Incident Detection with Pruned Residual Multilayer Perceptron Networks

Mohamad Soubra  
AGH University  
Faculty of Computer Science  
Electronics and Telecommunications  
Krakow, Poland  
Email: soubra@agh.edu.pl

Marek-Kisiel Dorohinicki  
AGH University  
Faculty of Computer Science  
Electronics and Telecommunications  
Krakow, Poland  
Email: doroh@agh.edu.pl

Marcin Kurdziel  
AGH University  
Faculty of Computer Science  
Electronics and Telecommunications  
Krakow, Poland  
Email: kurdziel@agh.edu.pl

Marek Zachara  
AGH University  
Faculty of Electrical Engineering,  
Automatics, Computer Science and Biomedical Engineering  
Krakow, Poland  
Email: mzachara@agh.edu.pl

Abstract—Internet of things (IoT) has opened new horizons in connecting all sorts of devices to the internet. However, continuous demand for connectivity increases the cybersecurity risks, rendering IoT devices more prone to cyberattacks. At the same time, rapid advances in Deep Learning (DL)-based algorithms provide state-of-the-art results in many classification tasks, including classification of network traffic or system logs. That said, deep learning algorithms are considered computationally expensive as they require substantial processing and storage capacity. Sadly, IoT devices have limited resources, making renowned DL models hard to implement in this environment. In this paper we present a Residual Neural Network inspired DL-based Intrusion Detection System (IDS) that incorporates weight pruning to make the model more compact in size and resource consumption. Additionally, the proposed system leverages feature selection algorithms to reduce the feature-space size. The model was trained on the NSL-KDD dataset benchmark. Experimental results show that the proposed system is effective, being able to classify network traffic with an $F_1$ score of up to 98.9% before the pruning and an $F_1$ score of up to 97.5% after pruning 90% of network weights.

I. INTRODUCTION

INTERNET of Things (IoT) is booming in markets, driving efforts for increasing device inter-connectivity. However, this strive for increased connectivity poses requirements related to provision of security protocols and measures that would secure communication between devices and build trust in users that their data is communicated privately [1]. In order to meet these requirements current security solutions typically endorse defense in depth approach [2] in which the security layers span across network perimeter, intranet and endpoint systems. Such security mechanisms involve many attack detection and prevention technologies. One of the most important class of these technologies, namely Intrusion Detection Systems (IDS) [3], come in various flavors. Host-IDS examines the actions of the users and compares them to decide which actions can be considered as malevolent and which are likely benign. On the other hand, Network-based IDS, examines the traffic traversing through the network and compares it with already known signatures to distinguish between normal and malevolent flow. Though popular, these systems still face various challenges, such as detection accuracy, high false-alarm rates or the inability to detect zero-day attacks [4].

Machine Learning (ML) and Deep Learning (DL)-based technologies recently enjoy numerous practical deployments, e.g., in speech recognition, object detection, natural language processing, etc. It is also increasingly used in the cybersecurity domain [5][6]. Consequently, ML- and DL-based IDS gained popularity in the recent years. In particular, they have proven to be more robust than their predecessors, having lower false-positive rates and higher accuracy [7]. However, this line of research often adopted renowned image classification algorithms [8] to the traffic classification tasks [9][10][11]. Consequently, the proposed systems tend to be computationally cumbersome. Accordingly, for IoT devices, which have limited storage and processing resources, research increasingly focuses on replacing such burdensome algorithms with much lighter solutions.

In this paper, we introduce a new DL-based IDS designed around lightweight residual network [12] architectures. Our solution is coupled with the Extra Tree classification algorithm, which allows us to extract the most important features from the dataset. This makes the proposed system compact, while retaining high accuracy and detection rates. The small
computational footprint of the proposed system is suitable for inference on a CPU, instead of resource-hungry GPU accelerators. Thus, our results show that attaining high accuracy while substantially reducing the size of the model is achievable in IDS tasks.

The following sections begin with review of the state-of-the-art results in ML-based intrusion detection systems. Next, we present the proposed attack detection architecture. Subsequently we describe the experimental setup and report obtained results. Finally, we give conclusions from experiments and outline future work.

