Stacked Autoencoder-Based Probabilistic Feature Extraction for On-Device Network Intrusion Detection

Thi-Nga Dao and HyungJune Lee, *Member, IEEE*

***Abstract*—Due to the outbreak of recent network attacks, it is necessary to develop a robust network intrusion detection system (NIDS) that can quickly and effectively identify the network attack. Although the state-of-the-art detection algorithms have shown quite promising detection performance, they suffer from computationally intensive operations and large memory footprint, making themselves infeasible to applications at the resource-constrained edge devices. We propose a lightweight yet effective NIDS scheme that incorporates a stacked autoencoder with a network pruning technique. By removing a set of ineffective neurons across layers in the autoencoder network with a certain probability based on their importance, a considerably large portion of relatively nominal training parameters are reduced. Then, the pruned and pretrained encoder network is used as-is and is connected with a separate classifier network for attack type inference, avoiding a full retraining from scratch. Experimental results indicate that our stacked autoencoder-based classification network with probabilistic feature extraction has outperformed the state-of-the-art NIDSs in terms of attack detection rate. Further, we have shown that our lightweight NIDS scheme has significantly reduced the computational complexity throughout the architecture, making it feasible to the edge, while maintaining a similar attack type detection quality compared with its original fully connected neural network.**

***Index Terms*—Anomaly classification, feature extraction, network intrusion detection system (NIDS), on-device AI.**

I. INTRODUCTION

NETWORK intrusion refers to any unauthorized activities on a network, such as denial-of-service attacks, backdoor attacks, brute-force attacks, which attempt to gather private information of users or make network services inaccessible to its intended users [1], [2]. Recent studies show that these network attacks have increasingly occurred in both their frequency and traffic volume. In order to address the network intrusion problem and to strengthen network security, it is important to design an agile yet reliable network intrusion detection system (NIDS). The functionality of early detecting the abnormal network behaviors and quickly responding to the detected events is considered as an essential requirement in most of the recent network devices.

As the recent Internet of Things (IoT) has imposed the requirement of low response time and bandwidth usages, machine learning-driven intelligent applications have been brought to the edge, such as mobile devices, embedded sensors or programmable network devices [3]. Applications that can perform inference on the edge devices include on-device services from face recognition, natural language processing to network intrusion detection. In case of the NIDS, building an efficient distributed detection model on resource-constrained edge devices is a challenging problem. It requires a significant latency reduction to take a responsive action against network attacks for ensuring network security.

Taking advantage of state-of-the-art machine learning techniques, the existing NIDSs mostly running on powerful GPU-assisted machines [4]–[6] have achieved promising prediction performance over real-world network traffic data sets. Generally, supervised learning-based models are constructed with two separate phases: 1) feature extraction in which a latent representation of input features are learned and 2) classifier to identify its attack type. In efforts to achieve computationally efficient performance, various feature extraction models including autoencoder (AE) [7] have been employed with the deep neural network architecture. However, even if the feature extraction has been applied, a neural network can still contain numerous parameter weights. Thus, it should be supported by substantial computation and memory resources for its intensive computing operations, making infeasible to edge network devices.

In this article, we propose a lightweight yet effective NIDS scheme that incorporates a stacked AE with a network pruning technique. The network pruning executes probabilistic feature extraction and infers network attack types so that it can be feasible to edge devices. Inspired by the fact that there exist substantial parts of redundant connections in a trained network, we perform a neuron pruning process. This process removes relatively insignificant neurons and their edges for constructing a compact AE architecture. To evaluate whether a neuron is effective or not, we quantify an importance score for each neuron. The score is calculated by taking into account the correlation between the input and the output features on the AE architecture as well as the classification label, and then a

Manuscript received 14 January 2021; revised 21 March 2021; accepted 4 May 2021. Date of publication 7 May 2021; date of current version 8 August 2022. This work was supported by the Samsung Research Funding & Incubation Center of Samsung Electronics under Project SRFC-TB1803-02. *(Corresponding author: HyungJune Lee.)*

Thi-Nga Dao is with the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University, Hanoi 10000, Vietnam (e-mail: daothinga.mta@gmail.com).

HyungJune Lee is with the Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, Republic of Korea (e-mail: hyungjune.lee@ewha.ac.kr).

Digital Object Identifier 10.1109/JIOT.2021.3078292

neuron gets pruned with a certain probability that is assigned based on the rank of its score. By doing so, high dimensional network traffic features are effectively extracted and captured at a low dimensional feature space as the internal representation.

By taking the pruned encoder part from the architecture and connecting it with an additional classifier layer, we form a new neural network for the attack classification purpose. We reuse the intact pretrained encoder layers and train only the classifier on the labeled data with attack types. Avoiding the entire network training from scratch helps to get the NIDS network ready for inference as early as possible, making it practically feasible to the edge.

Previous, work on network intrusion detection generally consists of two phases: 1) feature selection and 2) classification [4], [5], [8], [9], similar to this work. Since these existing NIDS often suffer from the high computation and execution time overhead, several NIDS approaches have been evolved with network simplication or sparsification via parameter reparameterization [10], feature reduction [11], and neuron pruning [12].

Some more general network simplication irrespective of the NIDS context has long been investigated with the following categories: neuron pruning [13]–[15], feature reduction [16]–[19], and operation simplification [20]–[22].

More closely related to feature extraction and neuron pruning, primarily considered in this work, Molchanov *et al.* [23] have designed a pruning method that approximates the loss change with the first-order derivative term upon trimming a feature map. Han *et al.* [24] have proposed a pruning method in which the edge connections with the smaller absolute weight values are removed from the network. Yu *et al.* [25] have introduced a neuron importance score propagation (NISP) algorithm that measures the importance score of each neuron and prunes one based on the backpropagation impact on its prior neurons.

To the best of our knowledge, this work is the first to implement an on-device multiclass classification of network attack types using a stacked AE architecture with a probabilistic neuron pruning approach. The main contributions can be summarized as follows.

1) We propose an on-device NIDS with much fewer parameters, which allows to quickly detect abnormal network behaviors as well as attack types, based on a stacked pruned AE combined with a classifier network.
2) A lightweight probabilistic feature extraction method is designed with constrained memory size and computing capacity for edge devices.
3) Experimental results with different pruning rates over real-world data sets demonstrate that our proposed algorithm significantly reduces the model size, while achieving the competitive performance in both supervised and unsupervised tasks.

II. System Model

As the Internet grows exponentially with a huge number of edge and IoT devices, botnet malware often use them as intermediate hosts to perform distributed massive attacks to certain network devices, and sometimes the entire network. Moreover, the attack patterns become more diversified and intelligent based on the recent advanced machine learning techniques. Analyzing real-time network traffic traces reliably and promptly is a challenging, but essential task, which is required at modern edge devices.

We address the problem of network intrusion detection at edge devices with constrained computation and memory capacity. We aim to design a lightweight deep neural network architecture that can effectively learn a nonlinear relationship between network traffic features and attack types with the small learning and inference overhead.

In order to handle large scale network traffic traces that have embedded numerous underlying features, at edge devices, feature extraction offers an effective way of discovering statistically significant features. AE is a neural network that can learn efficient representation of the input data by compressing it into the latent code, and thus can be used as a powerful feature detector.

This work incorporates an AE model as the underlying feature detector on network attack traces. To significantly reduce the model size in terms of computation and memory consumption, we integrate a neuron pruning approach with the AE architecture.

To formally define the problem of this work, we first introduce necessary notations for the AE model. Let $\boldsymbol{x}$ and $\boldsymbol{W}$ denote the input data and the weight matrices. Note that $\boldsymbol{W}$ consists of a set of weight matrices, and the elements in a weight matrix indicates the connection levels between two consecutive layers. The output layer $f(\boldsymbol{x}, \boldsymbol{W})$, which is the reconstructed data, can be expressed as a function of the input data and the weight matrices. To indicate the architecture of the pruned network, the binary mask matrices $\boldsymbol{M}$, which has the same size as $\boldsymbol{W}$, are used. At each matrix in $\boldsymbol{M}$, a value of 1 implies a valid connection between two corresponding units, whereas a value of 0 indicates a pruned connection. Then, the output layer in the pruned AE is represented as $f(\boldsymbol{x}, \boldsymbol{W} \odot \boldsymbol{M})$, where $\odot$ denotes the element-wise multiplication.

We aim to find a pruned AE model that minimizes the reconstruction loss, while meeting a target pruning rate p_{prune}, as follows:

$$\begin{aligned} \min_{\boldsymbol{M}} \quad & |f(\boldsymbol{x}, \boldsymbol{W} \odot \boldsymbol{M}) - \boldsymbol{x}| \\ \text{subject to:} \quad & \mu_{\boldsymbol{M}} \leq 1 - p_{\text{prune}} \end{aligned}$$

where $\mu_{\boldsymbol{M}}$ is the mean value of binary mask matrices $\boldsymbol{M}$. For example, if p_{prune} is set to 0.4, $\mu_{\boldsymbol{M}}$ should be at most 60% of the elements that should be set to 1 in $\boldsymbol{M}$.

