# Improving Network Intrusion Detection Using THE "KISTI+IDS2021-CDMC " Dataset, machine learning, and Deep learning techniques

**Amir Youssef [a], Amjad Dife [b], Mark Messih [c], Shrief Ahmed [d]**

[a] University of Ottawa, Ottawa, Canada, ayous016@uottawa.ca

[b] University of Ottawa, Ottawa, Canada, adife035@uottawa.ca

[c] University of Ottawa, Ottawa, Canada, mmess075@uOttawa.ca

[d] University of Ottawa, Ottawa, Canada, sahme277@uottawa.ca

## Abstract

**Network intrusion detection is defined as a pattern recognition problem where the events that constitute a normal network flow pattern are recognized, and an intrusion is defined as events that don't fit an expected pattern allowing the categorizing of events into binary or multiple classes. the earlier intrusion detection systems relied on hand-engineered features, then machine learning techniques have been used, then deep learning techniques have been used since it extracts the features, the "KISTI-IDS-2021" dataset has been used to conduct three experiments in this project to study the effect of using classical machine learning, deep learning techniques, and a hybrid approach to build a network intrusion detection system based on payload classification. As a result of the work, the performance of the classical machine learning algorithms increases by increasing the number of words in the word vector, the CNN model works as a classifier better than a feature extractor.**

## Keywords

**classical machine learning; TextCNN; CNN; Payload; Deep Learning**

## 1. Introduction

Network intrusion is defined as capturing patterns in the network that may not follow normal behavior,

and with how advanced the cyber world has become; it became increasingly important but also increasingly difficult to capture such intrusions.

Because as the world becomes more digitalized, and based on data, the more dangerous cyber-attacks become, and the more damage they can do.

Network intrusion detection systems have two types first the Signature-based, it's a human expert system defined, as experts putting the feature and signature of each malicious attack, and the Anomaly-based, it’s a method that works on separating normal actions from abnormal behaviors.

Modeling network behavior can be quite complex that's why many trials have been done to try and find an algorithm that can accurately represent network flow most of which have used machine learning techniques such as ensemble learning, it has been found that most of these methods tend to miss malicious attacks hidden in the payload of network packets.

This project proposed a network intrusion detection system that can accurately classify network packets into malicious or benign ones according to encoded payload content present in network packets using a real-world dataset of network alerts which will be described in more detail in the following sections.

## 2. Review of Literature

The authors (Zhuo et al., 2017) tried to fill the technological gap faced by conventional data analysis and knowledge extraction processes deployed in various cybersecurity-related problems using word embedding models, this largely reduces data processing work.

They used dataset KDD CUP’ 99 in their research and evaluated the performance of their model by varying vectorizers and classifiers. First, they labeled the data into binary and multi-class classes, where the data contains 37 attack types and normal types, which can be categorized into DOS, Probe, U2R, R2L, and Normal. Second, they divided the network logs in the data into training and testing data with different proportions to explore the impact of data size on the prediction performance. Third, they used three vectorizers which are Count Vectorizer, TFIDF Vectorizer, and word2Vec to obtain representation vectors during the training. Pre-trained word vectors were obtained from the word2vec toolkit by setting the Skip-gram model vector size to 300 and windows size to 8. Fourth, they used Naive Bayes, Logistic Regression, and SVM for the classification and explored the impact of data size on the prediction performance. They used SGD Classifier and applied Multinomial NB and Gaussian NB in Scikit-learn for Naïve Bayes. Fifth, they used error metrics of precision, Recall, and F1 Score to evaluate the performance.

The authors (Min et al., 2018) define the job of Intrusion detection systems (IDS) as detecting attacks in the network flow, two techniques could be employed for such detection to happen, one relies on the rules developed by cyber security experts for each protocol, these are called manual parsers which prove to be good at recognizing the known attacks but incapable of detecting new attack patterns, the second technique relies on deep learning techniques that aim to model what the normal network flow would look like and then define the attack as an anomaly compared to the normal behavior. The paper focuses on implementing a novel architecture for an IDS that relies on the second technique mentioned above by using statistical features collected from IDS alert packets as well as word embeddings of the payload present in the packets to achieve superior accuracy on the ISCX2012 dataset.

The methods are built on considering each byte to be a word and embedding it into a low dimensional, then combining the bytes of each packet together to form a sentence and apply padding to ensure that all the sentences have the same length, then pass each sentence representation to a slightly modified version of Text CNN where a feature layer is added before the SoftMax classification layer in order to extract the feature representation of CNN and use it along with the statistical features as an input for random forest classifier.

The main goal of (Cui et al., 2018) research is to build an improved IDS by using deep learning and word embedding. and this NIDS performs dimensionality reduction and learns features from the data so it can detect intrusion more effectively than the other methods.

The authors were clear when they defined the challenges behind building network intrusion detection systems. They defined two challenges: feature engineering and dimension reduction. They handled the first challenge by using deep learning techniques and handled the second challenge by using word embedding. So, by using deep learning they didn't have to perform feature engineering, and by using word embedding they can reduce the dimensionality without wasting the similarity relationship.

They have mentioned the usage of deep learning methods in intrusion detection. They divided the contributions of deep learning into three parts:

1. deep learning as classifiers: in this case, use deep learning to train a classifier or even to select a feature and use traditional machine learning methods to train the classifiers.

2. deep learning as a feature extractor and they mentioned the "Auto-Encoder" as an example.

3. deep learning as an end-to-end model. They defined the usage of deep learning as a combination of the two previous parts, which can be used first to be applied to the row data and then train the classifiers. They mentioned CNN and RNN as two approaches that have been used within this part. They defined the challenges of transferring the traffic data into images before the experiment.

They refer to the method they implemented by "WEDL-NIDS" which stands for word embedding and deep learning-based network intrusion detection system. in WEDL-NIDS they use word embedding as a first step to reduce the dimension of the payload of the packet, then as a second step they use CNN to learn the features of the network traffic. Finally, they use LSTM.

They defined their model as 3 stages model: data preprocessing, dimensionality reduction, and deep learning architecture. in the data preprocessing phase, the raw network traffic has been transformed into head features and payload text, then the payload text was input into the phase of dimensionality reduction in which the output is word vectors. Finally, the output of the two previous phases was input into the deep learning model. in the data preprocessing phase, they extract some features that identify the packets like length, they decided to choose only the first 100 words from each packet's payload, and they handled the problem of the length by truncating the extra if it was more than 100 words, and by padding zeros if it was less than 100. Finally, they used a one-hot encoding technique to transform each word in the text into a 256-dimension vector. in the dimensionality reduction phase, they mentioned that they had to choose between Word2Vec and GloVe techniques. And they chose the Word2Vec technique and skip-Gram model specifically. as an output of this step, they got the word vectors. They divided the deep learning phase into two main steps learning payload features in which they used CNN and learning global features in which they used LSTM and RNN.

As a result of the experiment that the authors (Zhuo et al., 2017) have conducted,

For binary classification (normal vs. abnormal), When the training data size is 10%, 50%, or 80% out of the entire dataset, the F1-score of classification is in the range between 0.90 and 0.97, and the F1 score is above 0.72 even when the training data size is only 0.1% of the total dataset.

