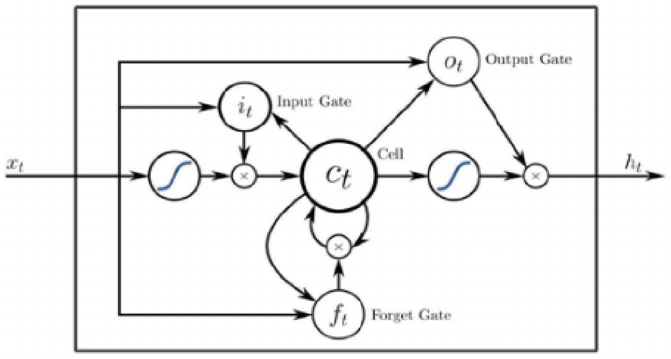


Figure 2.14: LSTM
[6]

2.4.2.1 Dense Layers

In neural networks, we did raise the dimensionality of the vectors that are being used. To change a vector from “n”dimensions to “m”dimensions, meaning by that if the input of the dense layer has this shape [L,m] and we have a layer of n dense units, the output of this layer will have this shape: [L,n]. In this project, the final layer is defined as a dense layer to reduce the number of outputs to three or two units.
In the figure below you will find a Dense layer having four units.

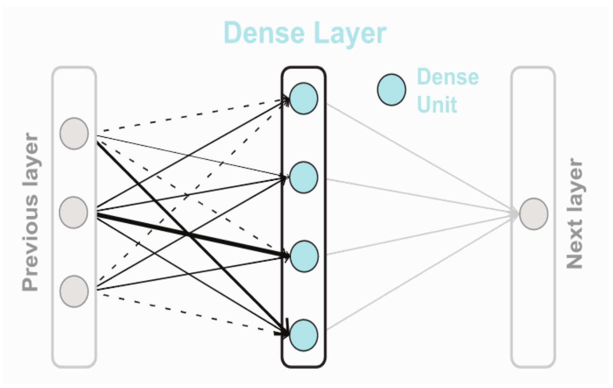


Figure 2.15: Dense Layers
[7]

2.5 Research for tools

When working on a practical AI project and dealing with data and modelling, we will be using few very useful tools like python and some of its libraries like TensorFlow. We will also be using Google Colab. So let’s learn more about them before we start using them.

2.5.1 Why Python?

First of all, Python is a popular general-purpose programming language that can be used for a wide variety of applications. It was created by Guido van Rossum, and released in 1991.
It’s one of the most popular and useful programming languages since it’s very attractive for Rapid Application Development due to:

- It’s simple and easy syntax.
- Its support modules and packages.

In this and the following chapter, we will perform Language classification with deep learning algorithms by implementing them with python.

2.5.2 TensorFlow

High-performance computations are commonly done through readily developed packages. TensorFlow is an open-source package developed by the Google Encephalon Team within Google in 2015. It's a bunch of APIs developed to build ANN models. For developing machine learning and deep learning programs with various architectures.[6]
The advantage of using such a platform is that the basic equations and rules are pre-defined in the library and the user can access and reach a solution in a much quicker manner. In this research, TensorFlow is imported in Python 3 for development of LSTM that predicts stock prices.
In the TensorFlow framework, we find Keras, which is a high-level neural network library and it's tyro-convivial since it avails building training models in a facile way.

2.5.3 Google Colab

Google Colaboratory is the impeccable Chromebook coding experience to utilize Python (especially for machine learning). It is an executable document that lets you indite, run, and share code within Google Drive.
A notebook of Colab is composed of cells where each one can contain code, text, images, and more. Colab connects the notebook to a cloud-based runtime, meaning the user can execute Python code without any required set up on his own machines. Supplemental code cells are executed using that same runtime, resulting in a rich, interactive coding experience in which the coder can use any of the functionality that Python offers.
In the following experimental parts, Colab is the workspace that was utilized!
In conclusion we chose google colab as our working environment for few of the following reasons:

- Here on colab we have amazing features like GPU accelerators, the access we have is still limited but it gives us enough resources to do the work!
- We can link the colab environment to Drive, where we stored our data and where we are intending to store the model h5 file.
- Here we can have code cells and text cells: that enables us to write comments on the code created, which is very helpful when it comes to collaborating and

to having reminders to yourself on what you were thinking when writing the code.

2.6 Evaluation metrics for classification

When evaluating our models we will first talk about some matrices that we use in the classification models like the accuracy and the F1-Score, but before we do that, There are four therms that we need to learn:

- True Positive, TP: we get it when we get a prediction that says that the class is zero, and in reality it is zero.
- True Negative, TN: we get it when we get a prediction that says that the class is not zero, and in reality it is not zero.
- False Positive, FP: we get it when we get a prediction that says that the class is zero, and in reality it is not zero.
- False Negative, FN: we get it when we get a prediction that says that the class is not zero, and in reality it is zero.

Now let's explore the accuracy and F1-Score:

2.6.1 The accuracy

The accuracy gives us an indication on how good our model is doing by giving us the fraction of the good predictions to the totality of them.
Below is the formula for the Accuracy:

$$\text{Accuracy} = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

Figure 2.16: The Accuracy formula

2.6.2 The F1-Score

The F1-score combines the precision and recall of a classifier into a single metric becoming their harmonic mean.
Here is the formula for the precision:

$$\text{Precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})}$$

Figure 2.17: The Precision formula

Here is the Formula for the recall:

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

Figure 2.18: The Recall formula

And finally here is the formula for the F1-Score

$$\text{F1 Score} = \frac{2 \text{ x } (\text{Precision x Recall})}{\text{Precision + Recall}}$$

Figure 2.19: The F1-Score formula

2.7 Conclusion

At the end of this section, we got some concepts cleared, though we learn much more by doing we can say that we are ready to start working on the project pipeline.

Chapter 3

Project Pipeline

3.1 Introduction

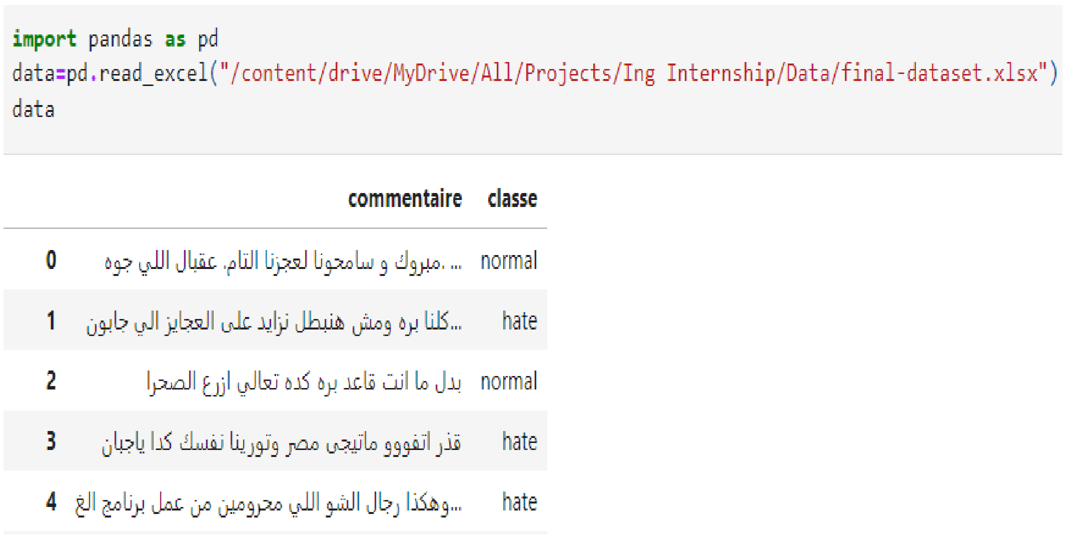
In the beginning of my work, I would like you to open my github repository, where I did put all of my code. Please note that I didn't put there any data as it is not open source and it belongs to CCK only.
Here is the link to my git hub repository: https://bit.ly/GithubRepositoryCCKProject
You can follow along with the complete versions of the code and see the outputs of each code cell.
In a general view, we explored three big approaches to this problem, The first approach is through neural networks, where we produced three different models: Hybrid, CNN and LSTM focused.
The second Approach was following up with an article that focused on using encoder and decoders with neural networks. It is an auto-encoder classifier model.
And the final approach was the Ensembling method where we implemented the ECOC Algorithm and three ways of the voting method.
In this chapter I went through the approaches of these methods and models and in the fourth chapter, I went into the details of the results where I compared The models' ad methods' accuracies.
Now before we Start with exploring the methods we applied, let's explore our data and how we transformed it from textual data to a numeric data.

3.2 Data Exploration

In this part we transformed our data, explored it and prepared it for the models.

3.2.1 Row data

When we first got the data it was clean and already in a CSV file ready for us to explore. And it looked something like this:



```
import pandas as pd
data=pd.read_excel("/content/drive/MyDrive/All/Projects/Ing Internship/Data/final-dataset.xlsx")
data
```

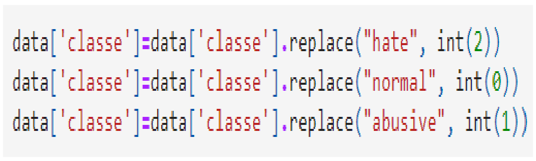
	commentaire	classe
0	مبروك و سامحونا لعجزنا التام. عقبال اللي جوه ...	normal
1	كلنا بره ومش هنبطل نزايد على العجايز الي جابون...	hate
2	بدل ما انت قاعد بره كده تعالي ازرع الصحرا	normal
3	قذر اتفووو ماتيجى مصر وتورينا نفسك كدا ياجبان	hate
4	وهكذا رجال الشو اللي محرومين من عمل برنامج الغ...	hate

Figure 3.1: Row Data
1

After exploring our Row data, we see that we have a lot of work to do, from transforming it into a numeric data and splitting it into training and validation data and maybe balancing the classes.

3.2.2 Data Transformation

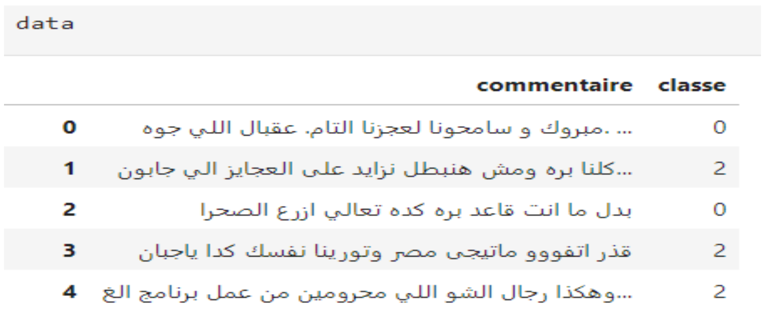
After seeing our row data, the first transformation I did was, transforming the classes to numeric values having the normal class become the class zero, the abusive class become the class one and the hate class become the class two With the following code:



```
data['classe']=data['classe'].replace("hate", int(2))
data['classe']=data['classe'].replace("normal", int(0))
data['classe']=data['classe'].replace("abusive", int(1))
```

Figure 3.2: Numerising the classes

After executing this code we transformed our data into this:

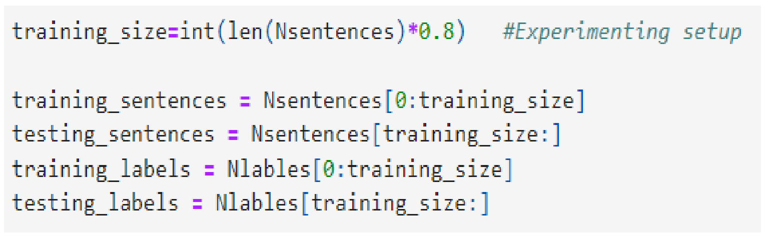


```
data
```

	commentaire	classe
0	... .مبروك و سامحونا لعجزنا التام. عقبال اللي جوه	0
1	...كلنا بره ومش هنبطل نزايد على العجايز الي جابون	2
2	بدل ما انت قاعد بره كده تعالي ازرع الصحرا	0
3	قذر اتفووو ماتيجى مصر وتورينا نفسك كدا ياجبان	2
4	...وهكذا رجال الشو اللي محرومين من عمل برنامج الغ	2

Figure 3.3: Data after Numerising the classes

After this part I Split the data into training data and validation data with this code:

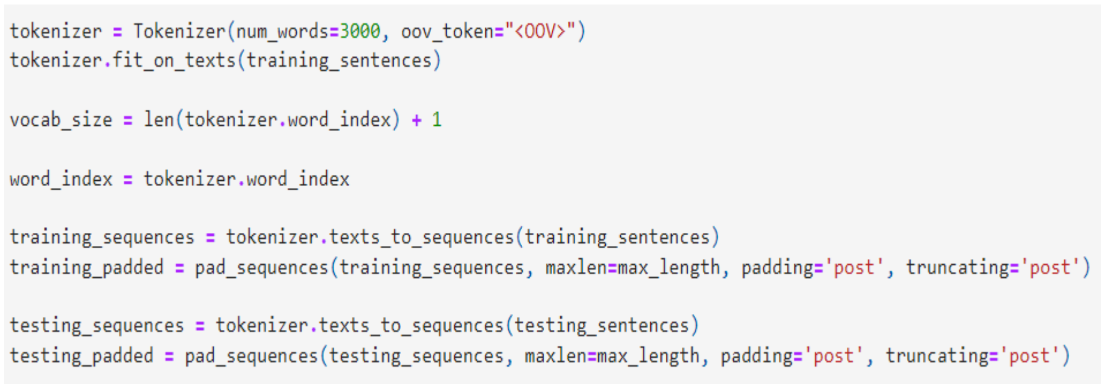


```
training_size=int(len(Nsentences)*0.8)   #Experimenting setup

training_sentences = Nsentences[0:training_size]
testing_sentences = Nsentences[training_size:]
training_labels = Nlables[0:training_size]
testing_labels = Nlables[training_size:]
```

Figure 3.4: Data splitting

After splitting our data, I used the Tokenizer function from the TensorFlow framework, and fitted it on our training dataset so we can get the tokens ready[2]. Here our tokens are saved in a dictionary having the words as keys and a unique number is assigned to each word as a value in this dictionary.
And then we padded our sequences so that the words become either a number from the tokenizer.word index dictionary or it will become a zero if it is out of the vocabulary, meaning by that it does not exist in the training data. here notice that I added a oov token="<OOV>" in the parameters of our Tokenizer, it is to take in mind that the new words will take the value zero and we will not have them ignored, as if it is not there.
In the code below I defined this part and prepare our data to have it all in a numeric form.

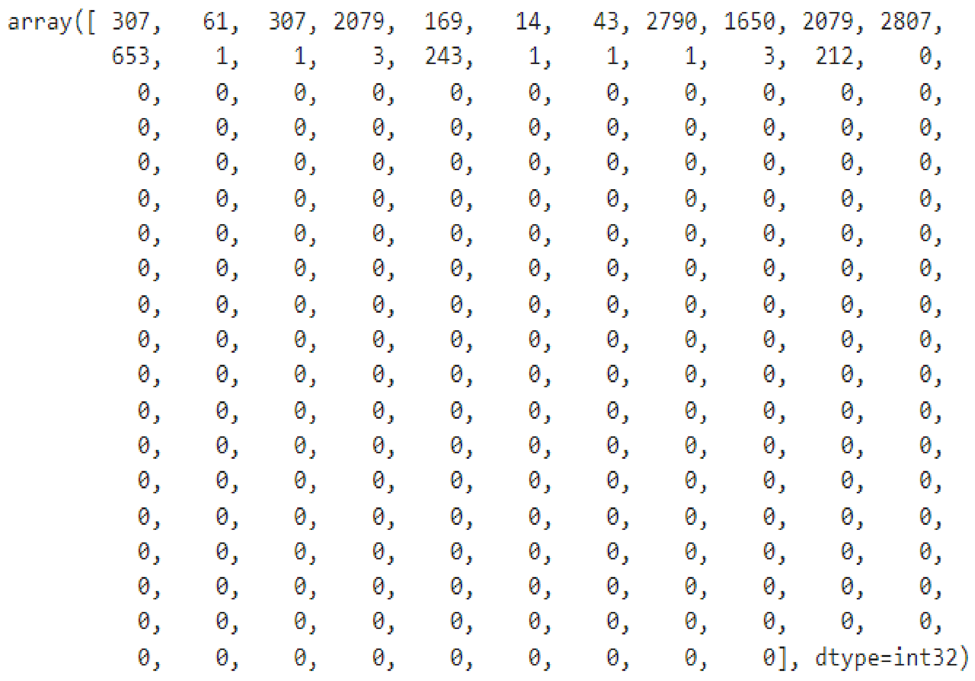
[2]A token is the smallest individual unit



```
tokenizer = Tokenizer(num_words=3000, oov_token="<OOV>")
tokenizer.fit_on_texts(training_sentences)

vocab_size = len(tokenizer.word_index) + 1

word_index = tokenizer.word_index

training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded = pad_sequences(training_sequences, maxlen=max_length, padding='post', truncating='post')

testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded = pad_sequences(testing_sequences, maxlen=max_length, padding='post', truncating='post')
```

Figure 3.5: Data tokenization

After all of this, here is an example of a comment that is now transformed int an array of numbers:



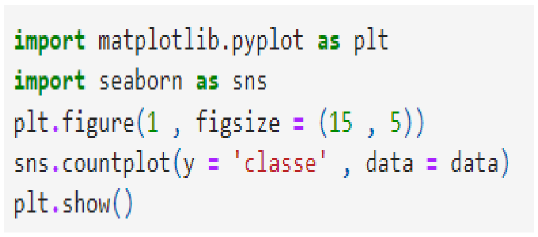
```
array([ 307,   61,  307, 2079,  169,   14,   43, 2790, 1650, 2079, 2807,
        653,    1,    1,    3,  243,    1,    1,    1,    3,  212,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          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,    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,
          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,    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], dtype=int32)
```

Figure 3.6: Numeric data

Every comment is now presented in an array just like this with every word transformed into a number.

