Robust, effective and resource efficient deep neural network for intrusion detection in IoT networks.

ZAKARIYYA, I., KALUTARAGE, H. and AL-KADRI, M.O.

2022

Robust, Effective and Resource Efficient Deep Neural Network for Intrusion Detection in IoT Networks

Idris Zakariyya, Harsha Kalutarage
School of Computing, Robert Gordon University
United Kingdom
{i.zakariyya,h.kalutarage}@rgu.ac.uk

M. Omar Al-Kadri
School of Computing and Digital Technology, Birmingham City University
United Kingdom
omar.alkadri@bcu.ac.uk

ABSTRACT

Internet of Things (IoT) devices are becoming increasingly popular and an integral part of our everyday lives, making them a lucrative target for attackers. These devices require suitable security mechanisms that enable robust and effective detection of attacks. Deep Neural Networks (DNNs) offer a promise, but they require large amounts of computational resources to provide better detection, and their detection capabilities can be exploited by adversarial attacks. Therefore, this paper proposes a method to train Fully Connected Neural Network (FCNN) for IoT security monitoring in a robust, effective and resource-efficient way. The resulting model is assessed against various benchmark datasets created using commercial IoT devices, such as doorbells, security cameras, and thermostats. Experimental results demonstrate the model's ability to maintain state-of-the-art accuracy and F1-score while reducing training memory and time consumption by 99.99 and 99.80 percentage points than its benchmark counterpart.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; *Embedded systems security*; • **Computer systems organization** → *Embedded and cyber-physical systems.*

KEYWORDS

IoT, Intrusion detection, Deep neural network, Computational efficient IDS.

ACM Reference Format:
Idris Zakariyya, Harsha Kalutarage and M. Omar Al-Kadri. 2022. Robust, Effective and Resource Efficient Deep Neural Network for Intrusion Detection in IoT Networks. In *Proceedings of the 8th ACM Cyber-Physical System Security Workshop (CPSS '22), May 30, 2022, Nagasaki, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3494107.3522772

1 INTRODUCTION

The Internet of Things (IoT) consists of Internet-enabled intelligent devices that use embedded systems such as processors, sensors and communication hardware to collect and exchange data. With the recent advances in super-cheap computer chips and the ubiquity of wireless networks, it is possible to make anything as small as a contact lens or as big as an aeroplane to be part of the IoT. IHS Markit estimates that 125 billion devices will be connected to the IoT by 2030 [15]. Most of these devices combine Artificial Intelligence (AI) with IoT infrastructure to enable more efficient IoT operations, improving human-machine interaction, and enhancing data management and analytics; therefore can be considered as Artificial Intelligence of Things (AIoT) [9]. The gradual provision of such devices is transforming the world into a sophisticated interconnected domain. This is good as it offers modern user convenience, but at the same time it can also open up a broader attack surface. Attackers can exploit vulnerabilities in software/hardware or embedded-AI of these devices, especially when they are connected to the external world to create a cyber disaster. In October 2016, for example, attackers launched a wave of IoT botnet attacks. It used various IoT devices to deny access to high profile websites like Twitter, Amazon, Github and Netflix [3]. Consequently, IoT security challenges must be addressed with robust and effective detection techniques.

Recent research has shown the capabilities of AI technologies in intrusion detection, particularly Deep Neural Network (DNN)-based methods, which can outperform most of their counterparts in cyber security monitoring [23]. A disadvantage of DNN-based methods, however, is that they require a lot of resources to build a model that can provide better detection with a multi-dimensional feature set [2]. This would be problematic in training scenarios such as edge machine learning, in which smart devices can process data locally using machine and deep learning algorithms (e.g. federated learning). Moreover, compared to the mainstream IT devices, IoT devices are equipped with limited computing resources (processing and storage) in order to enable maximum data output with minimum energy requirements while remaining cost-effective. As a result, DNN-based security solutions designed for mainstream IT devices cannot simply be deployed for security monitoring in an environment with limited computing resources. In addition, the detection capabilities of DNN-based methods can easily be exploited by feeding the network with adversarial samples [12]. To this end, we investigate the following research questions (RQs) to develop a suitable DNN-based method for the security monitoring of resource-constrained environments such as IoTs and cyber-physical systems.

