

*I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Dr Robert Atkinson."*

# **AI (MACHINE LEARNING) FOR CYBER SECURITY USING LIMITED DATA**

**Julien Priam**

**Supervisor: Dr Robert Atkinson**

**Date: 30/03/2023**

# 1 Abstract

---

The need for efficient Intrusion Detection Systems (IDS) is a modern problem related to the rise of internet use, that aims to keep communication networks operational and secure. The present study focuses on improving IDS by using Machine Learning (ML) models that can be trained with limited data. Indeed, examples of attacks are not always sufficiently available which causes many problems for ML models training. The project is based on the CICIDS2017 dataset to train and test the developed models. This dataset contains 14 different types of malicious traffic associated with benign traffic, and 38 features related to the flows characteristics were extracted to perform classification using a feed forward Neural Network (NN).

The problem has been approached step by step starting with binary classification that distinguishes benign from malicious traffic and which provides accurate results. However, the need for classifying types of malicious traffic has led the NN to evolve to multi-class classification. While the model performed well on classes containing many samples, its performances dropped drastically when integrating the underrepresented classes of the dataset (less than 200 samples per class).

A solution to this problem has been tested with Siamese Networks which are particularly efficient when dealing with limited data. Indeed, Siamese Networks deal with pairs of samples rather than unique samples, and a huge number of different pairs can be created from a small set of samples. Associated with a K-Nearest Neighbours algorithm, the Siamese Network could perform better than a feed forward NN and finally, with only 80 samples per class and 13 classes, the model achieved 75% accuracy.

## 2 Contents

---

1	Abstract.....	2
3	Introduction .....	5
4	Methodology.....	7
4.1	Environment choice .....	7
4.1.1	Hardware specifications.....	7
4.1.2	Software specifications .....	7
4.1.3	Choice of dataset .....	8
4.2	Preliminary activities.....	8
4.2.1	Reconstruction of communication flows and features extraction .....	9
4.2.2	Adding labels to the dataset .....	12
4.2.3	Features selection .....	12
4.2.4	Choice of ML algorithm.....	14
4.3	1 <sup>st</sup> experiment: binary classification .....	15
4.3.1	Architecture of the Neural Network for binary classification.....	15
4.3.2	Data preparation for binary classification .....	16
4.3.3	Implementation of a basic classifier .....	17
4.3.4	Parameters optimization .....	17
4.4	2 <sup>nd</sup> experiment: multi-class classification .....	19
4.4.1	Data preparation for multi-class classification .....	19
4.4.2	The classifier model .....	19
4.5	3 <sup>rd</sup> experiment: Siamese Neural Network.....	22
4.5.1	Siamese Networks.....	22
4.5.2	Data preparation for training.....	23
4.5.3	Data preparation for testing .....	24
4.5.4	Improvement with K-Nearest Neighbours.....	24
5	Results.....	26
5.1	Binary Classification .....	26

5.2	Multi-class classification .....	26
5.3	Siamese Network classifier .....	29
6	Discussion, analysis of results .....	31
6.1	Binary classification.....	31
6.2	Multi-class classification .....	32
6.2.1	Classification without data restriction .....	32
6.2.2	Classification with data restriction.....	33
6.3	Siamese Network Classifier .....	33
6.3.1	Analysis of the results with 100 samples per class .....	33
6.3.2	Behaviour of the models depending on the data size .....	34
7	Conclusion.....	36
8	Recommendation and future work.....	37
9	References .....	37
10	Bibliography .....	38
11	Glossary.....	38
11.1	Abbreviations and technical definitions .....	38
11.2	Attacks present in the dataset.....	38
12	Acknowledgment .....	40

### 3 Introduction

---

Since 1969 and the first connected computer network named ARPANET, communication networks have grown rapidly. In 2022, 62.5% of the earth population was using internet [1] which shows how successful this technology has become. Communication networks are mainly used to exchange information between devices: two persons can exchange together even being miles away from each other or someone can ask a server for specific information using web browsers.

This recent dependency to communication networks raises issues about the reliability and the security of such technology. What would happen if someone could turn down an online bank service or if someone could get into any device to collect private data?

As a result, experts developed systems to detect and prevent network intrusions. This project focuses on Intrusion Detection Systems (IDS). By listening to a traffic on a specific network, the aim is to differentiate a malicious traffic from a benign one and give information about it such as the time and the type of attack. This would give better tools to network security engineers to answer faster any attack they would face.

Different methods exist to detect intrusions in a network. IDS can rely on anomaly detection (based on machine learning) or on signature detection (comparing the network data to existing attack patterns). In this specific project, the aim is to develop a Machine Learning (ML) model that will classify any traffic into different categories such as benign traffic, DoS attacks, brute force attacks...

A Machine Learning model aims to build methods that learn from a specific set of training data in order to answer regression or classification problems. However, training a model often requires large amounts of data which are not always easily accessible, especially in the field of network intrusion. To answer this problem, the research will aim to test and find an efficient ML model that can be trained with few data available.

Training a classifier model and improving it is a process that involves many steps. The model needs relevant features that can distinguish the different classes, and the architecture of the model has to fit the problem that is being solved.

As a result, the first step will be to extract features from files containing network traffic captures. Then a selection must be made to keep only the relevant features. Once this is done, the first ML model will perform binary classification (benign and malicious traffic), then a more evolved version will perform multi-class classification (benign and different network attacks). The final evolution of the classifier will be to introduce the data limitation by highly

reducing the size of the training dataset. The solution explored is to implement a Siamese Neural Network because such models are quite robust with few training samples.

## 4 Methodology

---

### 4.1 Environment choice

#### 4.1.1 Hardware specifications

For hardware selection I decided to use my personal computer which offers the following characteristics:

<b>Operating System</b>	Windows 11
<b>Core</b>	Intel core i5
<b>RAM</b>	16Gb
<b>Frequency</b>	2.4GHz
<b>Stockage</b>	512Gb SSD (100Gb free)

*Tab 0: Hardware specifications used in the project*

These specifications seem reasonable and proved to be sufficient for most of the project. However, the dataset was enormous as it gathered five days of network captures (~50Gb). An access to a distant server has later been granted which allowed to run code for many hours-day and store huge files.

#### 4.1.2 Software specifications

To develop a Machine Learning model, many programming languages are available, the most common one being Python (version 3.10) with its libraries *Tensorflow* and *PyTorch*. *Tensorflow* has been selected for this project because it is better documented and offers better visualization by recording the whole training process. *Tensorflow* is a library for ML tasks, and we will use the *Keras* library that runs on top of *Tensorflow* and provides high-level NN implementation.

I first decided to work on a virtual machine running Debian 11.5 so I could use PyCharm IDE with an easy management of libraries. I later moved to Visual Studio Code associated with the wsl extension because it was lighter and faster to work directly with the native operating system, and it allowed the full use of computer resources.

Most of the time, packet captures on a network are saved in a pcap format file (pcap for Packet CAPture). These captures are performed by a software named Wireshark which I downloaded to allow a better visualization of the dataset. I also used Wireshark so I could check that information extracted from the files using Python were correct.

To end up with configuration, I created a public Github repository ([link](#)) to keep track of the research evolution and to have a backup of the code and the results.

#### 4.1.3 Choice of dataset

Training a classifier requires a huge amount of data. In the field of IDS, the Canadian Institute for Cyber-Security provides a complete dataset gathering five days of network traffic [2]. During this period 14 different types of attacks were performed on the network. The CICIDS2017 dataset is supposed to have a great diversity of network flows, gathering the most up to date attacks. However, some papers [3] highlight the shortcomings of this dataset, mainly because the different classes are highly unbalanced.

Two versions of the dataset are available: the first one contains 70 different features already extracted and stored in csv files, the second one is made of pcap files containing raw data.

The project is based on the pcap files. On the one hand it requires a lot of work to extract information from these binary files, but on the other hand choosing to work with raw data allows a better management of the features and a better understanding of the project.

A short explanation on each type of attack can be found in the glossary section and the repartition of the samples in each class is presented below.

