Deep Learning for Network Intrusion Detection System

Michael Grossberg, Temidayo Akinyemi, Pushpen Bikash Goala, Ivan Miller

Abstract

Network intrusion detection remains an important part of securing computer networks against threats and attacks. With the increasing complexity and volume of network attacks, signature-based intrusion detection is inefficient in detecting and responding to these attacks, especially novel ones like zero-day attacks. Also, many anomaly-based intrusion detection systems (IDS) generate lots of false positives, which ultimately cause alert fatigue on the part of security analysts when analyzing traffic data for indicators of compromise (IoC). This has led to the exploration of machine learning (ML) and deep learning (DL) approaches, with high accuracy and a very low rate of false positives, for network intrusion detection.

In our study, we performed experiments on the UNSW-NB15 traffic dataset to create intrusion detection models for both binary and multiclass classification. Our approach consists of three stages: data preprocessing and feature engineering, baseline machine learning methods for which we selected logistic regression, random forest and k-nearest neighbors, and deep learning methods. The experimental results show that the task of identifying an event of an attack happening could be solved by classical machine learning tools, achieving an accuracy over 99% without a need to employ neural networks. However, the problem of correctly identifying a specific type of attack has proven to be a lot more challenging and the highest balanced accuracy achieved with the deep learning classifier has only reached 77.24%.

1. Introduction

Cybersecurity remains an ongoing challenge for businesses, companies, and government institutions, regarding the identification of malicious network traffic and processes running on computers/servers. Attacks like zero-day are one of the most devastating attack types because they do not have any known signature. The term “zero-day” comes from the fact that these kinds of attacks are carried out against vulnerabilities that have not been discovered by the security community. This means it has been “zero number of days” since such a vulnerability was discovered, giving signature-based IDS no chance at detecting them. An IDS classified as signature-based means it can only detect known attack patterns stored in its database. Today, most cybersecurity analysts look for known attack signatures to detect and respond to cyber threats, but with the large volume of traffic and the speed at which these traffic data and system processes are generated, analysts tend to be overwhelmed and are unable to quickly detect these

threats. Typically, this data comes from different security devices like firewalls, privilege management tools, intrusion detection systems, security event and management tools (SIEM). And a lot of time, many of these devices generate false positives which eventually tend to cause alert fatigue on the part of security analysts, ultimately leading to ignoring or missing important alerts. Hence, being able to quickly identify attack patterns requires some machine learning algorithm with predictive capabilities to classify and identify good or bad network traffic/system calls. This will enable quicker response to security incidents and significantly less amount of manual work in detecting the threats.

Our problem formulation is two fold: first, a binary classification that differentiates between a normal traffic from an attack, and secondly, a multiclass classification that identifies a specific attack type. Hence, our approach was to a) first develop a binary classifier to reliably detect an attack/intrusion on the network with high accuracy and low false positives, and b) go deeper into multiclass classification so that any detected attack could also be identified regarding its specific attack type/category. In this project we worked with the UNSW-NB15 network traffic dataset [1]. It is a publicly available dataset that was generated using a hybrid of real network traffic and synthetic data on 9 different types of attacks: Fuzzers, Analysis, Backdoors, Denial of Service (DoS), Exploits, Generic, Reconnaissance, Shellcode, and Worms. This dataset has a total of 2,540,043 records with 49 features including the class label and attack category/type.

We have categorized our experiments into three distinct stages: data preprocessing and feature engineering, baseline machine learning methods, and deep learning methods. Part of our data preprocessing required handling missing values, dealing with class imbalance and standardizing the data before feeding it to our models. We also created machine learning baseline models (Logistic Regression, Random Forest, K-Nearest Neighbors) to serve as a benchmark and foundation for building and evaluating our deep learning models (Deep Sequential Neural Network and Variational Autoencoder).

2. Background and Related Work

Machine learning and deep learning techniques have been widely embraced for network intrusion detection. This is largely because of the ineffectiveness of rules-based traditional intrusion detection systems against new/novel threats. Two of the most prominent datasets used in network intrusion detection research are the KDD99 dataset and the UNSW-NB15 dataset.

The KDD99 dataset (Lippmann et al., 2000) [2] is a widely used benchmark dataset containing network traffic with various types of attack. It has been instrumental in the development and evaluation of intrusion detection systems. Al-Daweri et al. [3] did an analysis of the KDD99 dataset and highlighted its limitations, like the presence of duplicate and irrelevant records, unrealistic attack scenarios and imbalance class distribution.

Nour et al. [1] proposed the UNSW-NB15 dataset, which contains labeled network traffic captured in a controlled environment, simulating various attack types and normal network traffic. The UNSW-NB15 dataset is specifically designed for network intrusion detection systems, and it provides a realistic representation of network traffic. Nour et al. gave a detailed description of the dataset, including how it was collected, the evaluation metrics and feature collection process. The UNSW-NB15 dataset has been used to train many machine learning algorithms for network intrusion detection. Hodo et al. [4] proposed an ensemble-based intrusion detection system using Random Forest, Naïve-Bayes, and Support Vector Machines (SVM) classifiers. They were able to demonstrate improved intrusion detection accuracy compared to signature-based detection. Nguyen et al. [5] explored transfer learning to improve the generalization of network intrusion detection models, by investigating with the UNSW-NB15 dataset. In their work, a deep learning autoencoder was pretrained on a very large unlabeled dataset and then fine-tuned on the labeled UNSW-NB15 dataset. They were able to obtain better performance in detecting novel attacks and reducing false positives.Deep neural nets (DNNs) have also shown promising results in network intrusion detection. Gao et al. [6] leveraged the successive representation learning capabilities of DNNs to propose a deep belief network-based intrusion detection system. Their approach resulted in high performance in detecting novel attacks compared to traditional machine learning methods.

3. Data

3.1. The Dataset

In this project we were working with the UNSW-NB15 dataset created by the Cyber Range Lab of UNSW Canberra and presented by N. Moustafa and J. Slay in 2015 [1]. The data was generated using a hybrid of real network packets representing normal network activity and synthetic data on 9 types of attacks: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode and Worms. The dataset has a total of 2,540,043 records with 49 features including the class label and is freely available for download [7].

3.2. Data Cleaning and Preprocessing

Originally the dataset was split into 4 parts that had to be combined for data cleaning and preprocessing. The resulting file contained over 2.5M records with a total size of 559Mb. During the exploratory data analysis we learned that several columns had large amounts of missing values (`ct_flw_http_mthd` (53%), `is_ftp_login` (56%), and `ct_ftp_cmd` (56%)) making it impossible to impute them in a meaningful way, so the entire columns were removed from the dataset. Names of the attack categories stored in the `attack_cat` column which contained a name of each of the attack categories had to be brought to a uniform state of 10 labels for multi-class classification by cleaning leading and trailing spaces and fixing spelling differences. Additionally, labels for all non-attack classes were missing in this column and had to

be imputed with the “Normal” label. Lastly, some of the values in columns `sport` and `dsport`, which, according to the description of the features, were supposed to be holding integer values only, contained values in the hexadecimal format and had to be converted.

As the last step of initial preprocessing we one-hot-encoded all categorical variables and set aside 15% of the data as a holdout set for testing. Total number of features exploded to 207 after one-hot-encoding and pushed the size of the training set alone to over 1Gb. That significantly increased the hardware requirements for further work, since only a machine with a relatively high-RAM was capable of loading the dataset in memory.

4. Methods

4.1. Feature Selection

To relieve the strain on the hardware and ensure that we were using the most meaningful features we ran three separate methods:

1) First, we computed a pairwise correlation of features (Pearson) and identified those with the correlation of 85% or higher.
2) We then calculated variance inflation factor (VIF) for each feature and marked the features that had VIF values higher than 10.
3) As the third method we fit a random forest classifier with standard parameters to the training set and identified the features that were responsible for the top 95% of importance (see Fig.1 and appendix 1 for reference).