3.2.3 Data Balancing

To balance our data, we first need to know how many data points we have in each class, I called them data points because they are now transformed to vectors with coordinates.
To check the numbers I executed this code shown in the picture below:



```python
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(1 , figsize = (15 , 5))
sns.countplot(y = 'classe' , data = data)
plt.show()
```

Figure 3.7: Code to see Data points per class

This code gave these results:

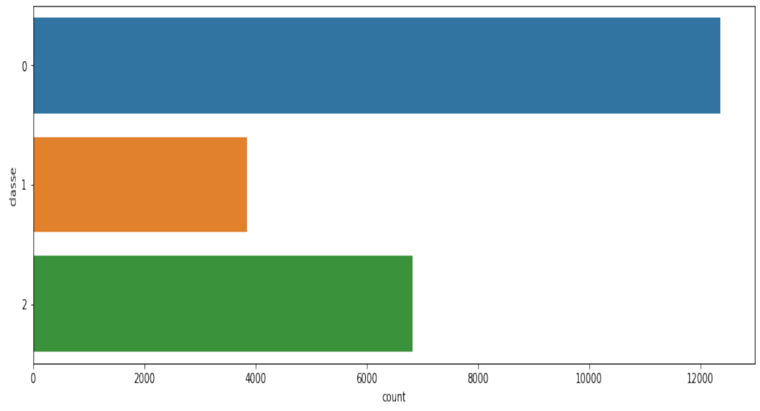
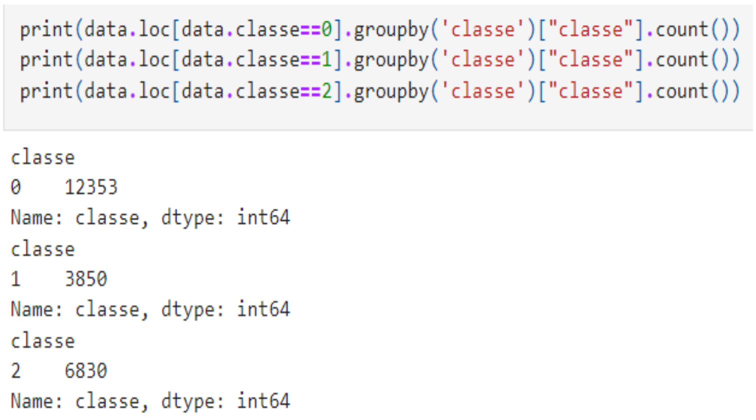


Figure 3.8: Data points per class

This plot gives us a good idea about the number of data points we have in each class, yet it does not give us the exact numbers. That is why we will run the upcoming code to get the exact numbers.

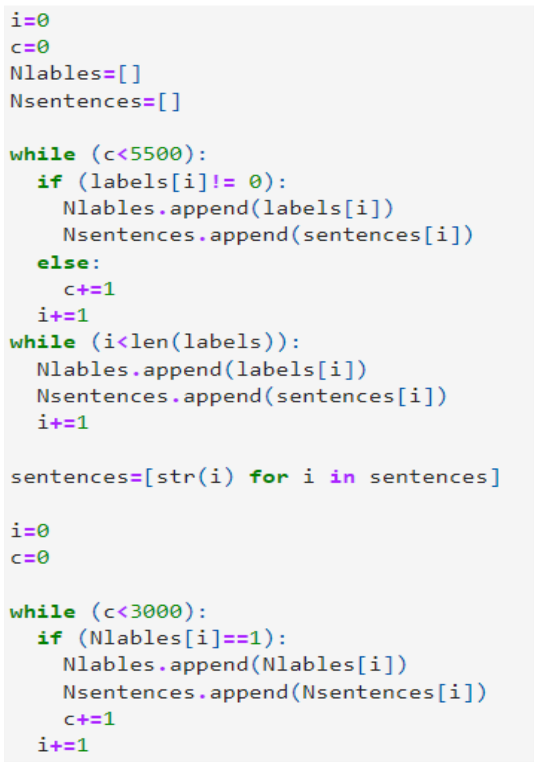


```
print(data.loc[data.classe==0].groupby('classe')["classe"].count())
print(data.loc[data.classe==1].groupby('classe')["classe"].count())
print(data.loc[data.classe==2].groupby('classe')["classe"].count())
```

```
classe
0    12353
Name: classe, dtype: int64
classe
1    3850
Name: classe, dtype: int64
classe
2    6830
Name: classe, dtype: int64
```

Figure 3.9: Exact number of Data points per class

After getting the exact numbers, I then balanced the classes with some code, here is a glimpse of it:



```
i=0
c=0
Nlables=[]
Nsentences=[]

while (c<5500):
  if (labels[i]!= 0):
    Nlables.append(labels[i])
    Nsentences.append(sentences[i])
  else:
    c+=1
  i+=1
while (i<len(labels)):
  Nlables.append(labels[i])
  Nsentences.append(sentences[i])
  i+=1

sentences=[str(i) for i in sentences]

i=0
c=0

while (c<3000):
  if (Nlables[i]==1):
    Nlables.append(Nlables[i])
    Nsentences.append(Nsentences[i])
    c+=1
  i+=1
```

Figure 3.10: Data Balancing

And after running this code we got these results:

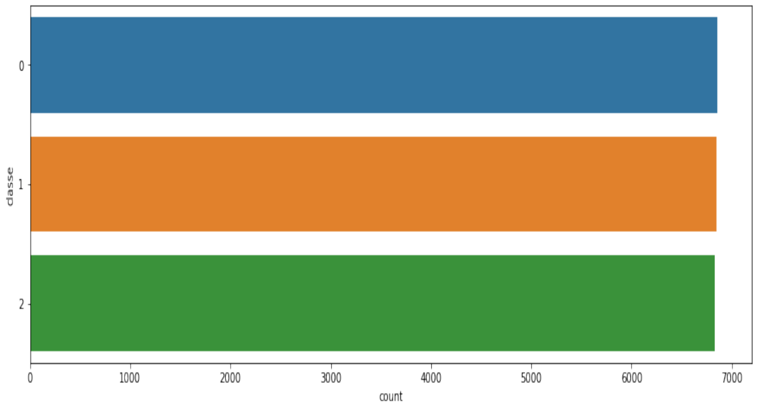


Figure 3.11: Balanced Data

3.2.4 Conclusion

After exploring our data and discovering that it was unbalanced, we decided to take two approaches and not each one's results.

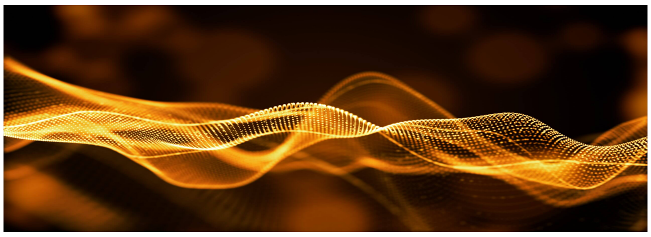


Figure 3.12: "Data Is The New Gold" Said by Mark Cuban

Every AI work is highly data driven, data is the new Gold, that is why we sometimes want to hold on to it, and that is why we took these two approaches, one with Unbalanced Dataset, and another one With a Balanced dataset.

3.3 Primary approach

After seeing our data and doing the Exploratory Data Analysis, I started modelling, I prepared three basic models: The hybrid method model, the CNN model and the LSTM model.
For more details you can explore this presentation through this link:
https://bit.ly/PrimaryApproachArchitectureDetailsImenMASMOUDI

3.3.1 The hybrid method model

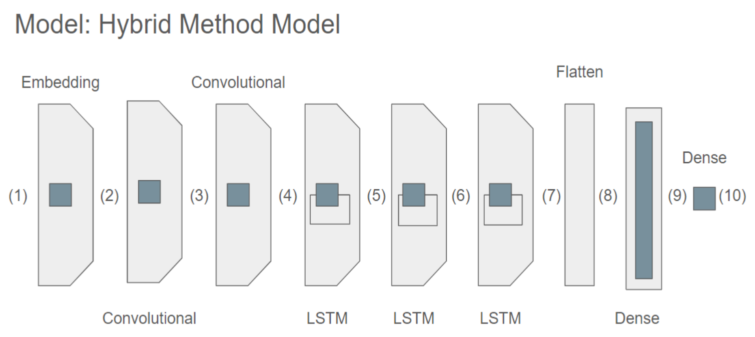
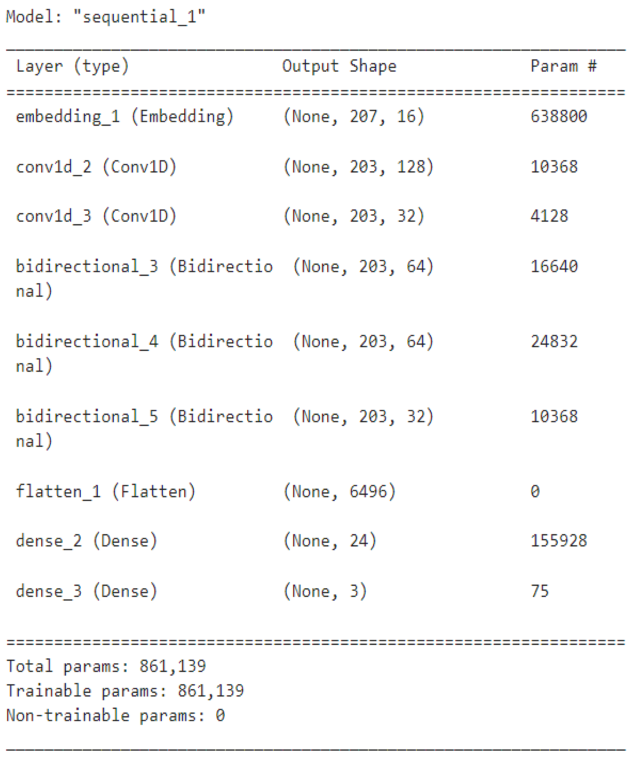


Figure 3.13: Hybrid Model's Architecture



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 207, 16)           638800

 conv1d_2 (Conv1D)           (None, 203, 128)          10368

 conv1d_3 (Conv1D)           (None, 203, 32)           4128

 bidirectional_3 (Bidirectio  (None, 203, 64)          16640
 nal)

 bidirectional_4 (Bidirectio  (None, 203, 64)          24832
 nal)

 bidirectional_5 (Bidirectio  (None, 203, 32)          10368
 nal)

 flatten_1 (Flatten)         (None, 6496)              0

 dense_2 (Dense)             (None, 24)                155928

 dense_3 (Dense)             (None, 3)                 75

=================================================================
Total params: 861,139
Trainable params: 861,139
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.14: Hybrid Model's Summary

In this first model, I used the hybrid method, which refers to using CNN layers and then using LSTM layers. In the first layer I took the input and applied it to

an embedding layer and raised the dimension up by one adding sixteen coordinates to the values that were represented just by one number.
Then I followed up with two Convolutional layers to filter some patterns in our data. The first has 128 filters and the second has 32 filters. The first CNN layer has a kernel of size [5,5], as the second's is a scalar.
I then applied three Directional Long Short-Term Memory layers to become able to follow up with the meaning from start to end and from end to start. In the end of our model, I used a flatten layer and a 24-unit dense layer having the RELU activation function and a final dense layer having three units to give us the probabilities of belonging to each class, having the SoftMax activation function.

3.3.2 The CNN model

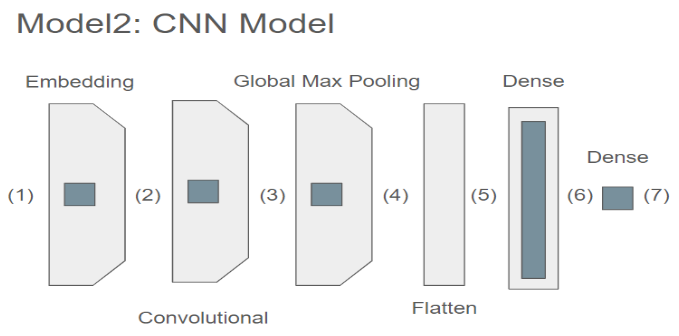
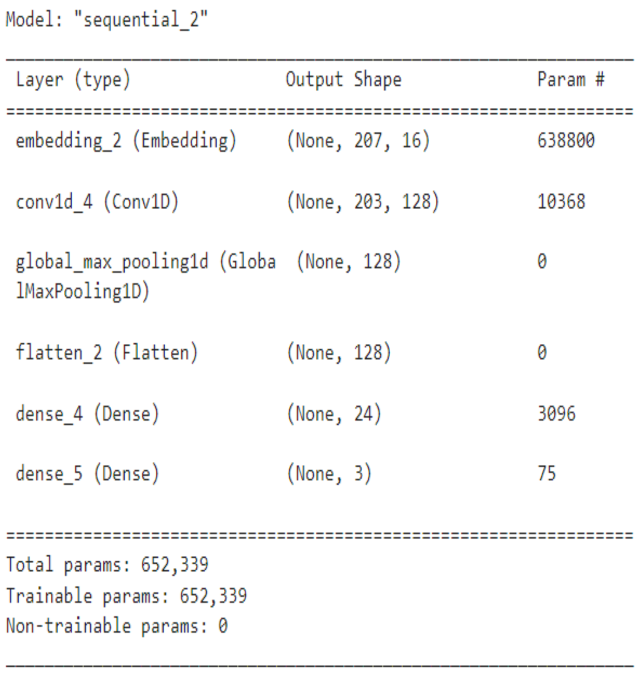


Figure 3.15: CNN Model's Architecture



```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 207, 16)           638800

 conv1d_4 (Conv1D)           (None, 203, 128)          10368

 global_max_pooling1d (Globa  (None, 128)              0
 lMaxPooling1D)

 flatten_2 (Flatten)         (None, 128)               0

 dense_4 (Dense)             (None, 24)                3096

 dense_5 (Dense)             (None, 3)                 75

=================================================================
Total params: 652,339
Trainable params: 652,339
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.16: CNN Model's Summary

In the second model, I focused on using the Convolutional Neural Networks. In the first layer I took the input and applied it to an embedding layer and raised the dimension from [207,] to [207,16].
And then I followed up with a Convolutional layer to filter some patterns in our data using 128 kernels of size [5,5] which is also known as filters. And I followed it up with a Global Max Pooling layer that reduced the dimensionality from [203,128] to just [128].
In the end of our model, I used a flatten layer and a 24-unit dense layer having the RELU activation function and a final dense layer having three units to give us the probabilities of belonging to each class, having the SoftMax activation function.

3.3.3 The LSTM model

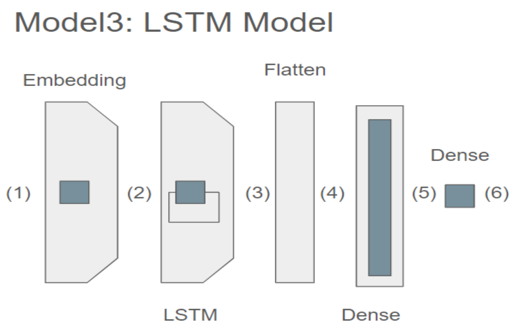
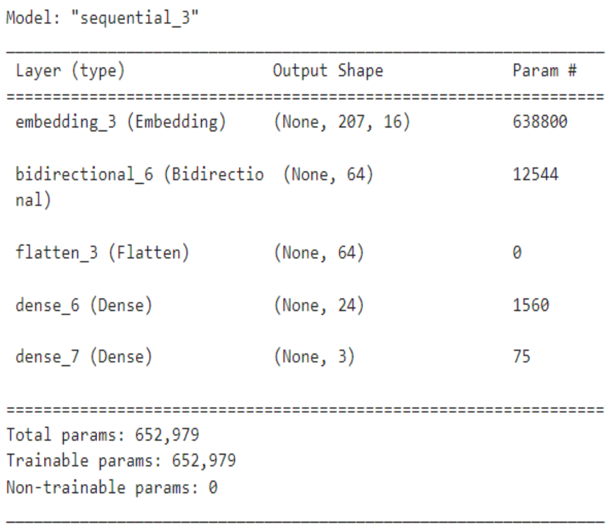


Figure 3.17: LSTM Model's Architecture



```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, 207, 16)           638800    
                                                                 
 bidirectional_6 (Bidirectio  (None, 64)               12544     
 nal)                                                            
                                                                 
 flatten_3 (Flatten)         (None, 64)                0         
                                                                 
 dense_6 (Dense)             (None, 24)                1560      
                                                                 
 dense_7 (Dense)             (None, 3)                 75        
                                                                 
=================================================================
Total params: 652,979
Trainable params: 652,979
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.18: LSTM Model's Summary

In the third model, I focused on using the Bidirectional Long Short-Term Memory layers.
In the first layer I took the input and applied it to an embedding layer and raised the dimension to 16.
and then I followed up with a Bidirectional Long Short-Term Memory layer to become able to follow up with the meaning from start to end and from end to start.
In the end of our model, I used a flatten layer and a 24-unit dense layer having the RELU activation function and a final dense layer having three units to give us the probabilities of belonging to each class, having the SoftMax activation function.

3.3.4 The training process

I trained the three models with the same datasets in the same conditions. I altered the conditions and took notes of the accuracies, the F1-Scores and the general accuracies of the models.
In conclusion I could note four different results, two of them were with Balanced Dataset and two with unbalanced dataset, where I altered the validation split from 0.8 to 0.9 as the more data that goes for training the better the model can get.
In coclusion we have four versions of the three models:

- Model that Is trained with unbalanced dataset and with 0.8 test split
- Model that Is trained with unbalanced dataset and with 0.9 test split
- Model that Is trained with balanced dataset and with 0.8 test split
- Model that Is trained with balanced dataset and with 0.9 test split

1. I got these graphs for the loss and the accuracy from the training history on the first model:

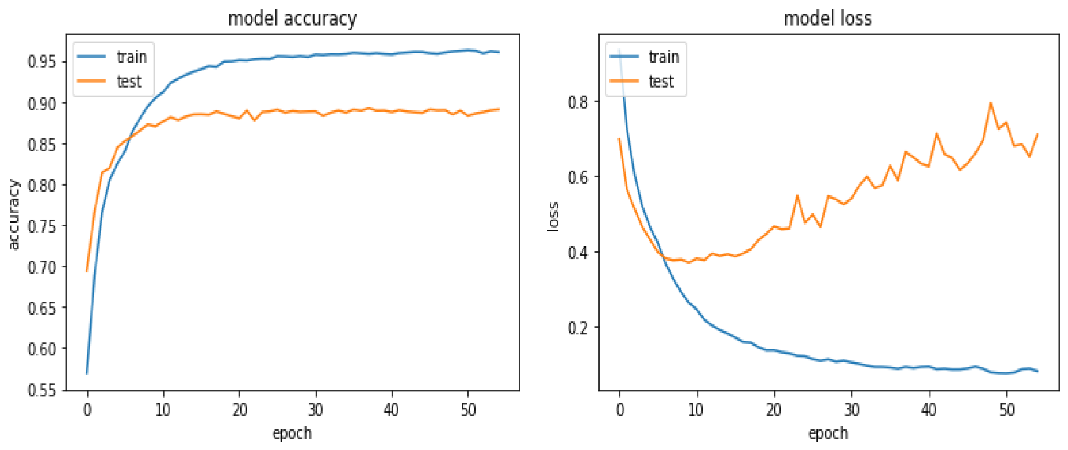


Figure 3.19: History of training the first model with Unbalanced data and 0.8 validation split

We can clearly see that our model is over-fitted to the data due to the divergence in the loss between the validation and the training data.
At this point I noticed that there is no point in training this model more than fifty epochs as the accuracy stops improving and our model starts over-fitting on the training data and it won't become able to generalize on new data.