Then, the problem of deriving a lightweight stacked AE-based feature extraction model can be decomposed into three stages: 1) score estimation; 2) pruning rate estimation; and 3) pruned network construction, as illustrated in Fig. 1. The first stage of score estimation is to extract network traffic features and quantify the intercorrelation among them. At the second stage, after computing a pruning probability of the input feature, the neuron-based pruning probability is inferred. At the final stage, the extracted features as the output of

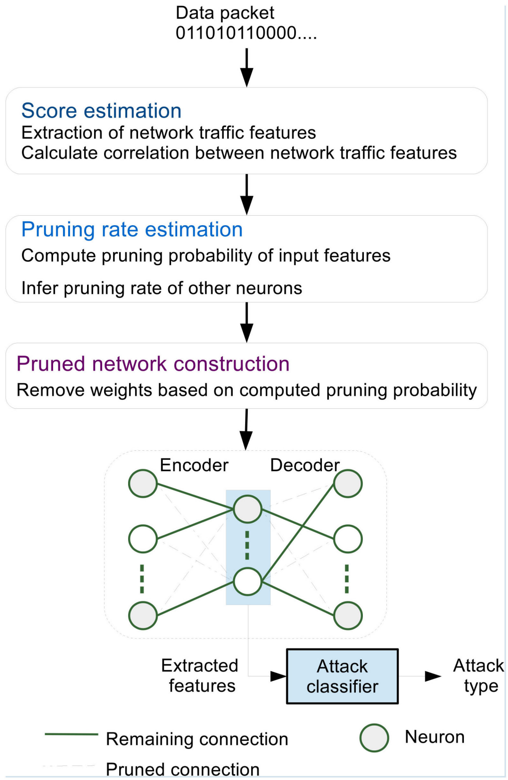


Fig. 1. Our proposed lightweight network intrusion detection architecture with a stacked AE-based probabilistic feature extraction.

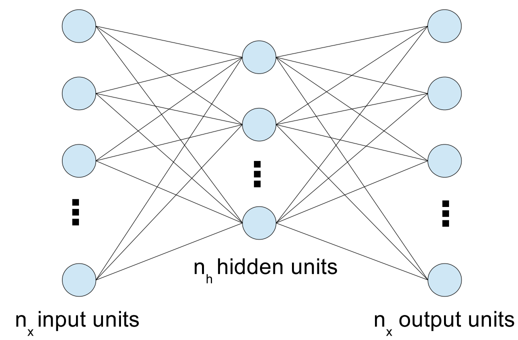


Fig. 2. Architecture of a fully connected AE model with one hidden layer.

the pruned AE network, are fed into the input of an attack classifier network, in order to determine a specific attack type.

III. Our Approach

In this section, we first discuss the motivation behind our pruning algorithm in Section III-A. In order to make an effective decision on neuron pruning, We derive a formal relationship of the importance score of neurons with the input features $\boldsymbol{x}$ and the weight matrices $\boldsymbol{W}$. Then, we introduce our proposed pruning algorithm based on an AE network, which is inspired by the formulation of the importance score of neurons in Section III-B. Finally, we exploit the pretrained pruned encoder network to construct an attack classifier for detecting the abnormality and the attack type from the network traffic traces.

A. Motivation

For the sake of simplification, we first take into account an AE model with one hidden layer. Fig. 2 shows an AE model consisting of three layers: 1) input; 2) one hidden; and 3) output layers. The equation of rectified linear unit (ReLU) activation function is presented as follows:

$$\mathrm{ReLU}(z) = \begin{cases} z, & z \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Since the computation of the derivative of ReLU is quite fast compared to other activation functions, and there is no saturation for the range of non-negative input values, ReLU is selected as the activation function. The input and output layers have n_x features, while the hidden layer contains n_h ReLU activation units (where $n_h < n_x$, in general). Let w_{ik} denote the connection weight between input feature x_i and feature h_k in the hidden layer. Similarly, w'_{kj} is the weight value that represents the strength of connection between feature h_k and the reconstructed feature $\hat{x}'_j$. Specifically, $h_k = \sum_{i=1}^{n_x} \mathrm{ReLU}(x_i w_{ik} + b_k)$ and $\hat{x}_j = \sum_{k=1}^{n_h} \mathrm{ReLU}(h_k w'_{kj} + b_j)$ where b_k and b_j are bias values of hidden unit h_k and output unit $\hat{x}'_j$, respectively.

Assume that there are m samples in the training set. The objective function is to minimize the mean-squared reconstruction error

$$C = \frac{1}{2m} \sum_{l=1}^{m} \left(\boldsymbol{x}^{(l)} - \hat{\boldsymbol{x}}^{(l)} \right)^2 \tag{2}$$

where $\boldsymbol{x}^{(l)}$ the lth sample in the training set. Using the chain rule, the importance of each input feature (which is proportional to the change in the loss function with respect to a specific input feature $|(\partial C)/(\partial x_i)|$) is derived for a given training sample

$$\begin{aligned} \left| \frac{\partial C}{\partial x_i} \right| &= \left| \sum_{j=1}^{n_x} \frac{\partial C}{\partial \hat{x}_j} \cdot \frac{\partial \hat{x}_j}{\partial x_i} \right| \\ &= \left| \sum_{j=1}^{n_x} \frac{\partial C}{\partial \hat{x}_j} \cdot \sum_{k=1}^{n_h} \frac{\partial \hat{x}_j}{\partial h_k} \frac{\partial h_k}{\partial x_i} \right| \\ &= \left| \sum_{j=1}^{n_x} \left(x_j - \hat{x}_j \right) \cdot \sum_{k=1, \hat{x}_j \geq 0, h_k \geq 0}^{n_h} w'_{kj} w_{ik} \right|. \end{aligned} \tag{3}$$

According to the above equation that the importance of feature x_i depends on the sum of the product of weight values, $w'_{kj} w_{ik}$. In case of the fully connected AE, x_i propagates through all hidden units h_k on the link w_{ik} ($1 \leq k \leq n_h$), before reaching the output unit $\hat{x}_j$ on the link w'_{kj}. In other words, the importance score of x_i is highly related to the propagation paths from x_i to $\hat{x}_j$. Accordingly, the product of connection weights ($w'_{kj} w_{ik}$) on the propagation path between x_i and $\hat{x}_j$ can be used as a physically meaningful metric for pruning a neuron in the AE model. However, it takes a long time for the training procedure to find the optimal weight values, and some specific learning cases with a large number of training samples or a complex AE architecture often make it even worse.

We aim to design a pruning algorithm that can sparsify a given network without a pretraining process. Inspired by the fact that the importance score of an input feature highly depends on the propagation path between the input feature and output units, we propose a probabilistic pruning method in which the importance score of an input unit is first determined based on a correlation coefficient between the input unit and the reconstructed input. Then, the probability for a neuron to be pruned is computed based on its importance score. In case of the important input features, we want them to keep remained by acquiring as many propagation paths as possible, with a relatively low pruning probability; the unimportant features would rather be skipped for the architectural efficiency by likely pruning the existing propagation paths with a relatively high pruning probability.

Most of the existing pruning algorithms are deterministic, i.e., the edge weights with the lowest importance score are completely removed from the network. An inherent problem of these deterministic pruning methods is that they often delete some boundary yet still important neurons located right below a threshold. Although both the pruned and the remaining neurons may have the similar contributions to the fully connected network, the deterministic algorithms discard any further opportunity. In order to address this issue, we propose a probabilistic pruning method that attempts to prune a neuron with a certain pruning probability, which is determined by the importance score and the rank of the neuron.

Fig. 3 shows an example of removing a connection path from x_i to $\hat{x}_j$. In Fig. 3(a), x_i is fully connected with $\hat{x}_j$ via all 4 hidden units in the fully connected AE model. In case that one hidden unit is pruned, the number of propagation paths from x_i is reduced to 3, based on the contribution of x_i to the output layer.

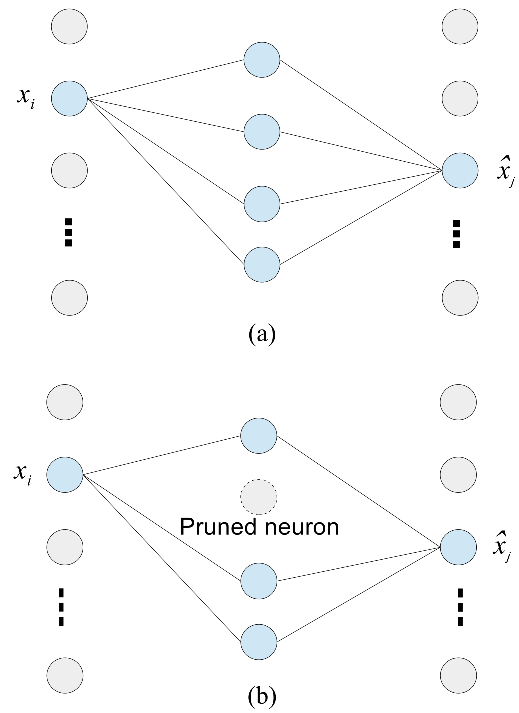


Fig. 3. Demonstration of the propagation paths between x_i and $\hat{x}_j$ in an AE network. (a) Original neural network before pruning. (b) Pruned neural network after pruning.

B. Algorithm

We present a pruning algorithm called Spearman correlation-based probabilistic pruning (SCPP) for a general AE model with an arbitrary number of hidden layers. Note that there are are some advanced AE models [26] with more complex architectures. However, since our work aims to design a lightweight intrusion detection method for edge devices, a fundamental architecture of AE is used to learn the representative features of data traffic. SCPP consists of three steps: 1) computing the importance score of input features; 2) computing the pruning probability for each input feature; and 3) constructing a pruned AE model.