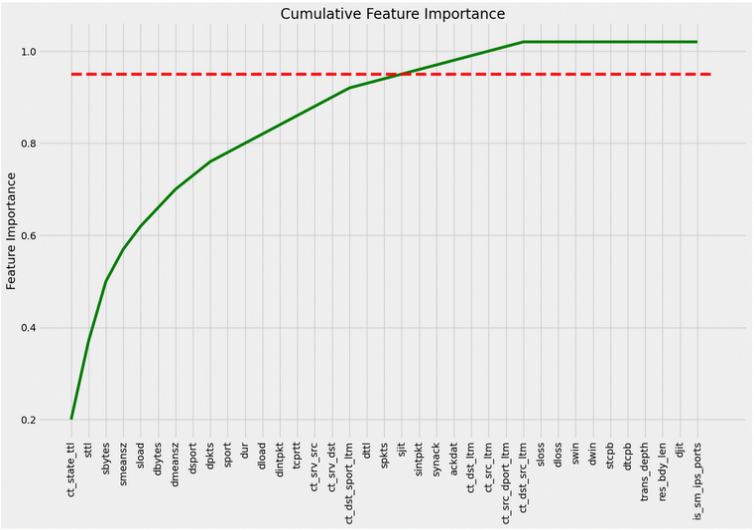


Fig.1. Cumulative feature importance produced by a random forest classifier initialized with default parameters. Green line represents cumulative feature importance, dotted red line marks 95% of the total importance.

We then cross-referenced the results and identified the features that carried the highest importance and weren’t correlated between each other, selecting 22 of them for future modeling.

4.2. Class Imbalance. Undersampling for Binary Classification

The distribution of data between attack classes in the original training set showed significant imbalance (see Fig.2) between one normal (87.35% of data) and nine attack classes (12.65%), so we chose two different strategies to address this imbalance for a binary and multi-class classification. To prepare the data for binary classification (attack vs normal) the downsampled

the majority class (normal) to match the total number of datapoints available across 9 attack classes. Such an approach resulted in a perfect 50/50 balance between classes with each class containing 273,090 records.

4.3. Class Imbalance. SMOTE and Undersampling for Multi-Class Classification

To even out the balance between the attack classes and prepare the data for the multi-class classification we decided to use the Synthetic Minority Oversampling Technique (SMOTE) [8] to bring the number of data points in each of the attack classes to the same level as the highest number of the organic data points available among all attack classes in the original training set ("Generic" attack class containing 183,159 of the original data points, see Fig.2).

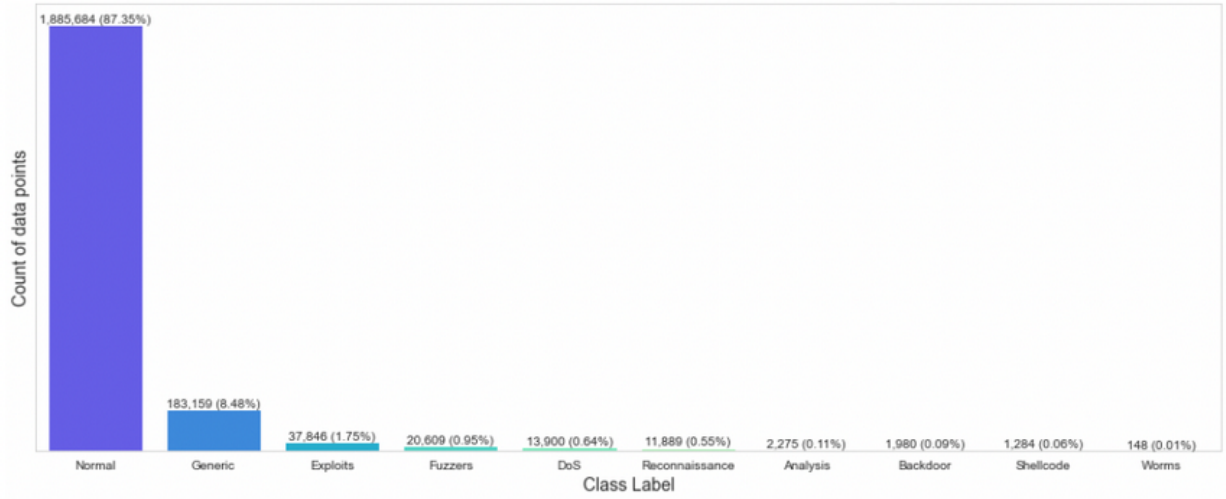


Fig.2. Visualization of the distribution of data between attack classes in the training set shows significant imbalance between one normal (87.35% of data) and nine attack classes (12.65%).

SMOTE works by selecting a pair of examples from the minority class that are close to each other in the feature space and creating a synthetic example at a random point along the line drawn between those examples. This approach combined with the subsequent downsampling of the majority class ("Normal") to the same level allowed us to create a perfect balance between all 10 attack classes (see Appendix 2).

5. Evaluation

5.1. Baseline Methods

We selected random forest, logistic regression, and k-nearest neighbors (KNN) to establish performance baselines and separated the work on binary classification - detection of an event of attack with high probability and multi-class classification - identification of the specific type of the attack.

5.1.1. Binary Classification

All three baseline classifiers showed high results with classification accuracy of at least 98% even when instantiated with default parameters (random forest - 99.32%; logistic regression - 98.93%; K-nearest neighbors (KNN) - 99.19%). We also created a simple neural network with sequential architecture and 2 hidden layers, which achieved a 99.30% accuracy. Additionally, precision, recall, and f1-score of all models also reached 99%. These results allowed us to conclude that the problem of identifying an event of an attack happening as well as reducing the number of false positives has largely been solved by classical machine learning tools without a need to employ neural networks. reasonably high performance.

5.1.2. Multi-class Classification

In order to establish the very first baseline for the multi-class classification we created a `DummyClassifier`. It achieved an accuracy of 10% which we then used throughout the work on the project as the lowest possible performance benchmark. While working on the baseline classifiers for the multi-class case we employed the following framework for cross-validation and hyperparameter tuning: to speed up the process, instead of working with the full (imbalanced) training set we started with subsampling 15-20% of the data in order to run 5-fold cross-validation. We then applied a combination of SMOTE and undersampling only on training folds to avoid leakage of data between synthetically generated points in the training and validation folds. Next, we performed a search over dozens of combinations of parameters for each classifier to determine a set of parameters that yielded the highest result for each model. Once grid search and cross-validation were complete, each of the classifiers was trained with the best parameters on a full imbalanced dataset to record final performance metrics.


```
Accuracy:0.97
Balanced accuracy:0.73
Precision:0.98
Recall:0.97
F1-score:0.97

Classification Report:
                precision    recall  f1-score   support

       Normal       1.00      0.99      0.99    332768
      Generic       1.00      0.97      0.99     32322
     Exploits       0.82      0.44      0.58      6679
     Fuzzers        0.42      0.87      0.56      3637
         DoS        0.34      0.53      0.41      2453
Reconnaissance      0.81      0.82      0.81      2098
     Analysis       0.12      0.25      0.16       402
     Backdoor       0.08      0.48      0.13       349
    Shellcode       0.49      0.98      0.66       227
        Worms       0.04      0.92      0.07        26

     accuracy                           0.97    380961
    macro avg       0.51      0.73      0.54    380961
 weighted avg       0.98      0.97      0.97    380961
```

Fig.3 Confusion matrix and the classification report for the baseline random forest classifier that achieved 72.68% balanced accuracy and 97% F1-score on the test set.