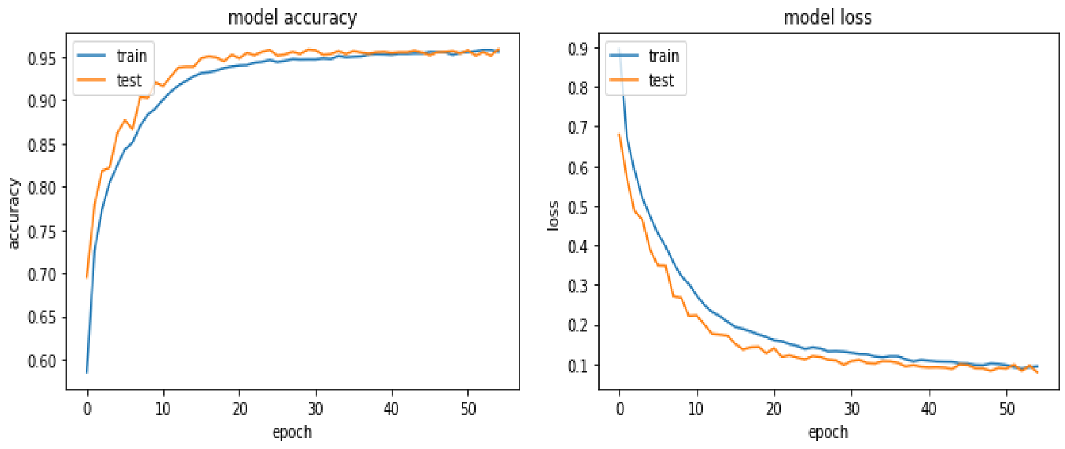


Figure 3.20: History of training the first model with Unbalanced data and 0.9 validation split

Here we notice that as soon as we changed the validation split and got more data to the model, the over-fitting problem got solved and the accuracy got better.

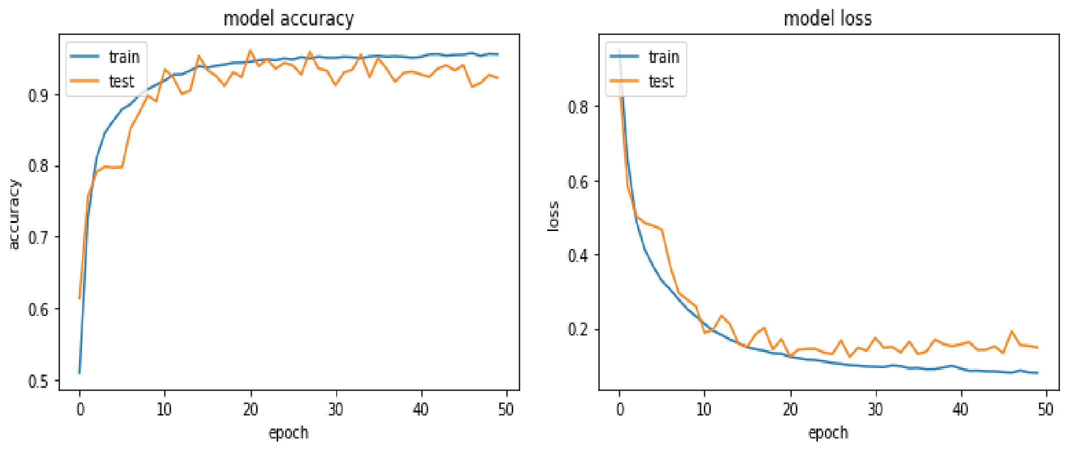


Figure 3.21: History of training the first model with Balanced data and 0.8 validation split

Here we notice that the validation is somehow unstable and not so smooth, but overall the results seems to reach a good accuracy that is around 90 percent for the validation data.

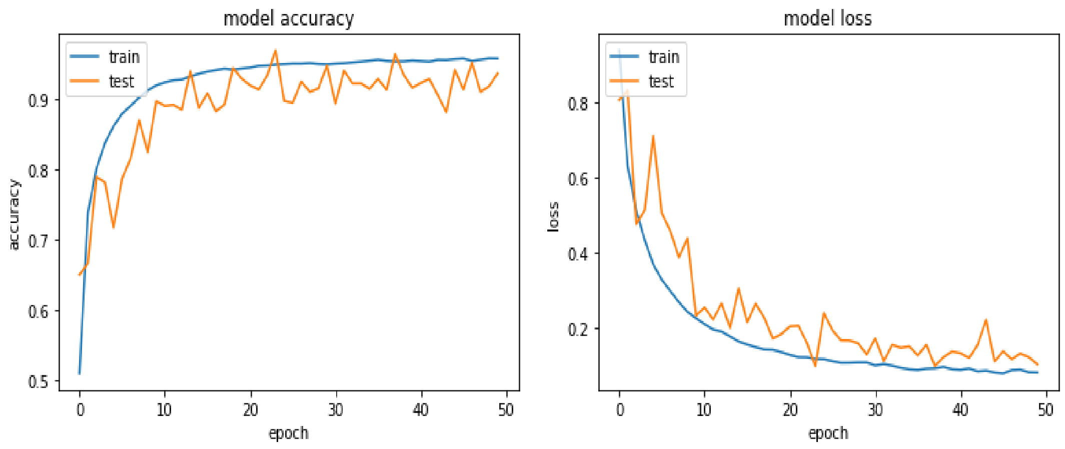


Figure 3.22: History of training the first model with Balanced data and 0.9 validation split

Here we noticed that the fluctuation got sharper as we increased the training data. The model isn't so stable and the accuracy is still around 90 percent.

2. I got these graphs for the loss and the accuracy from the training history on the second model:

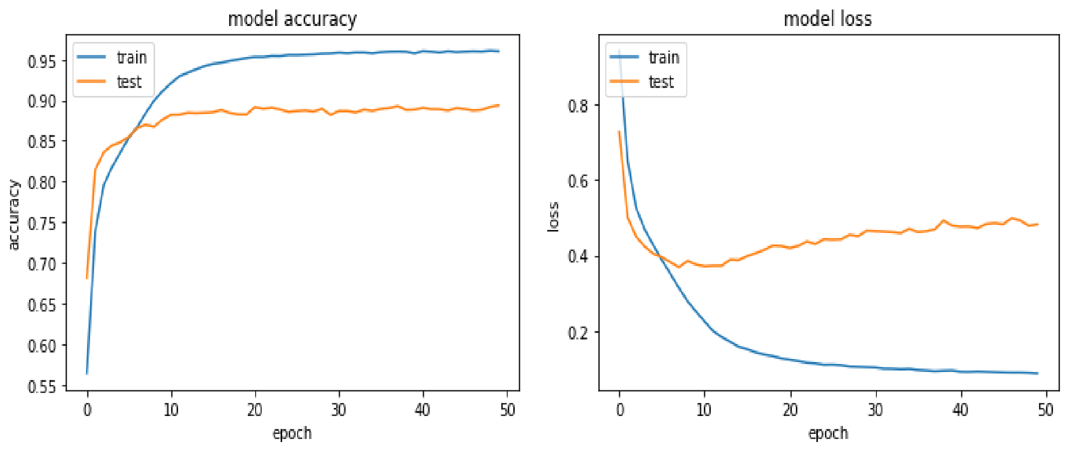


Figure 3.23: History of training the second model with Unbalanced data and 0.8 validation split

We can clearly see that our model is over-fitted to the data due to the divergence in the loss between the validation and the training data. And the accuracy on the validation is far form the training's accuracy, and the difference between the two of them is around 5 percent.

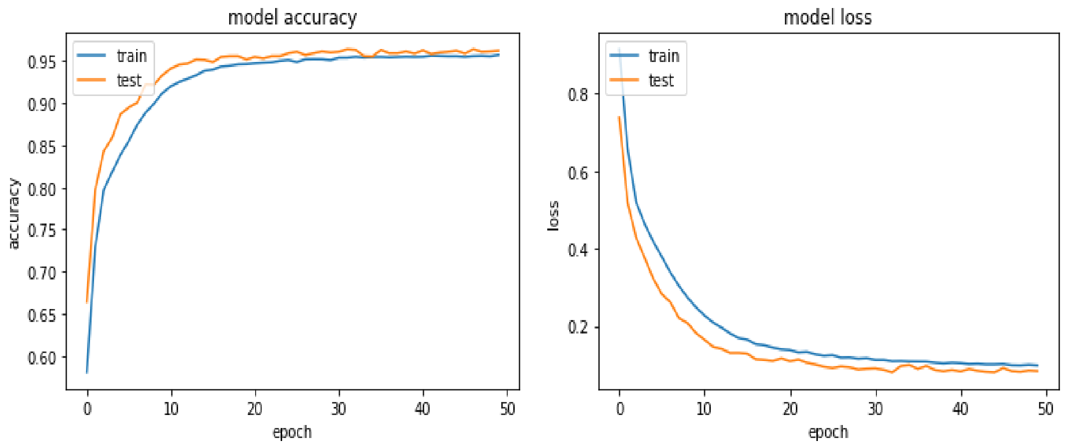


Figure 3.24: History of training the second model with Unbalanced data and 0.9 validation split

Here we notice that as soon as we changed the validation split and got more data to the model, the over-fitting problem got solved and the validation accuracy got closer the training accuracy and we could improve our model overall.

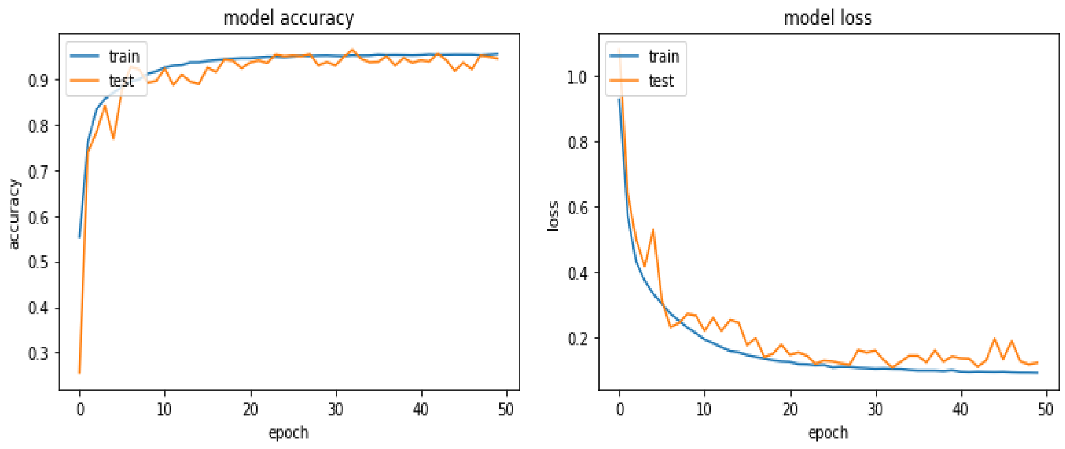


Figure 3.25: History of training the second model with Balanced data and 0.8 validation split

Here we notice that the validation is somehow unstable and not so smooth, but overall the results seems to reach a good accuracy that is around 90 percent for the validation data and we couldn't detect any over-fitting for the model.

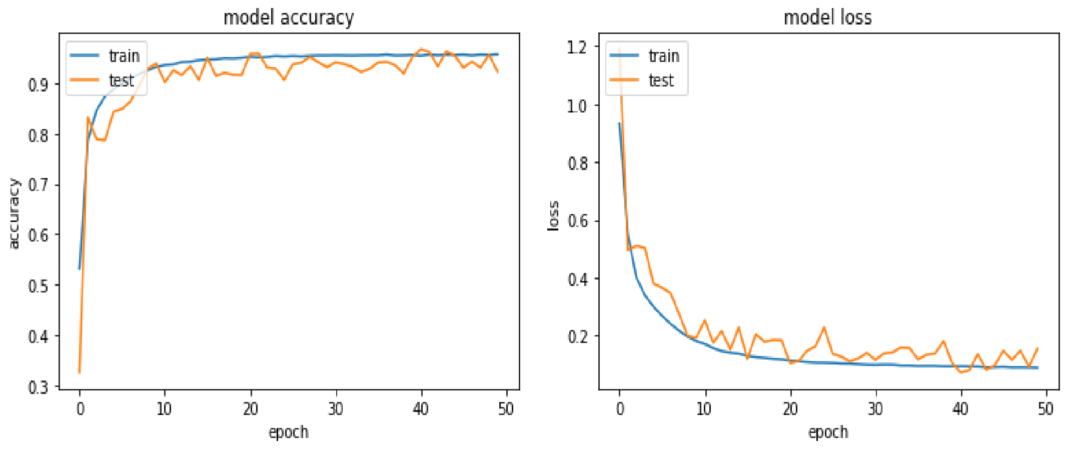


Figure 3.26: History of training the second model with Balanced data and 0.9 validation split

As we increased the training data, The model is still not so stable and the accuracy is still around 90 percent.

3. I got these graphs for the loss and the accuracy from the training history on the third model:

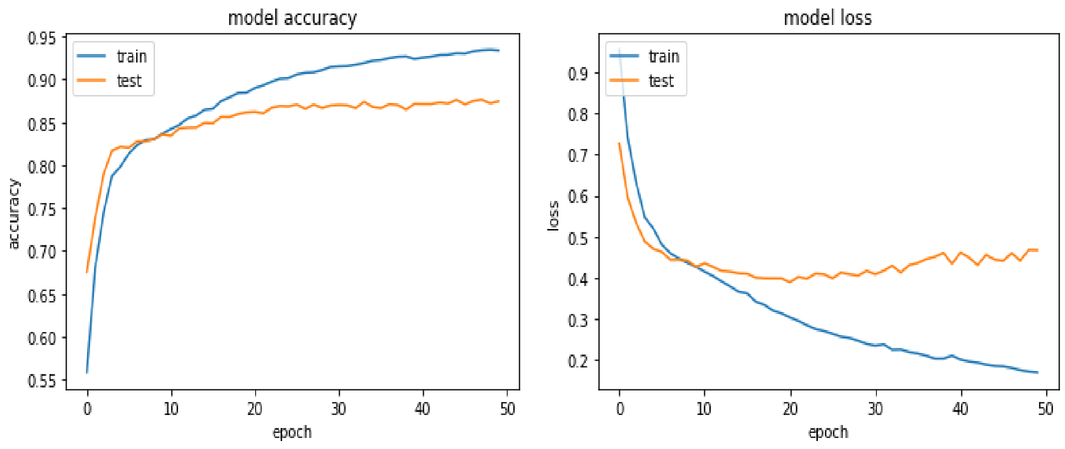


Figure 3.27: History of training the third model with Unbalanced data and 0.8 validation split

We can clearly see that our third model is over-fitted to the data due to the divergence in the loss between the validation and the training data.

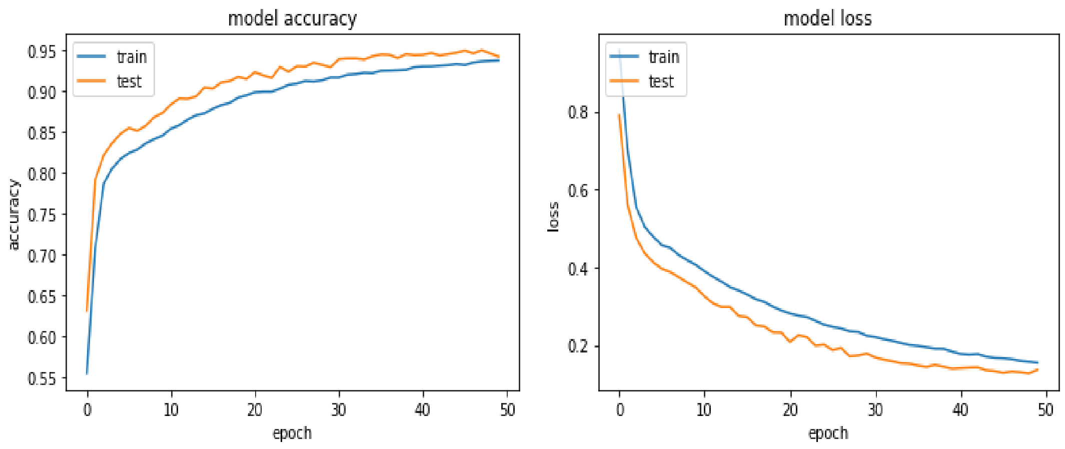


Figure 3.28: History of training the third model with Unbalanced data and 0.9 validation split

Here, just like in the other two models, we notice that as soon as we changed the validation split and got more data to the model, the over-fitting problem got solved and the accuracy got much better almost reaching 95 percent.

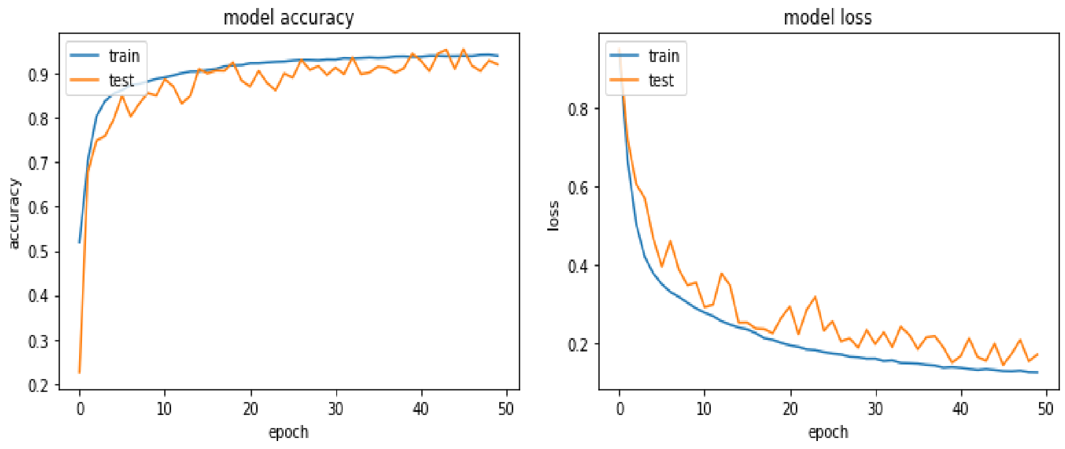


Figure 3.29: History of training the third model with Balanced data and 0.8 validation split

Here we notice that the validation is somehow unstable and not so smooth, but overall the results seems to reach a good accuracy that is also around 90 percent for the validation and training data.