1) Computation of the Input Feature's Importance Score: In the proposed SCPP algorithm, the importance score of the ith input feature x_i is computed by measuring a correlation between x_i and other input features. Since the representative features extracted from the AE network are then used to build a classification model, the SCPP algorithm also considers the correlation between x_i and the output labels of the classifier. Note that in general, the input features are either discrete or continuous variables. We select the Spearman correlation method [27] to compute the correlation coefficient between two features since it can be applied for both discrete and continuous variables. Let $\rho(x_i, x_j)$ denote the correlation coefficient between two input features x_i and x_j while $\rho(x_i, y_k)$ indicates the correlation score between x_i and the kth label y_k. Assume that there are n_y output units in the classification model. We define $s(x_i)$ as the importance score of x_i, which is equal to the mean of Spearman correlation values between x_i and all of the reconstructed features at the output layer of the AE network as well as the label of the classification model. Note that in the ideal case the AE model is trained such that the reconstructed feature equals the corresponding input feature, i.e., $\hat{x}_i = x_i$ with $i = 1, 2, \ldots, n_x$. Therefore, the importance score $s(x_i)$ can be derived as follows:

$$s(x_i) = \frac{1}{n_x + n_y}\left(\sum_{j=1}^{n_x}\left|\rho\left(x_i, x_j\right)\right| + .\sum_{k=1}^{n_y}\left|\rho(x_i, y_k)\right|\right). \tag{4}$$

Since the correlation coefficient can get a negative value, the absolute values $|\rho(x_i, x_j)|$ and $|\rho(x_i, y_k)|$ are considered.

The Spearman correlation between two variables is equivalent to the Pearson correlation between the rank values of them. Let R_{x_i} and R_{x_j} denote the rank of two variables x_i and x_j, respectively. Then, if there are n samples for each variable, $\rho(x_i, x_j)$ is derived as

$$\rho\left(x_i, x_j\right) = \frac{\sum_{k=1}^{n}\left(R_{x_i}^{(k)} - \mu_{R_{x_i}}\right)\left(R_{x_j}^{(k)} - \mu_{R_{x_j}}\right)}{\sigma\left(R_{x_i}\right)\sigma\left(R_{x_j}\right)} \tag{5}$$

where $\mu_{R_{x_i}}$ and $\mu_{R_{x_j}}$ are the mean of rank R_{x_i} and R_{x_j}, respectively, while $\sigma(R_{x_i})$ and $\sigma(R_{x_j})$ denote the standard deviations of rank R_{x_i} and R_{x_j}, respectively. The input features are arranged in the descending order of the importance score's

TABLE I
EXAMPLE OF THE PRUNING PROBABILITY OF FIVE INPUT FEATURES

Rank of Feature scores	Pruning Probability
1	0.0
2	0.2
3	0.4
4	0.6
5	0.8

rank value, i.e., the feature with the largest importance score has a rank value of 1, while the feature with the smallest importance score has a rank value of n_x, which is defined as the number of input features.

2) Computing Pruning Probability for Input Feature: In the second step, we find and assign a pruning probability for each input feature. A linear method is used such that the pruning probabilities are determined to be linearly increased with the ascending order of the rank values of input features' importance scores. More specifically, a feature with the rank mean of the importance score value, $\overline{R_{s(x_i)}} = [(n_x+1)/2]$, is pruned according to a target pruning rate p_{prune}. In addition, the pruning probability for input feature x_i, $p(x_i)$ is linearly proportional to the rank value with the step size $\Delta = ([2 \times \min(p_{\text{prune}}, 1-p_{\text{prune}})]/[n_x-1])$. Mathematically, the pruning probability of feature x_i is calculated as follows:

$$\begin{aligned} p(x_i) &= p_{\text{prune}} + \Delta\big(R_{s(x_i)} - \overline{R_{s(x_i)}}\big) \\ &= p_{\text{prune}} + \Delta\left(R_{s(x_i)} - \frac{n_x+1}{2}\right). \end{aligned} \tag{6}$$

Table I demonstrates an example of the pruning probability for $n_x = 5$ input features, where $p_{\text{prune}} = 0.4$ is given. The step size that indicates the pruning probability difference between two features with the consecutive rank values is given by $\Delta = [(2 \times \min(0.4, 0.6))/(5-1)] = 0.2$.

3) Construction of Pruned AE Network: After obtaining the pruning probability of each input feature, the pruned network is constructed by first computing the mask matrices, and then determining the pruned propagation paths using the mask matrices. For example, in case of the AE model with a hidden layer, a binary mask matrix $M_0 \in \mathbb{B}^{n_x \times n_h}$ is used to indicate the mask connection between the input layer and the first hidden layer. The connection mask of the ith input feature x_i is represented by n_h elements in the ith row of M_0. Specifically, these n_h elements are samples of a random variable that follows the Bernoulli distribution with the mean of $1 - p(x_i)$, where $p(x_i)$ is the pruning probability of input feature x_i.

Let $M_1 \in \mathbb{B}^{n_h \times n_x}$ denote the connection mask between the hidden layer and the output layer. Since the AE model usually has a symmetric architecture, the propagation paths are also symmetric in terms of layer structure (i.e., $M_1 = M_0^T$).

Similarly, we take into account the general AE with an arbitrary number of hidden layers. First, the connection masks of the encoder part are determined, and then that of the decoder side can be inferred because of the symmetric AE architecture. Mask matrix M_0 for weights between the input and the first hidden layers is derived in the same way as in the case of one hidden layer. For the connection mask M_1 that represents the remaining propagation paths between the first and the second hidden layers, the pruning probability of each hidden unit

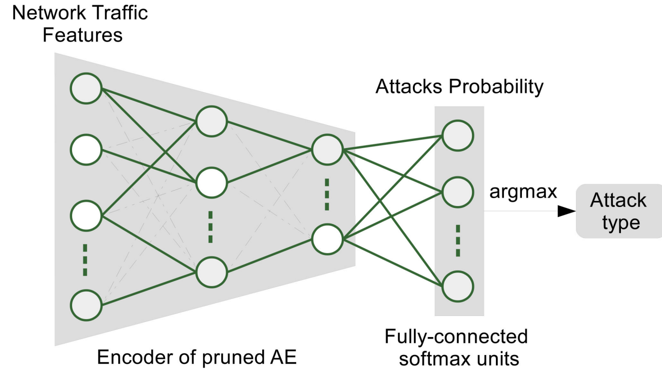


Fig. 4. Our proposed lightweight NIDS architecture based on a pruned AE.

in the first hidden layer should be estimated. We define $h_j^{(k)}$ as the jth unit at hidden layer k. Let M_k and $n_h^{(k)}$ denote the connection mask and the number of hidden units in layer k, respectively. If there are n_l hidden layers, $1 \leq k \leq n_l$. Since hidden units $h_j^{(1)}$ transmit information of all of the input features to the next layer, the pruning probability of $p(h_j^{(1)})$ is calculated as the average ratio of the remaining connections from the input layer through the hidden unit $h_j^{(1)}$

$$p\left(h_j^{(1)}\right) = \frac{1}{n_0}\sum_{i=1}^{n_0} M_0[i,j] \tag{7}$$

where $M_0[i,j]$ is the element of M_0 at the ith row and jth column. In general, based on the connection mask of the preceding layer, the pruning probability of the units in the current layer in the encoder side is calculated in a recursive manner. Suppose that there are $n_h^{(k)}$ units at hidden layer k. The pruning probability for the jth unit at the kth hidden layer $h_j^{(k)}$ is calculated as follows:

$$p\left(h_j^{(k)}\right) = \frac{1}{n_h^{(k-1)}}\sum_{i=1}^{n_h^{(k-1)}} M_{k-1}[i,j] \tag{8}$$

where $j = 1, 2, \ldots, n_h^{(k)}$. Finally, after obtaining the connection mask matrices for the encoder side, the mask matrices for the decoder part are inferred with the assumption of a symmetric AE architecture. In other words, $M_k = M_{n_l-k}$ with $k = [(n_l+1)/2], [(n_l+3)/2], \ldots, n_l$. The procedures of the SCPP method can be summarized in Algorithm 1. The SCPP algorithm consists of three consecutive steps and the complexity of each step is $O(n_x^2 + n_x n_y), O(n_x)$ and $O(n_h^{(1)} n_l)$, respectively. Note that due to $n_h^{(1)} < n_x$, the time complexity of SCPP is as high as $O(n_x^2 + n_x n_y)$.

C. Lightweight Intrusion Detection Architecture

Making use of the prior pruned AE model, we construct and train a neural network-based classifier by connecting a fully connected softmax layer after the AE's encoder network, as shown in Fig. 4. For the binary intrusion detection problem, a single output unit is sufficient to represent the network behavior with normal and abnormal (0: normal and 1: abnormal). For the multiattack classification problem, the number of output

Algorithm 1 SCPP Algorithm

Input: Pruning rate p_{prune}
Number of input features n_x
Number of attack labels n_y
Number of hidden layers of autoencoder model n_l

First step: Compute input feature's score

Rank input feature R_{x_i} using the training dataset
for $i = 1 \rightarrow n_x$ **do**
  for $j = i \rightarrow n_x$ **do**
    $$\rho(x_i, x_j) = \frac{\sum_{k=1}^{n}(R_{x_i}^{(k)} - \mu_{R_{x_i}})(R_{x_j}^{(k)} - \mu_{R_{x_j}})}{\sigma(R_{x_i})\sigma(R_{x_j})}$$
  end for
  for $k = 1 \rightarrow n_y$ **do**
    Compute $\rho(x_i, y_k)$
  end for
end for
for $i = 1 \rightarrow n_x$ **do**
  $$s(x_i) = \frac{1}{n_x + n_y}\left(\sum_{j=1}^{n_x}\left|\rho(x_i, x_j)\right| + \sum_{k=1}^{n_y}\left|\rho(x_i, y_k)\right|\right)$$
  ▷ Input features' score
end for

Second step: Compute pruning probability for input features

$$\Delta = \frac{2 \times \min(p_{prune}, 1 - p_{prune})}{n_x - 1}$$
for $i = 1 \rightarrow n_x$ **do**
  $$p(x_i) = p_{prune} + \Delta\left(R_{s(x_i)} - \frac{n_x + 1}{2}\right)$$
end for

Third step: Construct the pruned AE model

Prune connections from feature x_i with probability $p(x_i)$
Construct mask matrix M_0 between input to first hidden layer
Derive connection mask for encoder
for $k = 1 \rightarrow \frac{n_l - 1}{2}$ **do**
  for $j = 1 \rightarrow n_h^{(k)}$ **do**
    $$p(h_j^{(k)}) = \frac{1}{n_h^{(k-1)}}\sum_{i=1}^{n_h^{(k-1)}} M_{k-1}[i, j]$$
    Prune connections from neuron j in layer k to layer $k+1$ with probability $p(h_j^{(k)})$
  end for
  Derive M_k
end for
Derive connection mask for decoder
for $k = \frac{n_l + 1}{2} \rightarrow n_l$ **do**
  $M_k = M_{n_l - k}$ ▷ symmetric pruning
end for
return The pruned autoencoder model

units is equal to the number of attack types where each output unit implies the probability for a specific attack to appear. The parameter training in the classifier consists of three following steps.