A. Random Forest

Employing the above mentioned technique we performed the hyperparameter tuning with 5-fold cross-validation and searched over a grid of 80 combinations of parameters for the random forest

classifier. The process took over 32 hours to complete and identified the following set of parameters:

`n_estimators` (number of trees): 300;

`min_samples_split` (minimum number of samples to split a node): 3;

`max_depth` (maximum number of nodes): 10;

`min_samples_leaf` (minimum number of samples on each side after splitting a node): 2.

We then used the best parameters to train a new random forest classifier on a full imbalanced dataset and achieved balanced accuracy (average recall across all classes) of 72.68% and an F1-score of 97% (See Fig.3). Despite showing the absolute best performance we managed to achieve with any of the baseline classifiers, random forest still struggled to correctly identify some of the classes: "Analysis" attack class was classified correctly only 25% of the time while having the highest misclassification rates with 42% of the times being recognized by the model as "DoS" and 32% as backdoor. Exploits and Backdoor are two other classes that had less than a 50% recall (44% and 48% respectively).

B. Logistic Regression

Another of our baseline methods is logistic regression which we used for multiclass classification of our network traffic data. We trained the logistic regression model on the training set and evaluated the model's performance on the test set. To gain insights about the performance, we constructed a confusion matrix (shown in Fig. 4) and provided a classification report to show metrics such as accuracy, precision, recall and F1-score for each attack class.

Classification Report:

	precision	recall	f1-score	support
Normal	1.00	0.99	0.99	221846
Generic	1.00	0.98	0.99	21548
Exploits	0.77	0.43	0.55	4452
Fuzzers	0.43	0.80	0.56	2425
DoS	0.33	0.66	0.44	1635
Reconnaissance	0.65	0.68	0.66	1399
Analysis	0.10	0.28	0.15	268
Backdoor	0.07	0.18	0.10	233
Shellcode	0.54	0.95	0.69	151
Worms	0.02	0.82	0.03	17
accuracy			0.97	253974
macro avg	0.49	0.68	0.52	253974
weighted avg	0.98	0.97	0.97	253974

Dataset	Model	F1-score	Accuracy	Balanced Accuracy	Precision	Recall
Test	Logistic Regression	0.97%	0.97%	0.68%	0.98%	0.97%

Fig. 4 Confusion matrix for logistic regression (multiclass) on test set and classification report

Our results show a balanced accuracy of 68% and recall of 97%. However, considering individual attack class/type, the model gave low performance scores for attack classes like

Exploits, Analysis, and Backdoor with recall scores of 43%, 28% and 18% respectively (see Fig. 4)

C. KNN with Autoencoder:

To classify the attack categories which is a Multi-class problem we also use KNN. In our previous classifiers (Random Forest, Logistic Regression) we remove a large number of features from the dataset. Though the models trained and tested with a small number of important features from the dataset generate good predictions, make the entire dataset less significant. We also need to see how our system performs while keeping all the features in the dataset as significant. To do this experiment we come up with another baseline method KNN. In KNN we consider all the features in the dataset for training, testing, and validation. However, the problem with KNN is that it is very slow, and running it on a dataset with all features will make it even slower. To overcome this problem we compress the entire input features to 20 using a deep learning Autoencoder model. Figure 5 shows the structure of our Autoencoder model.

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 207)]             0

 dense (Dense)               (None, 414)               86112

 batch_normalization_6 (Batc  (None, 414)              1656
 hNormalization)

 leaky_re_lu (LeakyReLU)     (None, 414)               0

 dense_1 (Dense)             (None, 207)               85905

 batch_normalization_7 (Batc  (None, 207)              828
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 207)               0

 dense_2 (Dense)             (None, 20)                4160

 dense_3 (Dense)             (None, 207)               4347

 batch_normalization_8 (Batc  (None, 207)              828
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 207)               0

 dense_4 (Dense)             (None, 414)               86112

 batch_normalization_9 (Batc  (None, 414)              1656
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 414)               0

 dense_5 (Dense)             (None, 207)               85905

=================================================================
Total params: 357,509
Trainable params: 355,025
Non-trainable params: 2,484
_________________________________________________________________
None
```

Fig.5. Autoencoder model summary

It has 355,025 trainable parameters and 2484 non-trainable parameters. We use two dense layers in the encoder and decoder. We use Leaky ReLU as an activation function to deal with vanishing gradients. In the output layer, we use a linear activation function. To generate a good prediction with KNN it is important to choose the right value of n_neighbours. To choose the right value for n_neighbours we perform 5-fold cross-validation with the encoded data generated by the Autoencoder. While performing cross-validation we perform SMOTE only on the train folds to deal with the class imbalance and keep the validation folds as it is. This ensures no data leakage. While performing cross-validation we use balanced accuracy as our validation metric. After performing cross-validation we perform GridSearch to get the best n_neighbours. We find the best value of n_neighbours=95 with balanced accuracy = 0.573. Then we train our KNN model with class balance data generated by SMOTE and test the model with class imbalance data or original data. Our KNN model generates 0.63 balanced accuracy which is close to the validation accuracy metric. This ensures that there is no data leakage. Further, our KNN model generates Accuracy=0.96, Precision=0.98,

Recall=0.96, and F1-score=0.97. The accuracy was low for Backdoor and Analysis attacks. Though the test accuracy is high, the balanced accuracy is that good.

Classification Report:

	precision	recall	f1-score	support
Normal	1.00	0.98	0.99	221846
Generic	1.00	0.97	0.99	21548
Exploits	0.70	0.40	0.51	4452
Fuzzers	0.39	0.75	0.51	2425
DoS	0.31	0.55	0.40	1635
Reconnaissance	0.54	0.62	0.58	1399
Analysis	0.07	0.29	0.11	268
Backdoor	0.03	0.07	0.04	233
Shellcode	0.42	0.89	0.57	151
Worms	0.02	0.82	0.03	17
accuracy			0.96	253974
macro avg	0.45	0.63	0.47	253974
weighted avg	0.98	0.96	0.97	253974

Fig.6 Confusion matrix and classification report for KNN(Train and test with all features). The model achieves balanced accuracy of 63%.

Figure 6 shows the confusion matrix and classification report generated by our model. The confusion matrix represents the balanced accuracy of different types of attack categories. From the classification report it is obvious that the model performs very low for the attack categories.

5.1.3. Model Interpretability with LIME

LIME (Local Interpretable Model-Agnostic Explanations) is a machine learning interpretability technique. LIME works by creating a simple, interpretable model that approximates the predictions of a black-box model. This simple model can then be used to explain why the black-box model made a particular prediction. LIME can be used to explain the predictions of any machine learning model, regardless of the model's complexity. While predicting the model LIME does not make any assumptions about the underlying distribution of the data. Further, the prediction of the model through LIME is based on local technique, it only explains the predictions of the model for a small subset of the data, which is helpful for identifying the features that are most important for making a particular prediction. We use LIME to interpret our baseline Random forest model trained with all the features in the dataset. We initiate LIME with our trained Randomforest model then pick a test sample that is labeled as attack category Generic and pass this sample to LIME to make the prediction and generate features importance. The figure shows the result generated by LIME. LIME predicts the sample as Generic based on its local prediction technique. The prediction of LIME is the same as the model prediction. Apart from that LIME also generates the most important features for predicting whether the attack is Generic or not. Figure 7 shows the top 6 important features that contribute to predicting whether the attack is Generic or not.

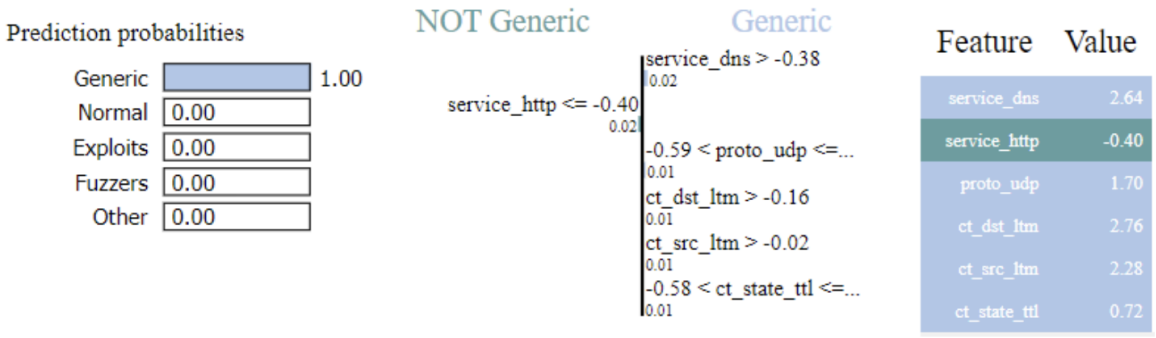


Fig.7 Interpreting predictions of the model with LIME

The features marked with sky color (service_dns, proto_udp, ct_dst_ltm, ct_src_ltm, and ct_state_ttl) contribute most to the prediction of attack category Generic and the feature marked with light green color (service_http) contributes to predicting that the attack category of the test sample is not Generic. Further, LIME also generates the value range for each feature that contributes to making the prediction. To predict that the attack category of the test sample is Generic the value range of the feature ct_state_ttl is between -0.58 to 0.01(inclusive).

5.2. Deep Learning

5.2.1. Sequential model

We utilized TensorBoard to monitor live performance of deep learning models with various designs and architectures to determine the optimal combination of layers (see Appendix 6a, 6b) and then took advantage of KerasTuner for hyperparameter optimization to identify a set of parameters that yielded the highest result (Fig.8):