After a set of comparisons, they concluded that the Word2Vec, Count Vectorizer, and TFIDF Vectorizer can extract features well when data is large enough, furthermore the features extracted by the Word2Vec vectorizer are more robust even when the training data size is limited.

Similarly, Naive Bayes, Logistic Regression, and SVM all perform well in classification tasks when training data is sufficient. Generally, SVM performs slightly better than Logistic Regression and Naive Bayes regardless of which vectorizer is used, and the classification performance obtained from Naive Bayes drops more than that of the other two algorithms as the training data size gets smaller. Therefore, they evaluated the run time of classification algorithms where the three used took less than one minute on 311,029 data samples, which meets the efficiency requirement in network intrusion detection while they found that the run time increased to over 2 hours on the same device when CNN model was run on the same device but it achieved a higher accuracy of 0.99, so this meant that there is a trade-off between the classification performance and the runtime.

Word2Vec vectorizer maintained stable high performance even with a smaller training data size. These results demonstrated that their NLP-based models could achieve high performance on the multi-class classification of non-textual network log data.

As a result of the experiment that the authors (Min et al., 2018) conducted, the authors claimed that they are the first to use a combination of the statistical features of the packets and the payload embedded in them, it was also claimed that random forest would be a good fit for such a classification problem. Those claims were well supported by results as the random forest was able to achieve an accuracy of 99 percent, and several experiments were performed by using different features such as raw bytes along with the statistical features, as well as experimenting with using the statistical features only, such experiments were able to prove that the CNN feature representation performs better than using raw bytes or not using payload information.

As a result of the experiment that the authors (Cui et al., 2018) conducted, they claimed that the features' design has a high effect on the performance of the intrusion detection systems. They mentioned that their approach achieves 99.97% in terms of accuracy, and 96.39% in terms of detection rate mentioned which dropped to 90.00% when considering the Infiltration attack. and they considered that as a quite good performance. in the claims, they have not mentioned to what extent the feature design affects the performance of NIDS. or even mention which features they were meant for.

They used an ISCX2012 dataset to perform some experiments to evaluate the model. They compared their method with other methods. They described very well how they set up the environment and mentioned the framework used and the main parameters they used. They mention the evaluation metrics they have used to perform a model evaluation based on it. which are: accuracy, detection rate, false alarm rate, and F1-Score.

They had initially supposed that the performance of classification algorithms would vary when the amount of data was small and near each other vs when the amount of the data was large. They confirmed their assumption through tables where all three classification algorithms gained a fairly steady f1-score in the range between 0.90 and 0.97 when the training data size was over 3,200 (1% or more is used as training data).

## 3. Research Objectives and Proposed Methodology

This project proposes a network intrusion detection system that can accurately classify network packets into malicious or benign ones according to encoded payload content present in network packets using the "KISTI-IDS-2021" dataset, based on implementing three experiments in each, a different number of words in the word vector were assigned to classical machine learning model, deep learning models, and a hybrid approach that used the deep learning techniques only for extracting the features and the classical models to be trained on the extracted features. Then a critical analysis has been conducted to study the effect of using each approach on the final results.

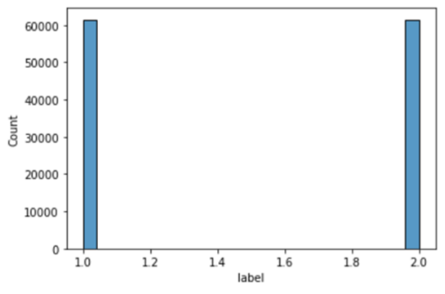
## 4. Dataset

"KISTI-IDS-2021" Dataset consists of network alerts collected from real-world data containing 25 attack types which focus on the classification of packet payload text information into malicious or benign where the packet payload is composed of words and each word is encoded into a 100-dimensional vector, the dataset consists of 153,829 rows and 4 columns, the Id which represents a unique packet id, Category which represents packet category, Word vector which represents payload encoded text in the packet, and Label which represent packet label that consists of malicious, and benign, where 1 represents malicious, and 2 represents benign.

## 5. Methodology

### 5.1. Data preprocessing

To understand the dataset and get insight, each feature has been analyzed.



```
======================================
Target Name : label
======================================
value count
2.0    61520
1.0    61520
Name: label, dtype: int64
======================================
```

Figure 1. Data balancing

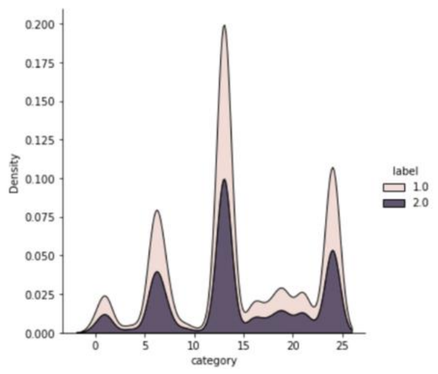


Figure 2. The distribution of the unique values in the category feature

The target variable has been used to check the data balance. As Figure 1 shows, the data is balanced since this project was interested in the rows labeled as malicious (label 1 in the figure).

Figure 2 shows that the "category" feature has 25 unique values. The most frequent category is 10 to 15, and the least frequent category is 0 to 5.

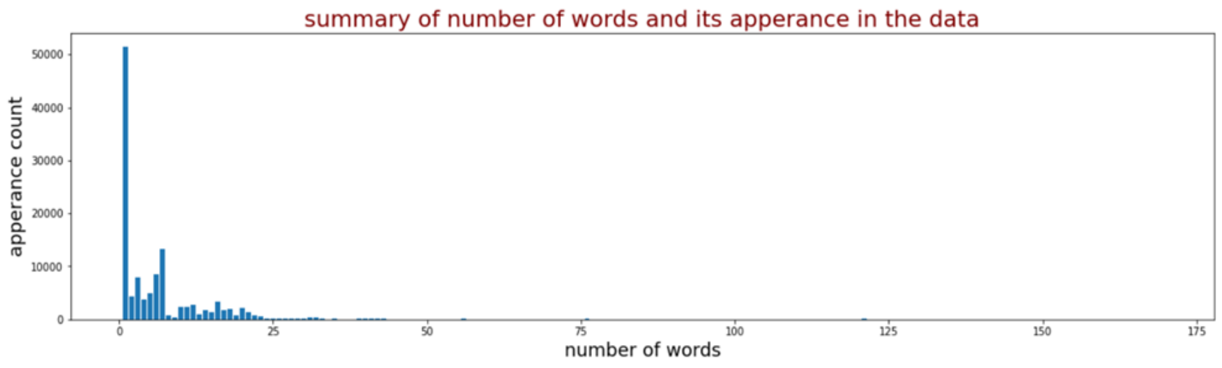


**Figure 3. Summary of the number of words in each row**

Figure 3 shows that not all the records have the same number of words, most of the records have a number of words in the range of 1 to 25 words. To deal with this challenge, different versions of the dataset each with an equal number of words has been developed to be used in the different parts of this project. The process used to handle the problem of different numbers of words in the different rows of the dataset was preparing a list of zeros and padding it to the original word vector.