1) *Step 1:* Initialize the weights across hidden layers by using the pretrained weights of the encoder network.
2) *Step 2:* Freeze the weights in hidden layers of the classifier and train only the softmax layer.
3) *Step 3:* Fine-tune the weights of the whole deep neural network.

Fig. 5. Visualization of images in the MNIST data set [30].

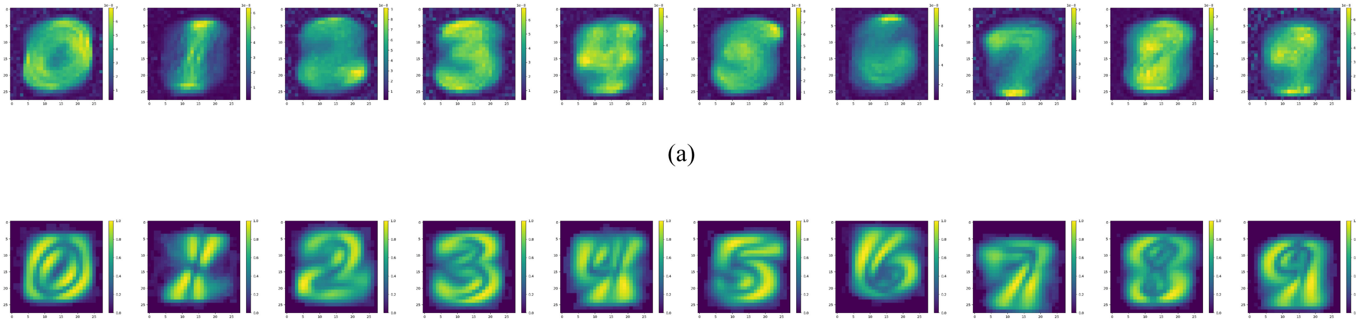
IV. Experiments

We evaluate the feasibility of our network intrusion detection algorithm on two real-world network traffic data sets: 1) UNSW-NB15 [28] and 2) CICIDS [29]. We first validate the effectiveness of the encoder network of our pruned AE architecture by quantifying the reconstruction error. We select three state-of-the-art pruning algorithms based on their novelty and popularity: 1) Molchanov's algorithm [23]; 2) Han's algorithm [24]; and 3) NISP algorithm [25]. It should be noted that the proposed pruning algorithm is designed for resource-limited edge devices (e.g., FPGA-based routers or gateways, or IoT devices), which cannot support computationally intensive operations. To stress out this aspect, we validate NIDS models in terms of the number of parameters and computation operations.

We apply our pruning algorithm and the counterpart algorithms to an original fully connected AE model, in which the former encoder network consists of 100, 50, 20 neurons across three hidden layers for UNSW-NB15, and 60, 30 neurons across two hidden layers for CICIDS, in order to evaluate the resulting pruning quality. The above-mentioned architecture, with the highest result on the validation set among different network architectures of the AE model, is selected.

We first show how the SCPP pruning algorithm can keep the inherent patterns of images on the handwritten digit recognition task using the MNIST data set. Then, once the unsupervised feature of our pruning technique is validated, we investigate the prediction accuracy of network intrusion classification. We show the accuracy of predicting the abnormality of network traffic with two classes, i.e., whether a network traffic trace turns out to be normal or not. Our NIDS algorithm that has been trained with two labels is compared against a state-of-the-art two-label NIDS classifier [8] based on multidistributed variational AE (MVAE).

Beyond the binary anomaly detection, we validate more detailed classification performance of inferring the correct anomaly type with a multiclass NIDS classifier. Our classifier



(a)

(b)

Fig. 6. Visualization of the importance scores for images with a specific label based on a derivation method (a) and our SCPP algorithm (b), respectively.

is compared against a fully connected neural network-based classifier as an upper bound performance baseline and a two-stage deep learning model (TSDL) [26] with an advanced AE network in NIDS.

A. Experimental Setting

We used two representative network traffic traces of UNSW-NB15 and CICIDS data sets. The UNSW-NB15 data set includes real normal and synthetic abnormal network traffic traces during a 16-h experimental period, consisting of nine attack classes. For our evaluation, the full data set was divided into a training data set of 175 341 samples and a test data set of 82 332 samples, and one third of the training data set is used as the validation data set.

The CICIDS data set covers normal network activities and common network attacks with 14 different types, which were collected during five days in 2017. The entire data set contains more than 3 million samples and 78 recorded features, and is divided into the training, validation, and test data sets with the ratio of 6:2:2. According to [30] and [31], in which 20% samples are recommended to be used as the test set, we have randomly selected 20% instances as the validation set for the selection of the good hyper-parameters, while the remaining 60% data are used to train network parameters.

To preprocess the traffic features, the nominal features are first converted to binary values using the one-hot encoding technique. Then, the min–max normalization is applied to bound the absolute input values to be less than or equal to 1. The autoencoder model consists of a certain number of hidden layers where the middle hidden layer contains the representative traffic features. After training the AE model, the representative features are extracted and fed into the softmax output layer in the classification model. The number of units in the encoder of the AE model is fixed to [100, 50, 20] and [60, 30] for UNSW-NB15 and CICIDS data sets, respectively, while the number of softmax units is equal to the number of traffic classes. We implemented our NIDS algorithm and other counterpart algorithms in a desktop PC with Intel Core i7-9700 3GHz CPU (with no GPU support) and 16 GB RAM, which are comparable to a normal edge device specification, with TensorFlow 1.15.0 on 64-bit Windows 10 OS to evaluate the detection and classification performance of the pruning methods. There are some commercial devices at the edge with the similar hardware configuration: NVIDIA Jetson AGX Xavier, Jetson Xavier NX, Jetson Nano, Cisco IC3000 Industrial Compute Gateway, Dell Edge Gateway Model 5100 (industrial version), and Industrial Smart IoT Edge Computing Gateway.

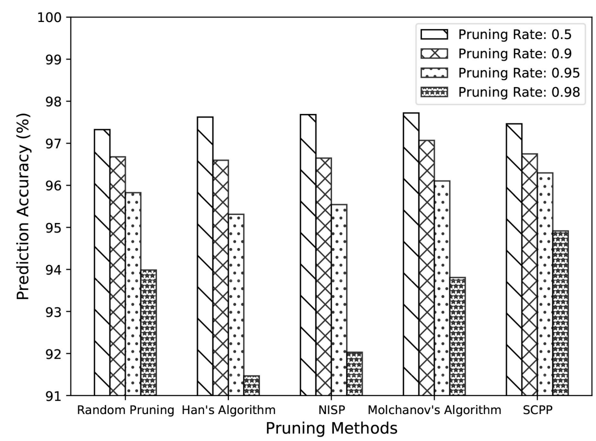


Fig. 7. Effect of the pruning methods on MNIST classification performance.

B. Pruning Effect on Feature Extraction

To get a glimpse of the effectiveness of our proposed SCPP pruning algorithm in a visual way, we illustrate the importance scores of features for images in the MNIST data set, as shown in Fig. 5. Since the images with different labels retain unique features, we validate how well a pruning algorithm avoids losing the innate characteristics within a feature.

We calculate the important scores of the input features with the same size of 28×28 with the MNIST images and visualize them for each label from the left to right side of Fig. 6. As can be compared in Fig. 6(a) and (b), the SCPP algorithm extracts and highlights the key patterns in a more clear contrasting manner, in particular for the cases of labels 6, 8, and 9. This result implies that our SCPP algorithm is good at keeping the core features after pruning some unimportant features, without requiring a pretraining AE model.