```
Activation: ReLU
Optimizer: Adam
Learning rate: 0.001
Batch size: 1,024
```

Fig.8 Confusion matrix for the neural network with sequential architecture. With total balanced accuracy of 70.98% the only significant improvement over the baseline showed by this model was with DoS attack class, 72% (+19% accuracy), while simultaneously there was a number of classes that showed noticeable declines ("Backdoor" -12%, "Worms" -19%).

During that stage we also employed a method similar to the one described in 5.1.2 making sure that there was no leakage of data between the validation sets and data points synthetically generated by SMOTE in the training set. The best balanced accuracy achieved by the model with these parameters stayed 1.7% below the baseline (70.98%) with DoS

attack class showing the only significant improvement over the baseline 72% (+19% accuracy), while simultaneously there was a number of classes that showed noticeable declines ("Backdoor" -12%, "Worms" -19%).

5.2.2. One vs. All Ensemble

Facing a challenge of pushing the performance to match the baseline, let alone beat it, with the neural network designed for a multi-class classification, we decided to split the dataset into multiple classification problems and fit a binary classifier to each of them following a one-vs-all strategy. To achieve that we implemented the following framework:

1. Calculate class weights in the original training set
2. Select one target class out of 10 available ones and remap it as a positive class for binary classification (1)
3. Rebalance all classes using a combination of SMOTE and undersampling with a goal of having a ratio between the new target class from #2 being equal to or very close to the ratio of all other classes.
4. "Collapse" all remaining classes into one large negative class for binary classification
5. Calculate resulting class weight to pass to the model
6. Fit the model on the remapped and rebalanced binary training set

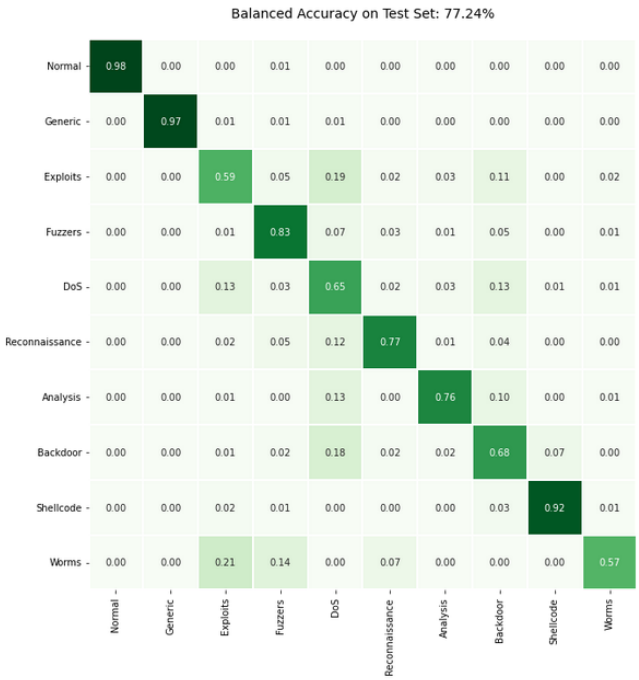


Fig.9 Confusion matrix for the voting classifier combining 10 individual binary classification models trained to identify one type of attack each according to one-vs-all strategy. That approach resulted in 4.5% increase in balance accuracy over the top baseline classifier (random forest). As you could see in the plot, that model has significant improvements in its ability to identify "analysis" (+51%) and "backdoor" (+32%) attack types. These gains, however, come at the expense of decrease in performance for other classes: "DoS" (-7%), "shellcode" (-5%), "worms" (-16%).

We then repeated that process 10 times to create a class-specific binary classifier for each of the attack classes and one non-attack (normal) class and combined them into a voting classifier. The voting classifier was then able to beat the baseline by 4.56% and achieve a balanced accuracy of 77.24% on the train set. As you could see in the plot on Fig.9, that model has significant improvements in its ability to identify "analysis" (+51%) and "backdoor" (+32%) attack types. These gains, however, come at the expense of decrease in performance for other classes: "DoS" (-7%), "shellcode" (-5%), "worms" (-16%).

5.2.3. Variational Autoencoder and KNN

We use a variational Autoencoder for Multi-class classification problems. Here again, we consider all the features in the dataset. The below figure shows the structure of our Variational Autoencoder model.

The model has 12,611 trainable parameters and 464 non-trainable parameters. We compress all input features to 20 and use the z_mean and z_log_var layers for parameter distribution. These two layers help to generate input-like data from the latent space's compressed data. We then use the encoder to encode or compress the class balance train data generated by SMOTE and also to encode or compress class imbalance test data. After performing encoding the train and test dataset has only 20 features. We then again pick KNN for the training and testing. Here again, we set the n_neighbours = 95. Then we train our model with the encoded data and also test our model with the encoded test data. The model generates Balanced accuracy=0.634 which is almost the same but slightly better than the baseline KNN trained with uncompressed or unencoded (all features) data. Further, the model generates Accuracy=0.95, Precision=0.97, Recall=0.95, and F1-score=0.96. Below is the confusion matrix generated by our model. The confusion matrix is also the same as the baseline KNN.

```
Model: "model_2"
_________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================================================
 input_2 (InputLayer)        [(None, 207)]             0         []

 batch_normalization_4 (BatchNo  (None, 207)           828       ['input_2[0][0]']
 rmalization)

 encoder_hidden (Dense)      (None, 25)                5200      ['batch_normalization_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 25)            100       ['encoder_hidden[0][0]']
 rmalization)

 z_mean (Dense)              (None, 20)                520       ['batch_normalization_5[0][0]']

 z_log_var (Dense)           (None, 20)                520       ['batch_normalization_5[0][0]']

 z_sampled (Lambda)          (None, 20)                0         ['z_mean[0][0]',
                                                                  'z_log_var[0][0]']

 decoder_hidden (Dense)      (None, 25)                525       ['z_sampled[0][0]']

 decoded_mean (Dense)        (None, 207)               5382      ['decoder_hidden[0][0]']

=================================================================================================
Total params: 13,075
Trainable params: 12,611
Non-trainable params: 464
_________________________________________________________________________________________________
```

Fig.10. Model architecture for the variational autoencoder

Confusion matrix in Fig.11 again represents balanced accuracy. The balanced accuracy for Backdoor is increased here from 0.07 (base line) to 0.09. The classification report also is almost the same as baseline KNN. The most important thing to be noticed here is that, we train the baseline KNN with all features; however, here we train KNN with reduced encoded features generated by the encoder and again we achieve almost the same accuracy.