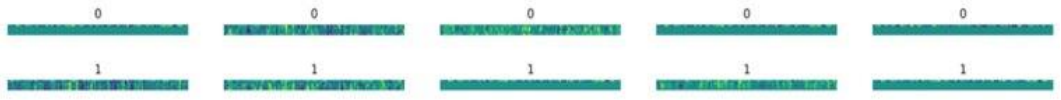


**Figure 4. Random five samples of each class in the "data5" dataset**

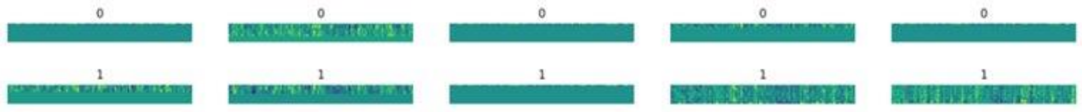


**Figure 5. Random five samples of each class in the "data10" dataset**

**Figure 6. Random five samples of each class in the "data25" dataset**

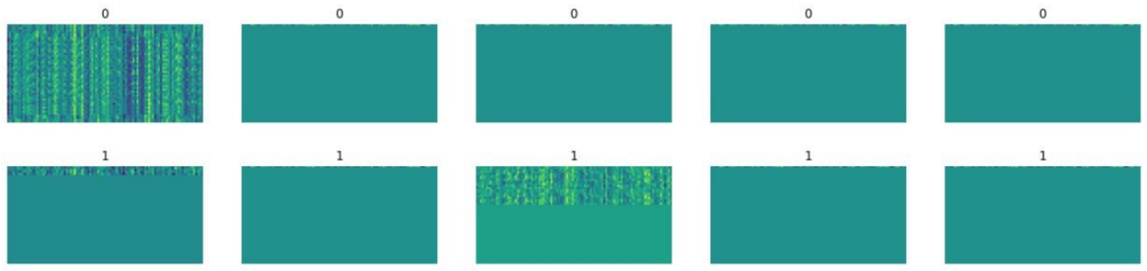


**Figure 7. Random five samples of each class in the "data50" dataset**

Firstly, a dataset with 5 words in each row has been prepared and saved as "data5", Figure 4 represents random five samples of each class in the "data5" dataset. Secondly, a dataset with 10 words in each row has been prepared and saved as "data10", Figure 5 represents random five samples of each class in the "data10" dataset. Thirdly, a dataset with 25 words in each row has been prepared and saved as "data25", Figure 6 represents random five samples of each class in the "data25" dataset. Finally, a dataset with 50 words in each row has been prepared and saved as "data50", Figure 7 represents random five samples of each class in the "data50" dataset.

Each dataset has been used separately to conduct a 4-part experiment based on the "word_vector" feature, the first part of each experiment uses classical machine learning algorithms to develop a classifier that is able to classify malicious and benign instances, the second part used Convolutional neural network for extracting the features from the word_vector as an image then classify the image based on these features. The third part uses the convolution neural network only for extracting the features from the “word_vector” and then uses e features to train a classical machine learning model. The last part uses text CNN, it deals with the “word_vector” as a text, not as an image. The data in each experiment has been divided into (80%) training (including 20% for validation) and (20%) testing.

### 5.2. Experiment 1

The “data5” dataset has been used in this experiment to conduct a 4-part experiment, each row in the “data5” dataset has five words for the “word_vector” feature.

#### 5.2.1. Experiment 1 - part 1: build a baseline model using the classical machine learning approach

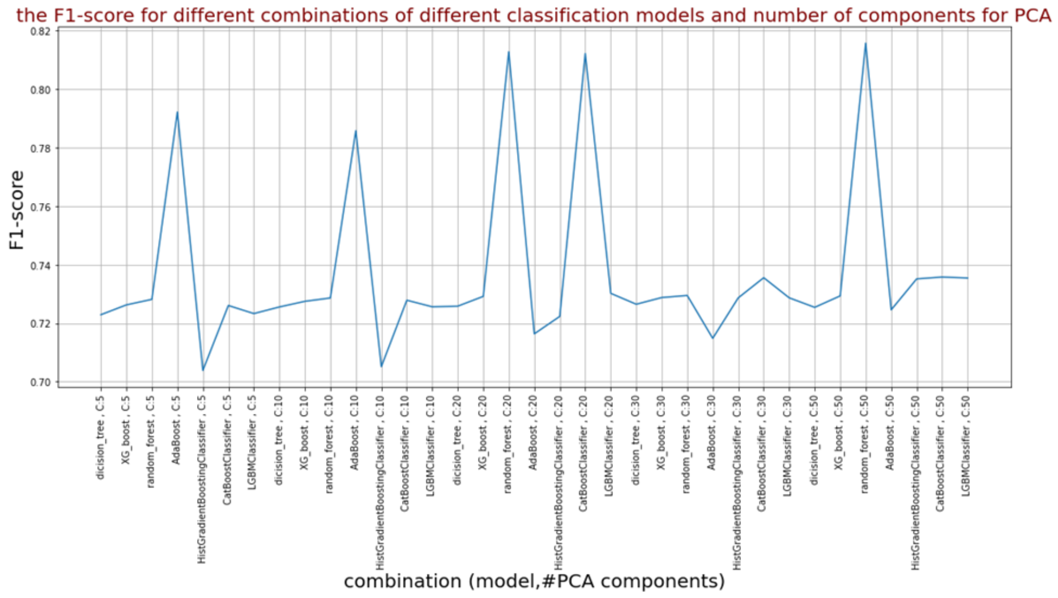


**Figure 8. Different combinations of classical models and PCA components**

In this part of experiment 1, multiple classical machine-learning algorithms have been used to build a classifier after applying a dimensionality reduction using PCA, As Figure 8 shows, decision tree, XGboost, random forest, AdaBoost, HistGradient Boosting Classifier, CatBoost Classifier, and LGBM algorithms have been used with 5,10,20,30, and 50 different numbers of components for PCA. the combinations that achieved the highest F1-Score was Random Forest using 50 PCA component, with 81.5% F1-score.

### 5.2.2. Experiment 1 - part 2: build an image classifier using a convolution neural network

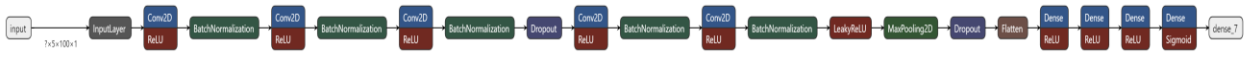


**Figure 9. Main architecture for the CNN model**

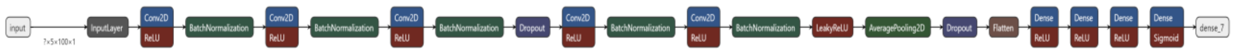


**Figure 10. Use Average Pooling instead of Max Pooling**

A convolution neural network with five convolution layers each followed by a batch normalization layer was used to extract the feature from the “word_vector” variable after reshaping it to be an image with one channel with shape (5,100,1), then a max pooling layer was added to reduce the number of features, then a classification head with four fully connected layers has been added to perform image classification by using sigmoid activation function in the output layer, Adam optimizer has been used to find the sweet point (the global minima). Figure 9 shows the main architecture of the CNN model.