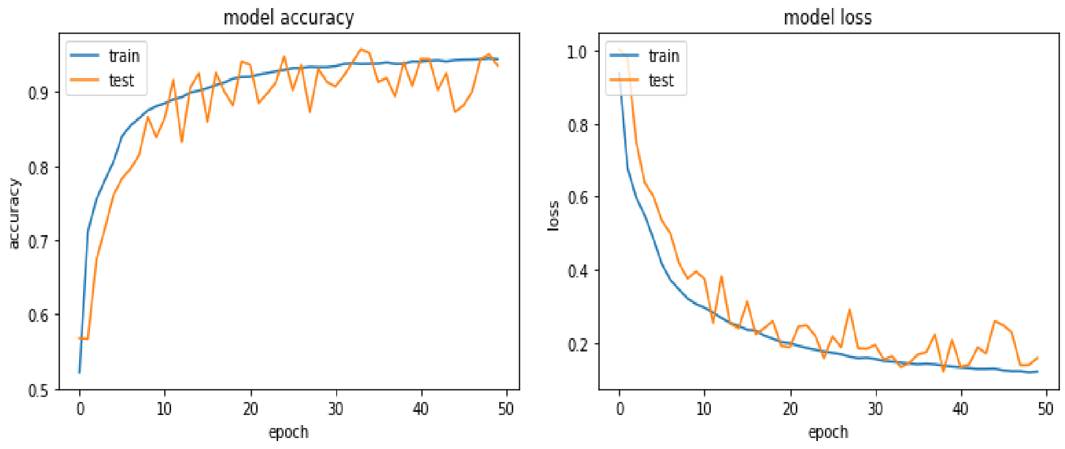
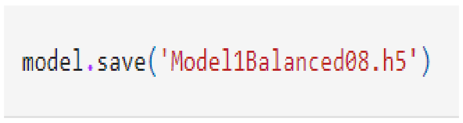


Figure 3.30: History of training the third model with Balanced data and 0.9 validation split

Here we noticed that the fluctuation got sharper for the accuracy as we increased the training data. The model isn't so stable and the accuracy is still unstable around 90 percent.

After preparing these models, I did save them in the Drive with the following command so that I could preserve the trained models and their weights.



```
model.save('Model1Balanced08.h5')
```

Figure 3.31: model.save command

3.4 Convolutional autoencoder classifier

In this approach, I followed along with this article [2]
This approach aims to create a classifier with the help of an CNN based encoder
In the pipeline, there were three big steps that we will explore in this part:

3.4.1 The first part

In the first part of this work, I built an autoencoder that is composed of an encoder, and then a decoder. The output of that model is the same as its input. The architecture of the autoencoder is composed of CNN Layers, BatchNormalization Layers and MaxPoolin1D Layers for the encoder part: where the dimension

decreases. And then It is composed of CNN Layers, BatchNormalization Layers and UpSampling1D Layers to get back to the original dimension.
I the figure below we can see how the dimension decreases in the encoder part:

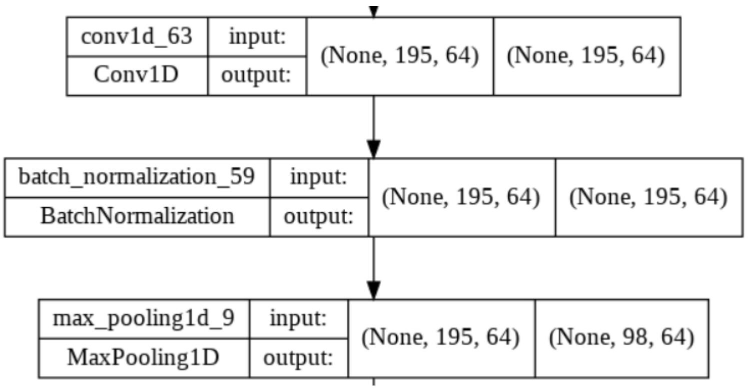


Figure 3.32: Dimensional reduction in the encoder

As for the decoder part, we can see in the figure, down below, how we got back the original dimension through using the UpSampling1D Layers:

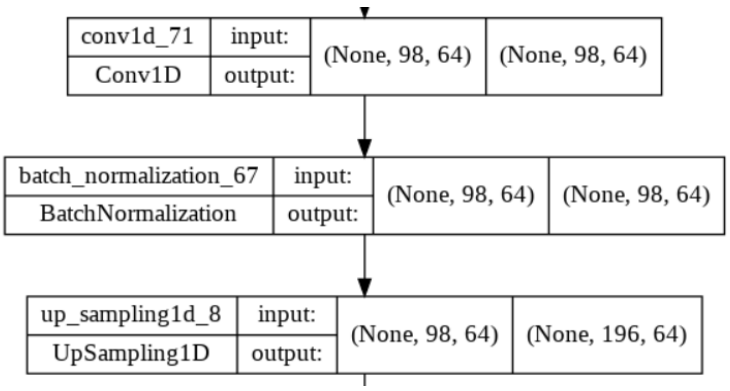


Figure 3.33: Dimensional augmentation in the decoder

You can see the whole code and architecture through this link:
https://bit.ly/ConvolutionalAutoencodeClassifierImenMASMOUDI
Overall the autoencoder architecture looks like this:

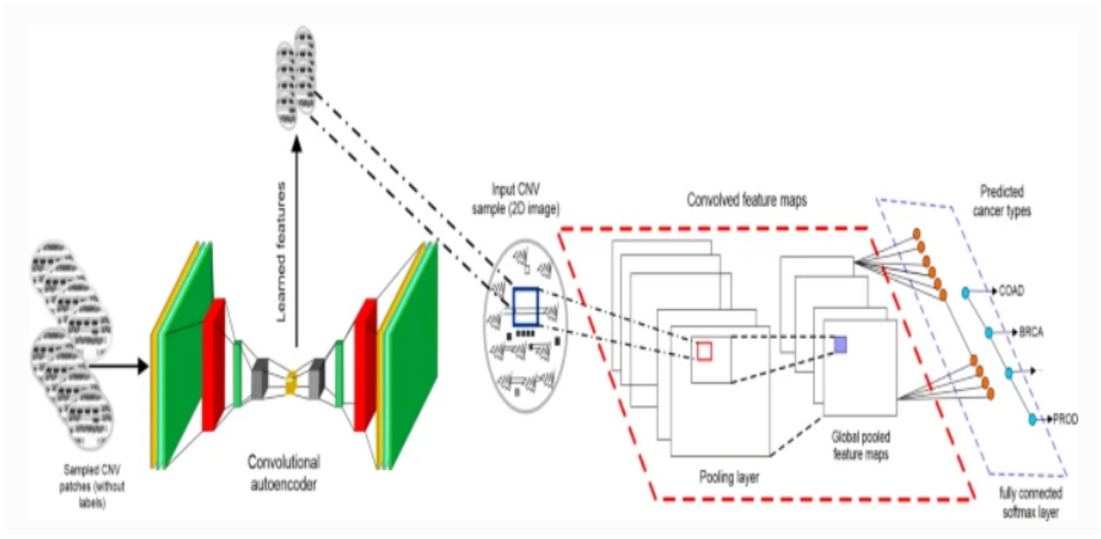
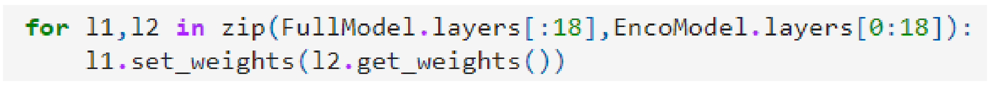

Learned features
Sampled CNV patches (without labels)
Convolutional autoencoder
Input CNV sample (2D image)
Convolved feature maps
Predicted cancer types
COAD
BRCA
PROD
Global pooled feature maps
Pooling layer
fully connected softmax layer

Figure 3.34: Autoencoder Architecture
[2]

The input gets into a tunnel like architecture that reduces the dimension as it evolves in the model, and then the dimension gets back to the same as the inputs.

3.4.2 The second part

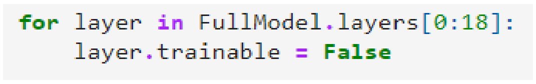
In the second part on this method, I prepared our classifier model's architecture, and it was built based on our autoencoder model. The beginning of the model was the same as the encoder, and then I followed up with some dense layers to have a model that predicts a class out of three.
I then loaded the weights of the encoder part of the model into our classifier and froze them so that they do not change as we train.
Here is how I loaded the weights from the autoencoder model into our classifier model:



```
for l1,l2 in zip(FullModel.layers[:18],EncoModel.layers[0:18]):
    l1.set_weights(l2.get_weights())
```

Figure 3.35: Freezing the first layers for the classifier

And here is the code I used to freeze the first layers:



```
for layer in FullModel.layers[0:18]:
    layer.trainable = False
```

Figure 3.36: Freezing the first layers for the classifier

Here, only the last part of the classifier model will change its weights, which is the part that we added that contains the dense layers.
Here is the final part of our classifier model where I added the dense layers:

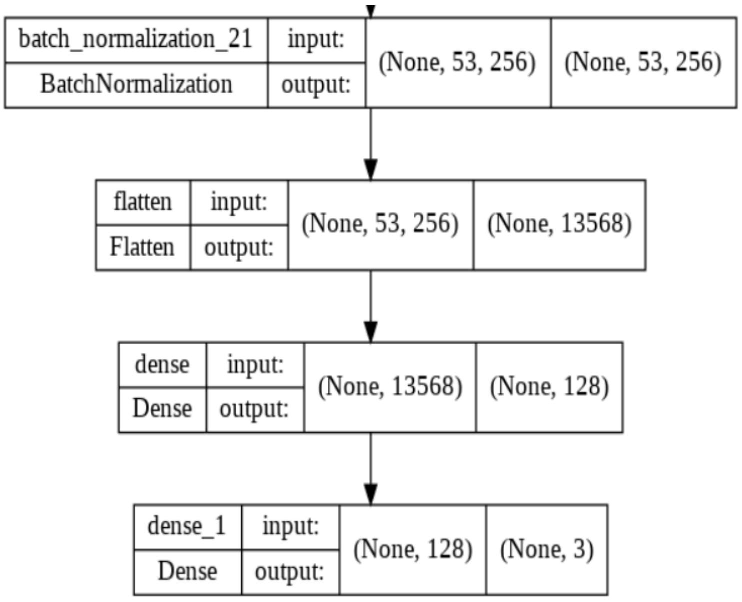
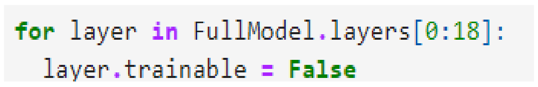


Figure 3.37: Closing architecture for the classifier

3.4.3 the third part

In the third part, we basically took our same classifier that had frozen first layers that were from the encoder and we unfroze them, and retrained our model so that it is now able to update all of its weights.
here is the code I used to unfreeze the first layers:



```
for layer in FullModel.layers[0:18]:
  layer.trainable = False
```

Figure 3.38: Unfreezing the first layers for the classifier

3.4.4 The training process

In the training process, I started with preparing the auto encoder. This model took forever to train, more than three hours! It took the longest training time in this project as it is the one with the biggest parameters' number.
Here are the plots I got after the training with an unbalanced data with 0.8 test split:

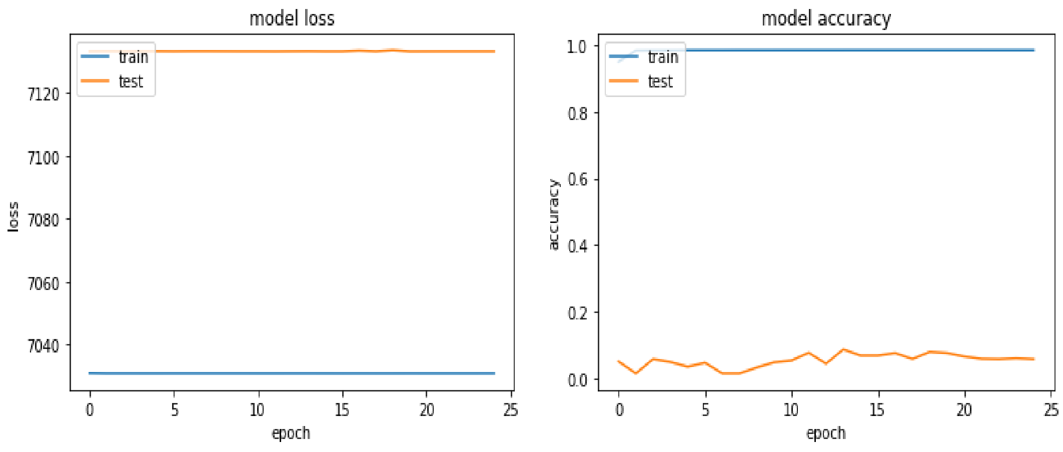


Figure 3.39: History of training the Autoencoder with Unbalanced data and 0.8 validation split

Though after training it, it seems that our model has a very low accuracy Here are the plots for the Auto Encoder classifier's Loss and Accuracy:

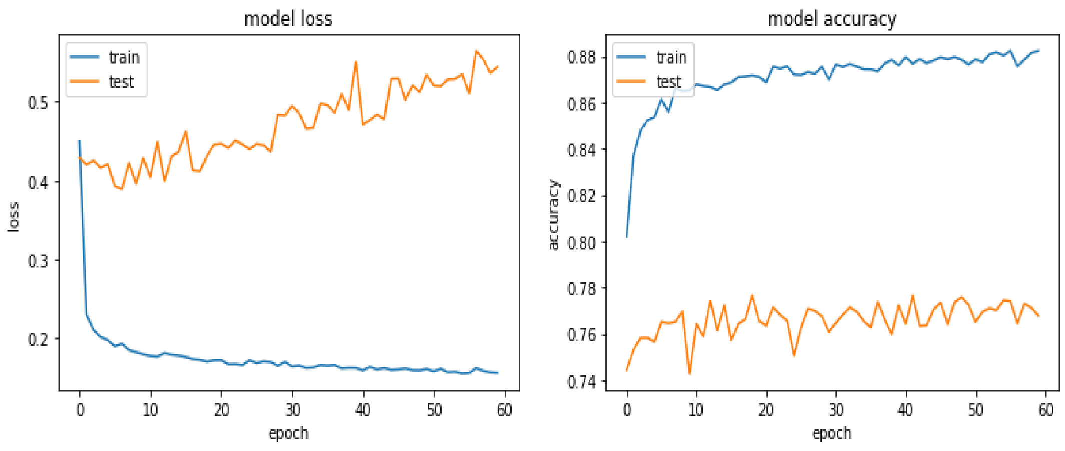


Figure 3.40: History of training the Classifier Model with First Layers Frozen with Unbalanced data and 0.8 validation split

After seeing the plots, it appears that our model is somehow unstable but it is giving results that have an accuracy around seventy six percent, which is not so bad!
Here are the plots I got for the Loss and Accuracy of the Same Classifier model with unfrozen first Layers:

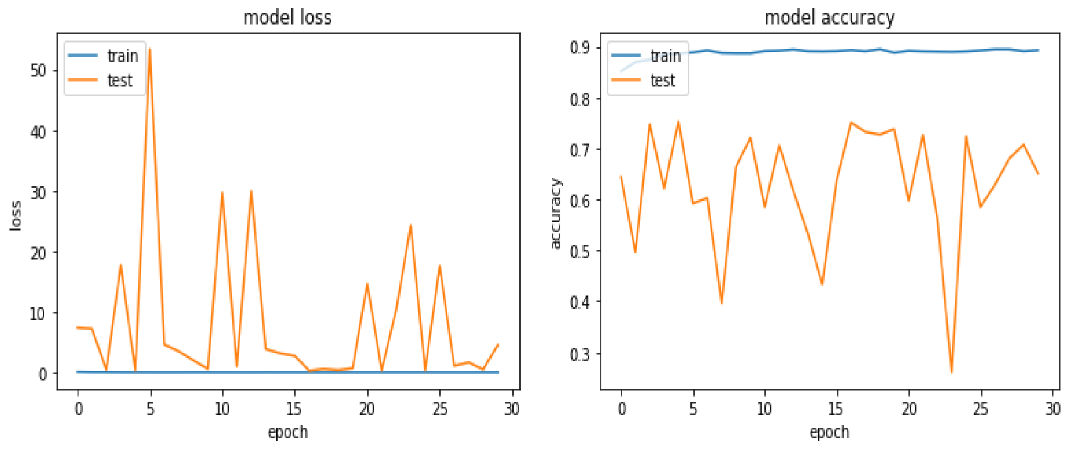


Figure 3.41: History of training the Classifier Model with First Layers Unfrozen with Unbalanced data and 0.8 validation split

The model doesn't seem to get better and got a lot more unstable.

3.5 Conclusion

After working on this approach, I realized that this is more oriented to deep learning, because for the autoencoder to give good results, as it has a very big architecture, we need big data, so much more data to get better results.