```
Classification Report:
                precision    recall  f1-score   support

        Normal       1.00      0.97      0.99    221846
       Generic       0.99      0.97      0.98     21548
      Exploits       0.38      0.41      0.39      4452
       Fuzzers       0.35      0.66      0.45      2425
           DoS       0.30      0.54      0.38      1635
Reconnaissance       0.52      0.66      0.58      1399
      Analysis       0.07      0.27      0.11       268
      Backdoor       0.03      0.09      0.05       233
     Shellcode       0.30      0.89      0.44       151
         Worms       0.01      0.88      0.03        17

      accuracy                           0.95    253974
     macro avg       0.40      0.63      0.44    253974
  weighted avg       0.97      0.95      0.96    253974
```

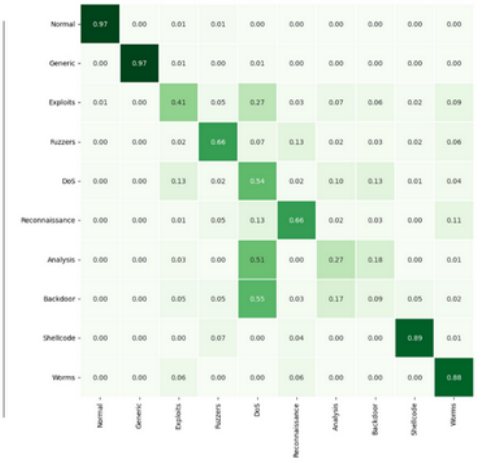


Fig.11 Confusion matrix and the classification report for the KNN (train and test with the encoded features using variational autoencoder (VAE)). The model achieved balanced accuracy 63.40%.

6. Conclusion

As we learned in the second part of the project, detection of the attack event with any of the binary classifiers we tested showed reasonably high performance: 98-99% accuracy with precision, recall, and f1-score of all models also reaching 99% (see Appendices 3-6 for details). So not only did binary classification not pose a particular challenge in general, it also did not require any kind of special work with upsampling or creation of the synthetic data to balance the classes.

However, for the multi-class classification we had to develop a special framework for grid-search and cross-validation carefully separating the pipelines between training and validation folds to avoid data leakage and make sure that test folds only were going through the application of SMOTE and undersampling to properly balance the attack classes. This approach allowed us to fine-tune hyperparameters of each model that yielded results significantly higher than the initial benchmark set by the DummyClassifier. Figure 12 summarizes balanced accuracy for multi-class classification on the test set achieved by the models developed during the work on this project.

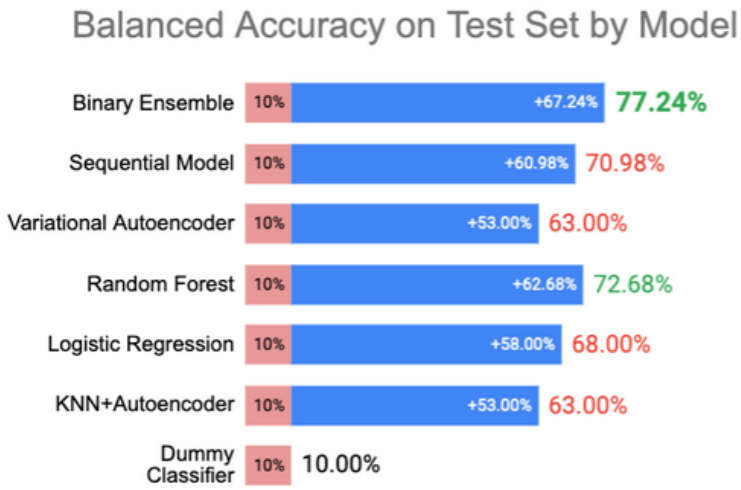


Fig.12 Visualization of balanced accuracy by model and comparison to the baseline(s).