Class Labels	Number of instances
Benign	2359087
DoS Hulk	231072
Port Scan	158930
DDoS	41835
DoS GoldenEye	10293
FTP-Patator	7938
SSH-Patator	5897
DoS slowloris	5796
DoS Slowhttptest	5499
Bot	1966
Web Attack – Brute Force	1507
Web Attack – XSS	652
Infiltration	36
Web Attack – SQL Injection	21
Heartbleed	11

Tab 1: Repartition of instances in CICIDS2017 dataset

## 4.2 Preliminary activities

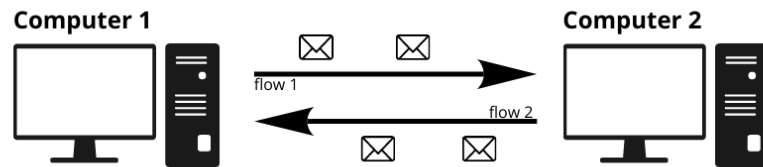
Feature selection and extraction is one of the most important steps for having an efficient network and a good accuracy in the output of a ML model.

To give a short example, when trying to train a model that detects malicious traffic, the protocol used in an exchange should be a good feature whereas the source IP is bad information as a new attack is unlikely to come from a source listed in the dataset.

It is important to define which features should be extracted before going further in the parser. First, let us understand how two devices communicate. To transfer data from



computer 1 to computer 2, packets are used: they contain the data that must be transferred and multiple additional headers with metadata. A communication is made of a certain number of packets going from 1 to 2 and vice-versa. The global exchange is called a flow.



*Figure 1: Simple representation of communication between two computers (source: author)*

A single packet does not contain enough information to predict if an exchange between two devices is malicious or not. However, once reconstructed a flow provides many interesting features such as the length of the communication or the number of packets exchanged. These are the features needed to train an efficient classification algorithm.

#### 4.2.1 Reconstruction of communication flows and features extraction

Pcap is an interface for network traffic capture. Pcap files can be read by the software Wireshark that provides a useful user interface.

While Wireshark was used to have a global overview of the data provided in the dataset, the features extraction was performed using Python and the library *dpkt*. The main function of this library is *dpkt.pcap.Reader()* that allows to open a pcap file and extract structures such as the time stamp and the body of the packet. Before extracting information, it is important to make sure that the protocol used is either TCP/IP or UDP/IP. If not, the packet is not treated.

Once we have a packet that can be exploited, the remaining part is to select what fields of the packet headers we want to extract and store them in a variable. The selected fields are as follow:

- Timestamp
- IP source and destination
- Port source and destination
- Protocol
- Flags: reset, push and urgent
- Packet size

To store the huge amount of data contained in pcap files we use a dictionary. The keys of the dictionary are the information listed above and the values are lists, each element of a list being related to one specific flow. In the end, the dictionary is made of ten lists of same length (which size is the number of packets treated).

As explained in the introductory part of this section, features are not computed from the packets but from the flows. To reconstruct a flow, it is necessary to gather together the packets that are part of the same exchange. This is done by comparing the fields extracted before: if two or more packets share the source IP, the destination IP, the source port, the destination port and the protocol, we can assume that these packets come from the same flow. It is important to notice that this flow is unidirectional, which means that a flow going from computer 1 to computer 2 will be different from a flow going the other way.

For the technical implementation, every packet that is part of a singular flow is stored in a Python list. If the dataset contains N flows, then N lists are created, and it corresponds to N/2 exchanges between devices. The reconstruction algorithm is summarised in the flowchart figure 2.

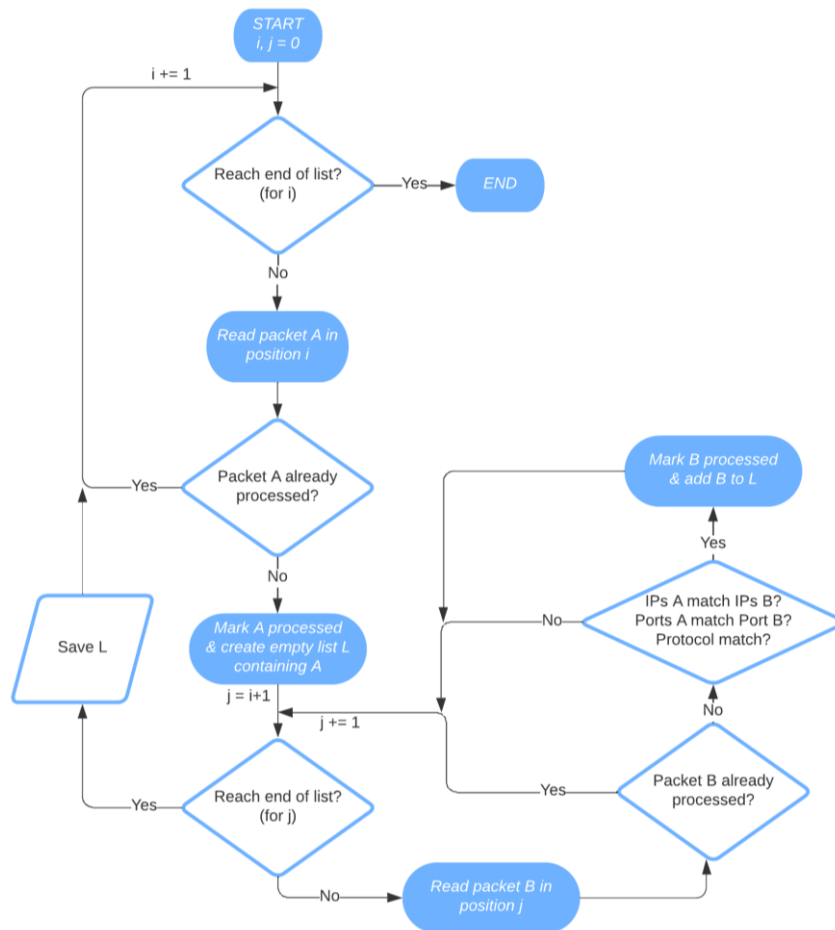


Figure 2: Algorithm used to reconstruct flows from a list of packets (source: author)

Based on the reconstructed flows it is possible to extract the features presented in tab [2](#).

Protocol	Total flow length	Forward flow length
Backward flow length	Total flow bytes/sec	Forward flow bytes/sec
Backward flow bytes/sec	Total flow pkt/sec	Forward flow pkt/sec
Backward flow pkt/sec	Forward number of packets	Backward number of packets
Ration fwd pkt/bwd pkt	Forward mean interval arrival time	Backward mean interval arrival time
Forward standard deviation interval arrival time	Backward standard deviation interval arrival time	Forward minimum interval arrival time
Backward minimum interval arrival time	Forward maximum interval arrival time	Backward maximum interval arrival time
Forward minimum offset	Backward minimum offset	Forward mean packet length
Backward mean packet length	Forward standard deviation packet length	Backward standard deviation packet length
Forward minimum packet length	Backward minimum packet length	Forward total number of bytes
Backward total number of bytes	Forward total number of PSH flags	Backward total number of PSH flags
Forward total number of RST flags	Backward total number of RST flags	Forward total number of URG flags
Backward total number of URG flags		

*Tab 2: Features extracted from the flows reconstructed*

The features are once again stored in a Python dictionary, the keys being the name of the features and the values being lists of the features values for each flow.

Having these 15 features, it would be possible to train a ML model that predicts the category of a flow. However, an attack can both be detected with the packets sent by the attacker and the answers of the victim. This is why every single flow has to be associated with its sibling so the total exchange can be reconstructed. This is once again done by comparing the IPs, ports, and protocol. Two sibling flows only have their IPs source/destination and ports source/destination inverted.

The final features that will be passed in a ML model are the association of both forward and backward flows.

#### 4.2.2 Adding labels to the dataset

The operation of supervised learning is to provide the network with a sample (made of multiple features) and it will predict an output. This output is then compared to the known result and the parameters of the network are updated to try improving the prediction.

This process requires to know the class of each sample, in this case it could be benign traffic or different types of attacks. However, as the flows are directly extracted from the pcap files, they are not associated with any label. The Canadian Institute for Cybersecurity provides a list of the attacks performed during the capture time. This list gives the following information:

- The type of attack.
- The time of the attack.
- The IP of the attacker / victim.

Adding the label is then a basic loop on every sample, if the IPs and time match with a known attack, the associated label is added. If no attack matches the sample, it is labelled as “benign”. The future implementation of classifier will require the labels to be integers. Indeed, a Machine Learning model performs mathematical operations. The tab 3 gives the correspondence between literal and numerical labels.