II. RELATED WORK

Deep Learning-based intrusion detection systems enjoyed rapid advances in recent years. Some researchers utilized DL capabilities for categorical data classification, where the task is to recognize specific attack instances. Haddad Pajouh et al. [13] proposed a Long-Short Term Memory (LSTM)-based IDS. First, they extracted OpCodes from the traffic and assigned them to input vectors. Next, they leveraged Principal Component Analysis (PCA) to extract the most significant features from the vectors. The model was trained using Adam optimizer [14]. Dropout layers were used to avoid overfitting.

The performance of the model was evaluated with 10-fold Cross Validation (CV). Swarna Priya et al. [15] proposed a DNN-based IDS that, similarly to Haddad Pajouh et al. approach, also used PCA as a feature extractor. The system also utilized feature scaling to normalize the input data before feeding it to the classifier. Furthermore, they used Grey Wolf optimization algorithm (GMO) [16] to construct a feature hierarchy. This hierarchy provided features’ fitness values. McDermott et al. [17] proposed a Bidirectional LSTM-based IDS. Word embeddings were used to embed the captured packets’ content in a vector space suitable for the model. Subsequently, they used word embeddings to establish a dictionary of tokenized words. Sigmoid function, Mean Absolute Error (MSE) and Adam were selected as the activation function, loss function and optimizer, respectively. Zhang et al. [18] proposed a Deep-Belief Network (DBN)-based IDS that employed an improved genetic algorithm. The algorithm incorporated improved crossover and elite retention strategies to prevent the loss of the best individuals. The proposed system was trained and evaluated on the NSL-KDD dataset. Another DBN-based IDS was proposed by Tama et al. [19]. The system incorporated a grid search strategy to select the most significant input features. Evaluation was carried out on three datasets, namely, UNSW-NB15 [20], CIDDS-001 [21], and GPRS [22] using 10-folds cross validation, Repeated Cross-Validation (RepCV) [23] and data sub-sampling. Their model was able to maintain the same detection rate after subsampling. Overfitting was prevented with L1 and L2 regulations and an adaptive learning rate. Muna et al. [24] proposed a Deep Autoencoder to reduce the features dimensionality. Their system also encompassed a deep feed-forward Neural Network to detect and classify traffic. It was trained and evaluated on the NSL-KDD dataset. Latif et al. [25] emphasized the importance of providing lightweight DL-based IDS solutions. To this end, they proposed an intrusion detection algorithm employing random neural networks, in which the Poisson distribution was used to estimate the probability of the signals that made the neurons either active or inhibited. The proposed system was evaluated on the DS2OS dataset [26]. Shone et al. [27] proposed a Non-Symmetric Deep Auto-Encoder for unsupervised feature learning. The system employed Random Forest [28] to classify the traffic between benign and malevolent. Both NSL-KDD and KDD Cup ’99 datasets were used in training and evaluation. Min et al. [29] proposed a system which uses an ensemble of byte-level word embeddings and text convolutional neural networks. Skip-Gram algorithms was used to create the byte-level word embeddings. Text convolutional neural networks were constructed from one-dimensional convolutions that extracted word-based features. Similarly to Shone et al., Random Forest was chosen as a classifier. The system was trained and evaluated on the ISCX2012 dataset [30]. Zhou et al. [31] proposed Deep Feature Embedding Learning method that reduces input features’ dimensionality, thereby decreasing the time needed to train the model. They trained and evaluated their model on the NSL-KDD and UNSW-NB15 datasets. Leaky ReLU was chosen as the activation function for the hidden layers, while Sigmoid function was used as an activation function in the classification layer. Additionally, Dropout was used to avoid overfitting.