Random Forest showed the highest result among baseline classifiers and achieved 72.68% overall balanced accuracy. However, the model struggled with correctly identifying some of the attack types. We weren't successful in our attempt to beat the baseline with the sequential model and autoencoders. An ensemble of 10 binary classifiers each trained to predict just one label was the only solution that improved the baseline result and showed better capabilities in recognizing attack classes that the rest of the models struggled the most (e.g. "analysis", "backdoor").

We believe that further improvements to the developed network intrusion detection system could be made by focusing on interpreting model's predictions for each type of attack in order to identify a set of features that are being frequently identified as predictors of attacks. That work could be helpful in improving precision and recall for the identification of attack types the model struggles the most with ("analysis", "backdoor", "worms" etc). Additionally, further

improvements in model's ability to generalize could be made, as the current performance was likely limited due to the following factors:

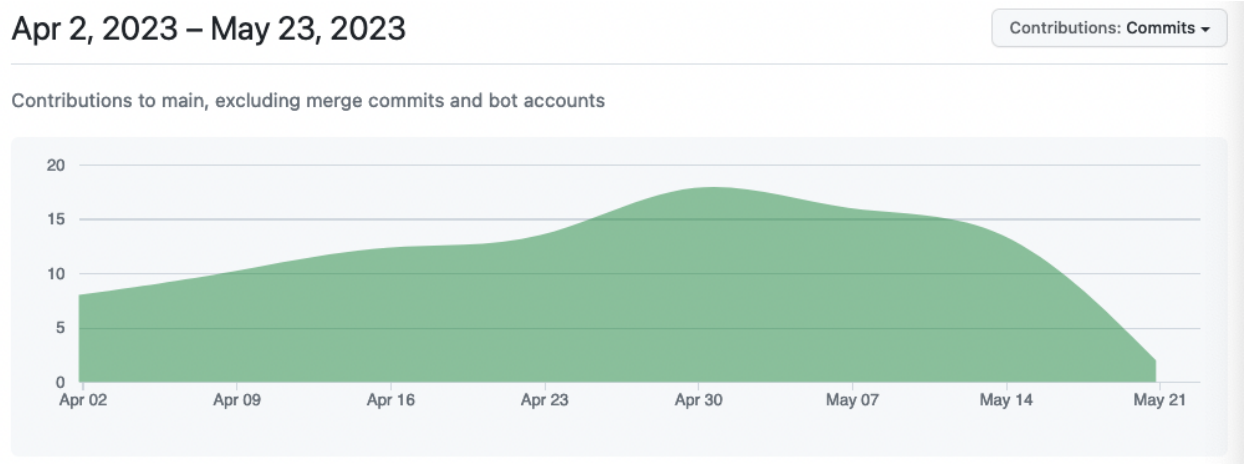
1. Severe imbalance between attack classes in the original dataset
2. Presence of synthetic data in the original dataset

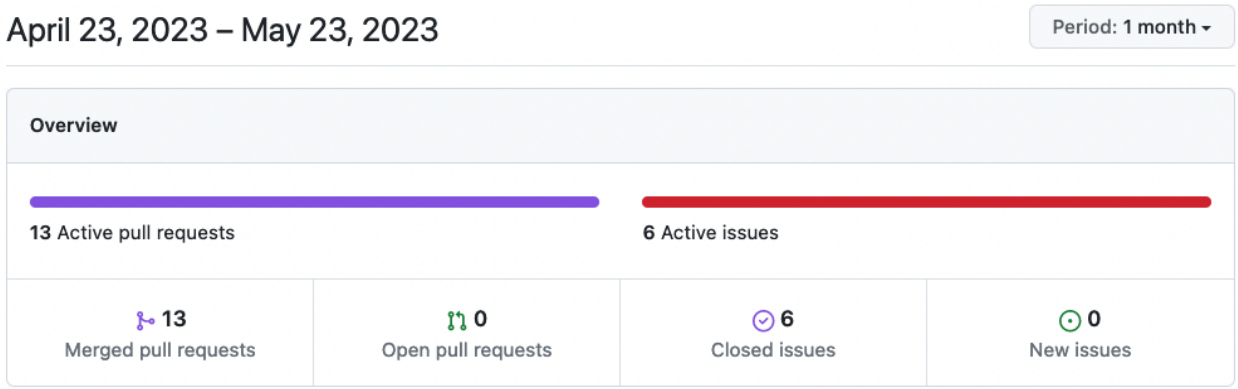
As soon as the team is satisfied with the interpretation of results with LIME/SHAP, bringing in more data and focusing on the features that have proven to be good predictors would be a logical next step for the development of this project.

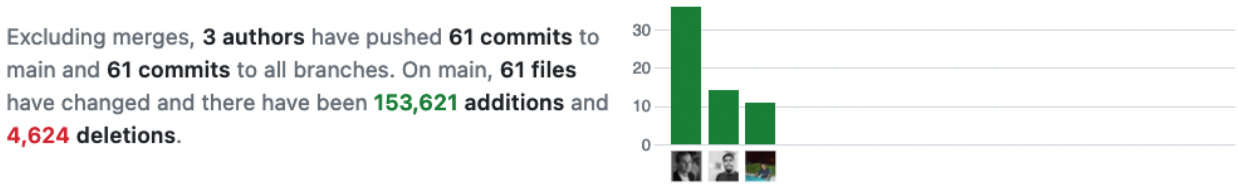
7. Attribution

7.1. Project Repository:

https://github.com/DataScienceAndEngineering/deep-learning-final-project-project-5-dl-for-cybersecurity

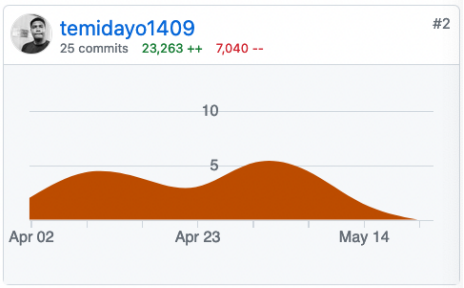






7.2. Individual Contributions

Temidayo Akinyemi: contributed 25 commits with 23,263 lines of code. Worked on several aspects including exploratory data analysis, feature engineering and mostly on logistic regression as the baseline method. LR on both binary and multiclass.



Pushpen Bikash Goala: Data exploration and analysis, featuring engineering to understand the dataset. Cross-validation and grid search to find the best neighbouring parameter for baseline KNN. Implementation of autoencoder to compress the dataset used for KNN during cross-validation. Work on baseline KNN for multiclass-classification. Implementation of Variational autoencoder to generate input like features in a compressed form. Implementation of KNN with the encoded features using Variational autoencoder for multiclass-classification. Implementation of LIME for model interpretability. Contribution of 12 commits with 64,711 lines code.