Type of traffic	Numerical Label	Type of traffic	Numerical Label
Benign	0	FTP Patator	1
SSH Patator	2	DoS Slowloris	3
DoS Slowhttptest	4	DoS Hulk	5
DoS Goldeneye	6	DoS Loit	7
Web Brute Force	8	XSS	9
Port Scan	10	Infiltration	11
Botnet Ares	12	SQL Injection	13
Heartbleed	14		

Tab 3: Correspondence between the types of traffic and the labels

#### 4.2.3 Features selection

Once features are available to answer a specific problem, they can feed a classifier model for training and testing. However, no one knows if these features are pertinent before testing them, and if they will lead to a good accuracy solving the problem. This is why feature selection has to be performed. This step will eliminate useless features and can also highlight which features have more impact on the training.

A feature can be useless for many reasons and correlation is an important one. The correlation between two features can be seen as their level of similarity. If two features are too close to each other, one of them does not bring new information and so does not help

improve the model. On the contrary, it can slow down the training and the prediction, and it can also lead to overfitting (which will be discussed in the next section).

An efficient way to perform feature selection is to implement Recursive Feature Elimination (RFE) using the Python library *Scikit Learn*. This figure [3](#) shows the algorithm behind RFE.

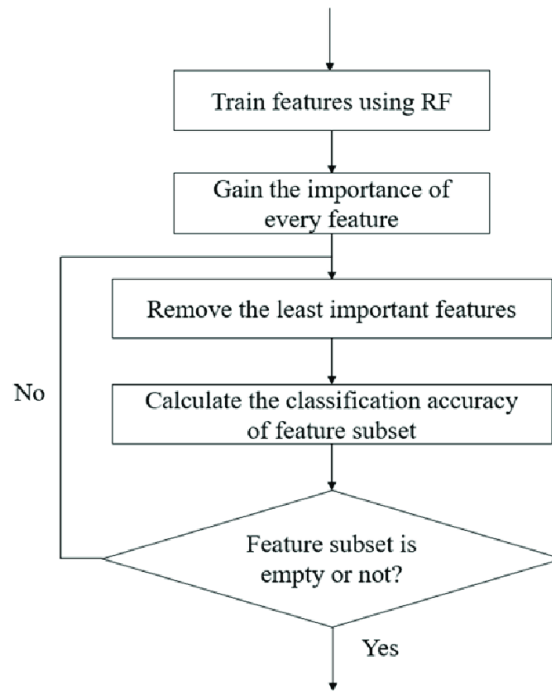


Figure 3: Recursive Feature Elimination algorithm (source: [link](#))

To summarise the RFE algorithm, it first trains a model with every feature and computes some metrics such as accuracy. Then the first feature is removed, and the metrics are computed again, if the model is still as good as it was, then the feature is removed because it is useless, else the feature is kept. This process is repeated on every feature.

A better version of RFE is RFECV (Recursive features elimination with cross-validation) and is used to implement the features selection. Cross validation is a process that helps to reduce the impact of data split between training and validation set. RFE requires a classifier model to be performed, however at this stage no model has been created. This is why a simple *RandomForest* model from the *Sklearn* library was used.

After this step, all the features that will be used to train a ML algorithm have been extracted and the project can evolve to an actual implementation of a classifier.

#### 4.2.4 Choice of ML algorithm

Developing a machine learning model is a process that requires exploring different solutions, including a lot of testing and training with the aim of converging to an efficient model that answers the problem. In general, when trying to solve a problem it is better to start from a simple solution and add complexity through time.

The current project is well suited to this kind of approach. It is possible to start with a binary classification (benign traffic or malicious traffic) and later evolve to a multi-class classification while using a lot of data for training. The final step would be to reduce the number of samples for training and explore techniques that can still lead to an efficient model.

The first choice that must be made when developing an artificial intelligence is the type of algorithm. It can be Neural Network (NN), Decision Trees (DT), Support Vector Machine (SVM). The choice of algorithm must consider the problem being solved. For IDS, the algorithm should be able to implement classification with supervised learning, taking approximately 30 or 50 input features that are not necessarily correlated, nor linear. It is also important to keep in mind that at some point the number of samples available will be reduced.

Two algorithms that seem adapted for these requirements are RandomForest and feed forward Neural Networks. The choice is made over feed forward NN because it proved in many cases to be more complex and give better results even if they require longer training (which is not a criterion in this problem).

In the next sections, each time that a model has to be trained, the dataframe storing the samples is split into training and testing (respectively 80% and 20%). This split will always be the same which allows to have an identical base of comparison.

A feed forward Neural Network belongs to the deep learning category which is a subset of machine learning which belongs to artificial intelligence. These types of networks are inspired by the human brain. The key element of such a structure is the neuron that takes many inputs and gives an output based on a mathematical operation between the inputs. Once associated together, multiple neurons form a Neural Network.

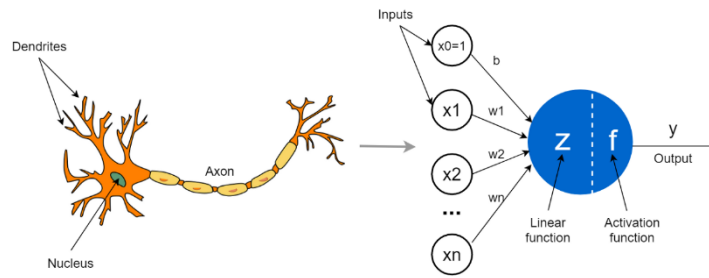


Figure 4: Structure of a neuron and an artificial neuron (source: [link](#))

## 4.3 1<sup>st</sup> experiment: binary classification

### 4.3.1 Architecture of the Neural Network for binary classification

An example of neural network used for binary classification is shown in figure 5. The first layer takes  $N$  features which go through multiple hidden layers and produces a single output that gives the predicted class of the input sample.

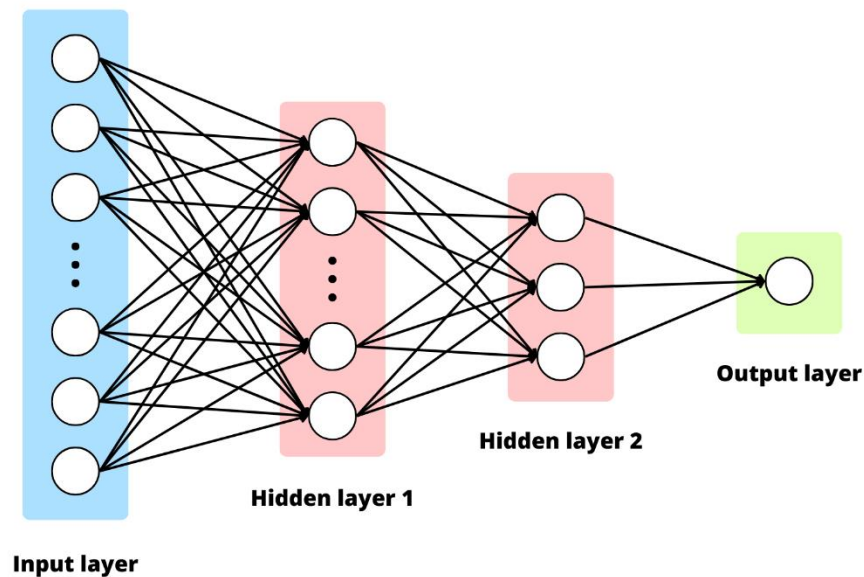


Figure 5: Example of a binary classification neural network (source: author)

The layers are said fully connected because every neuron of a layer is connected to all the neurons of the previous layer. The operation performed by a neuron consists in a linear operation on the inputs **(1)** followed by an activation function. In the hidden layers the activation function used is called Rectified Linear Unit (ReLU) **(2)**. However, the output neuron uses an activation function called Sigmoid **(3)** because it returns a float number between 0 and 1 that can be interpreted as the probability of the input sample to belong in one of the two classes.

$$z = w_1x_1 + \dots + w_nx_n + b \quad (1)$$

$$a = \max(0, z) \quad (2)$$

$$a = \frac{1}{1+e^{-z}} \quad (3)$$

In equation (1),  $x_i$  represents the inputs of the neurons and  $w_i$  and  $b$  are parameters that will be trained during the process. The training process and the parameters optimisation relies on a function called loss function. The loss function compares the Neural Network output with a known result and tells how far the model is from the truth. The objective is to minimize this loss function.


#### 4.3.2 Data preparation for binary classification

The features generated with the Python parser are stored in a dataframe (Python structure from Pandas library and useful when associated with Keras). However, each sample of the dataframe is labelled with 15 classes and the aim is to have only two classes. By looping again on the dataframe, all samples are labelled as 0 if the traffic is benign or 1 if the traffic is related to any kind of attack.