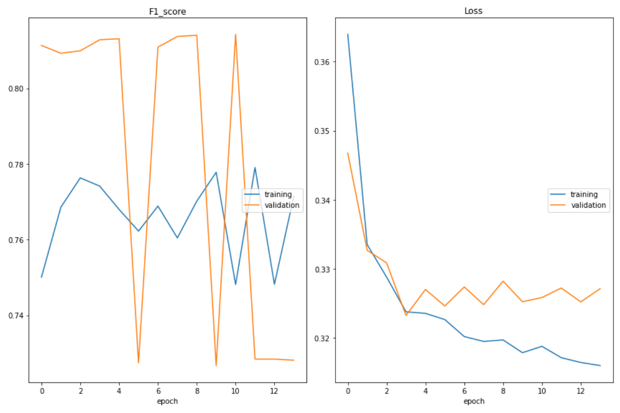


**Figure 11. The results of the main architecture**

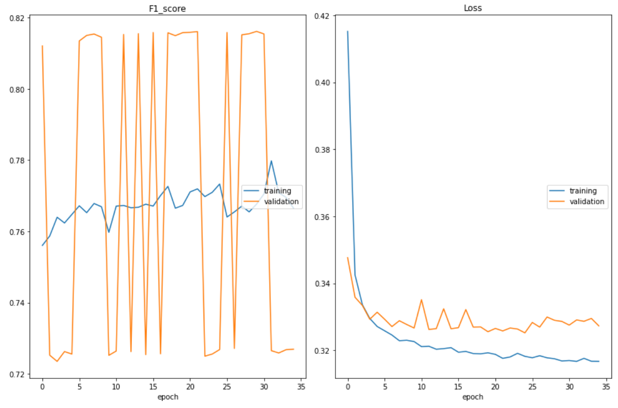


**Figure 12. The results of the CNN model after adding L1, and L2 regularization terms**

As Figure 11 shows, the model with the main architecture start overfitting at epoch 3 since the training loss kept decreasing while the validation loss kept increasing again, the F1-score has been chosen to evaluate the performance of the model since the data is labeled as a malicious and benign which mean that the cost of the false positive is different from the cost of the false negative. To overcome the model overfitting a Keras library has been used to add a combination of L1 and L2 regularization terms to the last fully connected layer.

As Figure 12 shows, the difference between the training loss and the validation loss decreased, but the model still overfitted starting from epoch number 25.

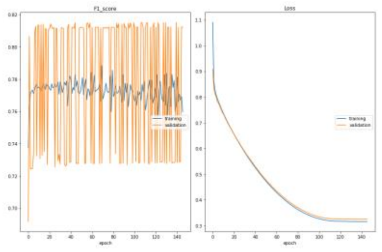


**Figure 13. The results of the CNN model after changing the optimizer with SGD instead of Adam**

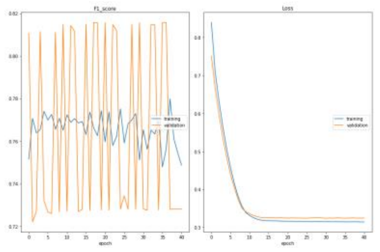


**Figure 14. The results of the CNN model after adding a function to decrease the learning rate every 10 epochs**

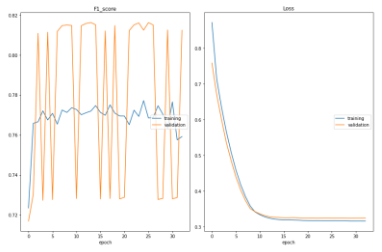


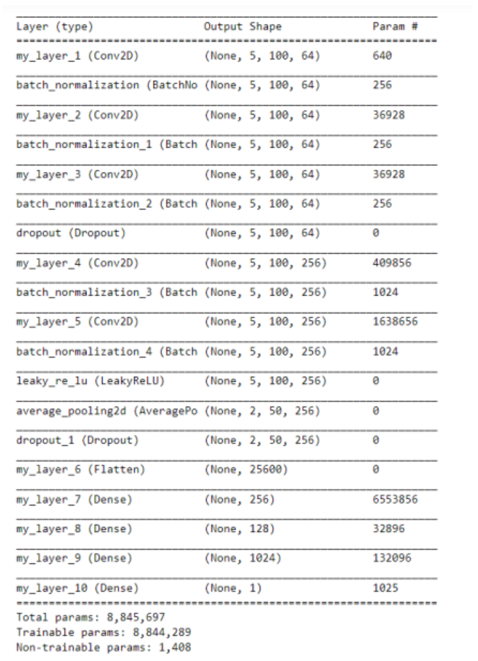
**Figure 15. The results of the CNN model after changing the max-pooling layer with an average-pooling layer**

As a third step in this part of experiment 1, a stochastic gradient descent optimizer has been used with the main architecture instead of Adam optimizers since the model using Adam optimizers was overfitting even after adding regularization terms. Figure 13 shows that both the training and validation curves kept decreasing together until the last epoch, and the model no longer overfitting.

To improve the result of the model with the stochastic gradient descent optimizer the learning rate has been scheduled to decrease after every 10 epochs. As Figure 14 shows, the value of the loss function for both training and validation loss has a lower value in the earlier epochs.

In the last step in this part of the experiment, an average pooling layer has been used instead of Max pooling. So, the model reduces the features without dropping values and just uses the largest one. As Figure 15 shows both the training and validation curves kept decreasing together, and the difference between the training and validation loss decreased.

### 5.2.3. Experiment 1 - part 3: build a classifier using convolution neural network and classical machine learning algorithms (XGBoost, Random forest).