Ivan Miller: Data cleaning and preprocessing (both binary and multi-class cases). Feature selection. Framework for cross-validation and grid search. Baseline binary classifiers. Baseline multi-class random forest. Sequential model. One-vs-All binary classifiers. GitHub project and Wiki. Contributed 57 commits (122,318 lines of code), which took approximately 15-25 hours/week.

8. Bibliography

[1] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 2015, pp. 1-6. https://doi.org/10.1109/MilCIS.2015.7348942

[2] Lippmann, R., Cunningham, R., and Kobsa, A. (2000). “Results of the 1999 DARPA Offline Intrusion Detection Evaluation”. Computer Networks, 34(4), 579-595

[3] Al-Daweri, M.S., Aborizka, M., and Kamal, N.S.M. (2020). “An Analysis of the KDD99 and UNSW-NB15 Datasets for Intrusion Detection System”. Symmetry, 12(10), 1666.

[4] Hodo, E., Bellekens, X., Hamilton, A., and Tachtatzis, C. (2016). “A Hidden Semi-Markov Model-Based IDS for Advanced Persistent Threats in Cloud Infrastructures”. IEEE Transactions on Emerging Topics in Computing, 4(2), 246-259.

[5] Nguyen, T.N., Kim, S.W., and Yoon, Y. (2018). “A Deep Learning Approach for Network Intrusion Detection System”. Security and Communication Networks, 2018, 1-11.

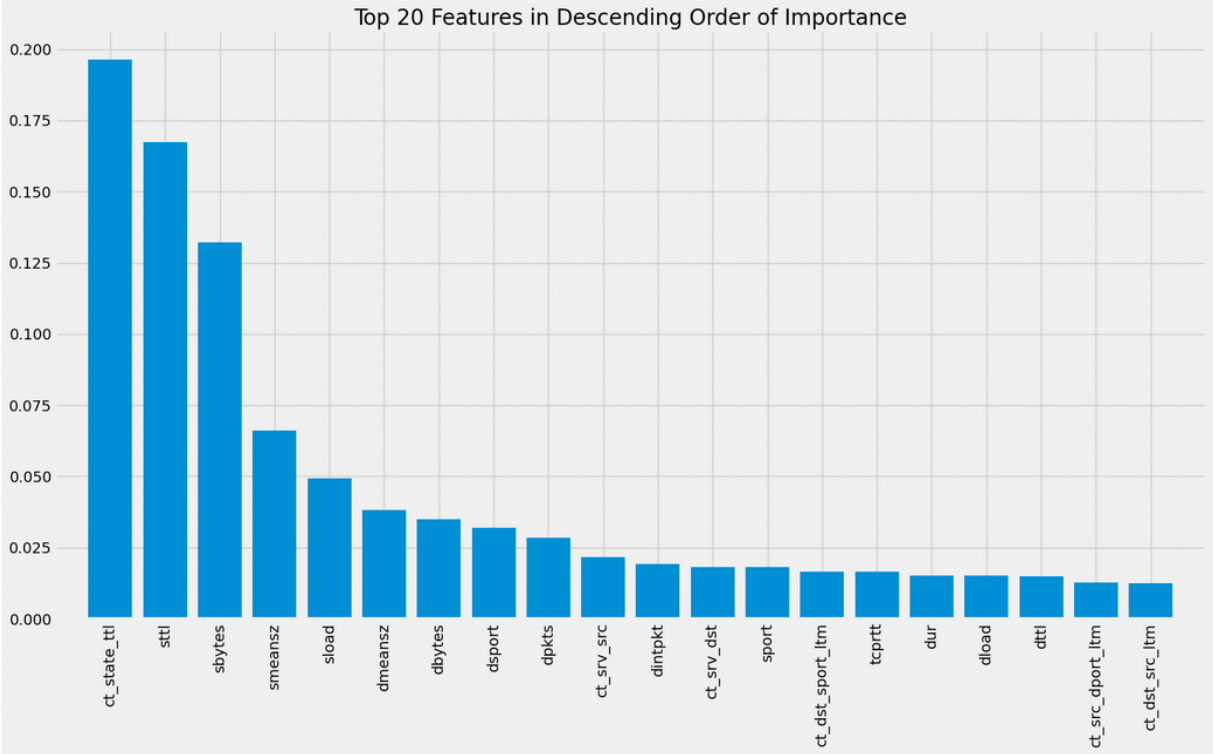
[6] Gao, J., Yu, L., Zhou, Z., and Wu, W. (2018). “Deep Belief Nework-Based Intrusion Detection System Using Optimized Intrusion Features”. IEEE Access, 6, 32482-32490

[7] The UNSW-NB15 Dataset. Intelligent Security Group UNSW Canberra, Australia. https://research.unsw.edu.au/projects/unsw-nb15-dataset

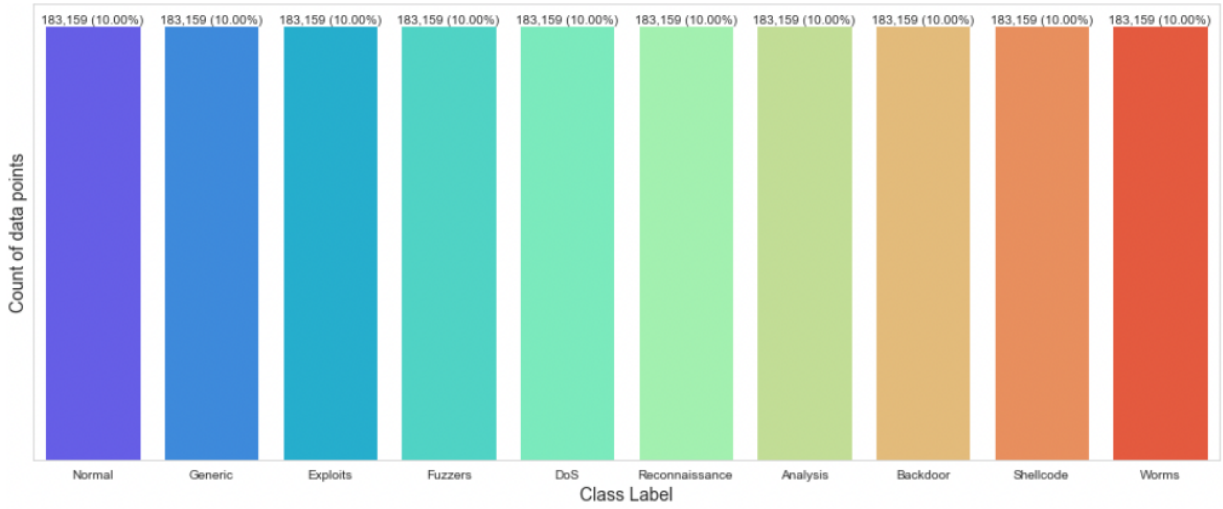
[8] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique”, Journal of Artificial Intelligence Research 16 (2002), 321 - 357 https://doi.org/10.48550/arXiv.1106.1813

9. Appendix

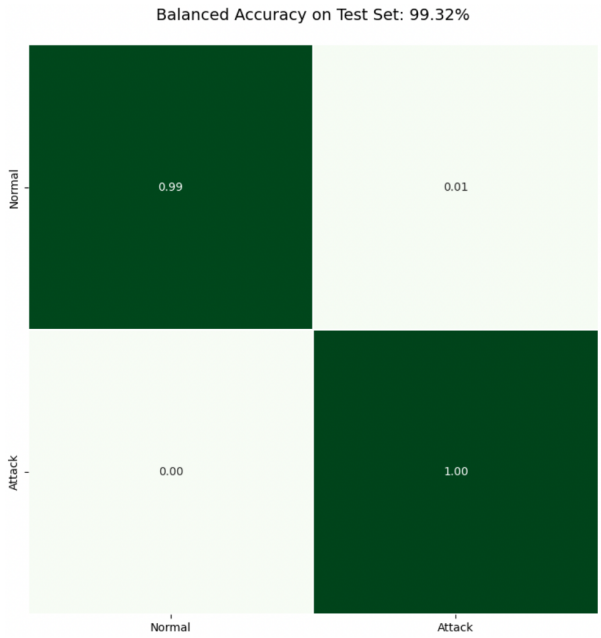
[1] Top 20 features of the original dataset sorted in descending order of importance (produced by a random forest classifier initialized with default parameters):



[2] Visualization of the distribution of data between attack classes in the training set after applying SMOTE to the minority classes and downsampling the majority class to the same level shows perfect balance between classes with each class contributing 10% of the data:

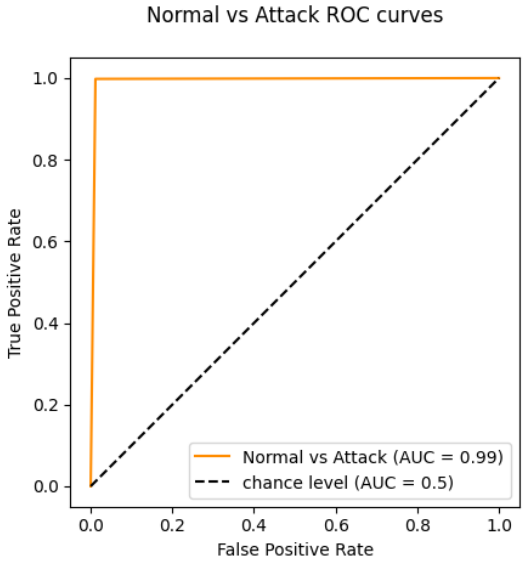


[3] Random Forest

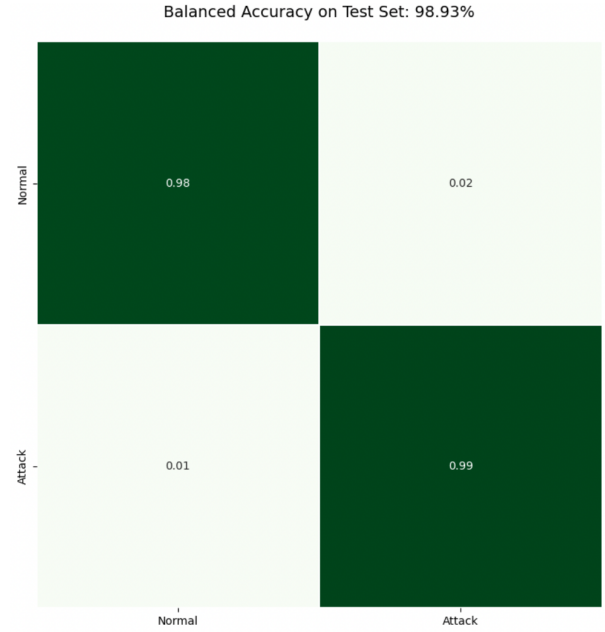