Other researchers choose to use deep learning for binary classification, where the goal is to distinguish attack signatures from normal traffic, irrespective of specific attack classes. Diro et al. [32] proposed a DL-based IDS trained in a distributed optimization scheme which involved fog nodes, i.e. mini-clouds implemented as edge devices in the cloud [33]. To avoid overfitting, the parameters were collected in the fog coordinator, which was responsible for their updating and distribution for subsequent epochs. Diro et al. [32] evaluated their system on the NSL-KDD dataset [34]. Similarly, Abeshu et al. [35] proposed a novel DL-based IDS that takes its parameters from the master fog node, while performing system fine-tuning on the worker nodes. Again, NSL-KDD was chosen as training and evaluation dataset. Almiani et al. [36] proposed an RNN-based IDS. Their system employed data oversampling to balance the minority classes, a modified back-propagation algorithm, and the min-max normalization. Kasongo et al. [37] proposed a feed-forward Neural Network-based IDS that was coupled with a wrapper-based feature extraction unit. The wrapper used the Extra Tree algorithm to classify and specify which features are most significant. The proposed system was trained and evaluated on the NSL-KDD dataset. Devan et al. [38] proposed an XGBoost DL-based IDS composed of three main steps, namely, input feature normalization, feature selection using a classifier based on a collection of decision trees that derive the significant features, and final classification. Their system also leveraged neural networks with ReLU and Softmax activation functions for the hidden and classification layers, respectively. Nagisetty et al. [39] proposed a DL-based IDS that incorporated three
DL architectures: Multi Layer Perceptrons (MLP), CNNs, and an Autoencoders. The proposed system was trained and evaluated on two datasets, namely, UNSW-NB15 and NSL-KDD99. The system employed Root Mean Square Root Error (RMSE) as the cost function. DNN was used mainly to sort the features and create a feature hierarchy. Zhihan et al. [40] proposed a hierarchical Supporting Vector Machine-based IDS. In addition, a stacked autoencoder was used to denoise the data. The system was evaluated on the NSL-KDD dataset.

In this paper we will benchmark our results with the papers that focus on the binary classification task. To this end, we will evaluate our algorithm with respect to the metrics that they have discussed in their papers as our goal is to see if our pruned networks could compete with the state-of-the-art.

III. PROPOSED ARCHITECTURE

In order to protect devices from attacks, while preserving processing and storage resources, we propose an Intrusion Detection System based on pruned residual neural networks [12]. The proposed system is trained in several steps. First, input data is pre-processed, including encoding of symbolic features. The data is then fed to an Extra Tree Classifier [41], which selects the most important features from the feature set. In the next step, the data is normalized and used to train the proposed classification model. Finally, the model is pruned in fine-tuning steps, which minimizes its size and the inference cost.

We evaluate the final model with respect to precision, recall and F1 score both before and after network pruning. Evaluation is carried on the NSL-KDD dataset.

A. NSL-KDD Dataset

The NSL-KDD dataset is the successor of the KDD’99 [42] dataset, which was introduced by DARPA in 1998. The dataset was firstly proposed by Tavallaee et al. [43] and is composed of 4 different attack classes, namely, Denial of Service (DoS), Probe, User-to-root (U2R), and Remote-to-Local (R2L). In DoS attacks the computing or network resources are exhausted, making the attacked system unable to serve the user’s requests. Signatures of a DoS attack in the NSL-KDD dataset would be, e.g., the Src_bytes and the Wrong_fragment features. Probe attacks are mostly used for surveillance, in order to to gain information on the potential victim system. The relevant signatures for probe attacks in the NSL-KDD dataset are the Src_bytes and the Duration features. User-to-root attacks attempt to grant superuser privileges to the attacker. One way of doing this is accessing the user's system via a normal account and then attempting to escalate privileges by exploiting a vulnerability. Relevant signatures for U2R attacks with respect to the NSL-KDD dataset are, e.g., Num_file_creations and Num_shells features. In Remote-to-Local attacks the attacker attempts to gain access of the user’s system via a remote machine. Relevant signatures for R2L attacks in the NSL-KDD dataset are, e.g., Duration, Service and Num_failed_logins features.

<table>
<thead>
<tr>
<th>Attack Type</th>
<th>KDDTrain+</th>
<th>KDDTest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DOS</td>
<td>43526</td>
</tr>
<tr>
<td>2</td>
<td>Probe</td>
<td>11655</td>
</tr>
<tr>
<td>3</td>
<td>R2L</td>
<td>995</td>
</tr>
<tr>
<td>4</td>
<td>U2R</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>Normal</td>
<td>67355</td>
</tr>
<tr>
<td>Total</td>
<td>123973</td>
<td>22544</td>
</tr>
</tbody>
</table>

The NSL-KDD dataset encompasses two subsets, namely, the KDDTrain+ and KDDTest. In standard classification setup, the proposed system should assign the signatures into four major categories, namely, DOS, Probe, R2L, U2R, and Normal traffic. Table 1 reports datasets statistics for these categories. Note that DOS, Probe, R2L, U2R and normal traffic makes 36.45%, 9.25%, 0.78%, 0.04% and 45.52% of the dataset instances, respectively. In binary classification setup the proposed system should be able to classify the traffic into two classes, namely, attack and non-attack. Note that classes in this case are balanced, with attack and non-attack traffic making 46.5% and 53.5% of the dataset, respectively.