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
my_layer_1 (Conv2D)          (None, 5, 100, 64)        640
_________________________________________________________________
batch_normalization (BatchNo (None, 5, 100, 64)        256
_________________________________________________________________
my_layer_2 (Conv2D)          (None, 5, 100, 64)        36928
_________________________________________________________________
batch_normalization_1 (Batch (None, 5, 100, 64)        256
_________________________________________________________________
my_layer_3 (Conv2D)          (None, 5, 100, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 5, 100, 64)        256
_________________________________________________________________
dropout (Dropout)            (None, 5, 100, 64)        0
_________________________________________________________________
my_layer_4 (Conv2D)          (None, 5, 100, 256)       409856
_________________________________________________________________
batch_normalization_3 (Batch (None, 5, 100, 256)       1024
_________________________________________________________________
my_layer_5 (Conv2D)          (None, 5, 100, 256)       1638656
_________________________________________________________________
batch_normalization_4 (Batch (None, 5, 100, 256)       1024
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 5, 100, 256)       0
_________________________________________________________________
average_pooling2d (AveragePo (None, 2, 50, 256)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 50, 256)        0
_________________________________________________________________
my_layer_6 (Flatten)         (None, 25600)             0
_________________________________________________________________
my_layer_7 (Dense)           (None, 256)               6553856
_________________________________________________________________
my_layer_8 (Dense)           (None, 128)               32896
_________________________________________________________________
my_layer_9 (Dense)           (None, 1024)              132096
_________________________________________________________________
my_layer_10 (Dense)          (None, 1)                 1025
=================================================================
Total params: 8,845,697
Trainable params: 8,844,289
Non-trainable params: 1,408
```

**Figure 16. The used CNN architecture**

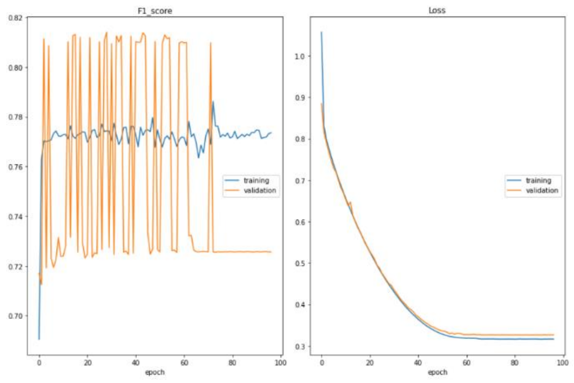


**Figure 17. F1-score and loss curves for training the CNN**

As Figure 16 shows, the experiment was done on the data using five words by CNN network as a features extraction to be the input for XGBoost and Random Forest classifiers. The main methodology used was first training the whole network with its top dense layer on the training data as a binary classifier to get the weights updated, then three new sub-networks were created from the main one by cutting the top dense layer and each one ended up with one of the last three dense layers (“my_layer_7”, “my_layer_8 “,” my_layer_9”) that preceded the top layer of the main network. For features extraction, each sub-network was used to predict the training and testing data to get the features used as the input for training and testing the XGBoost and Random Forest.

The training of the CNN As Figure 17 shows was done by the default learning rate of Keras (0.001) with a number of epochs of 100 and the aid of a callback function consisted of reducing the learning rate with factor 0.1 and early stop functions. Both monitored the validation loss.

The XGBoost was fitted by the default values of learning rate equaled 0.300000012 and number of estimators equaled 100, while the Random Forest also fitted with the default values of maximum depth equaled 100.

### 5.2.4. Experiment 1 - part 4: build a text CNN classifier

TextCNN is an algorithm that focuses on getting semantic meaning from encoded text according to the sequence of numbers in the sentence, in a more formally defined way, Sentences is defined as the concatenation of words where each word is represented by a D dimensional vector which is then reduced into a K dimensional vector resulting from sliding of 1D convolutional Filter over text using a sliding window, also it relies on the embedding layer which is a layer that's trained to do feature reduction where instead of having Thousands of uniquely identified words through word embeddings, the sentence is reduced to a fixed amount of numbers according to the specified embedding size. the preprocessing used relied on the concatenation of the first five words as representative of our payload sentence and padding where necessary, based on related works stated above that showed that a combination of TextCNN and the random forest is a good fit for this problem, a small deep-learning model to act as a TextCNN Using blocks of ConV1D‹batch normalization, and max-pooling 1d, at first, the model was unstable but by using SGD optimizer, and batch normalization as a form of regularization, the model was stabilized, then, different depths were tested to choose the optimal depth by comparing the accuracies and F1 Scores of two-block, three blocks, and four-block architecture.