```
Accuracy:0.99
Balanced accuracy:0.99
Precision:0.99
Recall:0.99
F1-score:0.99
```

```
Classification Report:
              precision    recall  f1-score   support

      Normal       1.00      0.99      0.99     48193
      Attack       0.99      1.00      0.99     48193

    accuracy                           0.99     96386
   macro avg       0.99      0.99      0.99     96386
weighted avg       0.99      0.99      0.99     96386
```



[4] Logistic Regression

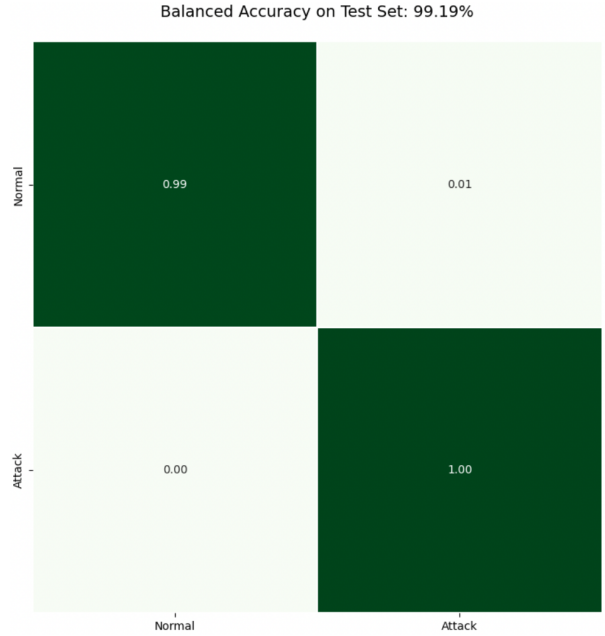


Accuracy:0.99
Balanced accuracy:0.99
Precision:0.99
Recall:0.99
F1-score:0.99

Classification Report:

	precision	recall	f1-score	support
Normal	0.99	0.98	0.99	48193
Attack	0.98	0.99	0.99	48193
accuracy			0.99	96386
macro avg	0.99	0.99	0.99	96386
weighted avg	0.99	0.99	0.99	96386

[5] K-nearest neighbors (KNN)



Accuracy:0.99
Balanced accuracy:0.99
Precision:0.99
Recall:0.99
F1-score:0.99

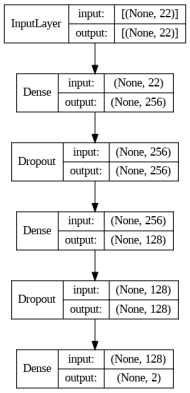
Classification Report:

	precision	recall	f1-score	support
Normal	1.00	0.99	0.99	48193
Attack	0.99	1.00	0.99	48193
accuracy			0.99	96386
macro avg	0.99	0.99	0.99	96386
weighted avg	0.99	0.99	0.99	96386

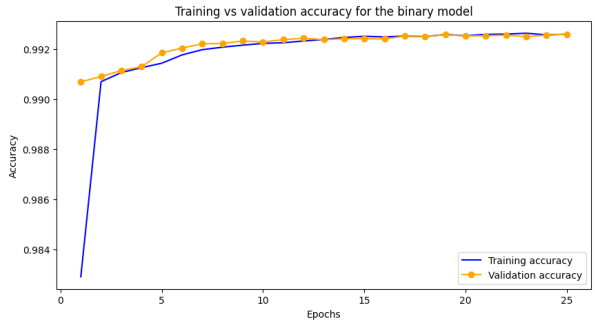
[5] Neural network for binary classification: (a) model summary; (b) model architecture; (c) training and validation accuracy; (d) training and validation loss; (e) confusion matrix; (f) classification report.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 22)]              0

dense (Dense)                (None, 256)               5888

dropout (Dropout)            (None, 256)               0

dense_1 (Dense)              (None, 128)               32896

dropout_1 (Dropout)          (None, 128)               0

dense_2 (Dense)              (None, 2)                 258

=================================================================
Total params: 39,042
Trainable params: 39,042
Non-trainable params: 0
_________________________________________________________________
```

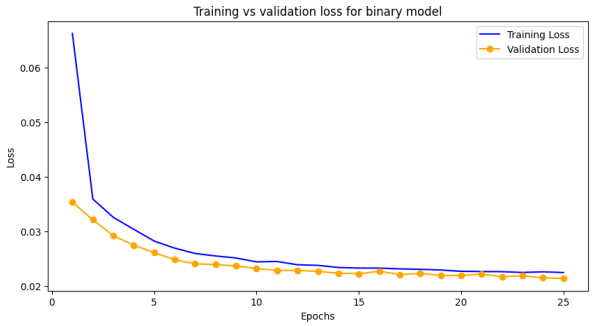
(a)



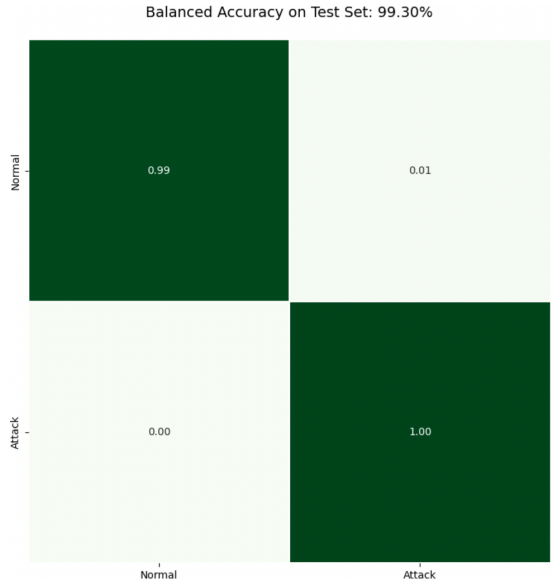
(b)



(c)



(d)



(e)

```
Accuracy:0.99
Balanced accuracy:0.99
Precision:0.99
Recall:0.99
F1-score:0.99


Classification Report:
              precision    recall  f1-score   support

      Normal       1.00      0.99      0.99     48193
      Attack       0.99      1.00      0.99     48193

    accuracy                           0.99     96386
   macro avg       0.99      0.99      0.99     96386
weighted avg       0.99      0.99      0.99     96386
```

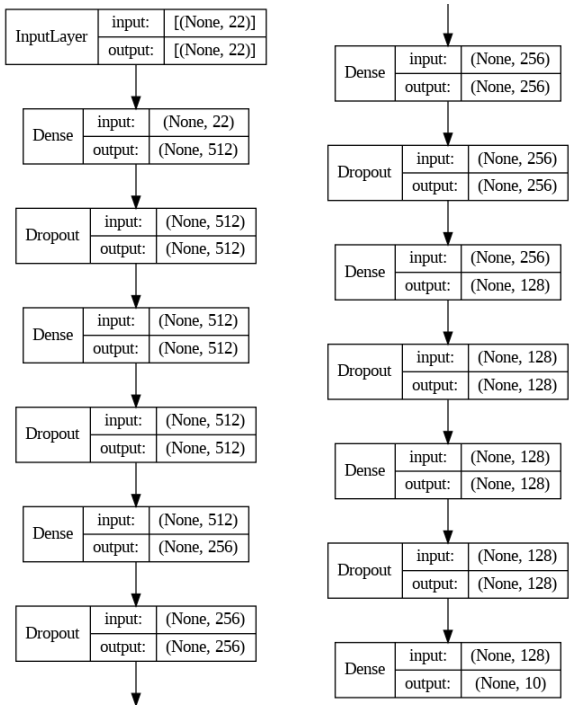
(f)

[6] Simple sequential model for multi-class classification: (a) model summary; (b) model architecture; (c) confusion matrix; (d) classification report.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 22)]	0
dense (Dense)	(None, 512)	11776
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 256)	65792
dropout_3 (Dropout)	(None, 256)	0
dense_4 (Dense)	(None, 128)	32896
dropout_4 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 128)	16512
dropout_5 (Dropout)	(None, 128)	0
dense_6 (Dense)	(None, 10)	1290

Total params: 522,250
Trainable params: 522,250
Non-trainable params: 0

(a)



(b)

(c)

Accuracy:0.97
Balanced accuracy:0.71
Precision:0.98
Recall:0.97
F1-score:0.97

Classification Report:

	precision	recall	f1-score	support
Normal	1.00	0.99	0.99	332768
Generic	1.00	0.97	0.99	32322
Exploits	0.81	0.45	0.58	6679
Fuzzers	0.45	0.84	0.58	3637
DoS	0.32	0.72	0.45	2453
Reconnaissance	0.68	0.82	0.75	2098
Analysis	0.12	0.25	0.16	402
Backdoor	0.09	0.36	0.15	349
Shellcode	0.60	0.97	0.74	227
Worms	0.05	0.73	0.10	26
accuracy			0.97	380961
macro avg	0.51	0.71	0.55	380961
weighted avg	0.98	0.97	0.97	380961

(d)