The second problem of the dataframe is that it is highly unbalanced. This means that the class “benign” comports way more samples than the class “malicious”. Having an unbalanced dataset can lead to poor performances in predicting the smaller class and to inaccurate metrics of the model. The Python library *imblearn* implements a method called *RandomUnderSampler()* whose function is to reduce the size of the biggest class to the one of the smallest by randomly removing some samples.

Last but not least, all the features must be numerical values. Indeed, each neuron performs a numerical operation on the input, so it cannot work with features being strings. Among the 38 features feeding the network, only the protocol is non numerical.

id	color			
1	red			
2	blue			
3	green			
4	blue			



id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

Figure 6: example of one-hot encoding (source: [link](#))

This feature being categorical (can be grouped into categories), it can be encoded and passed in the NN. A way to encode this feature is by using one-hot encoding. An example of one hot



encoding is shown in figure [6](#). Its Python implementation is easily performed using the `get_dummies()` function from the *Scikit-Learn* library.

The samples of flows being stored in a dataframe are now in a format that can be passed to a NN model without having to implement any other transformation.

#### 4.3.3 Implementation of a basic classifier

The most basic architecture when developing a Neural Network is to add two hidden layers, each of them using the Rectified Linear Unit (ReLU) activation function. For binary classification, the output layer should use a sigmoid activation function because it returns a result between 0 and 1 that can be associated with the probability for the sample to belong in one of the two classes.

The Python implementation is realised using *Tensorflow* with *Keras* model. The model is created with the architecture of figure [5](#) and compiled which configures the model for training with the specified parameters. The training part is performed with the `fit()` function on the specified training data and evaluated with `evaluate()` function. As *Keras* keeps a record of the training history it can be plotted for analysis.

#### 4.3.4 Parameters optimization

Hyperparameters tuning is a process that aims to improve any NN performance. The network created above uses common hyperparameters which work fine for a first test. However, these are not specific to the current problem and an appropriate association of hyperparameters would help having a high-performing model.

Optimizing a NN can be time consuming because it requires changing the parameters one by one and analysing every output to determine the impact of such a modification. Moreover, by doing so it is hard to find a good trade-off between multiple parameters that impact the performances in different ways. *Scikit-learn* library provides useful tools to answer this problem: *GridSearchCV* and *RandomSearchCV*.

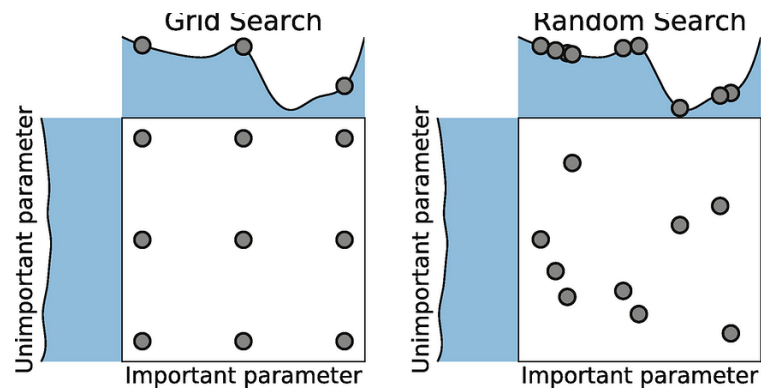


Figure 7: Comparison of Grid Search and Random Search (source: [link](#))

These two functions take as an input the parameters that must be optimized, and a set of values for each of them. They return the best association of parameters. It does so by training multiple times the network with different associations and comparing the metrics. The main difference between the two functions is that *GridSearchCV* performs an exhaustive search of every hyperparameters combination. However, this method can be highly expensive as the number of hyperparameters increases. *RandomSearchCV* is generally preferred because it can search for optimization in a larger space, and it does not test every single combination which makes it faster.

After this step of optimization, the binary classifier should perform well and no more work could be done on its architecture or on its hyperparameters. An evolution of the model to implement multi-class classification can then be investigated.

## 4.4 2<sup>nd</sup> experiment: multi-class classification

The aim of an Intrusion Detection System is to provide as much information as possible to react in an appropriate way to an attack. While binary classification alerts in case of an attack, it does not provide information on the type of attack and makes it difficult to prevent the damages. The multi-class classification aims to detect up to 14 different types of attacks.

### 4.4.1 Data preparation for multi-class classification

The major shortcoming of the CICIDS2017 dataset is its unbalanced classes as shown in tab [1](#). As mentioned earlier, a model requires balanced classes during the training process, so that the parameters are trained without overperforming on a particular class and would be unable to recognize some others.

Just like during binary classification, *RandomUnderSampler* function is used to decrease the size of the overrepresented classes. For the current use, the data limitation is not a constraint, so it is decided to only keep the classes that contain more than 4000 samples and reduce all of them to this specific size.

### 4.4.2 The classifier model

The multi-class classification problem is also based on a feed forward Neural Network and the previous architecture was used as a base in a gain of time purpose. Only a few changes were needed to adapt the NN to several classes. The output layer now requires nine neurons as it must classify nine classes (the nine first classes of tab [1](#)), and the associated activation function is changed to *SoftMax*. This function converts a vector of real numbers into a probability distribution of possible outcomes which is more suited for multi-class classification.

The loss function also needs to be changed to categorical cross-entropy. From a Python point of view, the training function requires the labels' values to be one-hot encoded which is performed using the *get\_dummies()* function as mentioned in section 4.3.4 figure [7](#). With these four changes, the model can now perform classification on 9 classes.

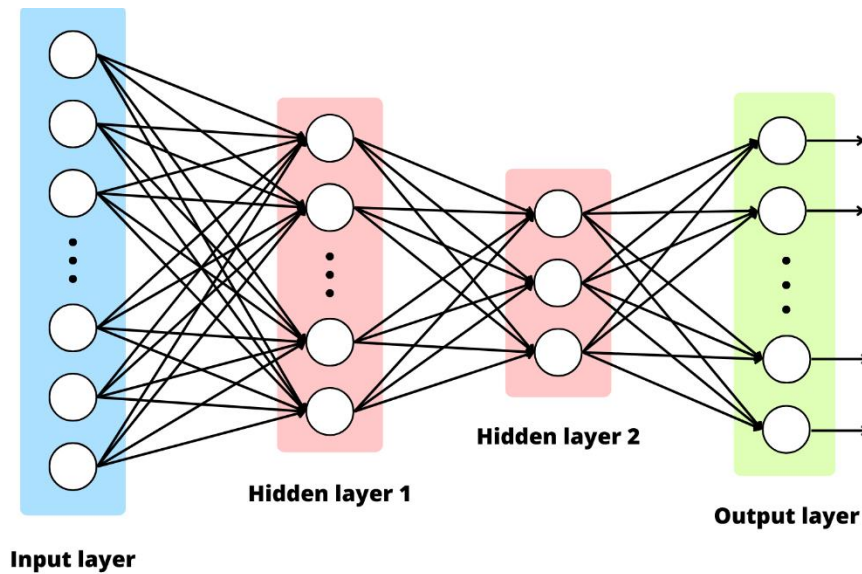


Figure 8: example of a Neural Network able to perform multi-class classification (source: author)

When performing classification on more than two classes, the accuracy of the model often provides only poor information. Indeed, if the classification was random and we imagine that the model had to classify 10 classes, the accuracy would be only 10%. With an accuracy of 50% (for instance) many different scenarios are possible:

- Each class could be well predicted 50% of the time which outputs a final accuracy of 50%
- Or five classes are always perfectly well predicted but the five others are completely mixed up which would also output a final accuracy of 50%

This information is crucial to adapt the evolution of the model. A useful tool that can visually and quickly provides it to the developer is the confusion matrix. This matrix shows for each class (on the rows) its probability to be predicted as being part of every other class. It makes it easy to visualize what classes are being confused. An example of confusion matrix obtained during the project is displayed in figure 9. It can be noticed that in the first confusion matrix, classes 1 and 2 are being confused, but this problem has been solved for the second one.