3.6 The Ensembling Approach

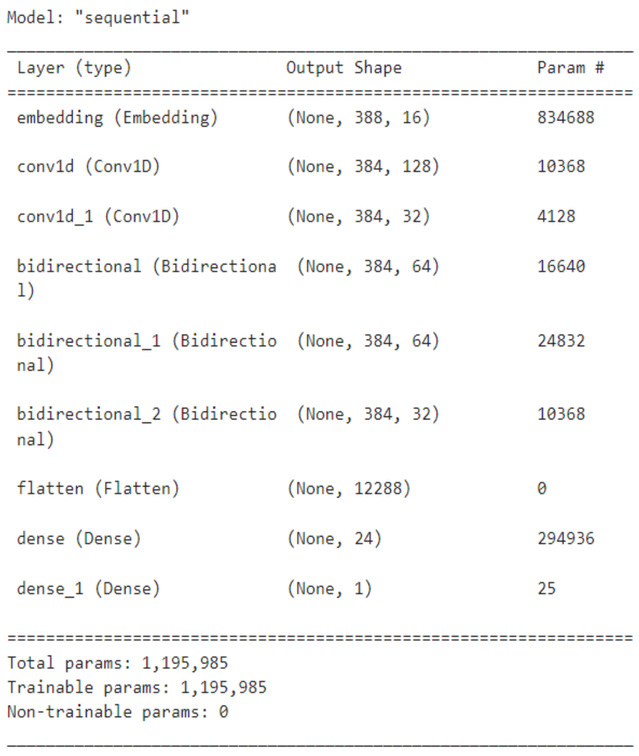
The ensembling method is an approach used in Machine Learning that is based on using a variety of models to get the final predictions.
It is not guaranteed to get better accuracy when using the ensembling methods, yet it can give great results in some cases.
Using the ensembling methods is the equivalent of getting more opinions from different people with different backgrounds, it doesn't guarantee that you'll get the best output, but at times it can!
There are so many ways we can include this approach through different methods like Voting and Averaging Based Ensemble Methods, Stacking Multiple Machine Learning Models, Bootstrap Aggregating, Boosting: Converting Weak Models to Strong Ones and using the ECOC Algorithm.
In this work we implemented some of these Ensembling methods which are the ECOC Algorithm and the voting method. We will start with the ECOC Algorithm.

3.6.1 The ECOC Algorithm

In this approach we produced three models because we have three classes and we relabeled our dataset to reproduce three datasets.
We choose this number because it is the same as the number of the classes that we originally have which is three.
Every model focuses on one class, so it outputs a binary number that indicates if it detects the specialized class or not.

3.6.1.1 The ECOC Algorithm approach

I made the three models' architectures the same, the input is the padded comment, and the output is binary, so either zero or one.

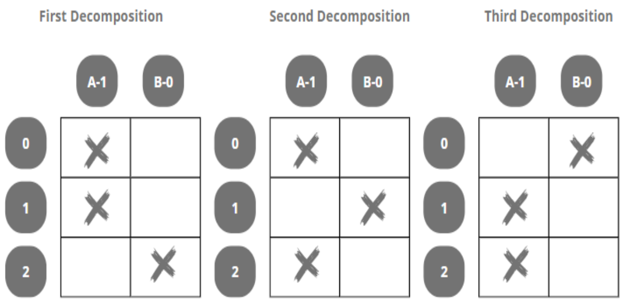


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 388, 16)           834688

 conv1d (Conv1D)             (None, 384, 128)          10368

 conv1d_1 (Conv1D)           (None, 384, 32)           4128

 bidirectional (Bidirectiona  (None, 384, 64)          16640
 l)

 bidirectional_1 (Bidirectio  (None, 384, 64)          24832
 nal)

 bidirectional_2 (Bidirectio  (None, 384, 32)          10368
 nal)

 flatten (Flatten)           (None, 12288)             0

 dense (Dense)               (None, 24)                294936

 dense_1 (Dense)             (None, 1)                 25

=================================================================
Total params: 1,195,985
Trainable params: 1,195,985
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.42: Models' Architecture in the ECOC Algorithm

For the first model, I relabeled the data to be either zero or one. The zero in the new labels indicates that the original class is zero or one and the one in the new labels indicates that the original class is two

In these tables I noted the relation between the new labels and the old labels for the three datasets:



First Decomposition

	A-1	B-0
0	X	
1	X	
2		X

Second Decomposition

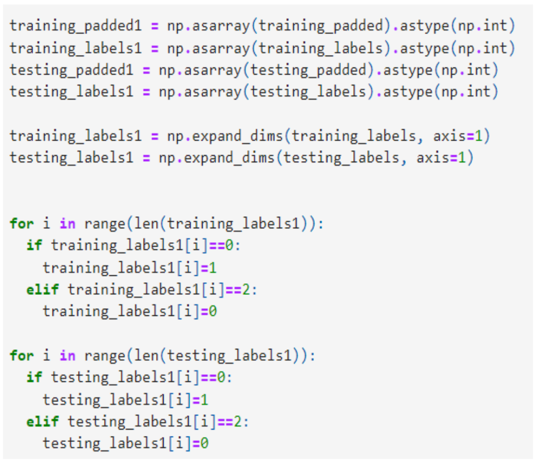
	A-1	B-0
0	X	
1		X
2	X	

Third Decomposition

	A-1	B-0
0		X
1	X	
2	X	

Figure 3.43: The relation between the Datasets

And then we trained the first model with the new data so that it would give zero if the original class is two and one otherwise.
For the second model, I relabeled the data to be either zero or one. The zero in the new labels indicates that the original class is zero or two and the one in the new labels indicates that the original class is one
For the Third model, I relabeled the data and trained it so that it would give zero if the original class is zero and one otherwise.
The relabeling part took some good hard coding skills, I was lucky that I pulled it off! I am proud of that part of my code!
This is a part of the code, you can see the whole code in this notebook:
https://bit.ly/ECOCAlgorithmByImenMASMOUDI



```
training_padded1 = np.asarray(training_padded).astype(np.int)
training_labels1 = np.asarray(training_labels).astype(np.int)
testing_padded1 = np.asarray(testing_padded).astype(np.int)
testing_labels1 = np.asarray(testing_labels).astype(np.int)

training_labels1 = np.expand_dims(training_labels, axis=1)
testing_labels1 = np.expand_dims(testing_labels, axis=1)


for i in range(len(training_labels1)):
  if training_labels1[i]==0:
    training_labels1[i]=1
  elif training_labels1[i]==2:
    training_labels1[i]=0

for i in range(len(testing_labels1)):
  if testing_labels1[i]==0:
    testing_labels1[i]=1
  elif testing_labels1[i]==2:
    testing_labels1[i]=0
```

Figure 3.44: Relabeling for the ECOC Algorithm

3.6.1.2 The training process

Just like I did in the primary approach, I trained the models under four conditions having Balanced and Unbalanced datasets and a validation split 0.8 and 0.9

1. Having unbalanced datasets and a validation split that's equal to 0.8, I got these graphs for the loss and the accuracy from the training history on the first model:

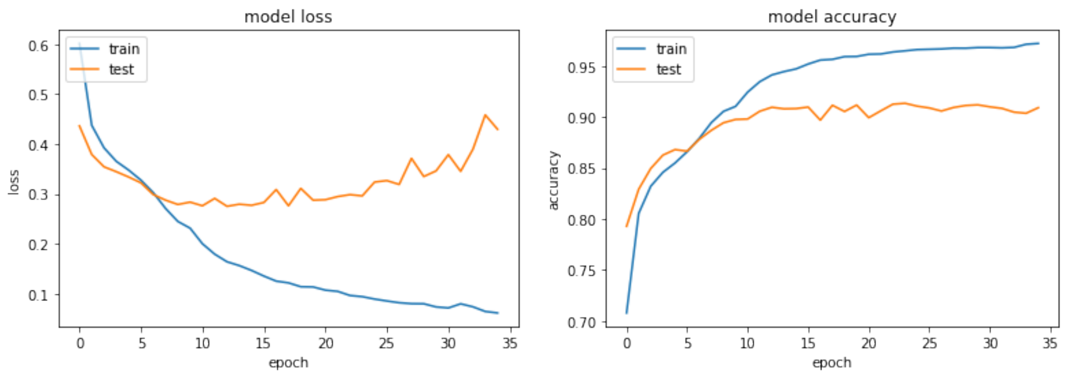


Figure 3.45: The Training Loss and Accuracy of the first model

We can clearly see that the model is starting to over-fit on the training data as there is divergence between the training loss and validation loss.
Here are the Training Loss and Accuracy of the first model:

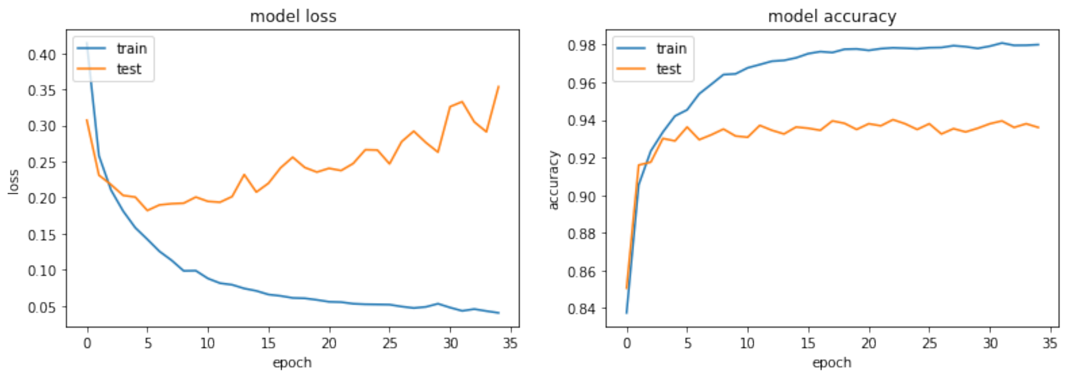


Figure 3.46: The Training Loss and Accuracy of the second model

We can also notice that the model here is starting to over-fit on the training data.
Here are the Training Loss and Accuracy of the first model:

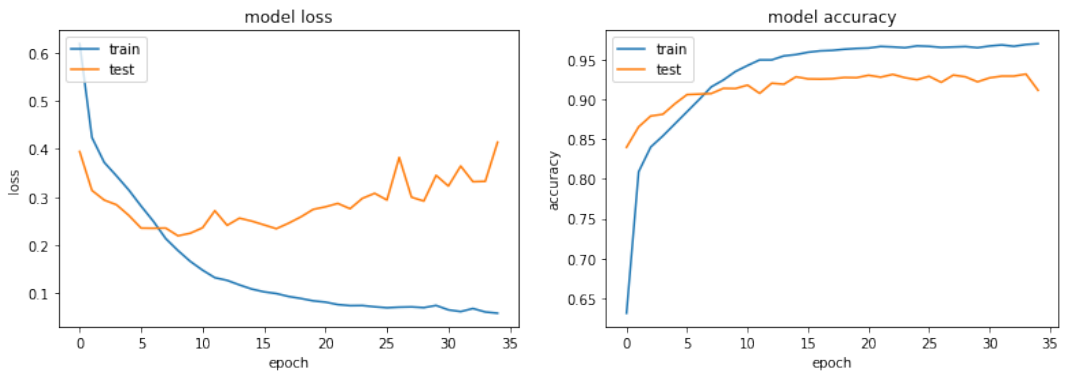


Figure 3.47: The Training Loss and Accuracy of the third model

We can also notice that the model here is starting to over-fit on the training data.

2. Having unbalanced datasets and a validation split that's equal to 0.9, I got these graphs for the loss and the accuracy from the training history on the first model:

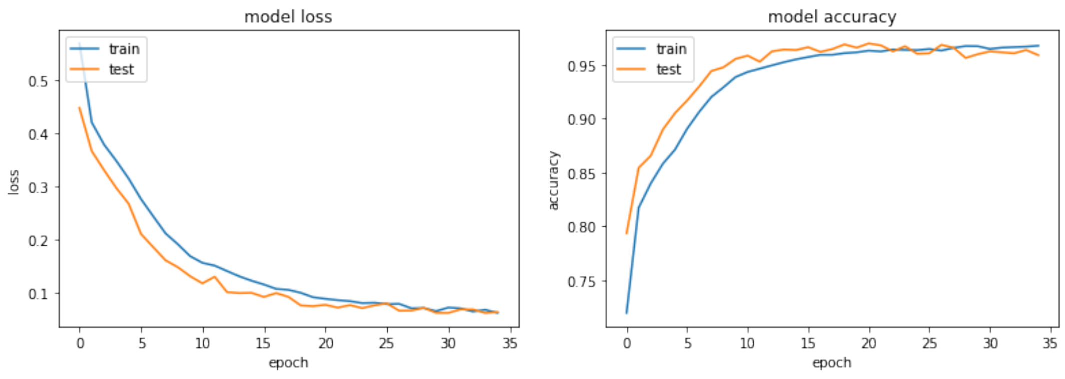


Figure 3.48: The Training Loss and Accuracy of the first model

We can clearly see that the over-fitting problem is resolved and that the accuracy is high, reaching around 95 percent.
Here are the Training Loss and Accuracy of the first model:

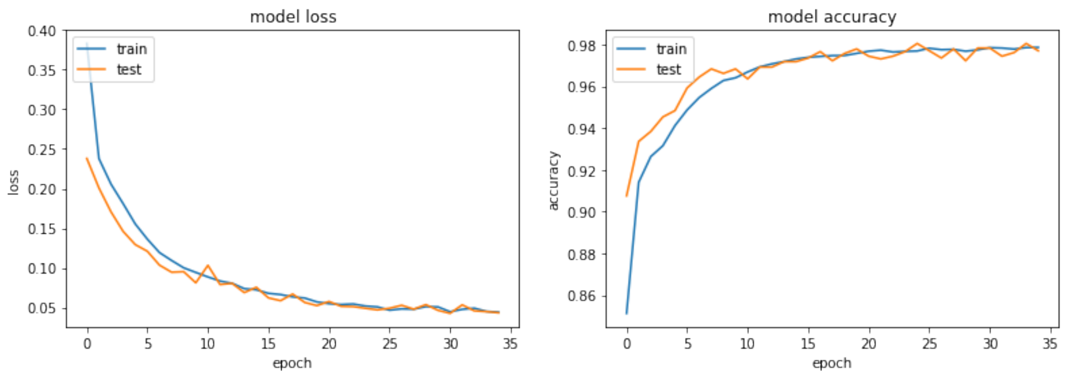


Figure 3.49: The Training Loss and Accuracy of the second model

We can also say the same thing about the second model.
Here are the Training Loss and Accuracy of the first model:

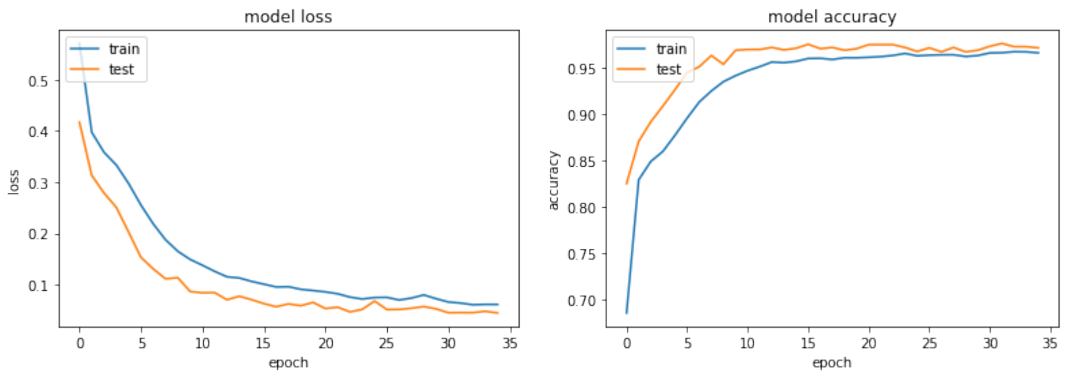


Figure 3.50: The Training Loss and Accuracy of the third model

We can also say the same thing about the third model.

3. Having balanced datasets and a validation split that's equal to 0.8, I got these graphs for the loss and the accuracy from the training history on the first model:

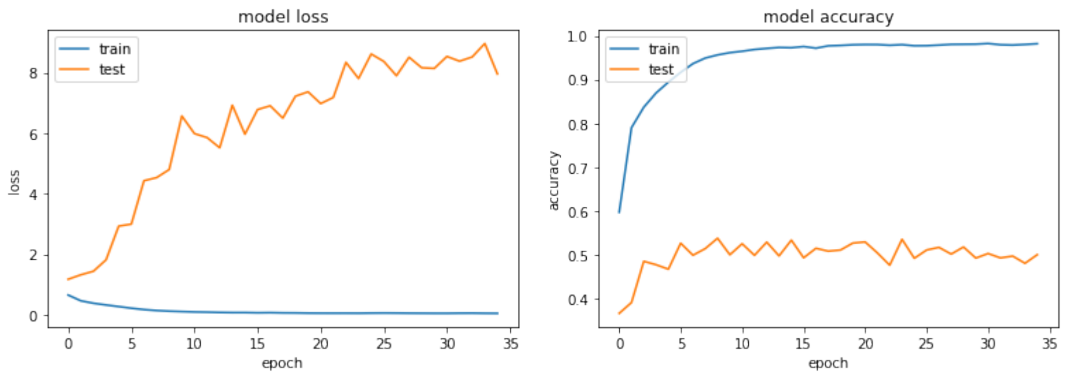


Figure 3.51: The Training Loss and Accuracy of the first model

We can clearly see that the model is starting to over-fit on the training data as there is divergence between the training loss and validation loss.
Here are the Training Loss and Accuracy of the first model:

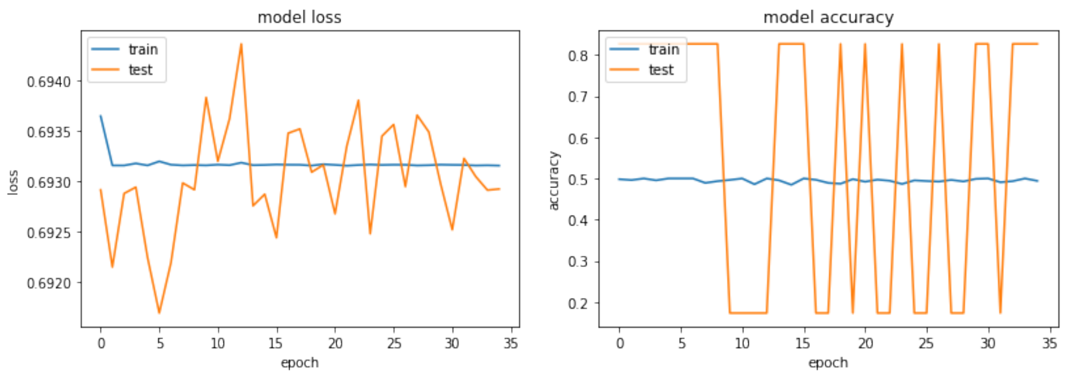


Figure 3.52: The Training Loss and Accuracy of the second model

We can see here that the model is not improving and that it is giving very bad results and that it is highly unstable.
Here are the Training Loss and Accuracy of the first model:

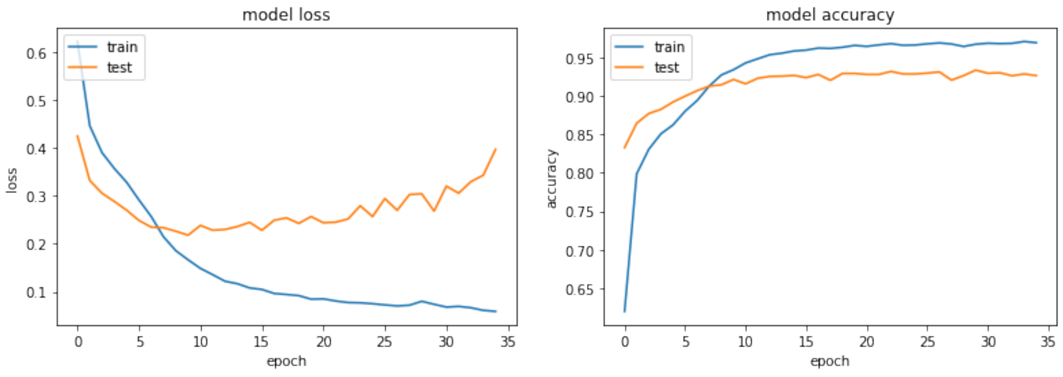


Figure 3.53: The Training Loss and Accuracy of the third model

We can also notice that the model here is starting to over-fit on the training data.