RQ1: Can existing DNNs be trained to be resource efficient so that the resulting model is suitable for security monitoring in resource-constrained environments like IoTs? (see subsection 3.2)

CPSS '22, May 30, 2022, Nagasaki, Japan

RQ2: Can the resulting Resource Efficient DNN (REDNN) model be robust against perturbations attacks? (see subsection 5.1)
RQ3: Can the REDNN effectively detect attacks on IoT networks better than its counterparts, and how much detection must be sacrificed to achieve the desired level of resource efficiency? (see subsection 5.3)

In our experiments, we use Fully Connected Neural Network (FCNN) along with IoT benchmark datasets as described in subsection 4.1, and exploit FCNN's optimization algorithm to obtain the REDNN version from the FCNN. The experimental results are promising, as the resulting REDNN maintains better classification performance, low execution time and memory consumption and well resistance to adversarial attacks against each dataset used in our experiments. To the best of our knowledge, this is the first attempt to examine FCNN's capabilities for resource efficient and robust detection in the IoT security context, using a large number of benchmark datasets generated by hostile attacks on commercial IoT devices.

The rest of the paper is organized as follows. Section 3 describes the proposed method, while Section 4 describes the evaluation procedure as well as the adopted perturbations techniques. Results and discussion can be found in Section 5. Section 2 presents the related work. Finally, Section 6 concludes the paper with future research directions.

2 RELATED WORK

IoT technology is not yet mature and fully secured. There are many security challenges that need to be overcome, and vulnerability issues are a key to them. These weaknesses can be traced back to a lack of standardization, security considerations and the resource limitation of IoT devices.

AI for IoT Security Monitoring: AI techniques have been applied widely in the literature to address security challenges in the IoT [40]. Elrawy et al. [11] present recommendations for developing Machine Learning (ML) and DNN based IoT intrusion detection systems. These techniques remain promising in the field of IoT security monitoring researches. In that aspect, Bhunia et al. [5] utilize Support Vector Machine (SVM) to propose a framework for IoT security monitoring. Lopez et al. [26] proposed an IoT network traffic forecasting technique based on Gradient Boosting (GB) classifier. Tang et al. [41] enhanced the Adaboost algorithm to detect low rate DoS attacks in an IoT environment. Mohammad et al. [30] utilized DNN to accurately classify IoT networks traffic data. Iandola et al. [17] described the minimum speed requirements for deploying DNN on a resource-constrained environment. Shen et al. [38] compressed Convolutional Neural Network (CNN) for structure learning in an IoT resource-constraint environment. Kodali et al. [21] utilized DNN, especially FCNN, for classification tasks on resource-limited devices. Most of these techniques compressed DNN by the quantization of weights and bias parameters. However, our proposed approach in this paper targets effective attacks detection with resource minimization to reduce FCNN computational complexity. The method exploits pruning, simulated micro-batching and parameters optimization to regularize the resulting model and reduce memory and time requirements while increasing accuracy performance.

Adversarial Attacks Against AI: The limitation of AI (ML and DNN) based models deployed for IoT security monitoring is that an attacker can hinder their functionality. For instance, an attacker can control the training data by modifying each label instance [6]. Such attacks consider feeding the model with poisonous training data. The intention is to reduce the classification performance [36]. With that, a bunch of incorrect classifications is created by impacting the model used for security monitoring. In another context, a new set of perturbed data can be generated at the testing phase. The Fast Gradient Sign Method (FGSM) proposed by Kurakin et al. [24] is a potential mechanism for launching such perturbation. Kurakin et al. [24] enhance FGSM and introduce Projected Gradient Descent (PGD). This is a targeted attack method that maximizes the probability of a specific target class. Hosseini et al. [14] proposed a semantic attack method to alter the meaning of the original data samples. Athalye et al. [4] proposed a generic perturbation method based on randomly generated noise samples. The Jacobian Saliency Map Attack (JSMA) based on feature saliency map and jacobian derivative computation of the learned DNN model can succeed in crafting adversarial samples [33].