We validate how choosing a different pruning rate affects classification performance over different pruning methods on the MNIST data set. We construct an AE model with a hidden layer of 300 neurons to learn the latent vectors from the

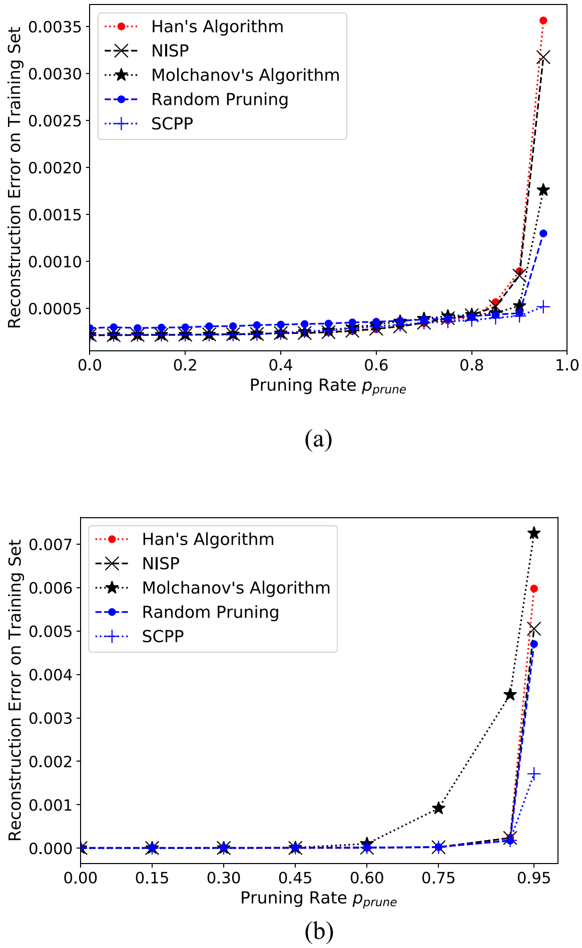


Fig. 8. Effectiveness of pruning algorithms in terms of reconstruction error on the training set by varying the pruning rate. (a) UNSW-NB15 training set. (b) CICIDS training set.

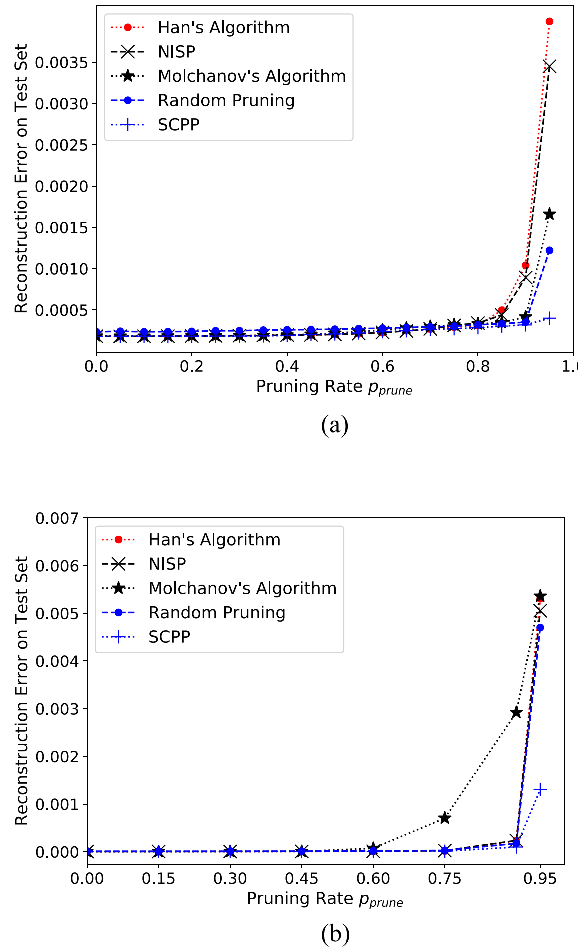


Fig. 9. Effectiveness of pruning algorithms in terms of reconstruction error on the test set by varying the pruning rate. (a) UNSW-NB15 test set. (b) CICIDS test set.

MNIST images. Then, the extracted vector is used to classify the images into 10 different classes. As shown in Fig. 7, our pruning algorithm achieves similar performance to the others with the low pruning rates (e.g., 0.5) and higher performance for higher pruning rates beyond 0.5. In particular, in case of the pruning rates of 0.98, our SCPP algorithm further improves the classification accuracy by 1 to 3.5% compared to others. The SCPP results on the handwritten digit classification task implicitly implies that the proposed SCPP pruning scheme can work well for a general machine learning problem. The rest of this section is devoted for analyzing experimental performance of SCPP and other pruning methods on the intrusion detection task.

C. Feature Extraction Performance

We measure the reconstruction error that calculates the mean squared error between the input samples and their reconstructed output samples after pruning, on both UNSW-NB15 and CICIDS data sets, as shown in Figs. 8 (for training sets) and 9 (for test sets). We vary the pruning rate p_{prune} in the range of $[0, 1]$, and report the average performance over 5 experiment runs. We compare our SCPP pruning algorithm to other pruning counterpart algorithms of Molchanov's, Han's, and NISP algorithms together with a naive random pruning approach that randomly prunes a neuron with a probability of p_{prune}.

As indicated in Fig. 8(a) and (b) on both data sets, as the pruning rate p_{prune} increases from 0 to 0.95, our SCPP algorithm increases very slowly from 2.14×10^{-4} to 5.18×10^{-4} with a factor of 2.42 on the UNSW-NB15 data set. On the other hand, the reconstruction error of the other pruning algorithms: random pruning, Molchanov's, Han's, and NISP increases steeply with a factor of 4.47, 8.03, 16,65, and 14.69, respectively, as in Fig. 8(a). We verify the similar result on the CICIDS training set, as shown in Fig. 8(b) and the test sets of both data sets in Fig. 9.

This result demonstrates that our probabilistic feature extraction provides an effective way of dropping some unimportant connections in the fully connected neural network even at a very high pruning rate of 0.95, for example. An AE network pruned by our approach reconstructs the input with a relatively smaller error. This implies that using only a small portion of neurons via our approach still provides a stable unsupervised learning performance with the smaller computation and memory usage.

In that Molchanov's, Han's and NISP algorithms perform pruning in such a deterministic way that the connections with

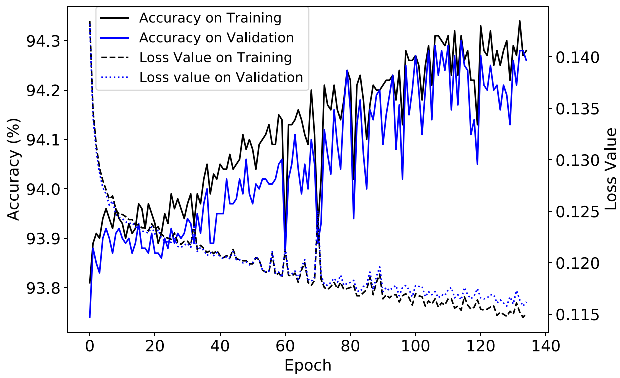


Fig. 10. Loss and prediction accuracy of our SCPP-based two-label classifier over epoch on the UNSW-NB15 data set.

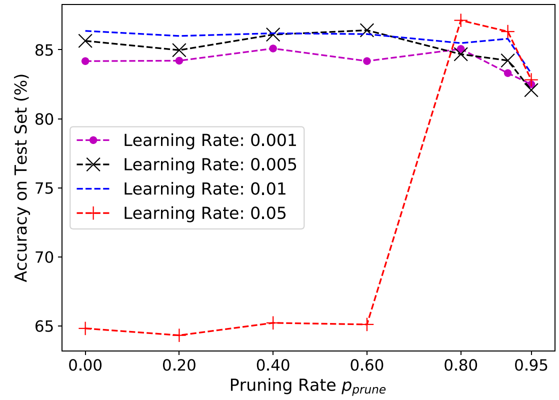


Fig. 11. Effect of the learning rate on two-label prediction accuracy in our SCPP algorithm with respect to pruning rate on the UNSW-NB15 test set.

the lower importance scores are completely removed from the network even though they may have interconnected with their prior or subsequent connections with the higher importance scores. It is interesting to see that our approach and the random pruning that both prunes neurons in a probabilistic manner are more effective in feature extraction than the other deterministic pruning techniques. This implies that a pruning algorithm with reliable feature extraction can be applied to an even more deep and wide neural network under the same learning and inference time constraint.

D. Network Intrusion Detection Performance

We evaluate the network intrusion detection performance for two classification problems: 1) two-label classification with normal and abnormal and 2) multilabel classification with attack types.

1) Two-Label Classification: We first show the learning curves of our SCPP algorithm in the accuracy and loss dynamics for two-label classification on the UNSW-NB15 data set, as shown in Fig. 10. From the beginning to epoch 40 or around, the SCPP-based two-label classifier network gets trained quickly and efficiently. When there is no further improvement in accuracy of the validation set for the last 20 epochs, we stop the parameter training process. We fine-tune training parameters with various pruning rate and learning rate, as shown in Fig. 11. The detection accuracy highly depends on both the learning rate and the pruning rate. Specifically, the learning rate higher than 0.01 results in unstable and low detection performance, especially with the pruning rate lower than 0.8. Moreover, the accuracy performance generally decreases in case that the pruning rate higher than 0.8 due to the lack of network parameters. Since the learning rate of 0.01 produces the highest results among possible values, we have selected 0.01 as the default value for the learning rate of the classification model.