The NSL-KDD consists of a total of 41 features that comes in four main categories: (a) intrinsic features that can be extracted from the packet’s headers, (b) content features which reflect the data content of the packets, (c) time-based features which reflect the connection rates with the hosts, and finally (d) the host-based features. It is also worth mentioning that the KDDTrain+ subset has 3 categorical features, namely:

- Protocol Type which consists of 3 categories,
- Services which consists of 70 categories,
- Flag which consists of 11 main categories.

These features require preprocessing into one-hot encoding before they can be used in the subsequent steps.

B. Data Preprocessing

In this work we focus on a binary classification task, i.e., distinguishing normal network traffic from attacks. We therefore convert the provided labels into attack and non-attack classes before selecting the important features. Next, we remove data duplicates and rows that contain null values. The KDDTrain+ subset consists of both numerical and categorical data. We normalize the numerical features via z-scores:

\[ Z = \frac{x - \mu}{\sigma} \]  

where \( x \) represent the current instance of the feature while \( \mu \) and \( \sigma \) represent the mean and the standard deviation of the feature respectively. The categorical features, on the other hand, are encoded in one-hot vectors.

In the next step we use Extra Tree classifier with 100 estimators (trees) to identify the most significant features. Specifically, we use the Gini coefficients [44] returned by the Extra Tree classifiers to select the most prominent features. Importantly, for one-hot-encoded features, the feature is retained if the Extra Tree classifier selects any of its dimensions.
according to the Gini coefficient. After a series of features selection iterations, the features listed in Table II were used to train the neural network for the binary classification task.

**TABLE II: Features selected by the Extra Tree classifier.**

<table>
<thead>
<tr>
<th>Index</th>
<th>NSL-KDD Features</th>
<th>Index</th>
<th>NSL-KDD Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>count</td>
<td>12</td>
<td>protocol_type</td>
</tr>
<tr>
<td>2</td>
<td>same_srv_rate</td>
<td>13</td>
<td>logged_in</td>
</tr>
<tr>
<td>3</td>
<td>dst_host_count</td>
<td>14</td>
<td>error_rate</td>
</tr>
<tr>
<td>4</td>
<td>dst_host_same_srv_rate</td>
<td>15</td>
<td>same_srv_rate</td>
</tr>
<tr>
<td>5</td>
<td>dst_host_rerror_rate</td>
<td>16</td>
<td>error_rate</td>
</tr>
<tr>
<td>6</td>
<td>dst_host_same_src_port_rate</td>
<td>17</td>
<td>service</td>
</tr>
<tr>
<td>7</td>
<td>dst_host_same_srv_rate</td>
<td>18</td>
<td>flag</td>
</tr>
<tr>
<td>8</td>
<td>dst_host_rerror_rate</td>
<td>19</td>
<td>src_bytes</td>
</tr>
<tr>
<td>9</td>
<td>dst_host_srv_count</td>
<td>20</td>
<td>srv_rerror_rate</td>
</tr>
<tr>
<td>10</td>
<td>dst_host_srv_diff_host_rate</td>
<td>21</td>
<td>srv_rerror_rate</td>
</tr>
<tr>
<td>11</td>
<td>dst_host_srv_rerror_rate</td>
<td>22</td>
<td>dst_host_srv_rerror_rate</td>
</tr>
</tbody>
</table>

C. Pruning

We use pruning to reduce the overall size of the trained neural model. There are several pruning strategies that can be used to this effect:

- The classical approach in which the model is firstly trained with all parameters and then subset of the trained parameters is removed during additional fine-tuning epochs.
- Pruning at initialization, where parameters are pruned before the model is trained [45].
- Pruning during the main training run.

Furthermore, pruning can carried out globally, i.e., across the whole model, or locally, i.e., in each network layer [46].

In this work we use global, magnitude-based pruning which employs fine-tuning epochs after the main training run, during which weights with low magnitudes are gradually set to zero.