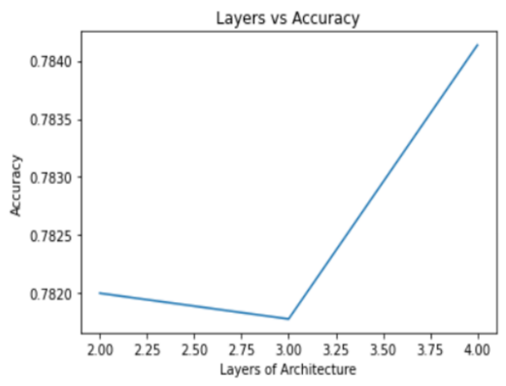


**Figure 18. Number of layers vs Accuracy of the TextCNN model**

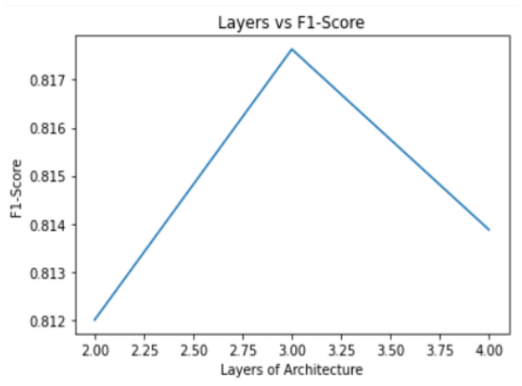
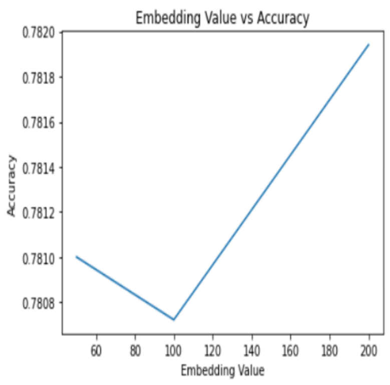


**Figure 19. Number of layers vs F1-score of the TextCNN model**

Even though Figure 18, and 19 show a bit of conflict between accuracy and F1 Score, F1 Score was used as this would help in creating a more balanced architecture, As shown in the figures, the three-block architecture achieves a slightly better f1 score.



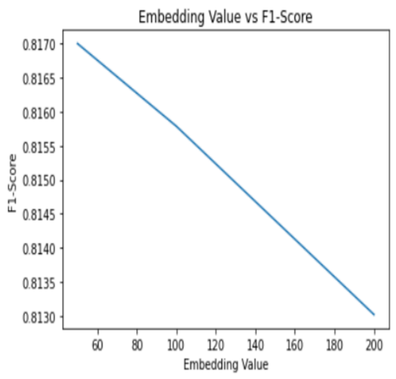


**Figure 20. Word Embedding Value vs Accuracy of the TextCNN model**

**Figure 21. Word Embedding Value vs F1-score of the TextCNN model**

It's Known from research that on the Text CNN, the embedding layer can greatly affect the score of the architecture, so we tried changing this hyperparameter, between three values 50, 100, and 200, Figures 20, and 21 show that the best f1 score between these three values, which was at 50 effectively transforming 500 values from using 5 words into 50 embedded values.

### 5.3. Experiment 2

The “data10” dataset has been used in this experiment to conduct a 3-part experiment, each row in the “data10” dataset has ten words for the “word_vector” feature.

#### 5.3.1. Experiment 2 - part 1: build a baseline model using the classical machine learning approach

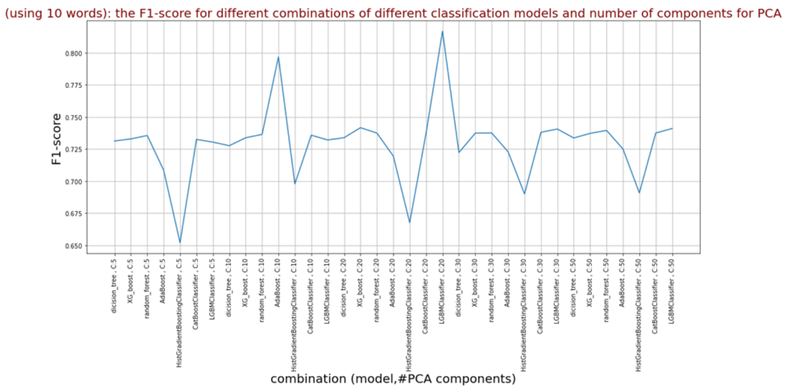


**Figure 22. Different combination of classical models with different PCA component with the data10**

The same as part 1 in experiment 1, As Figure 22 shows, a combination of, decision tree, XGboost, random forest, AdaBoost, HistGradient Boosting Classifier, CatBoost Classifier, and LGBM algorithms have been used with

5,10,20,30, and 50 different numbers of components for PCA, to build a classifier that achieves a high value for F1-score, but the difference here is the input to PCA since that dataset here contains 10 words in every row, each embedded in 100 value. the combination that achieved the best F1-Score was the LGBM Classifier using 20 components of PCA with an F1-score of 81.7%.

### 5.3.2. Experiment 2 - part 2: build an image classifier using the convolution neural network

The same architecture that has been developed in part 2 of experiment 1 has been used with the “data10” dataset.

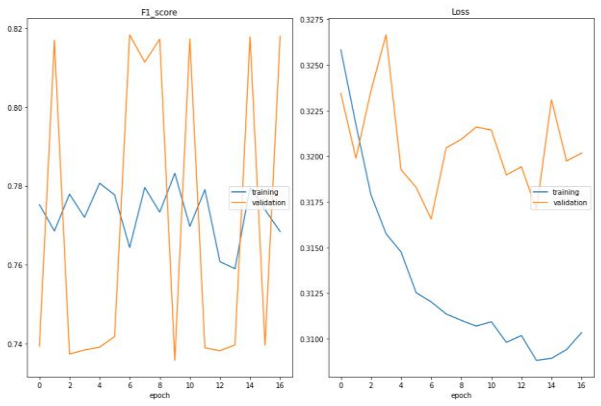


**Figure 23. The results of the main architecture of the CNN model with data10**

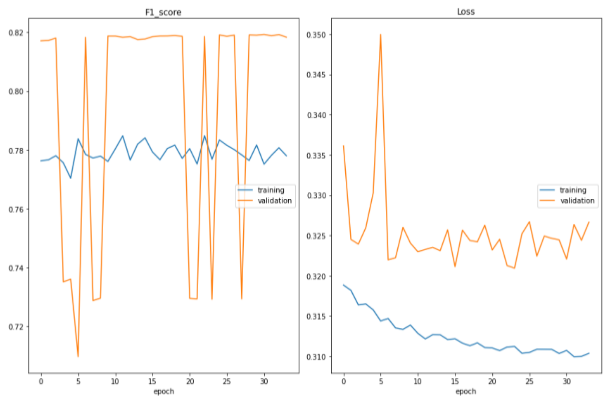


**Figure 24. The results of the CNN model after adding L1, and L2 regularization terms**

As Figure 23 shows, the model with the main architecture start overfitting at epoch 6 since the training loss kept decreasing while the validation loss kept increasing again, to overcome the overfitting a combination of L1 and L2 regularization terms has been added to the last fully connected layer. As Figure 24 shows, the model is still overfitted.

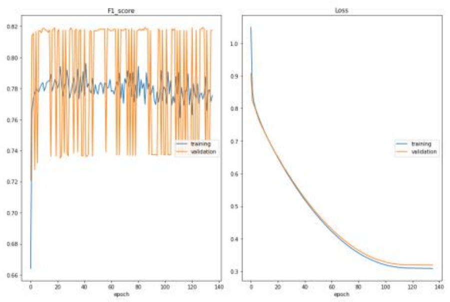


**Figure 25. The results of the CNN model after changing the optimizer with SGD instead of Adam**

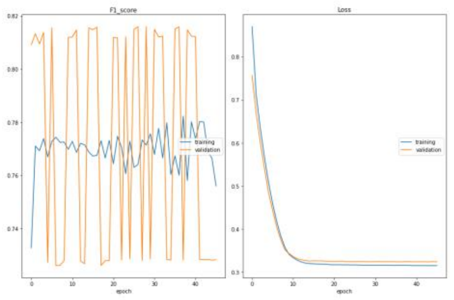


**Figure 26. The results of the CNN model after using average-pooling instead of max-pooling**

To update the weights after each instance of the training dataset, a stochastic gradient descent optimizer has been used with the main architecture instead of Adam optimizers. Figure 25 shows that both the training and validation curves kept decreasing together until the model stopped training, and the model no longer overfitting.

To reduce the number of features without dropping a meaningful value, the average pooling layer has been used instead of the max-pooling layer. So, it takes the average of the values instead of finding the largest value and leaving the others, and the same as experiment 1 part 2 in the third step, it has been scheduled to decrease the learning rate with the stochastic gradient descent after every 10 epochs. as Figure 26 shows the value of loss for both training and validation data decreased earlier.

### 5.3.3. Experiment 2 - part 3: build a classifier using convolution neural network and classical machine learning algorithms.

The same procedures of part 3 in experiment 1 were done also for experiment 2 but with data of ten words instead of five words.

## 5.4. Experiment 3

The “data25” dataset has been used in this experiment to conduct a 1-part experiment, each row in the “data25” dataset has 25 words for the “word_vector” feature.

### 5.4.1. Experiment 3 - part 1: build a baseline model using the classical machine learning approach

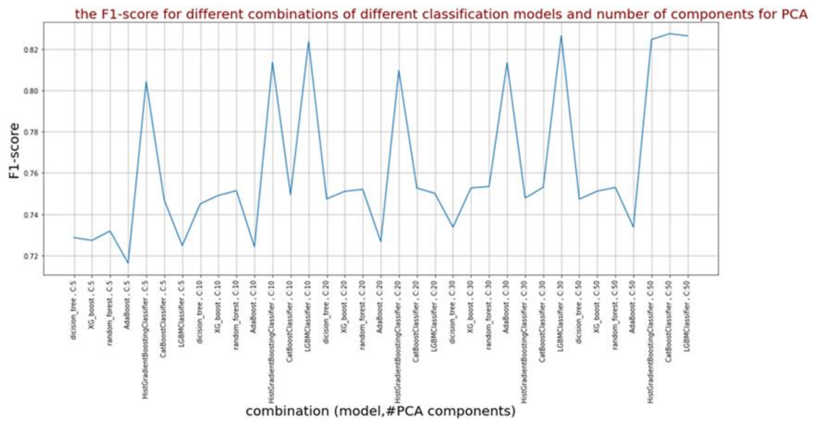