These perturbation methods are white-box based, assuming that the adversary has complete knowledge about the model applied in cyber security monitoring. However, a robust and effective classification model can defeat various adversarial perturbations. Such procedure considers training the built model with perturbed samples to augment regularization for resilience testing [44]. In our approach, we try to address these attacks for IoT security without utilizing the perturbed samples in training. Particularly to exploit the capability of the optimized FCNN model in defeating adversarial attacks effectively and efficiently. Therefore, we use eleven network traffic features data from various IoT devices. These IoT benchmarks data capture relevant device-specific properties. Therefore suitable for evaluating the performance of the proposed method with more realistic data samples instead of using oversimplified simulation methods.

3 METHODOLOGY

To demonstrate the proof of concept, we will use (FCNN) along with a number of IoT benchmark datasets, and exploit FCNN's optimization algorithm to obtain the REDNN version of the FCNN.

3.1 Baseline FCNN

DNN is a neural network structured into several layers of neurons representing the input data. A neuron is a fundamental computing unit capable of transmitting the result of the operation computed by its activation function with the input. FCNN is a sequential DNN that connects neurons by linking them with their corresponding weights and bias parameters. These weights and biases function as information storage units. Hence, we used Algorithm 1 to obtain the optimized FCNN model ($\mathcal{M}_n$) as the baseline for the comparison. The function BASE in line 1 of Algorithm 1 corresponds to the $\mathcal{M}_n$ mini-batch training using the gradient descent algorithm while computing the forward and backward derivatives [8]. The procedure is determined to minimize the objective function $J(W, b)$ in Equation 1 in-order to map unseen samples by using a procedure

that learned from $\mathcal{D}_{tr}$. The resulting FCNN approach uses supervised neural networks as a classifier, $\mathcal{M}_n$ can accept an input $\mathcal{D}_{tr}$ and outputs a probability class of vector $\hat{Y}$. The desired output $\hat{Y}$ are rounded up to the closest integer using a specified threshold value t as in Equation 2. This output represents either the benign (1) or the attack (0) traffic instance.

Algorithm 1 Baseline FCNN training

Input: Labelled data $\mathcal{D}_{tr}$, Iteration number $\mathcal{T}$, Batch size $\mathcal{S}$
Output: Baseline model $\mathcal{M}_n$

1: **function** BASE($\mathcal{D}_{tr}$[ ]) ▷ Training baseline model
2: **for** $i = 1$ to $\mathcal{T}$; **do**
3: Mini-batch $B = \{(x_1, y_1), ..., (x_m, y_m)\} \subset D_{tr}$
4: $F_p(B)$ ▷ Forward propagation
5: $\mathcal{E}_i \leftarrow L$ ▷ L = Base loss
6: B_p(B) ▷ Backward propagation
7: Compute gradients for parameters update
8: Estimate m_i ▷ Execution memory at epoch i
9: Estimate t_i ▷ Execution time at epoch i
10: $\mathcal{M}_n$ = Trained model that estimate $\mathcal{E}_i, m_i, t_i$
11: **end for**
12: **return** $(\mathcal{M}_n, \mathcal{E}_i, m_i, t_i)$
13: **end function**

$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} L - (Y\log\hat{Y} + (1-Y)\log(1-\hat{Y}) \quad (1)$$

$$Detection = \begin{cases} 0 \text{ if } \hat{Y} \le t \\ 1 \text{ if } \hat{Y} > t \end{cases} \quad (2)$$

3.2 Robust, Effective and Resource Efficient FCNN (REDNN)

The procedure of finding a robust, effective and resource efficient DNN model can be a challenging task [1]. This is due to the various parameters requirements in designing and building the desirable architecture. Particularly with complex and multidimensional datasets. Especially the IoT traffic data produced from many commercial devices consists of resource constraints, lower memory and processor. In this context, we utilize the baseline $\mathcal{M}_n$ model to propose the optimized REDNN model. As demonstrated in Algorithm 2 the optimized model adopts micro-batching [16, 31] for efficient model building. The function procedure requires $\mathcal{D}_{tr}$ in mini-batch and micro-batch forms to return the efficient M_e representing the REDNN model. The optimization process utilizes penalty function (weight elimination) [13] represented by E in Equation 3 with a threshold parameter w_0. This training description is in line 7 of Algorithm 2. This is useful in distinguishing the sets of relevant weights from the irrelevant ones. Particularly the insignificant large weights of the baseline $\mathcal{M}_n$ model. Weights values W greater than w_0 yield a complexity cost closer to 1 and require regularization using the penalty parameter λ. This is important to reduce the complexity of the model. To retain its performance, we consider the set of parameters that can give a training error $\mathcal{E}_j$ lower than $\mathcal{E}_i$. The REDNN model is less complex based on the regularization in lines 10 and 11 of Algorithm 2.