**IV. EXPERIMENTAL SETUP**

Table III summarizes the hyper-parameters used in the experiments. These training hyperparameters were selected with few trial training runs. We evaluate variants of this architecture with varying widths. In particular, we vary the number of neurons inside the residual blocks while keeping a fixed network width of the skip-connection nodes. Furthermore, batch normalization layers [47] are used to improve the training. This architecture proved to work well, while saving on the number of model parameters. Each constructed model was run five times with different random seeds.

We also carried out evaluation of pruned variants of our models. To this end, a 20 epoch fine-tuning run with magnitude-based pruning was done. For each network instance the sparsity schedule started with 85% initial sparsity and increased with each iteration, until it reached a final sparsity of 90% by the end of the last fine-tuning epoch. For the performance numbers we report mean and variance of training time, test accuracy, precision, recall and F1 score:

\[
\text{Precision} = \frac{TP}{TP + FP} \quad \text{(2)}
\]

\[
\text{Recall} = \frac{TP}{TP + FN} \quad \text{(3)}
\]

\[
\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \quad \text{(4)}
\]

\[
F1 - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \text{(5)}
\]

where \( TP \) is the true-positive count, \( FP \) is the false-positive count, \( TN \) is the true-negative count and \( FN \) is the false-negative count. Due to the substantial pruning rate, the sparse models of the same width tended to have the same performance across random seeds. Consequently, the variance estimates are not meaningful in this case and we don’t report them.

**V. RESULTS**

The proposed models were trained using the Google Collab environment. Table IV reports the results for unpruned networks. The model with the highest width achieved 98.96% accuracy, 99.39% precision, 98.38% recall and 98.91% \( F_1 \) score. Note that this is the most computationally expensive of our models. That said, the model with quarter the width preformed equally well up to the variance across random seeds. The remaining two models performed slightly worse, with \( F_1 \) score around 0.6% below that of the larger models. These models were, however, much more computationally efficient, with the training time stabilizing below width equal to 32 units. Our results also shows that the variance across the training runs is low for all models, which shows that the performance is not highly affected by the initial seeds.

Results for the pruned networks are reported in Table V. The \( F_1 \) score of the model with 1024-unit width dropped by about 1.5% after pruning, with performance decrease manifesting mostly in model's recall. For the pruned model with quarter the width, the performance metrics were about 0.5% below those of the larger pruned model and up to 2% below the unpruned network. The two smallest models scored the lowest after pruning, with an \( F_1 \) score approximately 2% below larger pruned networks. Overall, our results shows that even with aggressive pruning and small initatied models residual fully-connected networks perform well in this task, with precision recall and \( F_1 \) score above 95%.

**TABLE III: List of training hyper-parameters.**

<table>
<thead>
<tr>
<th>Models width</th>
<th>Activation function</th>
<th>Optimizer</th>
<th>Loss function</th>
<th>Learning rate</th>
<th>Decay for unpruned models</th>
<th>Decay for pruned models</th>
<th>Batch size</th>
<th>Number of epochs (for both pruned and unpruned networks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024, 256, 32, 8</td>
<td>RELU [48]</td>
<td>Stochastic Gradient Descend (SGD) with Nesterov accelerated gradient = 0.9 [49]</td>
<td>Binary Cross Entropy [50]</td>
<td>0.1</td>
<td>1e-6</td>
<td>Polynomal Decay</td>
<td>120</td>
<td>20</td>
</tr>
</tbody>
</table>

\[
\text{Recall} = \frac{TP}{TP + FN} \quad \text{(3)}
\]

\[
\text{Accuracy} = \frac{TP + TN}{TP + FN + TN + FP} \quad \text{(4)}
\]

\[
F1 - \text{Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad \text{(5)}
\]
TABLE IV: Performance metrics for unpruned models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Time (sec)</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>304.5 ± 3.8</td>
<td>98.96 ± 0.03</td>
<td>99.39 ± 0.10</td>
<td>98.22 ± 0.08</td>
<td>98.91 ± 0.05</td>
</tr>
<tr>
<td>256</td>
<td>155.4 ± 6.5</td>
<td>98.91 ± 0.05</td>
<td>99.44 ± 0.02</td>
<td>98.22 ± 0.08</td>
<td>98.82 ± 0.05</td>
</tr>
<tr>
<td>32</td>
<td>72.3 ± 7.1</td>
<td>98.38 ± 0.07</td>
<td>91.07 ± 0.16</td>
<td>97.45 ± 0.23</td>
<td>98.25 ± 0.08</td>
</tr>
<tr>
<td>8</td>
<td>73.7 ± 8.5</td>
<td>98.38 ± 0.07</td>
<td>99.07 ± 0.16</td>
<td>97.45 ± 0.23</td>
<td>98.25 ± 0.08</td>
</tr>
</tbody>
</table>