**Figure 27. Different combinations of classical models with different combination of PCA components with data25**

The same as part 1 in experiment 1 and experiment 2, as Figure 27 shows, a combination of, decision tree, XGboost, random forest, AdaBoost, HistGradient Boosting Classifier, CatBoost Classifier, and LGBM algorithms have been used with 5,10,20,30, and 50 different numbers of components for PCA, to build a classifier that achieves a high value for F1-score, dataset here contains 25 words in every row, each embedded in 100 value. the combination that achieved the highest F1-Score was the CatBoost Classifier using 50 components of PCA with an F1-score of 82.7%.

## 6. Results and Evaluations

### 6.1. Results and Evaluations - experiment 1

#### 6.1.1. Results and Evaluations - experiment 1 - part 1

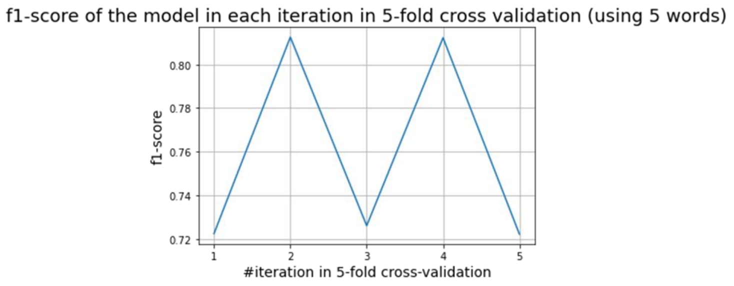


**Figure 28. The value of the F1-score for each iteration in a 5-fold cross-validation**

A five-fold cross-validation technique has been used to check the performance of the combination of the model with a specific number of PCA components that achieved the highest f1-score. Figure 28 shows the value of the F1-score for each iteration, the (Random Forest algorithm with 50 PCA components) achieved a 0.7590 mean of the F1-score across the 5-folds and a 0.0435 Standard deviation.

#### 6.1.2. Results and Evaluations - experiment 1 - part 2

| Architecture | Main architecture | After L1, L2 | Using SGD | After decreasing the learning rate | After changing the max-pooling with average-pooling |
|---|---|---|---|---|---|
| F1-score on the test dataset | 0.811894166752 | 0.711572392319 | 0.713155854225 | 0.71395360851 | 0.815514703365 |

**Table 1. The F1-score of the CNN model using the best weights saved during the training, and the test data (5 words)**

As Table 1 shows, the CNN model after adding all the modifications to overcome the overfitting problem achieved an F1-score of 0.8155.

### 6.1.3. Results and Evaluations - experiment 1 - part 3

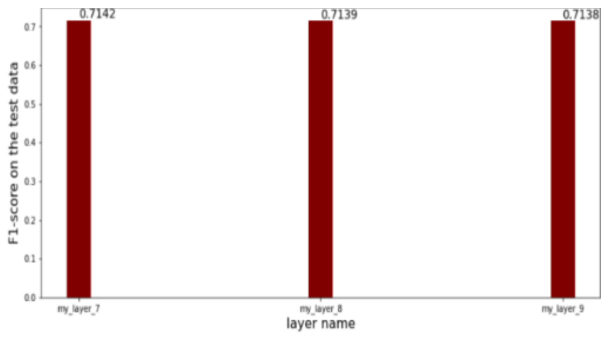


**Figure 29. Results of using CNN to perform feature extraction and XGBoost to perform classification (using only 5 words)**

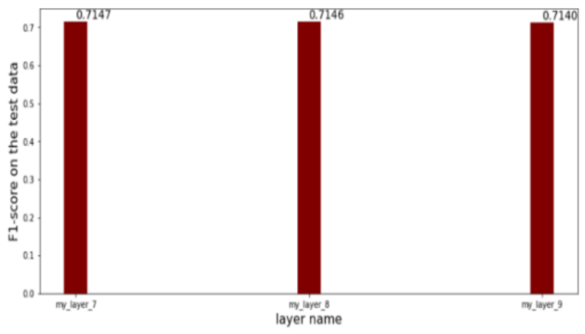


**Figure 30. Results of using CNN to perform feature extraction and Random Forest to perform classification (using only 5 words)**

The main CNN when used as a binary classifier, achieved an 81.5% F1-score while XGBoost and Random Forest with the CNN of a top layer of one of the last three dense layers achieved an F1-score of 71.42%, 71.39%, 71.38%, 71.46%, 71.46%, and 71.39% respectively as Figure 29, and 30 show. XGBoost and Random Forest achieved the same results with 5 words.

### 6.1.4. Results and Evaluations - experiment 1 - part 4

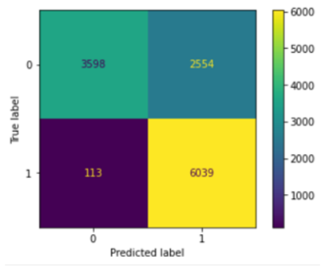


**Figure 31. Confusion Matrix for TextCNN**

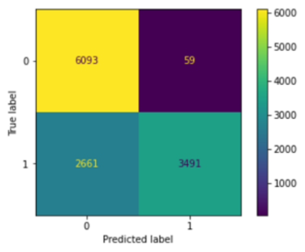


**Figure 32. Confusion Matrix for Random Forest Using TextCNN Features**

As a result of combining the best results from the hyperparameters, the final TextCNN had an 81.9% F1 score which is more balanced compared to the random forest classifier using the same features extracted from TextCNN with only 71% F1 Score.

As Figure 31, and 32 show, TextCNN created a system that is more balanced but its limitation is it's a bit more lenient with that malicious types (can pass some malicious types as normal) While the random forest relying on TextCNN features is on the strict side (pass no attack) but creates a lot of false alarms.

## 6.2. Results and Evaluations - experiment 2

### 6.2.1. Results and Evaluations - experiment 2 - part 1

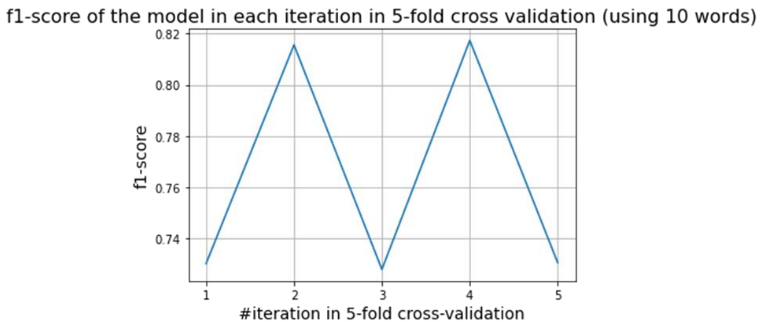


**Figure 33. The F1-score value of LGBM model in each iteration of 5-fold cross-validation**

The same as part 1 in experiment 1, a five-fold cross-validation technique has been used to check the performance of the combination of the model with a specific number of PCA components that achieved the highest f1-score.

Figure 33 shows the value of the F1-score for each iteration, the (LGBM algorithm with 20 PCA components) achieved a 0.7643 mean of the F1-score across the 5-folds and a 0.0425 Standard deviation.

### 6.2.2. Results and Evaluations - experiment 2 - part 2

| Architecture | Main architecture | After L1, L2 | Using SGD | After decreasing the learning rate and changing the max-pooling with average-pooling |
|---|---|---|---|---|
| F1-score on the test dataset | 0.8195329087048833 | 0.7152702212549873 | 0.7255539743850377 | 0.8155778378563361 |

**Table 2. The F1-score of the CNN model using the best weights saved during the training, and the test data (10 words)**

As Table 2 shows, the CNN model after adding all the modifications to overcome the overfitting problem achieved an F1-score of 0.815577.

#### 6.2.3. Results and Evaluations - experiment 2 - part 3

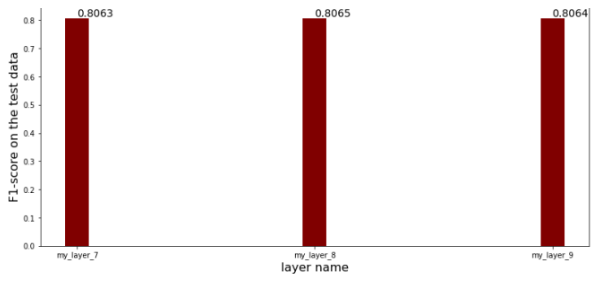


**Figure 34. Results of using CNN to perform feature extraction and XGBoost to perform classification (using only 10 words)**

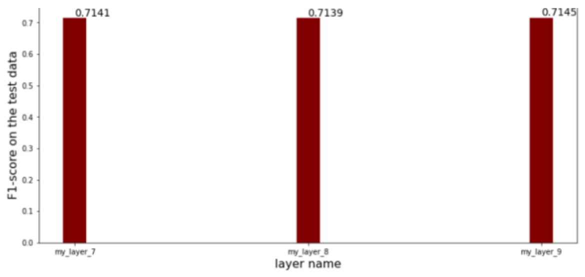