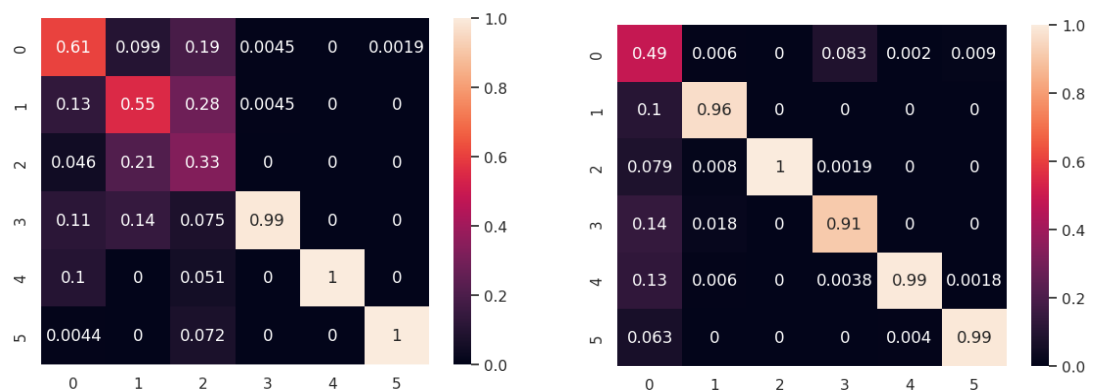


Figure 9: Example of two possible confusion matrix for the same problem (source: author)

Once again, the *Scikit Learn* library provides a function that takes the predicted classes and the real classes, and plots the associated confusion matrix.

## 4.5 3<sup>rd</sup> experiment: Siamese Neural Network

The Intrusion Detection System developed relies on the presence of a large dataset. However, these data are not always available. The research part of the project now focuses on this specific question and aims to explore solutions that can provide efficient IDS while only using a few samples per class to train a ML model.

In this section, the remaining classes of the dataset were added because the model now aims to classify the underrepresented classes.

### 4.5.1 Siamese Networks

A Siamese Neural Network is a kind of NN that takes two input samples and predicts if they belong to the same class or not. It is constructed on two exact same NN, each of them taking one of the two samples and computing the output vectors. The model then computes the distance between the two vectors (often the Euclidean distance) and gives it to the loss function. The shared NN parameters are then updated based on this loss function.

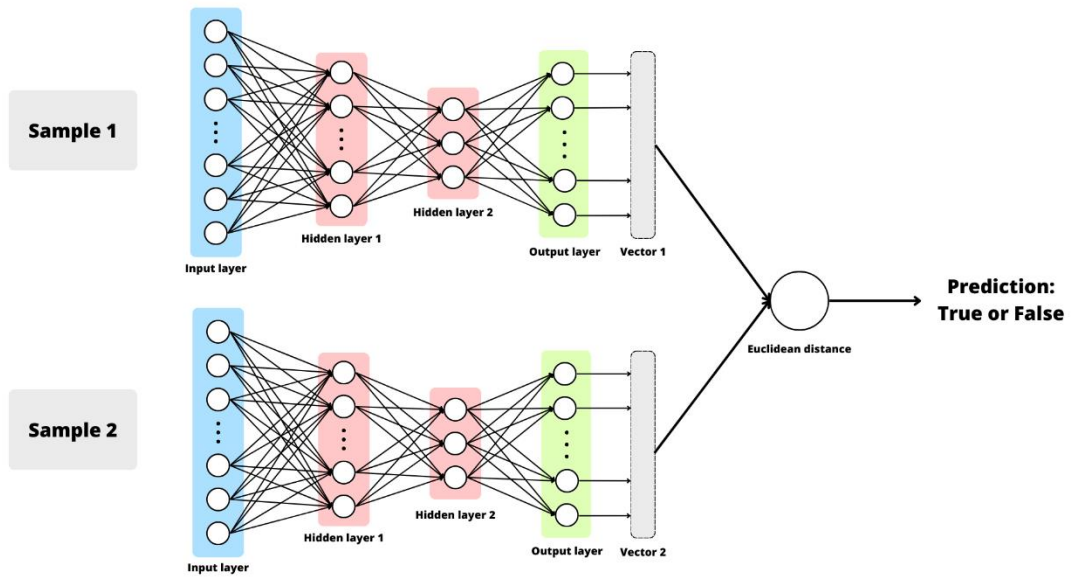


Figure 10: Global view of a Siamese Neural Network (source: author)

Siamese networks are particularly efficient on small datasets because a huge number of different pairs can be created from few samples. Indeed, each sample could be associated with every other sample and having  $N$  samples,  $\frac{N(N-1)}{2}$  pairs could be created to train the network.

#### 4.5.2 Data preparation for training

A Siamese NN is based on a classic NN, it is then possible to take the previous code for this part. The main difference is the values passed to the *model()* function. Instead of taking one input layer and one output layer, the Siamese NN takes two inputs and one output, each of the inputs being passed in the NN model. For the training part, the input which was previously a list of samples is now a list of pairs of samples. This process requires an important pre-processing work to create the pairs that will train the model and different approaches can be considered. For the rest of the document, we will call a positive pair a pair that contains two samples from the same class, and a negative pair one that contains two samples from different classes.

The simplest way to create such pairs is to associate each sample with another one selected randomly. Thus, with N samples in the dataset, N pairs would be created. As the pairs are created randomly, it is still important to make sure that a minimum number of positive pairs are created, or the model would be bad at recognizing them. Without this protection, only 7.7 positive pairs would be created when classifying 13 classes. A proportion of 50% positive pairs in the final dataset seems reasonable as it keeps a balance between positive and negative.

The process however does not answer the problem of having few samples available which does not lead to any advantages using a Siamese Network. Instead of associating each sample with another one, the second step is to associate each sample with k other samples. Thus, with N samples available,  $k \cdot N$  pairs are created (still remembering the limit of  $\frac{N(N-1)}{2}$  pairs).

Finally, because the process still relies on random selection of pairs, it is likely that similar pairs are created multiple times. This could lead to overfit the model on these samples and later drop the performances of the classifier. As a result, every new pair created is checked to prevent this last case from happening.

The entire process of pairs creation is summarized in the flowchart figure [11](#).

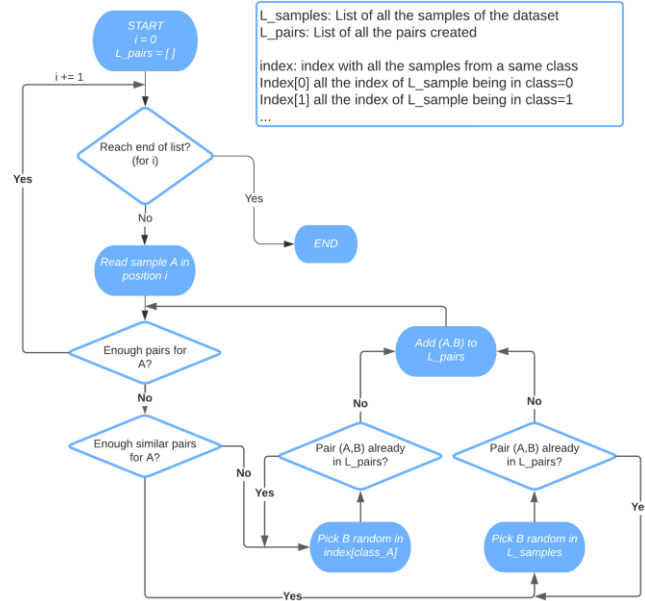


Figure 11: Algorithm used to create training pairs (source: author)

#### 4.5.3 Data preparation for testing

Once the training part has been performed, the Siamese network is able to predict with a certain accuracy if two samples belong to the same class. However, the aim is still to detect the type of traffic that is being exchanged on a network.

The simplest way to do this is to associate the sample that is being tested with one sample of each class. Thus, because we try to classify 13 types of traffic, 13 pairs should be created. It is also possible to get the distance between two classes from the Siamese Network (not only a Boolean) and this is used to predict the distance of the test sample from every other class. The shortest distance then gives the predicted class of the test sample. However, one sample of a class is not always representative of the entire class which is a major drawback of this solution. This is why an update of this algorithm associates the test sample to at least 10 samples from each class and computes the mean distance to get the predicted class.

#### 4.5.4 Improvement with K-Nearest Neighbours

In the previous section, a testing sample was associated with random other samples of each class to compute its distance with every class. However, this approach is not optimal because a set of random samples is still not necessarily representative of its class. A better way to answer the problem is to associate the test sample to its nearest neighbours in the training set.



An efficient way to find the neighbours of a sample is to use a K-Nearest Neighbours (KNN) algorithm. The process computes the distance between the different samples and learns to create groups of samples that belong to the same class.

This is why during the Siamese Network training, a KNN classifier is also trained with the same data. It is later possible to find the neighbours of the test sample (the class of these samples being known) and to create multiple pairs between the test sample and each of its neighbours. The end of the process remains the same as before, the distance of each pair is computed, and the minimum distance found corresponds to the predicted class. If no pair exists between the test sample and a specific class, the distance is manually set to a very high value that cannot impact the result.