TABLE V: Performance metrics for pruned models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Time (sec)</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>471.5</td>
<td>97.87</td>
<td>99.22</td>
<td>95.95</td>
<td>97.46</td>
</tr>
<tr>
<td>256</td>
<td>217.9</td>
<td>97.11</td>
<td>99.22</td>
<td>95.01</td>
<td>97.07</td>
</tr>
<tr>
<td>32</td>
<td>118.6</td>
<td>95.82</td>
<td>95.01</td>
<td>95.13</td>
<td>95.52</td>
</tr>
<tr>
<td>8</td>
<td>119.3</td>
<td>95.82</td>
<td>95.01</td>
<td>95.13</td>
<td>95.52</td>
</tr>
</tbody>
</table>

TABLE VI: Performance metrics reported in related work.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>[32]</td>
<td>99.20%</td>
<td>99.02%</td>
<td>99.27%</td>
<td>99.14%</td>
</tr>
<tr>
<td>[35]</td>
<td>99.20%</td>
<td>-</td>
<td>99.27%</td>
<td>-</td>
</tr>
<tr>
<td>[36]</td>
<td>92.42%</td>
<td>90.20%</td>
<td>-</td>
<td>92.29%</td>
</tr>
<tr>
<td>[37]</td>
<td>99.37%</td>
<td>-</td>
<td>92%</td>
<td>-</td>
</tr>
<tr>
<td>[38]</td>
<td>97.68%</td>
<td>97%</td>
<td>97%</td>
<td>97%</td>
</tr>
<tr>
<td>[39]</td>
<td>98.96%</td>
<td>-</td>
<td>-</td>
<td>92.28%</td>
</tr>
<tr>
<td>[40]</td>
<td>97.83%</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Our unpruned 1024 model: 98.96±0.03% 99.39±0.10% 98.38±0.11% 98.91±0.07%

Our pruned 1024 model: 95.82% 95.91% 95.13% 95.52%

To benchmark our results against the state of the art, we selected peer-reviewed papers which addressed the binary classification task with respect to the same NSL-KDD dataset. Some of these papers reported all the metrics mentioned earlier, while others took into consideration only a subset of them. Comparison between the benchmarks and our results is summarized in Table VI.

Comparing with the state-of-the-art for this benchmark dataset in binary classification setup, we observe that all of the proposed unpruned networks give competitive or better precision in detecting attacks (Table VI). More precisely, the models with 1024 and 256 widths achieved better accuracy compared to [36] [38] [39] [40], recall compared to [37] [38] and F1 score compared to [36] [38] [39]. The pruned models achieved slightly lower results, but still maintained strong performance while requiring only 10% of the initial parameters.

VI. CONCLUSIONS AND FUTURE WORK

Proliferation of IoT devices is making a huge impact on the communication sector. The increased interconnectivity comes not only with new business opportunities, but also increases security risks related to prevalence of network vulnerabilities and persistent cyberattack threats. Conventional IDS and firewalls deployed to counter cyber-threats are often inadequate for IoT environments, e.g., due to high false-positive rates or large resource requirements. In this paper we proposed an ML-based IDS that employs residual MLP networks and demonstrated that it provides strong results with respect to the precision and recall of attack detection, even when implemented with relatively small networks. We also demonstrated that it retains most of its accuracy after pruning of as much as 90% of its parameters.

In our future work we intend to extend this line of research with novel and promising neural architectures, e.g., transformer models. These models excel at text embedding and classification. We therefore intend to explore their ability to classify network and system logs. We also intend to explore more pruning strategies, e.g., unit-based pruning which removes entire neurons, rather than individual weights. Such pruning strategies may result in lower computational footprint, while still maintaining strong attack detection performance.

REFERENCES


M. Pahl and F. Aubeit, "Dx2os traffic traces IOT traffic traces gathered in a the dx2os in environment," 2018.