4. Having balanced datasets and a validation split that's equal to 0.9, I got these graphs for the loss and the accuracy from the training history on the first model:

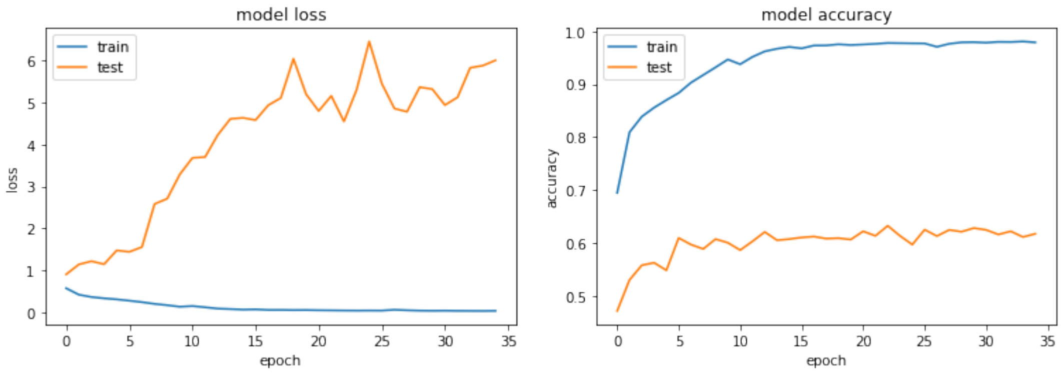


Figure 3.54: The Training Loss and Accuracy of the first model

We can clearly see that the over-fitting problem still there and that the validation accuracy is very far from the training accuracy.
Here are the Training Loss and Accuracy of the first model:

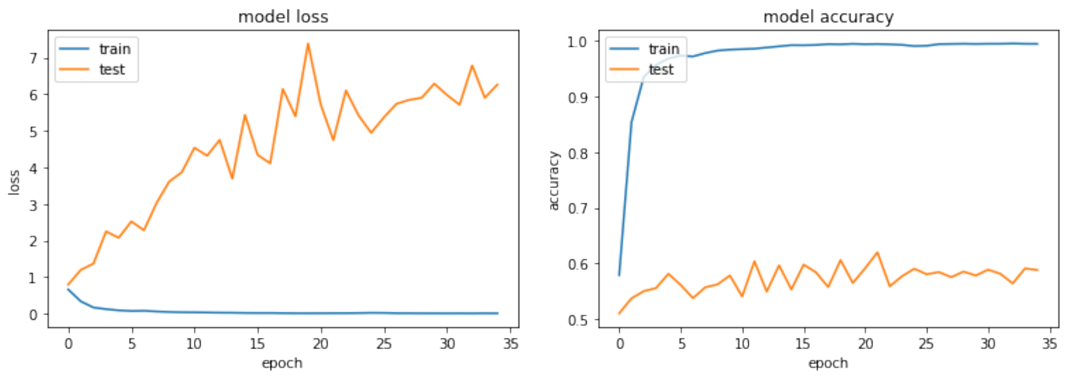


Figure 3.55: The Training Loss and Accuracy of the second model

We can also say the same thing about the second model, it is much more stable now but it seems like the model is starting to over-fit on the training data and that it can not generalize.
Here are the Training Loss and Accuracy of the first model:

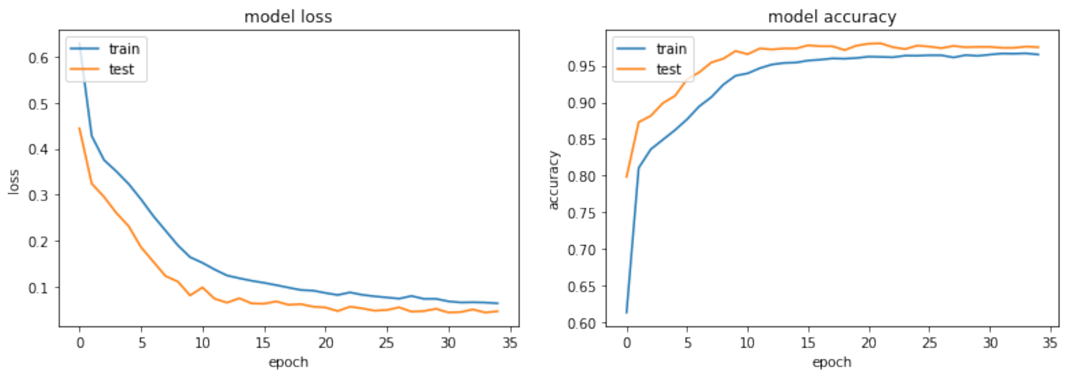
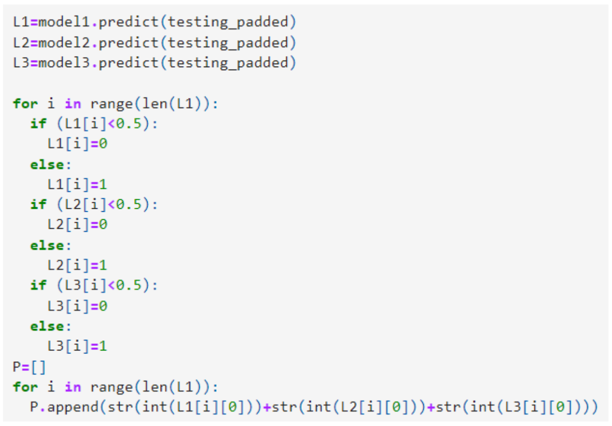


Figure 3.56: The Training Loss and Accuracy of the third model

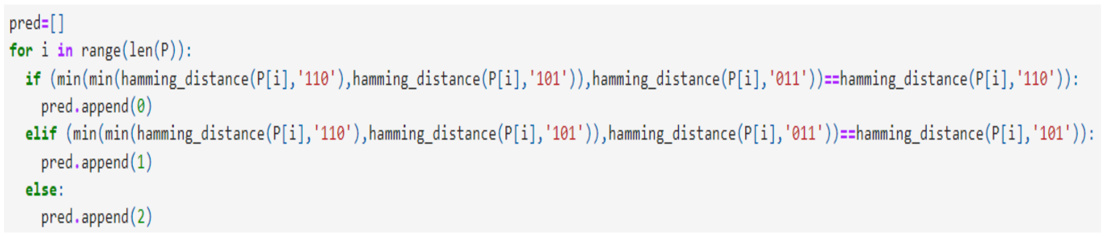
We can say that for the third model we can see improvement as we changed the validation split, the accuracy is high reaching 95 percent and the model is training in a stable way as it evolved in the epochs.

3.6.1.3 getting predictions

After creating the three specialized models, preparing their training and validation datasets and fitting them to it, I linked them through a function that I have defined that takes the predictions from the three models and outputs the final predictions that indicates what class out of the three ones our class is.
And here is the code of that function:



```
L1=model1.predict(testing_padded)
L2=model2.predict(testing_padded)
L3=model3.predict(testing_padded)

for i in range(len(L1)):
  if (L1[i]<0.5):
    L1[i]=0
  else:
    L1[i]=1
  if (L2[i]<0.5):
    L2[i]=0
  else:
    L2[i]=1
  if (L3[i]<0.5):
    L3[i]=0
  else:
    L3[i]=1
P=[]
for i in range(len(L1)):
  P.append(str(int(L1[i][0]))+str(int(L2[i][0]))+str(int(L3[i][0])))
```



```
pred=[]
for i in range(len(P)):
  if (min(min(hamming_distance(P[i],'110'),hamming_distance(P[i],'101')),hamming_distance(P[i],'011'))==hamming_distance(P[i],'110')):
    pred.append(0)
  elif (min(min(hamming_distance(P[i],'110'),hamming_distance(P[i],'101')),hamming_distance(P[i],'011'))==hamming_distance(P[i],'101')):
    pred.append(1)
  else:
    pred.append(2)
```

Figure 3.57: Getting Predictions From the ECOC Algorithm

For the Normal Class, which is labelled as zero in the original dataset, we should have the outputs of the three models combined be 110, this indicates that the third model sensed the presence of the label zero because it is the one that focuses on the model, and the other models didn't sense the presence of the classes one and two.

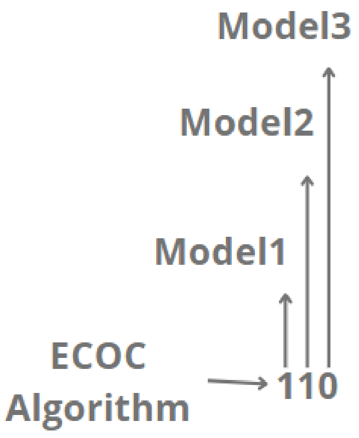
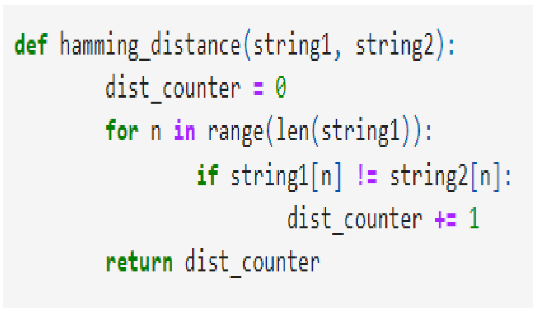


Figure 3.58: 110 Output Example for the ECOC Algorithm

For the abusive Class, which is labelled as one in the original dataset, we should have the outputs of the three models combined to be 101, this indicates that the second model sensed the presence of our label our second model is specialized in detecting this class, and the other models didn't sense the presence of the class zero or two.
For the third Class, which is labelled as two in the original dataset, we should have the outputs of the three models combined to be 011, this indicates that the first model sensed the presence of our label because it is the one that focuses on, and the other models didn't sense the presence of the other two classes.
When we explore the possibilities and the possible outputs that we can get when combining three binary numbers, we know that there are more than three possible outputs, we can find eight different combinations: 000, 001, 010, 011, 100, 101, 110, 111.
We can find three amongst them that are precise predictions for us which are:

- 110 that indicates the class number zero
- 101 that indicates the class number one
- 011 that indicates the class number two

As for the rest five of them, they give vague and unsure predictions. For that we used the Hamming distance to figure out the closest class to it, and we assigned it. The Hamming distance is a metric that measures the number of changes that we need to make to a string to change it to another string. For example, we need one change to go from 001 to 011 and two changes to go from 001 to 010.
Here is the definition for this function:



```
def hamming_distance(string1, string2):
        dist_counter = 0
        for n in range(len(string1)):
                if string1[n] != string2[n]:
                        dist_counter += 1
        return dist_counter
```

Figure 3.59: Relabeling for the ECOC Algorithm

After doing those steps, we got ourselves predictions that were 89 percent accurate from the first try!
For the training part, I used a sparse cross entropy loss function and the Adam optimizer, Then I trained each model for 50 epochs.

3.6.2 The voting Method

Still in the Ensembling, I applied the voting method with three variations, but before talking about the different variations and what makes them unalike, let's explore the size of the models' outputs and the chosen models, because that's what we will be using for the next part.

3.6.2.1 preliminary explorations

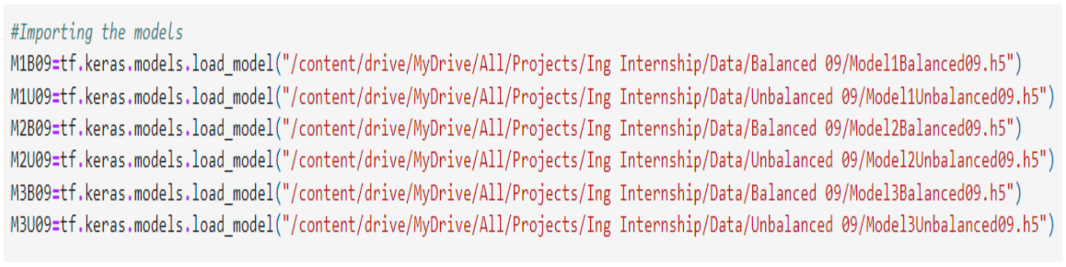
After finishing the Primary approach and getting four versions of the three basic models, I explored the strengths and weaknesses of the 12 models and combined six models in this part
The outputs of each of the models are matrix having the number of lines equal to the test cases and the number of the columns equal to three. Since we are in a classification problem, the final outputs are the probabilities of belonging to the respective class. This is assured by the use of the SoftMax activation function at the end that gives us the probabilities using the following equation:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Figure 3.60: SoftMax Activation Function

This opened the window to possibilities for us to explore, as we can get the predictions as probabilities and add them together and then applying the ArgMax function, or we can apply ArgMax to decide on the classes that each model produced and then vote between them.
The ArgMax function is a built-in Numpy function that is used to get the indices of the maximum element from an array.
I will present the difference between the two approaches with the following example:
Let's say I have three models that will predict two classes:
Let us note P1 the Prediction of the first model that is an array that has two values that contain the probabilities of belonging to the first and the second classes
P2 is the prediction of the second model and P3 is the prediction of the third model.
P1=[0.8,0.2]
P2=[0.4,0.6]
P3=[0.4,0.6]

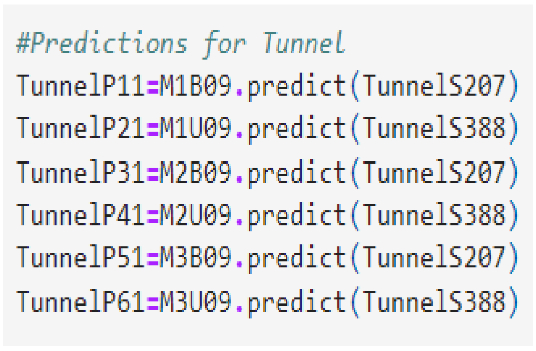
If we apply ArgMax function to these arrays we will get respectively:
L1=0
L2=1
L3=1
Please note that the true label is zero, and in this case the first model is the best at predicting this class
here is we used the first approach we will apply ArgMax and then vote among the results that we have got which means that we will vote between L1, L2 and L3 and the result would be The class one. but if we use the second approach we will add L1 to L2 and L3. and we will get L=[1.6,1.4].
Then when we apply the ArgMax we will get the class zero which is the right class. Usually when the probabilities are close to fifty fifty, this issue raises. And this why sometimes the majority does not have the accurate results but the one who is the most sure among us and with the higher expertise can decide better than all of us! That is why I wanted to apply the two methods to see if they give different results. And I will call them, ArgMax then Vote, or Add then ArgMax.
In the beginning I called six of the best models that were stored in the drive from the primary approach. In the next chapter, I evaluated the accuracies that the models gave us and these six models are the ones that gave the best results at that stage.
I called them through this command:



```
#Importing the models
M1B09=tf.keras.models.load_model("/content/drive/MyDrive/All/Projects/Ing Internship/Data/Balanced 09/Model1Balanced09.h5")
M1U09=tf.keras.models.load_model("/content/drive/MyDrive/All/Projects/Ing Internship/Data/Unbalanced 09/Model1Unbalanced09.h5")
M2B09=tf.keras.models.load_model("/content/drive/MyDrive/All/Projects/Ing Internship/Data/Balanced 09/Model2Balanced09.h5")
M2U09=tf.keras.models.load_model("/content/drive/MyDrive/All/Projects/Ing Internship/Data/Unbalanced 09/Model2Unbalanced09.h5")
M3B09=tf.keras.models.load_model("/content/drive/MyDrive/All/Projects/Ing Internship/Data/Balanced 09/Model3Balanced09.h5")
M3U09=tf.keras.models.load_model("/content/drive/MyDrive/All/Projects/Ing Internship/Data/Unbalanced 09/Model3Unbalanced09.h5")
```

Figure 3.61: Loading the models

And then I got the predictions with the model.predict command:

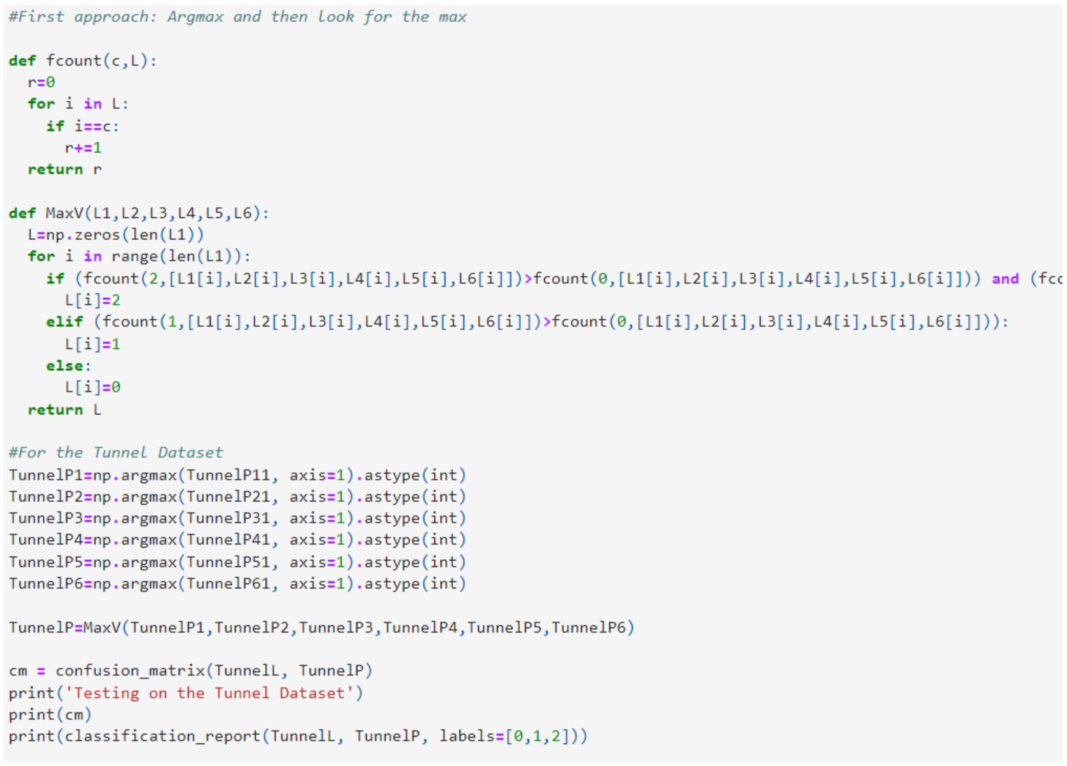


```
#Predictions for Tunnel
TunnelP11=M1B09.predict(TunnelS207)
TunnelP21=M1U09.predict(TunnelS388)
TunnelP31=M2B09.predict(TunnelS207)
TunnelP41=M2U09.predict(TunnelS388)
TunnelP51=M3B09.predict(TunnelS207)
TunnelP61=M3U09.predict(TunnelS388)
```

Figure 3.62: Making Predictions

3.6.2.2 ArgMax then Vote

This is a part of the code where I defined the voting function and used it to get predictions from the six models.
You can explore the whole code through this link:
https://bit.ly/EnsemblingMethodeVotingImenMasmoudi



```
#First approach: Argmax and then look for the max

def fcount(c,L):
  r=0
  for i in L:
    if i==c:
      r+=1
  return r

def MaxV(L1,L2,L3,L4,L5,L6):
  L=np.zeros(len(L1))
  for i in range(len(L1)):
    if (fcount(2,[L1[i],L2[i],L3[i],L4[i],L5[i],L6[i]])>fcount(0,[L1[i],L2[i],L3[i],L4[i],L5[i],L6[i]])) and (fcc
      L[i]=2
    elif (fcount(1,[L1[i],L2[i],L3[i],L4[i],L5[i],L6[i]])>fcount(0,[L1[i],L2[i],L3[i],L4[i],L5[i],L6[i]])):
      L[i]=1
    else:
      L[i]=0
  return L

#For the Tunnel Dataset
TunnelP1=np.argmax(TunnelP11, axis=1).astype(int)
TunnelP2=np.argmax(TunnelP21, axis=1).astype(int)
TunnelP3=np.argmax(TunnelP31, axis=1).astype(int)
TunnelP4=np.argmax(TunnelP41, axis=1).astype(int)
TunnelP5=np.argmax(TunnelP51, axis=1).astype(int)
TunnelP6=np.argmax(TunnelP61, axis=1).astype(int)

TunnelP=MaxV(TunnelP1,TunnelP2,TunnelP3,TunnelP4,TunnelP5,TunnelP6)

cm = confusion_matrix(TunnelL, TunnelP)
print('Testing on the Tunnel Dataset')
print(cm)
print(classification_report(TunnelL, TunnelP, labels=[0,1,2]))
```

Figure 3.63: Ensembling ArgMax and then Vote

In this code, we can see that I have defined two functions:

1. The first function: fcount(c,L)
   This function takes a character or an integer and a list L. It counts how many times c exists in the list L and then outputs that number.

2. The second function: MaxV(L1,L2,L3,L4,L5,L6):
   This function takes as input six Lists and outputs a list.
   the output list contains the results of a vote between the six lists. every list will contain so many variable that can be either zero, one or two that refers to a class.

After this TunnelP contains the predictions that result from the voting.
The concept of the max voting is illustrated in the graph below:

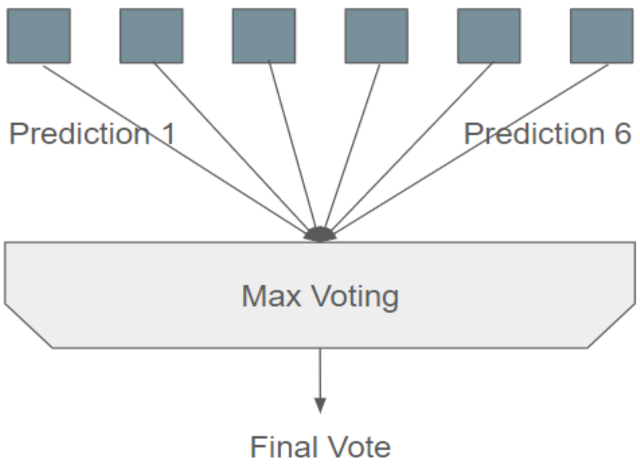
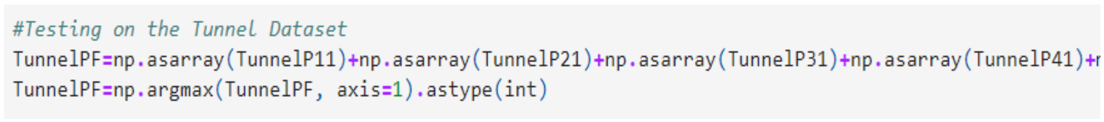


Figure 3.64: Max Voting Explanation

We have six prediction and the final output will depend onthe six of them.

3.6.2.3 Add then ArgMax

In this part, I added the outputs of the six models and the applied the ArgMax function to get the class having the highest probability, here is how I did it:



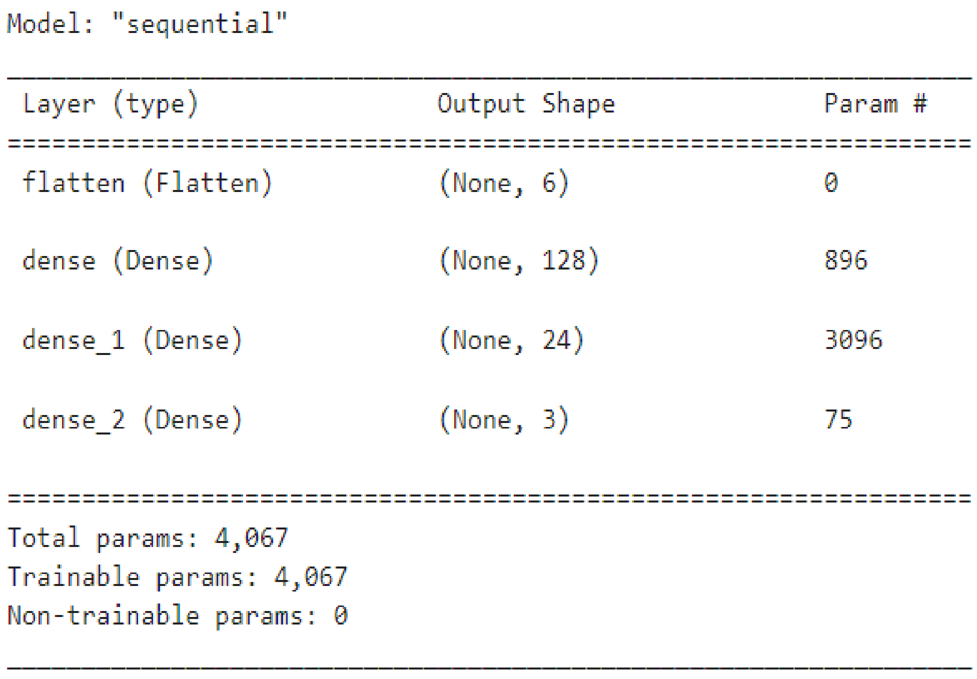
```
#Testing on the Tunnel Dataset
TunnelPF=np.asarray(TunnelP11)+np.asarray(TunnelP21)+np.asarray(TunnelP31)+np.asarray(TunnelP41)+
TunnelPF=np.argmax(TunnelPF, axis=1).astype(int)
```

Figure 3.65: Ensembling Adding then applying ArgMax

Here, as the ArgMax function returns the index of the biggest cell, when we add the results together we will add the probabilities of belonging together of the six models.

3.6.2.4 Model empowered voting technique

In this part I though that we use models to figure out pattern in the data, so why not give a model the outputs of all of our models as input and giving it the real labels as outputs and then letting it decide the pattern on it's own!
Here I didn't use a big model, neither CNN or LSTM, I kept it simple and used Dense Layers.
Here is the summary of the model that I have used in this part: I then used the same technique on all of the datasets and created four models: one or the Tunnel Dataset, one for the Tunisian Dataset, one for the Lebanon Dataset and one for the Egyptian Dataset.
Here is the summary of the model structure:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 6)                 0

 dense (Dense)               (None, 128)               896

 dense_1 (Dense)             (None, 24)                3096

 dense_2 (Dense)             (None, 3)                 75

=================================================================
Total params: 4,067
Trainable params: 4,067
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.66: Ensembling model architecture

3.7 Conclusion

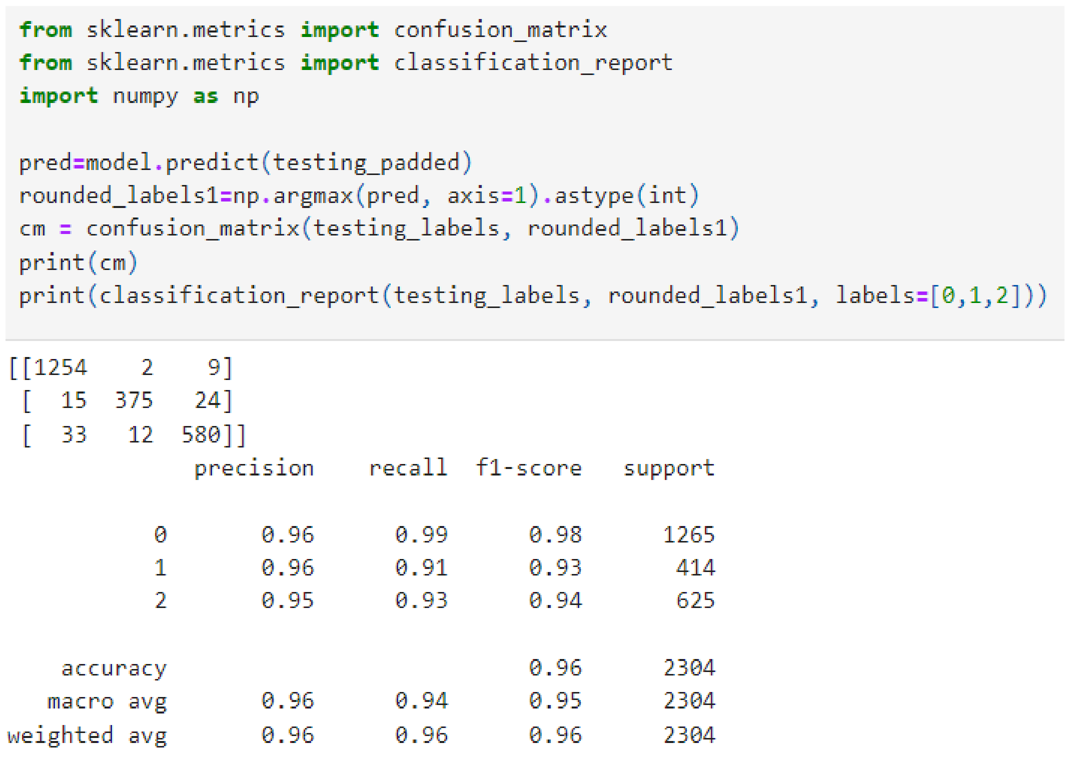
After preparing so many models and methods, we need to further explore their results and study their strengths and weaknesses. Which brigs us to the last part of our Project which is evaluation.

Chapter 4

Evaluations

4.1 Introduction

After preparing the models an Preparing the testing data, I tested the models' performances with all the datasets. Here is an exemple on the code I ran to get the classification metrics and the confusion matrices:

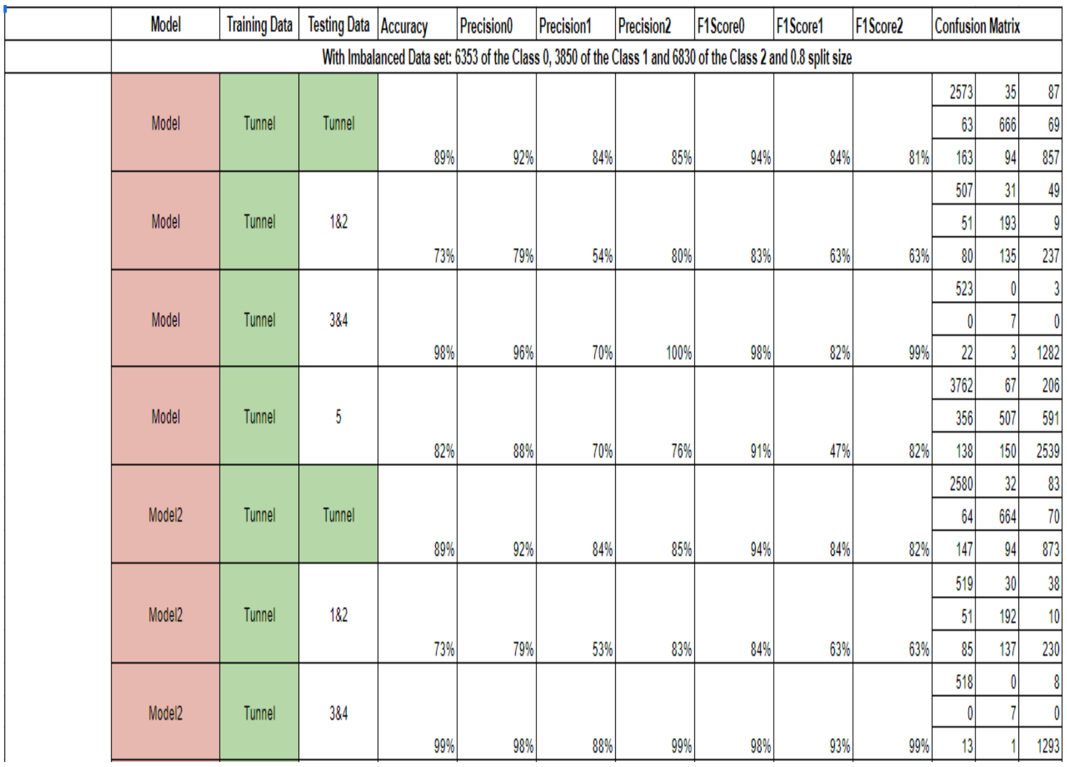


```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import numpy as np

pred=model.predict(testing_padded)
rounded_labels1=np.argmax(pred, axis=1).astype(int)
cm = confusion_matrix(testing_labels, rounded_labels1)
print(cm)
print(classification_report(testing_labels, rounded_labels1, labels=[0,1,2]))
```

```
[[1254    2    9]
 [  15  375   24]
 [  33   12  580]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.98      1265
           1       0.96      0.91      0.93       414
           2       0.95      0.93      0.94       625

    accuracy                           0.96      2304
   macro avg       0.96      0.94      0.95      2304
weighted avg       0.96      0.96      0.96      2304
```

Figure 4.1: Classification report and Confusion matrix

I then noted the results on a sheet that ended up too long to share in this report! here is a part of it:



Model	Training Data	Testing Data	Accuracy	Precision0	Precision1	Precision2	F1Score0	F1Score1	F1Score2	Confusion Matrix		
With Imbalanced Data set: 6353 of the Class 0, 3850 of the Class 1 and 6830 of the Class 2 and 0.8 split size												
Model	Tunnel	Tunnel	89%	92%	84%	85%	94%	84%	81%	2573	35	87
										63	666	69
										163	94	857
Model	Tunnel	1&2	73%	79%	54%	80%	83%	63%	63%	507	31	49
										51	193	9
										80	135	237
Model	Tunnel	3&4	98%	96%	70%	100%	98%	82%	99%	523	0	3
										0	7	0
										22	3	1282
Model	Tunnel	5	82%	88%	70%	76%	91%	47%	82%	3762	67	206
										356	507	591
										138	150	2539
Model2	Tunnel	Tunnel	89%	92%	84%	85%	94%	84%	82%	2580	32	83
										64	664	70
										147	94	873
Model2	Tunnel	1&2	73%	79%	53%	83%	84%	63%	63%	519	30	38
										51	192	10
										85	137	230
Model2	Tunnel	3&4	99%	98%	88%	99%	98%	93%	99%	518	0	8
										0	7	0
										13	1	1293

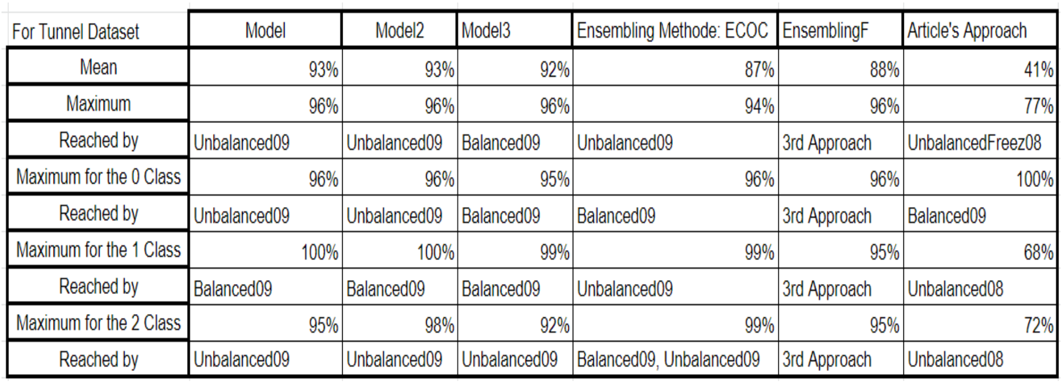
Figure 4.2: A portion of the results noted in the sheet

You can explore the whole version of the sheet through this link:
https://bit.ly/ResultsSheetImenMASMOUDI
Now I explored the large results we got and produced these tables that we will be analysing in a few lines.
I calculated the average accuracy, the max accuracy and the max accuracies for each class separately for each test dataset, meaning by that, I produced the same table for the Tunnel dataset, the Tunisian Dataset, the Lebanon Dataset and the Egyptian dataset.
I also noted for each best value, the model that got us these beautiful results so that we can know which model gave us the best accuracy. Now we will start with exploring the Tunnel dataset's results.