## 5 Results

This section summarises all the results that have been obtained through the project evolution. It is separated depending on the type of model: binary classification NN, multi-class classification NN and Siamese network classification.

### 5.1 Binary Classification

1/ First Neural Network that performed binary classification had the following hyperparameters. It was trained first with a highly unbalanced dataset and second with a balanced dataset.

<b>Input layer</b>	30 neurons	<b>Learning rate</b>	0.001
<b>1<sup>st</sup> hidden layer</b>	16 neurons - ReLu function	<b>Batch size</b>	100
<b>2<sup>nd</sup> hidden layer</b>	8 neurons - ReLu function	<b>Epochs</b>	100
<b>Output layer</b>	1 neuron - Sigmoid function	<b>Optimizer</b>	Adam
<b>Loss function</b>	Binary cross-entropy	<b>Number of samples</b>	1/ Highly unbalanced 2/ 107 000 samples per class

*Tab 4: Hyperparameters of the 1<sup>st</sup> binary classifier*

Result: 94% accuracy with unbalanced dataset, 56% accuracy with balanced dataset.

2/ Second model obtained after running hyperparameters optimization (see section 4.3.4).

<b>Input layer</b>	30 neurons	<b>Learning rate</b>	0.001
<b>1<sup>st</sup> hidden layer</b>	15 neurons - ReLu function	<b>Batch size</b>	32
<b>2<sup>nd</sup> hidden layer</b>	12 neurons - ReLu function	<b>Epochs</b>	50
<b>Output layer</b>	1 neuron - Sigmoid function	<b>Optimizer</b>	Adam
<b>Loss function</b>	Binary cross-entropy	<b>Number of samples</b>	107 000 samples per class

*Tab 5: Hyperparameters of the 2<sup>nd</sup> binary classifier*

Result: 90% accuracy with balanced dataset

3/ The last model for binary classification after fixing the batch size, the number of epochs, the optimizer and running again hyperparameters optimization.

<b>Input layer</b>	30 neurons	<b>Learning rate</b>	0.001
<b>1<sup>st</sup> hidden layer</b>	30 neurons - ReLu function	<b>Batch size</b>	128
<b>2<sup>nd</sup> hidden layer</b>	25 neurons - ReLu function	<b>Epochs</b>	100
<b>Output layer</b>	1 neuron - Sigmoid function	<b>Optimizer</b>	Adam
<b>Loss function</b>	Binary cross-entropy	<b>Number of samples</b>	107 000 samples per class

*Tab 6: Hyperparameters of the 3<sup>rd</sup> binary classifier*

Result: 94% accuracy with balanced dataset

### 5.2 Multi-class classification

1/ First multi-class classifier is the same as the last binary classifier but with a bigger output layer and an update of the loss function.

<b>Input layer</b>	30 neurons	<b>Learning rate</b>	0.001
<b>1<sup>st</sup> hidden layer</b>	30 neurons - ReLu function	<b>Batch size</b>	128
<b>2<sup>nd</sup> hidden layer</b>	25 neurons - ReLu function	<b>Epochs</b>	100
<b>Output layer</b>	9 neurons - Softmax function	<b>Optimizer</b>	Adam
<b>Loss function</b>	Categorical cross-entropy	<b>Number of samples</b>	2000 samples per class

*Tab 7: Hyperparameters of the 1<sup>st</sup> multi-class classifier*

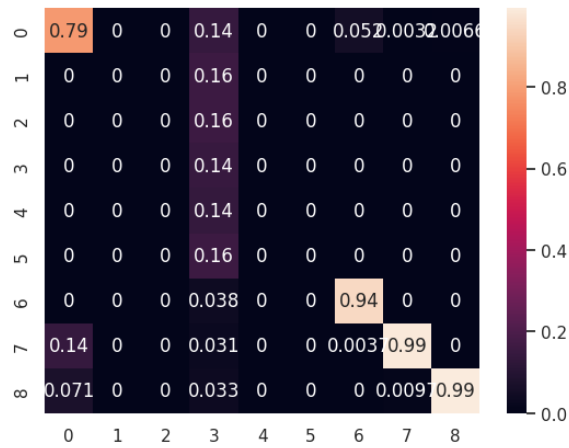


Figure 12: confusion matrix related to the training performed with parameters of tab 7

Result: 43% accuracy

2/ Second model is the same as the first one but only classifies benign traffic plus five different types of attacks.

Input layer	30 neurons	Learning rate	0.001
1 <sup>st</sup> hidden layer	20 neurons - ReLu function	Batch size	128
2 <sup>nd</sup> hidden layer	10 neurons - ReLu function	Epochs	100
Output layer	6 neurons - Softmax function	Optimizer	Adam
Loss function	Categorical cross-entropy	Number of samples	4000 samples per class

Tab 8: Hyperparameters for 6 classes classification

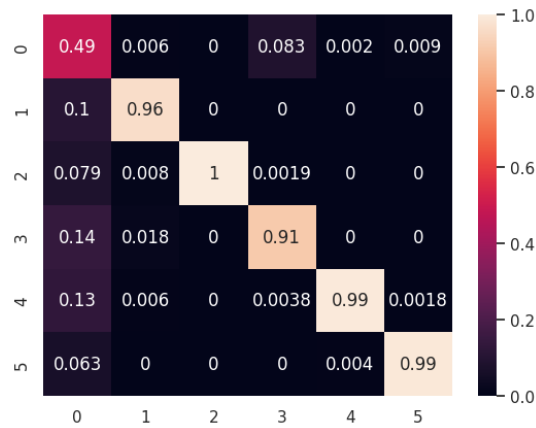


Figure 13: confusion matrix after removing similar classes

Result: 85% accuracy

3/ Third model goes back to 9 classes classification and updates hyperparameters to fit to the current problem.

Input layer	30 neurons	Learning rate	0.0001
1 <sup>st</sup> hidden layer	60 neurons - ReLu function	Batch size	128
2 <sup>nd</sup> hidden layer	30 neurons - ReLu function	Epochs	100
Output layer	9 neurons - Softmax function	Optimizer	Adam
Loss function	Categorical cross-entropy	Number of samples	2000 samples per class

Tab 9: Hyperparameters updated for 9 class classification

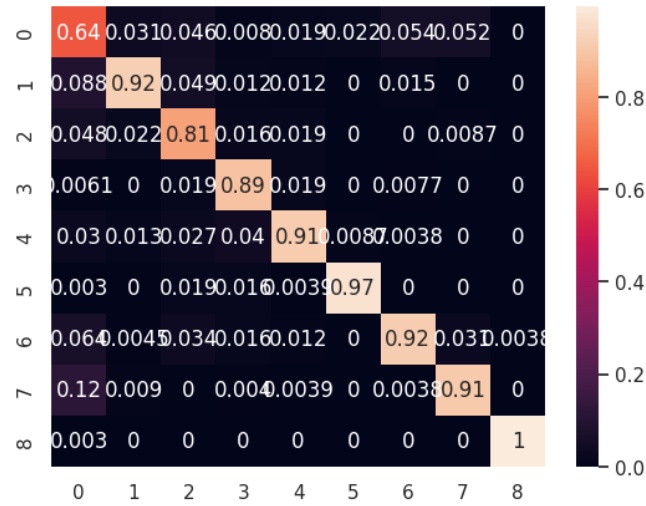


Figure 14: confusion matrix related to model of figure X

Result: 85% accuracy

4/ The same model is later trained but the number of samples per class is reduced to 100 and four new types of attack are added.

Input layer	30 neurons	Learning rate	0.0001
1 <sup>st</sup> hidden layer	60 neurons - ReLu function	Batch size	128
2 <sup>nd</sup> hidden layer	30 neurons - ReLu function	Epochs	100
Output layer	13 neurons - Softmax function	Optimizer	Adam
Loss function	Categorical cross-entropy	Number of samples	100 samples per class

Tab 10: Hyperparameters for the model with reduced dataset



Figure 15: confusion matrix related to model of figure X

Result: 68% accuracy

### 5.3 Siamese Network classifier

The Siamese Network architecture is based on the multi-class classifier which is specified in tab [10](#). Now, the results evolve relatively to the pairs that feed the network and not on its architecture directly. Moreover, two different accuracies are not compared: the capability of the Siamese Network to predict if two samples belong to the same class, and its capability to predict the actual class.