Algorithm 2 Proposed method to obtain REDNN

Input: Penalty term λ, ($\mathcal{D}_{tr}, \mathcal{T}, B$, in Alg. 1)
Output: Efficient model $\mathcal{M}_e$

1: **function** EFFICIENT($\mathcal{D}_{tr}$[ ])
2: **for** $j = 1$ to $\mathcal{T}$; **do**
3: Micro-batch $M = \{(x_1, y_1), ..., (x_m, y_m)\} \subset B$
4: $F_p(M)$ ▷ Forward propagation
5: $\mathcal{E}_t = L$ ▷ L = Initial loss
6: m_t, t_t ▷ m_t, t_t estimated memory and time using $\mathcal{E}_t$
7: $\mathcal{E}_j \leftarrow \mathcal{E}_t + \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)}$
8: B_p(M) ▷ Backward propagation
9: Compute gradients for parameters update
10: **if** $(\mathcal{E}_j \le \mathcal{E}_t)$ **then**
11: $\lambda = \lambda + \triangle\lambda$
12: Estimate m_j ▷ Execution memory at epoch j
13: Estimate t_j ▷ Execution time at epoch j
14: **if** $m_j \le m_t$ **then**
15: $m_{tr} = m_j$ ▷ $m_t r$ = Efficient memory
16: **if** $t_j \le t_t$ **then**
17: $t_{tr} = t_j$ ▷ t_{tr} = Efficient time
18: $\mathcal{M}_e$ = Trained model that estimate $\mathcal{E}_j, m_{tr}, t_{tr}$
19: **end if**
20: **end if**
21: **end if**
22: **end for**
23: **return** $(\mathcal{M}_e, \mathcal{E}_j, m_{tr}, t_{tr})$
24: **end function**

$$E = \lambda \sum_{j=1}^{W} \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)} \quad (3)$$

4 EVALUATION

This section describes the evaluation criteria of the baseline FCNN and REDNN models. It also presents the datasets used in building models.

4.1 Utilized Datasets

The N-BaIoT dataset contains various realistic data samples from nine commercial IoT devices that collectively represent multitudes of botnet and benign network traffic flow [27]. Each device is either infected by BASHLITE or Mirai attacks, with some regular instances. The nine devices subsets are a (i) Danmini Doorbell, (ii) Ecoobee Thermostat, (iii) Ennio Doorbell, (iv) Philips B120N10, (v) Provision PT-737E, (vi) Provision PT-838, (vii) Samsung SNH-1011-N, (viii) SimpleHome XCS-1002-WHT, and (ix) SimpleHome XCS-1003-WHT. Each device consists of sufficient records of attacks and regular instances with 115 features vector. As a result, the N-BaIoT dataset serves as a benchmark for the proposal of IoT network intrusion detection systems. We utilized all commercial devices subsets data of the N-BaIoT for training and testing FCNN and REDNN models.

Kitsune dataset consists of multiple traffic captured on an IoT network setting [29]. This dataset contains attacks that violate confidentiality, integrity and authenticity. These attacks are categorized into (i) Reconnaissance attacks, (ii) DoS attacks, and (iii) Mirai attacks. The subset of this data used to evaluate our models has 764,137 of Mirai and regular instances. This dataset has 115 features with a normal distribution of 121,621 raw traffics data.

WUSTL dataset consists of multiple reconnaissance attacks with normal traffics that emulate real-world industrial IoT systems for cyber-physical systems security research [42]. This dataset is useful in investigating the feasibility of ML algorithms for detecting various real-time attacks. The raw data consists of 7,037,983 data samples with seven (7) features. It comprised 93.30% for benign records with 6.7% attacks data records.