Then, in order to compare our algorithm with a state-of-the-art two-label classifier, MVAE with feature extraction in NIDS [8], we collect the area-under-curve (AUC) score which considers both true-positive and false-positive. This is due to the fact that the score measure considers both detection quality and specificity at various threshold settings. As shown in Table II, our SCPP-based algorithm outperforms

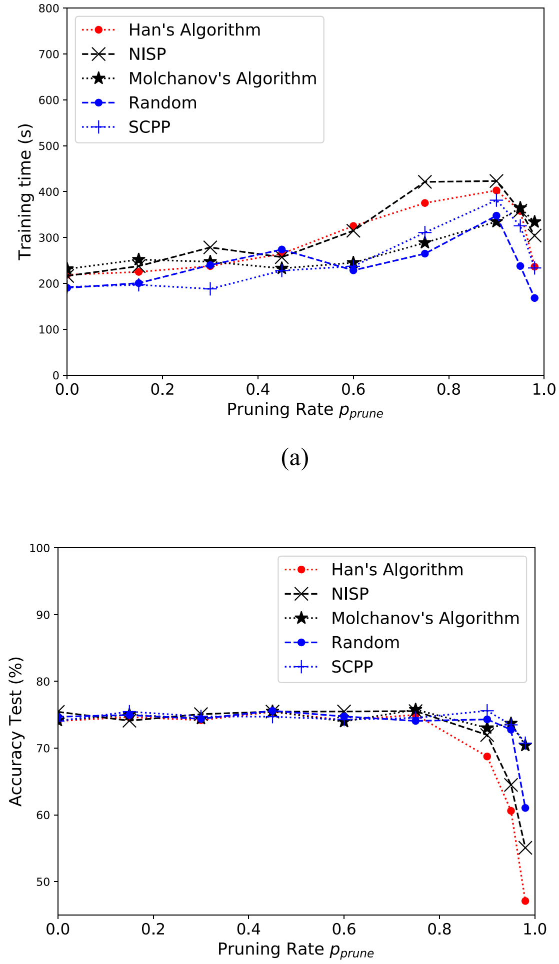
TABLE II
PERFORMANCE COMPARISON AMONG TWO-CLASS CLASSIFIERS ON THE UNSW-NB15 DATA SET

Models	**AUC Score**	**#Param**	**Memory (KB)**	**#FLOPs**
MVAE w/ Naive Bayes	0.928	25,812	100.83	25,640
MVAE w/ SVM	0.945	25,791	100.75	25,620
MVAE w/ Decision Tree	0.954	25,790	100.74	25,600
MVAE w/ Random Forest	0.961	25,870	101.05	25,600
AE w/ Random Forest	0.900	25,870	101.05	25,600
SCPP w/ $p_{prune} = 0$	0.963	25,791	100.75	25,620
SCPP w/ $p_{prune} = 0.5$	0.965	12,991	50.74	12,820
SCPP w/ $p_{prune} = 0.8$	0.962	5,311	20.74	5,140
SCPP w/ $p_{prune} = 0.95$	0.935	1,471	5.75	1,300

all of other counterpart algorithms, which combines various machine learning techniques with the MVAE-based underlying feature extraction. Observing the performance of our SCPP algorithm with different pruning rates, we demonstrate that the SCPP-based two-label classifier can substantially reduce a large portion of parameters from 80% up to 95%, without a significant sacrifice of the classification performance.

We also compare the SCPP-based classification models with the existing methods in terms of computation and memory overhead. Specifically, the memory usage to store network parameters and the number of FLOPs are used as comparison metrics. It is assumed that four bytes are used to store each parameter. It should be noted that, without neuron pruning, the fully connected classifier with MVAE has the model complexity similar to the proposed SCPP method with the pruning rate of 0. Even though MVAE and AE use different cost functions, they have the same architecture. When the pruning rate increases, our SCPP scheme can yield even lower model complexity than MVAE to learn the latent representation of data traffic. For example, when the pruning rate is 0.8, both memory and complexity overhead of the SCPP-based classification model are reduced by almost 80%.

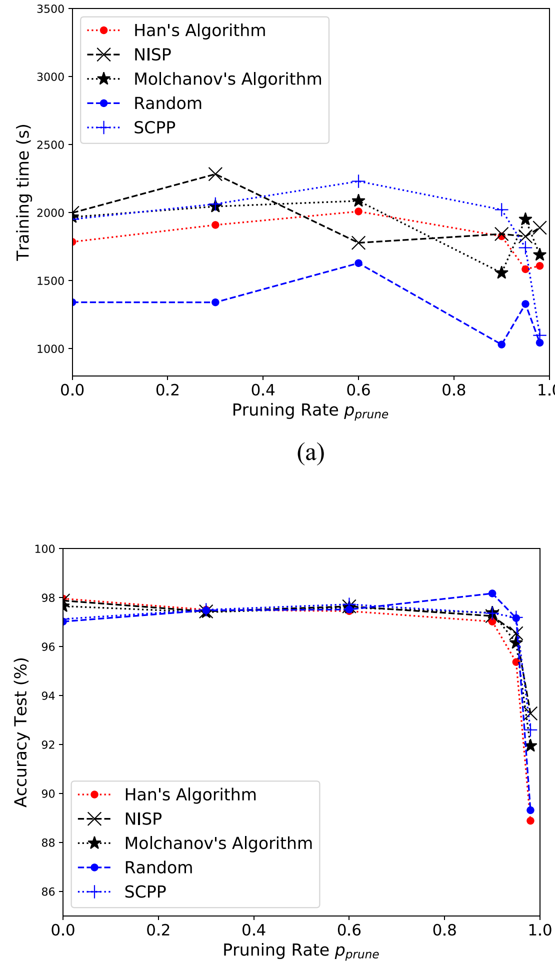
2) Multilabel Classification: In order to show the lightweight benefits of SCPP over other pruning methods, we first conduct performance comparison between pruning algorithms in terms of training time, inference time, and multiclass classification accuracy. Then, we measure and compare architecture complexity and accuracy among the SCPP algorithm, the fully connected neural network, and TSDL model. Finally, research discussion is made on the possible improvement of the classification performance of the proposed method



(a)

(b)

Fig. 12. Training time and classification accuracy depending on the pruning algorithm on the UNSW-NB15 data set. (a) Training time. (b) Classification accuracy.



(a)

(b)

-1

Fig. 13. Training time and classification accuracy depending on the pruning algorithm on the CICIDS data set. (a) Training time. (b) Classification accuracy.

TABLE III
AVERAGE PERFORMANCE OF PRUNING ALGORITHMS ON UNSW-NB15

	Accuracy (%)	Training time (s)	Inference time (s)
Han	69.32	293.70	0.068
NISP	71.40	312.39	0.069
Molchanov	73.99	281.1	0.070
Random	72.95	239.08	0.075
SCPP	74.17	254.88	0.073

TABLE IV
AVERAGE PERFORMANCE OF PRUNING ALGORITHMS ON CICIDS

	Accuracy (%)	Training time (s)	Inference time (s)
Han	95.70	1786.93	0.43
NISP	96.66	1936.13	0.42
Molchanov	96.35	1882.64	0.47
Random	96.11	1285.46	0.43
SCPP	96.58	1851.02	0.43

TABLE V
PERFORMANCE COMPARISON FOR MULTIATTACK CLASSIFICATION WITH FULLY CONNECTED NN, TSDL ON UNSW-NB15

	No. of parameters	No. of FLOPs	Accuracy (%)
TSDL [27]	31,031	61,710	76.48
Fully-Connected NN	25,980	51,600	73.37
SCPP (p_{prune} = 0.5)	13,080	25,890	74.63
SCPP (p_{prune} = 0.9)	2,760	5,322	74.65
SCPP (p_{prune} = 0.95)	1,470	2,751	73.09

TABLE VI
PERFORMANCE COMPARISON FOR MULTIATTACK CLASSIFICATION WITH FULLY CONNECTED NN, TSDL ON CICIDS

	No. of parameters	No. of FLOPs	Accuracy (%)
TSDL [27]	8,616	16,970	98.92
Fully-Connected NN	7,035	13,860	98.63
SCPP (p_{prune} = 0.5)	3,570	6,983	98.02
SCPP (p_{prune} = 0.9)	798	1,481	97.31
SCPP (p_{prune} = 0.95)	452	793	97.08

by analyzing the data distribution and accuracy on each attack label.

First, Figs. 12 and 13 show the comparisons of the average training time and classification accuracy with different pruning algorithms by varying the pruning rate from 0 to 0.98 on the UNSW-NB15 and CICIDS data sets, respectively. Since network parameters are randomly initialized, to make fair comparison between the pruning algorithms, we collect and present the average training time and classification accuracy over 10

TABLE VII
DISTRIBUTION OF ATTACK TYPES IN THE TRAINING AND TEST SETS OF UNSW-NB15

Class name	Training size	Training distribution (%)	Test size	Test distribution (%)
Analysis	2,000	1.14	677	0.82
Backdoor	1,746	1	583	0.71
DoS	12,264	6.99	4,089	4.97
Exploit	33,393	19.04	11,132	13.52
Fuzzers	18,184	10.37	6,062	7.36
Generic	40,000	22.81	18,871	22.92
Normal	56,000	31.94	37,000	44.94
Reconnaissance	10,491	5.98	3,496	4.25
Shellcode	1,133	0.65	378	0.46
Worms	130	0.07	44	0.05
Total	173,341	100	82,332	100

TABLE VIII
CONFUSION MATRIX ON THE TEST SET OF UNSW-NB15

	Anal.	Back.	DoS	Expl.	Fuzz.	Gene.	Norm.	Reco.	Shell.	Worm.	Total	Accuracy (%)
Anal.	**1**	0	23	638	11	0	4	0	0	0	677	0.15
Back.	0	**26**	23	507	15	0	2	2	8	0	583	4.46
DoS	0	39	**174**	3,642	132	8	19	29	46	0	4,089	4.26
Expl.	16	48	120	**10,146**	445	2	80	111	162	2	11,132	91.14
Fuzz.	0	25	48	1,444	**3,126**	0	965	156	298	0	6,062	51.57
Gene.	0	11	57	460	141	**18,142**	34	5	18	3	18,871	96.14
Norm.	306	63	24	1,161	7,426	1	**27,507**	391	120	1	37,000	74.34
Reco.	0	8	14	645	44	4	58	**2,652**	71	0	3,496	75.86
Shell.	0	15	0	34	41	0	7	68	**213**	0	378	56.35
Worm.	0	1	0	30	5	0	0	0	1	**7**	44	15.9
Total	323	236	483	18,707	11,386	18,157	28,676	3,414	937	13	82,332	Average: 75.29%

running times for each pruning rate. Generally, the classification accuracy clearly decreases when the pruning rate is greater than 0.8. This observation is attributed by the fact that the large pruning rate causes the lack of network parameters, and thus, the classification model becomes underfit to the data samples.