4.1.1 Testing on the Tunnel Dataset

The Tunnel Dataset is a multi-dialect Dataset that is composed of Tunisia, Lebanon and Egyptian Datasets that we will explore aside later on.

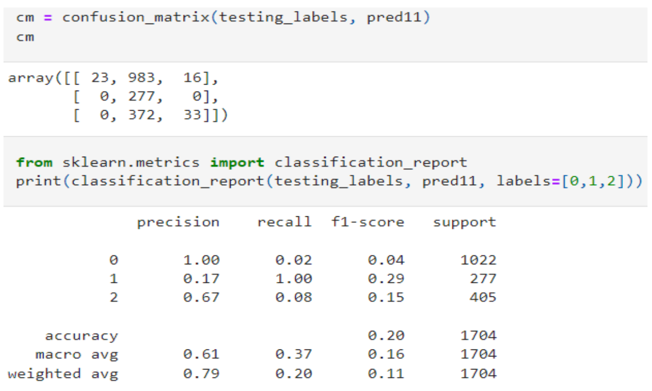
After testing the models and methods that we have worked on on the Tunnel Dataset, We could conclude this table:



For Tunnel Dataset	Model	Model2	Model3	Ensembling Methode: ECOC	EnsemblingF	Article's Approach
Mean	93%	93%	92%	87%	88%	41%
Maximum	96%	96%	96%	94%	96%	77%
Reached by	Unbalanced09	Unbalanced09	Balanced09	Unbalanced09	3rd Approach	UnbalancedFreez08
Maximum for the 0 Class	96%	96%	95%	96%	96%	100%
Reached by	Unbalanced09	Unbalanced09	Balanced09	Balanced09	3rd Approach	Balanced09
Maximum for the 1 Class	100%	100%	99%	99%	95%	68%
Reached by	Balanced09	Balanced09	Balanced09	Unbalanced09	3rd Approach	Unbalanced08
Maximum for the 2 Class	95%	98%	92%	99%	95%	72%
Reached by	Unbalanced09	Unbalanced09	Unbalanced09	Balanced09, Unbalanced09	3rd Approach	Unbalanced08

Figure 4.3: Results of the Tunnel Dataset's Tests

After exploring this data, we see that for the Primary approach, the Models one, Two and three, which are the hybrid, CNN and LSTM models reached the best accuracy which is 96 percent when respectively they were: trained with unbalanced data with a 0.9 validation split for the first model and with the same conditions for the second and with a balanced dataset having 0.9 validation split.
We could reach 100 percent accuracy for the class one in the first and second models that were trained with balanced datasets and 0.9 validation split.
As for the class two, the second model seems to give the best results when trained with an unbalanced dataset, having 0.9 as the validation split.
On the other hand the ECOC Algorithm seems to give great results when predicting the Class two, which is 'harmful', having an accuracy for that class of 99 percent when trained with balanced or unbalanced dataset with a validation split equal to 0.9.
As for the column EnsemblingF, it goes to the ensembling methods where we used the voting technique and they seem to give the best results with a 96 percent accuracy which is the highest when using the third approach, meaning by that, when using the model empowered voting technique.
As for the Article's Approach, which is the method where we used encoders and decoders and then keeping the encoder and adding dense layers at the end to get an encoder model, we see that this method gave 100 percent accuracy, whereas in all of the other models, the best accuracy we got for the class zero was 96 percent! At a first glimpse this seems great, so let's further explore this, We have got these results being trained with a Balanced data, having 0.9 as a validation split.
To further explore this part, I went to the details of the results of this model and here is the full report I got:



```
cm = confusion_matrix(testing_labels, pred11)
cm
```

```
array([[ 23, 983,  16],
       [  0, 277,   0],
       [  0, 372,  33]])
```

```
from sklearn.metrics import classification_report
print(classification_report(testing_labels, pred11, labels=[0,1,2]))
```

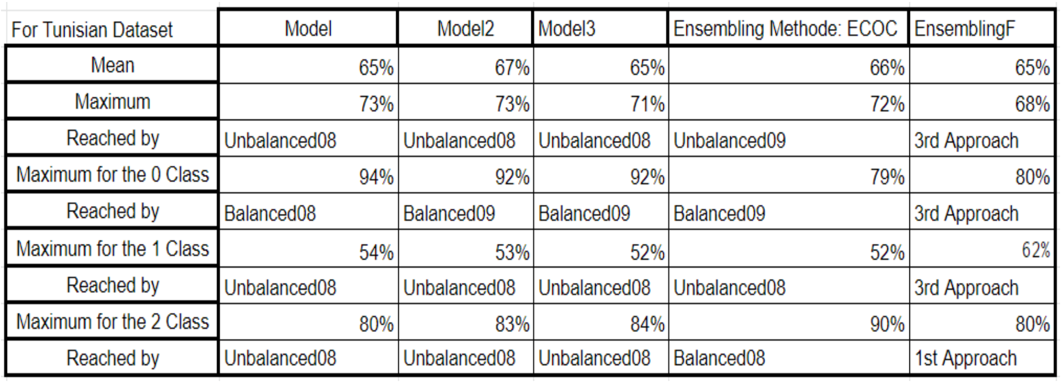
```
              precision    recall  f1-score   support

           0       1.00      0.02      0.04      1022
           1       0.17      1.00      0.29       277
           2       0.67      0.08      0.15       405

    accuracy                           0.20      1704
   macro avg       0.61      0.37      0.16      1704
weighted avg       0.79      0.20      0.11      1704
```

Figure 4.4: The full report for the Article's approach on the Tunnel dataset

After exploring the full report, I see that the class zero here is not composed of just twenty-three data points.
As the accuracy is focused on detecting the correctly classified observations both positive and negative, the F1-Score combines the Accuracy with the recall, that is why it gives us a cleared vision if our model is doing good or not.
Though the accuracy for the class zero is 10 percent, we see that the F1-Score here in this model is very low, it is equal to 4 percent which means that we can't really be happy about our model, it is basically classifying everything as the class zero and we can not trust it.

4.1.2 Testing on the Tunisian Dataset

The Tunisian Dataset is a mono-dialect dataset.
After testing the models and methods that we have worked on, We could conclude this table:



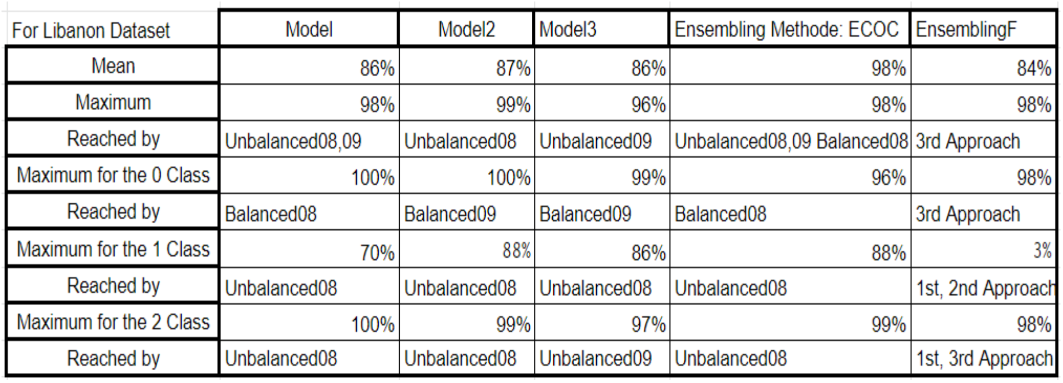
For Tunisian Dataset	Model	Model2	Model3	Ensembling Methode: ECOC	EnsemblingF
Mean	65%	67%	65%	66%	65%
Maximum	73%	73%	71%	72%	68%
Reached by	Unbalanced08	Unbalanced08	Unbalanced08	Unbalanced09	3rd Approach
Maximum for the 0 Class	94%	92%	92%	79%	80%
Reached by	Balanced08	Balanced09	Balanced09	Balanced09	3rd Approach
Maximum for the 1 Class	54%	53%	52%	52%	62%
Reached by	Unbalanced08	Unbalanced08	Unbalanced08	Unbalanced08	3rd Approach
Maximum for the 2 Class	80%	83%	84%	90%	80%
Reached by	Unbalanced08	Unbalanced08	Unbalanced08	Balanced08	1st Approach

Figure 4.5: Results of the Tunisian Dataset's Tests

When exploring the results of the tests made on the Tunisian dataset, we can see that the best general accuracies are around 73 percent, which means that as our models trained with the Tunnel Dataset, they did not catch the Tunisian dialect as much and they didn't master it.
The models that seem to give the best general accuracies are the Hybrid method and the CNN Models that were explored in the Primary approach, trained both with an unbalanced dataset and a test split equal to 0.8.
The Hybrid method model, when trained with a balanced dataset having a test split equal to 0.8, gives the best accuracy for the class zero, that is equal to 94 percent.
As for the second class the best accuracy among all of the models was around 62 percent reached by the third approach model in the ensembling method where we used the model to get a final prediction from the other six model's predictions.
As for the class two, We see that the Ensembling ECOC Algorithm gave the best accuracy that reached 90 percent when trained with a balanced dataset having a test split equal to 0.8.
We see that overall there are some good accuracies for the classes zero and two. and that the ECOC Algorithm stood out when predicting the class two.

4.1.3 Testing on the Lebanon Dataset

The Lebanon Dataset is a mono-dialect dataset.
After testing the models and methods that we have worked on, We could conclude this table:

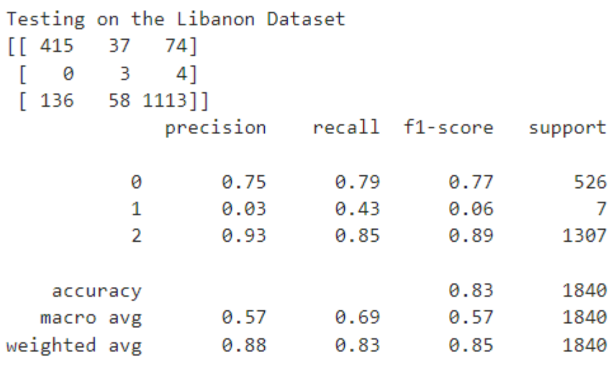


For Libanon Dataset	Model	Model2	Model3	Ensembling Methode: ECOC	EnsemblingF
Mean	86%	87%	86%	98%	84%
Maximum	98%	99%	96%	98%	98%
Reached by	Unbalanced08,09	Unbalanced08	Unbalanced09	Unbalanced08,09 Balanced08	3rd Approach
Maximum for the 0 Class	100%	100%	99%	96%	98%
Reached by	Balanced08	Balanced09	Balanced09	Balanced08	3rd Approach
Maximum for the 1 Class	70%	88%	86%	88%	3%
Reached by	Unbalanced08	Unbalanced08	Unbalanced08	Unbalanced08	1st, 2nd Approach
Maximum for the 2 Class	100%	99%	97%	99%	98%
Reached by	Unbalanced08	Unbalanced08	Unbalanced09	Unbalanced08	1st, 3rd Approach

Figure 4.6: Results of the Tunisian Dataset's Tests

When Exploring the results that our models gave when testing on the the Lebanon dataset, I was really happy to see that all of the models and approaches reached a very good general accuracy, with a 99 percent in the CNN model when

trained with an unbalanced dataset having a test split equal to 0.8. The LSTM Model gave 96 percent accuracy when trained with an unbalanced dataset, having the test split equal to 0.9. As for the rest of the models they all reached an accuracy around 98 percent.
All of this means that our models mastered the Lebanon's dialect, though it was trained to master three different dialects. This could also mean that the Lebanon dialect is somehow included in the Tunisian and the Egyptian dialects.
We reached a 100 percent accuracy for the class zero in the CNN and the LSTM Models and for the class two in the CNN model too.
What I found somehow shocking here is the results that the ensembling methods with the voting techniques for the class one which has an accuracy of three percent, so let's further explore this part to see why it is very low. Here is the full results I got:



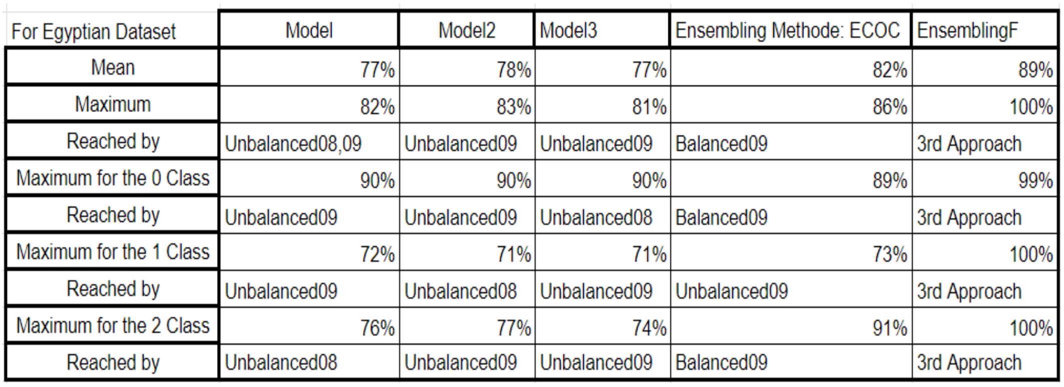
```
Testing on the Libanon Dataset
[[ 415   37   74]
 [   0    3    4]
 [ 136   58 1113]]
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       526
           1       0.03      0.43      0.06         7
           2       0.93      0.85      0.89      1307

    accuracy                           0.83      1840
   macro avg       0.57      0.69      0.57      1840
weighted avg       0.88      0.83      0.85      1840
```

Figure 4.7: The full report for the Ensembling method approach on the Lebanon dataset

After seeing the full report, it seems that the model is classifying most of the class one as two, that may be because that ensemblig model, when trained with the Tunnel dataset, it did not become an expert in the Lebanon's dialect so it confuses the class one for the class two.

4.1.4 Testing on the Egyptian Dataset

The Egyptian Dataset is a mono-dialect dataset.
After testing the models and methods that we have worked on, We could conclude this table:



For Egyptian Dataset	Model	Model2	Model3	Ensembling Methode: ECOC	EnsemblingF
Mean	77%	78%	77%	82%	89%
Maximum	82%	83%	81%	86%	100%
Reached by	Unbalanced08,09	Unbalanced09	Unbalanced09	Balanced09	3rd Approach
Maximum for the 0 Class	90%	90%	90%	89%	99%
Reached by	Unbalanced09	Unbalanced09	Unbalanced08	Balanced09	3rd Approach
Maximum for the 1 Class	72%	71%	71%	73%	100%
Reached by	Unbalanced09	Unbalanced08	Unbalanced09	Unbalanced09	3rd Approach
Maximum for the 2 Class	76%	77%	74%	91%	100%
Reached by	Unbalanced08	Unbalanced09	Unbalanced09	Balanced09	3rd Approach

Figure 4.8: Results of the Tunisian Dataset's Tests

For the Egyptian Dataset, the first thing that we notice is that we were able to reach 100 percent accuracy in the Ensembling method using the third approach, which is when having the model predict the output using the best six models' outputs. Here this model seems to be perfect and it have mastered the Egyptian Dialect.
The second best model was the ensembling ECOC algorithm, reaching a general accuracy of 86 percent being trained with an unbalanced dataset with a test split equal to 0.9.

4.1.5 Conclusion

In the end of this work, it looks like the Dialect that was understood by most of the models was the Lebanon dialect, and that the Ensembling method gives great results and the dialect that was understood the most was the Egyptian in the third approach in the Ensembling models with the voting technique.

General Conclusion

After Training more than thirty one different models, and making more than one hundred and twenty five tests and noting them in the pipeline sheet, we could find some very good results.
The different approaches that we worked on in this project were subjects of research in so many papers and research projects, so we could find some very good resources and support online.
At the end of this work, I want to get back to basics, this was a Deep Learning problem, it is a supervised one, because we had labeled data. And We classified our data into three classes, the class zero, which means normal, the class one, which means abusive and the last class which is the class two, and it refers to hurtful.
One of our models reached a perfect accuracy of one hundred percent when testing on the Egyptian dataset.
One other model reached an almost perfect accuracy of ninety nine percent accuracy on the Lebanon dataset.
Overall the best accuracy that we got on the Tunnel dataset was ninety six percent.

Bibliography

[1] Alaoui, I. E. Les méthodes ensemblistes pour algorithmes de machine learning. https://blog.octo.com/les-methodes-ensemblistes-pour-algorithmes-de-machine-learning/, Jan 2014.

[2] Beyan, M. R. K. A. R. J. B. J. S. D. . O. A snapshot neural ensemble method for cancer-type prediction based on copy number variations. https://link.springer.com/article/10.1007/s00521-019-04616-9, November 2019.

[3] Bhavsar, D. Dispelling myths: Deep learning vs. machine learning. https://www.merkle.com/blog/dispelling-myths-deep-learning-vs-machine-learning, May 2020.

[4] Demir, N. Ensemble methods: Elegant techniques to produce improved machine learning results. https://www.toptal.com/machine-learning/ensemble-methods-machine-learning#:~:text=Ensemble%20methods%20are%20techniques%20that,winning%20solutions%20used%20ensemble%20methods., Feb 2016.

[5] Drvenkar, N. Ai sub fields. https://www.researchgate.net/figure/AI-and-related-subfields_fig1_345719908, June 2021.

[6] Eliasy, A. The role of ai in capital structure to enhance corporate funding strategies. https://www.researchgate.net/publication/338672883_The_role_of_AI_in_capital_structure_to_enhance_corporate_funding_strategies, July 2020.

[7] Malard, F., and Stichelen, S. O.-V. Fully connected (dense). https://epynn.net/Dense.html, Jan 2021.

[8] Rajaa, S. Deep into learning : 4. perceptron and other alien creatures. https://medium.com/@shangethrajaa/

deep-into-learning-4-perceptron-and-other-alien-creatures-a1924da9a385, Nov 2018.