**Figure 35. Results of using CNN to perform feature extraction and Random Forest to perform classification (using only 10 words)**

There were differences in F1-scores when the experiment was repeated again using ten words, where the F1-score of XGBoost and Random Forest with the CNN of a top layer of one of the last three dense layers achieved F1-score of 80.63%, 80.65%, 80.64%, 71.41%, 71.39%, and 71.45% respectively as Figure 34, and 35 show, but the CNN classifier achieved the same result as experiment 1.

### 6.3. Results and Evaluations - experiment 3

#### 6.3.1. Results and Evaluations - experiment 3 - part 1

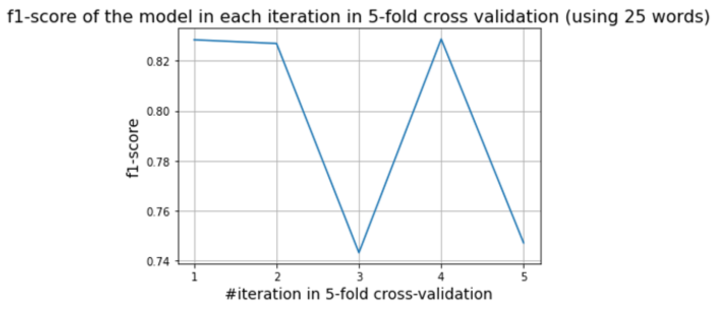


**Figure 36. The results of the CatBoost classifier in each iteration in a five-fold cross-validation**

The same as part 1 in experiment 1 and experiment 2, a five-fold cross-validation technique has been used to check the performance of the combination of the model with a specific number of PCA components that achieved the highest f1-score. Figure 36 shows the value of the F1-score for each iteration, the (CatBoost Classifier algorithm with 50 PCA components) achieved a 0.7948 mean of the F1-score across the 5-folds and a 0.0407 Standard deviation.

## 7. Discussion and Results Relevance

It is worth to be mentioned that the authors of (Ahlawat & Choudhary, 2020) showed that using hybrid CNN-SVM achieved better accuracy results than SVM alone when they worked on the MNIST dataset. Also, researchers (Jiao et al., 2021) showed that when working on MNIST, CIFAR-10, or CIFAR-100, the CNN-XGBoost Model Based on Adaptive Particle Swarm Optimization always achieved better accuracy results than CNN alone, also XGBoost achieved better F1-score results than other classifiers as SVM, linear regression, and Random Forest, whether they worked on balanced or unbalanced data sets.

Despite the use of XGBoost in the project wasn’t tuned and the CNN architecture being different from that used by the referred researchers, it seemed there were similarities where XGBoost achieved better results than Random Forest on increasing the number of words used, but it didn’t achieve better results than the CNN model when using five words however it achieved the same F1-score as the CNN model when using ten words. This might lead thinking to that optimizing XGBoost and increasing feature size could achieve better results.

It is worth mentioning that the authors (Min et al., 2018) proved that the use of neural networks for feature extraction was supposed to improve the performance of random forest, but they used statistical features in addition to the extracted features from the neural network, while this project used only the features from the neural network to train the random forest algorithm.

## 8. Limitations

From the first two experiments, it’s noticed that CNN with XGBoost classifier achieved better results using ten words than using five words, however, this acted as the main limitation where the Incapability to deal with large amounts of data and features due to hardware limitations, where the running sessions were exposed easily to the fullness of the used memory on the used tools in the experiments either on local machine supplied with 32 GB ram and SSD or on the computing platforms such as Kaggle and Google Collab which used GPU P100 and Tesla K80 12GB respectively.

## 9. Conclusion

Based on the experiments conducted in this project with the KISTI+IDS2021-CDMC dataset, it was found that using more words led to better results with both the Classical machine learning and hybrid CNN-classifier model. CNN as a classifier was better than the hybrid CNN - classifier model, but using more words and better tuning XGBoost might lead to an opposite result. Also, TextCNN as a classifier was better than TextCNN to extract the features to train the random forest.

## 10. Future work

The main priority of future work is to improve the accuracies of the hybrid CNN-XGBoost model by retraining it by using a larger amount of words and hyper-parameters tuning of XGBoost. For TextCNN, a larger architecture will be implemented and trying different preprocessing techniques for it.

## 11. REFERENCES

Zhuo, X., Zhang, J., & Son, S. W. (2017). Network intrusion detection using word embeddings. 2017 IEEE International Conference on Big Data (Big Data). https://doi.org/10.1109/bigdata.2017.8258516

Min, E., Long, J., Liu, Q., Cui, J., & Chen, W. (2018). TR-IDS: Anomaly-Based Intrusion Detection through Text-Convolutional Neural Network and Random Forest. Security and Communication Networks, 2018, 1–9. https://doi.org/10.1155/2018/4943509

Cui, J., Long, J., Min, E., & Mao, Y. (2018). WEDL-NIDS: Improving Network Intrusion Detection Using Word Embedding-Based Deep Learning Method. Modeling Decisions for Artificial Intelligence, 283–295. https://doi.org/10.1007/978-3-030-00202-2_23

Ahlawat, S., & Choudhary, A. (2020). Hybrid CNN-SVM Classifier for Handwritten Digit Recognition. Procedia Computer Science, 167, 2554–2560. https://doi.org/10.1016/j.procs.2020.03.309

Jiao, W., Hao, X., & Qin, C. (2021). The Image Classification Method with CNN-XGBoost Model Based on Adaptive Particle Swarm Optimization. Information, 12(4), 156. https://doi.org/10.3390/info12040156