In order to indicate the overall performance of pruning algorithm over different p_{prune} values, we take an average of classification accuracy, training time, and inference time over pruning rate values as shown in Tables III and IV. In practice, a specific pruning rate should be used; however, for the purpose of conducting extensive and quantitative experiments, we measure how well each pruning algorithm shows dynamic performance over a variety of pruning rates, requiring different computation and memory overhead. With the UNSW-NB15 data set, our SCPP achieves the highest average classification accuracy (i.e., 74.17%), while requiring relatively lower training time, thus becoming more feasible to edge devices than the other methods. On the CICIDS data set, the highest classification accuracy (96.66% on average) belongs to the NISP method. Meanwhile, our SCPP algorithm produces the second-highest accuracy with the 85-s reduction of training time (i.e., 4.4%) compared to the NISP method. The random pruning method consumes the lowest training time on both data sets for parameters learning, but the classification performance is less than SCPP and Molchanov's algorithms. With regard to the inference time for all test samples, the pruning algorithms consume similar amount of time to recognize an attack type of incoming traffic on both data sets.

In summary, beside the lower reconstruction error than the other pruning methods, our SCPP algorithm yields relatively higher classification accuracy with the lower training time than the deterministic pruning schemes. Since edge devices are equipped with some more limited computing and memory resource, it is more beneficial to use the proposed SCPP scheme when deploying a network defense system on the edge.

Second, we validate the multiattack classification performance by comparing our algorithm against a fully connected neural network and TSDL [26] on both data sets as shown in Tables V and VI. TSDL consists of two consecutive submodels. The first submodel trains an AE network from traffic features and then the condensed features are fed into a sigmoid output layer to learn the intrusion probability value. This probability value and traffic features provided by the data set are later used to construct another AE network. In the second submodel, the abstract encoded features extracted from the second AE are connected to an softmax output layer to classify attack types of data traffic.

As can be seen in Tables V and VI, with some small sacrifice in classification accuracy of 1.9%–3.4%, the SCPP algorithm is much more lightweight than TSDL. For example, in UNSW-NB15, when the pruning rate is set to 0.9, our SCPP method can reduce parameters and FLOPs by a factor of 11.2 and 11.6, respectively, with less than 2% accuracy reduction. The reason that TSDL offers a slightly higher accuracy is that TSDL is based on a relatively more complex two serial submodels where the second submodel leverages the intrusion probability learnt from the first submodel. Our SCPP algorithm, on the other hand, only considers one AE network together with neuron pruning, resulting in considerably fewer parameters and FLOPs than TSDL. Therefore, a symmetric AE network consisting of an encoder and a decoder turns out to be an effective architecture for extracting the feature representation.

The results indicate that our SCPP-based NIDS algorithm produces a similar predictive performance, while using considerably fewer parameters. For example, with $p_{\text{prune}} = 0.95$ where 95% of the weight connections are pruned, our

TABLE IX
DISTRIBUTION OF ATTACK TYPES IN THE TRAINING AND TEST SETS OF CICIDS 2017

Class name	Training size	Training distribution (%)	Test size	Test distribution (%)
Normal	1,134,810	80.31	567,415	80.31
Botnet	978	0.069	489	0.069
DDoS	64,012	4.53	32,007	4.53
DoS GoldenEye	5,146	0.36	2,574	0.36
DoS Hulk	115,062	8.14	57,531	8.14
DoS Slow HTTP	2,748	0.19	1,377	0.19
DoS Slow Loris	2,898	0.21	1,449	0.21
FTP	3,966	0.28	1,986	0.28
Heart bleed	4	0.00028	5	0.00071
Infilt	18	0.0013	9	0.0013
PortScan	79,398	5.62	39,701	5.62
SSH	2,948	0.21	1,475	0.21
Web Attack Brute Force	752	0.053	379	0.054
Web Attack Sql Inject	10	0.00071	6	0.00085
Web Attack XSS	326	0.023	163	0.023
Total	1,413,076	100	706,566	100

TABLE X
CONFUSION MATRIX ON THE TEST SET OF CICIDS

	Norm.	Botn.	DDoS	DoSG.	DoSH.	DSH.	DSL.	FTP	Hear.	Infi.	Port.	SSH.	WABF.	WASI.	WAX.	Total	Accuracy(%)
Norm.	563,423	7	30	13	1,890	329	26	20	0	1	1,675	1	0	0	0	567,415	99.3
Botn.	489	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	489	0
DDoS	1,047	0	**29,510**	0	1,450	0	0	0	0	0	1	0	0	0	0	32,007	92.2
DoSG.	297	0	0	**2,269**	3	3	2	0	0	0	0	0	0	0	0	2,574	88.15
DoSH.	15,362	0	0	0	**42,141**	0	0	0	0	0	0	0	0	0	0	57,531	73.25
DSH.	310	0	0	1	0	**1,062**	4	0	0	0	0	0	0	0	0	1,377	77.12
DSL.	432	0	0	1	0	175	**836**	5	0	0	0	0	0	0	0	1,449	57.69
FTP.	127	0	0	0	0	0	0	**1,859**	0	0	0	0	0	0	0	1,986	93.61
Hear.	4	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	5	20
Infi.	9	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	9	0
Port.	240	0	0	0	9	0	0	0	0	0	**39,452**	0	0	0	0	39,701	99.37
SSH.	728	0	0	0	2	0	0	6	0	0	0	**745**	0	0	0	1,475	50.51
WABF.	361	0	0	0	0	0	0	0	0	0	0	2	**18**	0	0	379	4.75
WASI.	6	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	6	0
WAX.	162	0	0	1	0	0	0	0	0	0	0	0	0	0	**0**	163	0
Total	582,997	7	29,540	2,285	45,495	1,569	868	1,884	1	1	41,155	746	18	0	0	706,566	Average: 96.43%

TABLE XI
PRECISION, RECALL, AND F1 PERFORMANCE ON THE TEST SET OF UNSW-NB15

	Precision	Recall	F1
Anal.	0.0031	0.0015	0.002
Back.	0.1102	0.0446	0.0635
DoS	0.3602	0.0426	0.0762
Expl.	0.5424	0.9114	0.6801
Fuzz.	0.2745	0.5157	0.3583
Gene.	0.9992	0.9614	0.9799
Norm.	0.9592	0.7434	0.8376
Reco.	0.7768	0.7586	0.7676
Shell.	0.2273	0.5635	0.3239
Worm.	0.5385	0.159	0.2455

TABLE XII
PRECISION, RECALL, AND F-1 PERFORMANCE ON THE TEST SET OF CICIDS

	Precision	Recall	F-1
Norm.	0.9664	0.993	0.9795
Botn.	0	0	N/A
DDoS	0.999	0.922	0.959
DoSG.	0.993	0.8815	0.934
DoSH.	0.9262	0.7325	0.8180
DSH.	0.6769	0.7712	0.721
DSL.	0.9631	0.5769	0.7216
FTP.	0.987	0.9361	0.9609
Hear.	1	0.20	0.3333
Infi.	0	0	N/A
Port.	0.9586	0.9937	0.9758
SSH.	0.9987	0.5051	0.671
WABF.	1	0.0475	0.091
WASI.	0	0	N/A
WAX.	0	0	N/A

algorithm reduces the number of parameters, and the number of FLOPs by a factor of $25\,980/1470 \approx 17.67$ and $51\,600/2751 \approx 18.76$, respectively, on the UNSW-NB15 data set, and by a factor of $7035/452 \approx 15.56$ and $13\,860/793 \approx 17.48$, respectively, on the CICIDS data set, while achieving almost similar prediction quality, compared to the fully connected neural network. Meanwhile, the number of parameters and FLOPs in our SCPP algorithm is equal to $1/21.11$ and $1/22.43$ that of TSDL with $p_{\text{prune}} = 0.95$ in UNSW-NB15. In cases of CICIDS, these numbers are $1/19.06$ and $1/21.40$ for parameters and FLOPs, respectively.

Reducing the number of FLOPs by a factor of roughly 19 technically means that the edge devices with SCPP can inspect data packets 19 times higher than the fully connected classification model. For example, assume that an edge device (e.g., industrial smart IoT edge computing gateway by EtherWan systems company) has a speed of 1.35 GHz and spends 20 clock cycles on average for each FLOP. In SCPP, the maximum number of packets to be inspected by this edge device is $(1.35 \times 10^9)/(2,751 \times 20) = 24\,736$. If there are 1000 bytes per packet on average, the maximum data rate can be inspected is $24\,736 \times 1000 \times 8 = 187.2$ Mb/s. Meanwhile, if using the fully connected classification model, the mentioned edge device can only process maximum $187.2/18.75 = 9.98$ Mb/s. Therefore, the proposed SCPP algorithm allows NIDS to be implemented on the edge device in networks with a relatively larger volume of traffic.