**1/ Each sample is associated with another one randomly, but it is checked that about 50% of the pairs are positive pairs.**

Result: 73% accuracy to predict if samples belong to the same class

Accuracy to predict the class not checked

**2/ Each sample is now associated with at least three others to create more pairs. To predict the class of a test sample, it is compared to one sample of each class and the minimum distance computed is used to predict the class.**

Result: 80% accuracy to predict if samples belong to the same class

42% accuracy to predict the class of a sample

**3/ Each sample is now associated with at least 30 others to create more pairs.**

Result: 86% accuracy to predict if samples belong to the same class

50% accuracy to predict the class of a sample

**4/ Integration of a K-Nearest Neighbours algorithm in the testing process.**

Result: 86% accuracy to predict if samples belong to the same class

72% accuracy to predict the class of a sample

**5/ Improvement in the pairs creation process by removing duplicate pairs.**

Result: 89% accuracy to predict if samples belong to the same class

75% accuracy to predict the class of a sample

A summary of the evolution is shown figure [16](#).

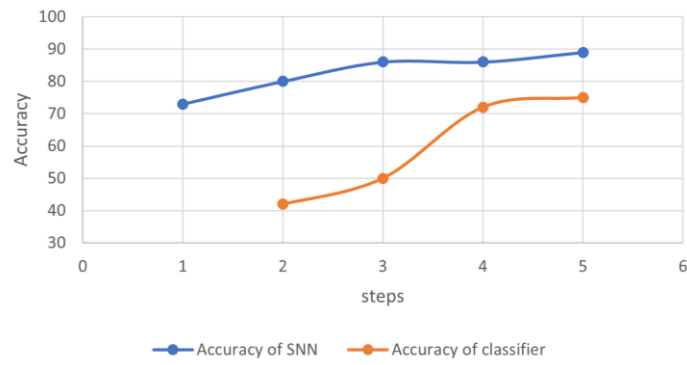


Figure 16: Evolution of the accuracy through the previous steps

6/ The model is then tested on different sizes of dataset, from 24 samples per class to 158. The feed forward Neural Network is also tested with the same samples to compare both solutions.

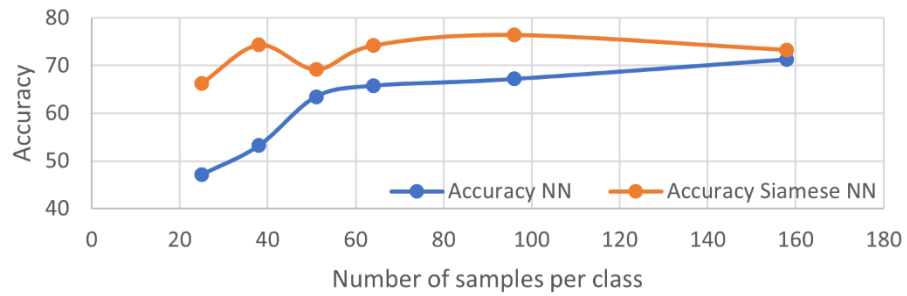


Figure 17: Evolution of Siamese Network and Feed Forward NN depending on the size of the dataset

## 6 Discussion, analysis of results

All results that have been acquired through this project are being discussed in this section. The aim was to compare the performances of the different models and to highlight the advantages or disadvantages of each solution explored and finally give an answer to implement an efficient Intrusion Detection System based on Artificial Intelligence with few data available.

### 6.1 Binary classification

The first Neural Network developed was kept as simple as possible. The aim was only to have something working with the dataframe containing the features extracted from the network capture files. However, even using a simple model it outlined an aspect of the training process which is the importance of having a balanced dataset. Because the first training led to 94% accuracy, we could think that the features and the classifier parameters were well suited to the problem. But just after balancing the data, the accuracy dropped to 56% which is a poor result (as the worst one is 50%) but a realistic one. This observation was remembered for every further implementation and a balanced dataset has always been used.

A Neural Network has many hyperparameters that can be tuned to reach better performances (see section 4.3.4). The best parameters search was performed with the values specified in tab 5. The functions returned a combination of parameters that reached 95% accuracy on the training set and 90% accuracy on the testing set to classify malicious traffic.

Hyperparameter	Values tried
Epochs	[25, 50, 100, 150]
Batch size	[32, 64, 128]
Optimizer	[adam, rmsprop, SGD]
1st layer size	[10, 15, 20, 25]
2nd layer size	[4, 8, 12]
<b>BEST RESULT</b>	<b>25 epochs, 32 batches, adam optimizer 15 and 12 neurons on layers 1 and 2</b>

Tab 11: values tested for hyperparameters optimization

After more reading and discussion, it appeared that some parameters are not worth being optimized directly in the *RandomizedSearchCV()* function, and that integrating them in the search could lead to inappropriate results.

As it is possible to plot the evolution of accuracy and loss along the epochs, the number of epochs should be set to a high value during the search. If it appears afterwards that the model overfit, it can still be reduced with no consequences on the other parameters. Some theories

also imply that the batch size should be set to a high value and for this reason it was decided to use 128 batches. The Adam optimizer being known as the most efficient one in many cases (and proved to be during last search) is also used as default.

The previous search did not consider the learning rate which is a major parameter. It tells the model how fast it should converge to the best solution depending on the loss function result. A too high learning rate could prevent the model from converging, while a too low learning rate would increase the computation time.

Considering the last points, a new search was performed using the following parameters:

Hyperparameter	Values tried
Learning rate	[0.0005, 0.001, 0.005, 0.01, 0.05, 0.1]
1st layer size	[5, 10, 15, 20, 25, 30]
2nd layer size	[5, 10, 15, 20, 25, 30]
<b>BEST RESULT</b>	<b>0.001 learning rate, 30 and 25 neurons on layers 1 and 2</b>

Tab 12: Parameters for 2<sup>nd</sup> Random Search

The model performances improved compared to the previous simulation. A slight overfit was also noticed, as the testing accuracy was smaller than the training one. However, this overfit was expected as discussed before and reducing the number of epochs to 50 allowed the model to gain a few more percent of accuracy. The performances seemed sufficient at this point because it would be hard to perform better by only changing the NN model and not the features or the dataset. It was then decided to move on multi-class classification.

## 6.2 Multi-class classification

### 6.2.1 Classification without data restriction

To start with, the same Neural Network was kept (the one used for binary classification), but eight types of attack were separated plus the benign traffic. The first train proved to be unsatisfactory with only 43% accuracy. This performance still shows that the prediction is not random, otherwise the classifier would only reach 11% accuracy. The confusion matrix from figure 12 actually gave much information on the origin of the problem. While some classes were well detected, it seemed that five of them were confused together, explaining the final accuracy. These classes are “ftp-patator”, “ssh-patator”, “DoS slowloris”, “DoS slowhttptest” and “DoS Hulk”. The confusion makes sense as these five classes are quite close to each other (refer to the attacks’ description in glossary).

In a first time, it was decided that only one type of DoS attack and one type of Patator attack would be kept. As expected, the result is way better than before (85% accuracy) as shown on figure 13. Because the model performs well in this case, the classes are added back to the



dataset and better parameters are searched again to find if it is possible to differentiate the confused classes.

Because the model does not overfit, increasing the size of the layers was an idea to help the model adapt to the data. More neurons means more trainable parameters that can be tuned to classify the data. The second idea that really solved the confusion problem was to decrease the learning rate by a factor 10. These two changes proved to be highly efficient and an accuracy of 85% could be reached when classifying nine classes. This result was considered good enough to move on to the next part. Data limitation should now be integrated in the project.

### 6.2.2 Classification with data restriction

After reducing the size of the dataset to 100 samples per class and integrating four more classes that had less than 2000 samples originally, it was important to test how the current Neural Network model works to have a base for comparison. The confusion matrix associated with this training is presented in figure [15](#) and performs an accuracy of 68%. This represents a drop of 17% accuracy but also a major increase of the loss function. This result highlights the need of a different solution to perform classification when data is restricted.

## 6.3 Siamese Network Classifier

### 6.3.1 Analysis of the results with 100 samples per class

From the results presented in section 4, it can be seen that a Siamese Network performs well on detecting if two samples belong to the same class. With only 1300 samples in total (100 per class) and after creating 1300 pairs the model had an accuracy of 73% to predict if the pair is positive or negative. This specific accuracy could easily be improved to almost 90% by creating more pairs and taking care of pairs duplication and presence of positive pairs.

However, this good accuracy on predicting if two samples belong to the same class was not sufficient to get acceptable results on class prediction, which did not exceed 50%. Two main reasons were found to explain such a result:

- A test sample was associated with samples that were not used to train the model. However, creating pairs with samples used for training would help the model to recognise patterns which should lead to better prediction.
- Testing pairs are random which does not lead to a representative set of each class.