Finally, in order to give insights into multiclass performance, we present label distribution, confusion matrices on the test set of both data sets as shown in Tables VII, VIII, IX, and X. Please note that both data sets are highly biased in terms of the number of samples in each different group as can be seen in Tables VII and IX. As a result, performance in some certain classes with majority samples is expected to be much higher than others. Tables VIII and X show the confusion matrix of the classification model with SCPP ($p_{\text{prune}} = 0.5$) on the test sets

TABLE XIII
CONFUSION MATRIX ON THE TEST SET OF UNSW-NB15 AFTER UPSAMPLING MINORITY CLASSES

	Anal.	Back.	DoS	Expl.	Fuzz.	Gene.	Norm.	Reco.	Shell.	Worm.	Total	Accuracy (%)
Anal.	**170**	368	76	51	2	0	9	1	0	0	677	25.11
Back.	129	**359**	63	6	6	0	1	9	9	1	583	61.57
DoS	862	1,569	**748**	557	57	12	66	62	105	51	4,089	18.29
Expl.	946	1,858	1,081	**5,440**	261	8	265	560	372	341	11,132	48.86
Fuzz.	328	877	164	107	**1,710**	0	1,898	231	724	23	6,062	28.20
Gene.	20	42	94	329	41	**18,172**	36	21	80	36	18,871	96.29
Norm.	1,877	33	103	169	3,335	2	**30,330**	432	674	45	37,000	81.97
Reco.	85	175	76	22	25	4	75	**2,885**	125	24	3,496	82.52
Shell.	0	0	0	3	10	0	12	52	**300**	1	378	79.36
Worm.	0	0	0	1	1	0	1	1	6	**34**	44	77.27
Total	4,417	5,281	2,405	6,685	5,448	18,198	32,693	4,254	2,395	556	82,332	Average: 73.05%

where average classification accuracy is 75.29% and 96.43% with UNSW-NB15 and CICIDS, respectively. Note that the value at row i and column j indicates the number of samples that belong to the attack label i are predicted as class j. In Table VIII, some classes with high performance include Exploit, Generic, Normal, whereas the classification model does not well detect certain attach types, such as Analysis, Backdoor, DoS, and Worms. Tables XI and XII explicitly show precision, recall, and F1 scores that can be computed from the confusion matrix.

We believe that the performance difference among the attack classes is greatly related to the imbalance problem of the data set since the classification model tends to be trained such that the output mostly belongs to a majority group. According to [32] to address the imbalance issue, we can apply resampling the training data set (e.g., upsampling the minority instances or downsampling the majority ones) or modify the classification model (e.g., the cost function, threshold value, one-class learning).

We have applied the random upsampling on some minority classes such that each attack type in the training set has more than 10 000 samples. After training the classification model with the newly created training data set, the confusion matrix on the test set of UNSW-NB15 is collected and presented in Table XIII. There is a significant increase in classification accuracy for the minority classes, e.g., accuracy for Analysis, Backdoors, and DoS gains by around 25%, 57%, and 14%, respectively, compared to the case without the resampling method. However, the performance rather decreases in some classes (i.e., Exploits and Fuzzers) due to a possible lack of physically different samples for these classes. In summary, it is important to have enough actual data samples to construct a classification model with high classification performance. Thus, when deploying NIDS in practice, we should frequently collect samples especially for minority classes to update network parameters and to enhance classification accuracy.

V. CONCLUSION

To construct a robust yet efficient NIDS, we have proposed a neural network architecture that consists of a stacked AE with feature extraction and neuron pruning, and a following classifier network. By tightly coupling feature extraction and neuron pruning, a bare AE network has effectively been sparsified, leaving only effective neurons and edge connections among them. We have verified the effectiveness of our pruned AE network with feature extraction in a unsupervised manner on two real-world network traffic data sets.

Once our condensed AE network has shown a higher reconstruction quality compared to several state-of-the-art pruning approaches, we have extended the AE network to an intrusion detection system architecture by connecting the pretrained AE network to a classifier network. We have demonstrated that our algorithm for both two-label traffic abnormality detection and multilabel attack type classification outperforms the state-of-the-art algorithms, or shows the similar performance compared with them, in terms of classification quality, while significantly reducing the number of parameters and the number of operations with a factor of up to 19. This result implies that our lightweight NIDS architecture is well-fit to edge devices with a small computation and memory usage.

For future work, it would be interesting to implement our lightweight network intrusion detection architecture directly on an FPGA-based dedicated switching hardware (e.g., using the P4 framework), in order to support the data plane programmability for fast traffic inspection and response in the packet switching level.

REFERENCES

[1] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," 2019. [Online]. Available: arxiv.abs/1903.02460.

[2] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: Techniques, datasets and challenges," *Cybersecurity*, vol. 2, p. 20, Jul. 2019.

[3] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, and K. Wehrle, "Towards executing computer vision functionality on programmable network devices," in *Proc. 1st ACM CoNEXT Workshop Emerg. In-Netw. Comput. Paradigms*, 2019, pp. 15–20.

[4] N. Moustafa, B. Turnbull, and K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4815–4830, Jun. 2019.

[5] Y. Yang, K. Zheng, C. Wu, and Y. Yang, "Improving the classification effectiveness of intrusion detection by using improved conditional variational autoencoder and deep neural network," *Sensors*, vol. 19, no. 11, p. 2528, Jun. 2019.

[6] V. L. Cao, M. Nicolau, and J. McDermott, "Learning neural representations for network anomaly detection," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 3074–3087, Aug. 2019.

[7] A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow, "Adversarial autoencoders," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–9.

[8] L. Vu, V. L. Cao, N. Q. Uy, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz, "Learning latent distribution for distinguishing network traffic in intrusion detection system," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.

[9] S. Hosseini and M. Azizi, "The hybrid technique for DDoS detection with supervised learning algorithms," *Comput. Netw.*, vol. 158, pp. 35–45, Jul. 2019.
[10] H. Mostafa and X. Wang, "Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 1–15.
[11] Y. Xiao and X. Xiao, "An intrusion detection system based on a simplified residual network," *Information*, vol. 10, no. 11, p. 356, Nov. 2019.
[12] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," 2018. [Online]. Available: arxiv.abs/1805.12185.
[13] X. Xiao, Z. Wang, and S. Rajasekaran, "AutoPrune: Automatic network pruning by regularizing auxiliary parameters," in *Proc. 32nd Adv. Neural Inf. Process. Syst.*, 2019, pp. 13681–13691.
[14] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," in *Proc. NeurIPS*, 2019, pp. 2130–2141.
[15] X. Gao, Y. Zhao, L. Dudziak, R. Mullins, and C. Z. Xu, "Dynamic channel pruning: Feature boosting and suppression," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–6.
[16] B. Su and Y. Wu, "Learning low-dimensional temporal representations," in *Proc. 35th Int. Conf. Mach. Learn.*, Jul. 2018, pp. 10–15.
[17] M. Ye and Y. Sun, "Variable selection via penalized neural network: A drop-out-one loss approach," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, Jul. 2018, pp. 10–15.
[18] M. T. Law, J. Snell, A. M. Farahmand, R. Urtasun, and R. S. Zemel, "Dimensionality reduction for representing the knowledge of probabilistic models," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–9.
[19] Y. Bartal, N. Fandina, and O. Neiman, "Dimensionality reduction: Theoretical perspective on practical measures," in *Proc. 32nd Adv. Neural Inf. Process. Syst.*, 2019, pp. 10576–10588.
[20] M. Courbariaux and Y. Bengio, "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1," 2016. [Online]. Available: arxiv.abs/1602.02830.
[21] T. Simons and D.-J. Lee, "A review of binarized neural networks," *Electronics*, vol. 8, no. 6, p. 661, Jun. 2019.
[22] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, Jul. 2018, pp. 10–15.
[23] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," 2016. [Online]. Available: arxiv.abs/1611.06440.
[24] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[25] R. Yu *et al.*, "NISP: pruning networks using neuron importance score propagation," 2017. [Online]. Available: arxiv.abs/1711.05908.
[26] F. A. Khan, A. Gumaei, A. Derhab, and A. Hussain, "A novel two-stage deep learning model for efficient network intrusion detection," *IEEE Access*, vol. 7, pp. 30373–30385, 2019.
[27] W. Daniel, *Applied Nonparametric Statistics* (The Duxbury Advanced Series in Statistics and Decision Sciences). Boston, MA, USA: PWS-Kent, 1990.
[28] N. Moustafa and J. Slay, "UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *Proc. IEEE Mil. Commun. Inf. Syst. Conf. (MilCIS)*, 2015, pp. 1–6.
[29] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proc. ICISSP*, 2018, pp. 1–9.
[30] A. Gron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 1st ed. New York, NY, USA: O'Reilly Media, 2017.
[31] G. Afshin, K. Vladik, and K. Olga, "Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation," Univ. Texas at El Paso, El Paso, TX, USA, Rep. UTEP-CS-18-09, Jan. 2018.
[32] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Handling imbalanced datasets: A review," *GESTS Int. Trans. Comput. Sci. Eng.*, vol. 30, no. 1, pp. 25–36, 2006.

Thi-Nga Dao received the B.S. degree in electrical and communication engineering from Le Quy Don Technical University, Hanoi, Vietnam, in 2013, the M.S. and Ph.D. degrees in computer engineering from the University of Ulsan, Ulsan, South Korea, in 2016 and 2019, respectively.

From July 2019, she has been working as a Lecturer with the Faculty of Radio-Electronic Engineering, Le Quy Don Technical University. Her research interests include machine learning-based applications in network security, network intrusion detection and prevention systems, human mobility prediction, and mobile crowdsensing.

HyungJune Lee (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2006 and 2010, respectively.

He joined Broadcom, San Jose, CA, USA, as a Senior Staff Scientist for working on research and development of 60-GHz 802.11ad SoC MAC. He also worked for AT&T Labs, Florham Park, NJ, USA, as a Principal Member of Technical Staff with the involvement of LTE overload estimation, LTE-WiFi interworking, and heterogeneous networks. He is currently an Associate Professor with Computer Science and Engineering Department, Ewha Womans University, Seoul. His current research interests include future wireless networks on the IoT, fog computing, VANET, and machine learning-driven network system design.