The first issue was easily addressed because the training samples were available, and it did not require a massive code update. To fix the second issue a KNN algorithm was used (see

section 4.5.4). Only close neighbours would be used to create pairs with the test sample, which makes it sure that they are accurate samples for the distance calculation. Both updates improved the network that could reach 75% accuracy.

This step was considered as a major one because it was the first time that the Siamese Network performed better than the feed forward Neural Network with a dataset containing 100 samples per class.

### 6.3.2 Behaviour of the models depending on the data size

The objective of using a Siamese network is to perform well with limited data. Until now, the dataset contained 100 samples per class, 80% of which was used for training. The model was able to perform 75% accuracy with this dataset, compared to 67% for the feed forward NN. However, these results are quite close, and it is interesting to compare the evolution of the accuracy depending on the size of data available.

Figure 17 showed that with 160 training samples per class, both Siamese Networks and feed forward Neural Networks had the same accuracy. However, the accuracy of feed forward NN starts decreasing slowly when reducing the size of data while the one of Siamese Network remains globally constant. Under 50 samples per class for training, the feed forward NN model performances drop drastically, much faster than the second model.

The result can be expected. Indeed, when the number of samples decreases, the Siamese network algorithm is adapted to create more pairs. Many tests on the number of pairs showed that having around 50 000 pairs was sufficient to keep the model above 70% accuracy.

This number makes sense with the results obtained in figure 17. With  $N$  samples in the dataset, it is possible to create  $\frac{N(N-1)}{2}$  pairs. By solving the second-degree equation (4) we find  $N = 316.7$ . Having 13 classes means that 24 samples per class are needed to reach this number.

$$\frac{N(N-1)}{2} = 50000 \Leftrightarrow N^2 - N - 100000 = 0 \quad (4)$$

Figure 17 shows a drop of Siamese Network performances when reaching the limit of 24 samples per class.

The principal result that should be remembered is that a Siamese Network is not necessary when dealing with a dataset that has more than 150 samples per class. In this case a feed forward Neural Network requires less computational power and performs the same which makes it a better option. However, when dealing with smaller datasets the Siamese Network becomes the more efficient solution and can still perform with very few samples.

## 7 Conclusion

---

Developing an intelligent Intrusion Detection System with a constraint on the size of the dataset available can be challenging. While classic feed forward Neural Networks can only predict the nature of a network traffic with limited accuracy, some side solutions can be explored to work around this constraint.

Siamese Neural Networks provide a good alternative to limited data as they use pairs of samples for training, a lot of which can be created from a few samples. The results obtained in this project showed the importance of the pairs created both for training and testing. It was mainly highlighted that the testing process could not perform well by using random pairs of samples. Therefore, a K-Nearest Neighbours algorithm was included at a later stage. Trained with the same data as the Siamese Network, it could give the closest neighbours of a test sample which were used to predict the associated class with better accuracy.

This work also mainly focused on the samples preparation by performing features extraction. As most of the network captures were performed with the software Wireshark that generates pcap format files, the need of an efficient parser was essential to extract relevant features that can discriminate one type of traffic to another. This process should also be quick if the Intrusion Detection System had to be deployed.

To conclude, this work tested one solution able to deal with small datasets when implementing a classifier. However, Siamese Networks are not the only solution that exist when facing such problems and further work should look into different techniques addressing this problem.

## 8 Recommendation and future work

---

This project investigated the use of Siamese Networks for the implementation of an efficient Intrusion Detection System. However, the results could not exceed 75% accuracy to classify different types of network traffic which is not good enough to envisage a real use of this IDS.

Some observations showed that the Siamese Network developed required at least 50 000 pairs during the training process to keep an accuracy higher than 70%. But no specific search was performed to highlight the evolution of the performances in function of these pairs. It could be an interesting point to extract some correlation between the pairs and the results. Moreover, the project assumed that having the same number of positive pairs and negative pairs was ideal but it was not checked that this assumption was true.

Transfer learning is also a process often used when working with limited datasets. The paper [4] studied this solution and further work could try to mix transfer learning with Siamese Networks.

Finally, one-shot learning could also be searched to help categorise small classes. This method lays on already acquired knowledge (from the larger classes) to learn quicker on smaller classes, using the similarities with known classes. Indeed, we saw that some of the attacks considered in this project were often confused because of their similarities. These similarities could then be used to train the models on new types of attacks.

## 9 References

---

[1]: [datareportal.com/reports](https://datareportal.com/reports), Simon Kemp, 2022

[2]: [unb.ca/cic/datasets](https://unb.ca/cic/datasets), University of New Brunswick, 2017

[3]: “A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems”, Ranjit Panigrahi, Samarjeet Borah, 2018.

[4]: “Transfer Learning based Intrusion Detection”, Zahra Taghiyarrenani, Ali Fanian, Ehsan Mahdavi, Abdolreza Mirzaei, Hamed Farsi, 2018.

## 10 Bibliography

---

“A Taxonomy of Malicious Traffic for Intrusion Detection Systems”, Hanan Hindy, Elike Hodo, Ethan Bayne, Amar Seeam, Robert Atkinson and Xavier Bellekens.

“Deep Learning with Python: Neural Networks (complete tutorial)”, Mauro Di Pietro, 2021.

“A friendly introduction to Siamese Networks”, Sean Benhur, 2020.

“Apply machine learning techniques to detect malicious network traffic in cloud computing”, Amirah Alshammari, Abdulaziz Aldribi, 2021.

## 11 Glossary

---

### 11.1 Abbreviations and technical definitions

**Activation function:** Function used inside a neuron (most of the time non-linear) that decides whether the neuron should be activated or not.

**Internet Protocol (IP):** Set of rules used for addressing and routing packets across networks.

**K-Nearest Neighbours (KNN):** Algorithm used in supervised learning that works based on the distance between samples.

**Loss function:** Function that evaluates how far the prediction of a network is from the truth.

**Neural Network (NN):** Deep Learning model that aims to imitate the function and the structure of the brain cells.

**Overfitting:** Neural Networks parameters fit too well on the training data which leads to a bad prediction of new elements.

**Siamese Neural Network (SNN):** Artificial Neuron Network that uses the exact same NN on two inputs to predict if they belong to the same class.

### 11.2 Attacks present in the dataset

**Denial of Service (DoS):** A denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users.  
(Wikipedia)

**DoS Slowloris:** Application layer DDoS attack which uses partial HTTP requests to open connections between a single computer and a targeted Web server, then keeping those

connections open for as long as possible, thus overwhelming and slowing down the target.  
(*Netscout.com*)

**DoS Slowhttptest:** Highly configurable tool that simulates some application layer Denial of Service attacks. It implements most common low-bandwidth application layer Denial of Service attacks, such as Slowloris, Slow HTTP POST, Slow Read attack and Apache Range Header attack. (*Kali.org*)

**DoS Hulk:** Attack designed to overwhelm web servers' resources by continuously requesting single or multiple URL's. (*kb.mazebolt.com*)

**DoS Goldeneye:** GoldenEye is a HTTP DoS Test Tool. This tool can be used to test if a site is susceptible to DoS attacks. (*Kali.org*)

**Ddos Loic:** LOIC performs a DoS attack on a target site by flooding the server with TCP, UDP, or HTTP packets. (*Wikipedia*)

**Brute Force:** A brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing correctly. (*Wikipedia*)

**Patator:** Patator is a multi-purpose brute-forcer, with a modular design and a flexible usage. (*Kali.org*)

**SSH-Patator:** Patator brute-forcer applied to SSH protocol.

**FTP-Patator:** Patator brute-forcer applied to FTP protocol.

**Cross Side Scripting (XSS):** XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. (*Wikipedia*)

**Heartbleed:** The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. (*Heartbleed.com*)

**Botnet Ares:** A botnet attack is any attack leveraging a botnet (a group of bots and devices linked together to perform the same task) for distribution and scaling. Botnet attacks are used by cybercriminals to carry out intense scraping, DDoS, and other large-scale cybercrime. (*Datadome.co*)

**Port Scan:** A port scanner is an application designed to probe a server or host for open ports. (*Wikipedia*)

## 12 Acknowledgment

---

I thank Dr Robert Atkinson (University of Strathclyde, Glasgow) who supervised and guided the project throughout the year. I also thank Jack Wilkie and Christopher Mackinnon (PhD students) who attended every meeting and gave advice on the issues faced.