4.2 Data Preprocessing

The choice of these datasets allows frequent model training and rigorous evaluation. These datasets represented by numeric traffics flow are highly in-balanced suitable for IoT security investigations. Each utilized dataset is separated into 80% for training and 20% testing samples. Data input vectors are normalized using the unity-based normalization feature scaling. With n data features $x_1, x_2, ..., x_n$, within a dataset, the normalization is performed using the formula in Equation 4. The notation $x_i{}'$, represents the normalized value of the ith feature, x_i the original value, while min_{x_i} and max_{x_i} represents the minimum and maximum value of the i^{th} feature over the entire dataset.

$$x_i{}' = \frac{x_i - min_{x_i}}{max_{x_i} - min_{x_i}} \tag{4}$$

4.3 Experimental Setup

We used Python 3.76 on a desktop computer with Intel Xeon E5-2695(4 core) CPUs running at 2.10 GHz with 16.0 GB installed memory to build each model. For profiling model memory consumption, we utilized the integrated memory usage [34]. At training, parameters remain constant to enable a fair comparison. This applied to the baseline FCNN model, optimized REDNN and adversarial process.

4.4 Implementation Details

FCNN and REDNN Models Design. For building the generic sequential (dense) FCNN and REDNN models with each dataset, we utilized the scientific NumPy python module [20]. This enables the building of an in-depth DNN model without any library. This is good to understand underlying concepts and explore the internal operations within the network. Each model consists of an input layer, four hidden layers and an output layer. Regarding the topology selection against each dataset, we utilized the best-run Hyperas modules [22]. With that, we can select the best topology configurations against each dataset. The topology selection can minimize operations while maximizing the performance metrics. These are the requirements for the task of binary classification. The architectural settings remain identical for evaluating the baseline FCNN and the proposed REDNN model. Table 1 describes the models topology against each tested data.

For training, a mini-batch gradient descent optimizer with momentum was used. Random initialization of weight and bias parameters are within [0,1]. The baseline and optimized training procedure utilized lr = 0.001 across each dataset except the Ecobee and Ennio devices data with a different topology that used lr = 0.0001. Both FCNN and REDNN utilized a momentum value of 0.001. We used 0.01 values for λ, $\triangle\lambda$ and threshold w_0 [7] with 4 micro-batches to build the REDNN model. Models are trained in 128 batches within the 100 epochs for accuracy to converge. Binary cross entropy was used for calculating loss function. The activation function used in the input layer is ReLu [18] and Sigmoid for the output layer. For efficient hyperparameter selection, we utilized an automatic optimizer search module [37]. This technique required a range of values for each hyperparameter to be tuned to return an efficient combination. We used Numpy.float16 modules to implement the float 16 precision for the baseline and optimized model.

A TensorFlow Core version (v2.8.0) is used for building the Keras and TensorFlow deep learning models. The TensorFlow Lite (TFLite) converter module is used to build the TFLite deep learning model. The implementation used a fair comparison with Numpy (FCNN and REDNN) as both the Keras and TFLite models are trained in 128 mini-batches using stochastic gradient descent, at 100 epochs iterations. For the linear SVM, Adaboost and GB models, we utilized the Scikit-learn [35] machine learning python framework.

For exploration and reproduction purposes, the codes used for this study is accessible at [45]. Regarding the TFLite experimentation, we include both the Jupyter notebook file and the python script in [45] repository.

Perturbation Procedures. The perturbation methods utilized in crafting the adversarial samples are the FGSM and PGD [24], semantic [14], and random noise [4]. The FGSM attacking technique is computationally efficient. It required a one-step gradient update towards the direction of gradient sign as in Equation 5. We utilize FGSM in Algorithm 3 to generate a new perturbed dataset X^{fgsm} based on the original data X^o. The epsilon ϵ parameter determines the proportion of generated X^{fgsm} samples. These features are normalized using the clipping method to align with the corresponding X^o data features. The success of FGSM motivates the proposal of PGD [24]. PGD is an extended version of the FGSM method described in Equation 6. Both methods operate by computing the forward and backward propagation while generating perturbed samples. In the PGD attacking method, perturbed noise θ was added to the FGSM procedure before clipping the perturbed samples.

$$X^{fgsm} = X^o + \epsilon sign(\nabla_{X^o} J(X^o, Y)) \tag{5}$$

For crafting the FGSM and PGD adversarial samples, we utilized Algorithm 3 along with the cleverhans documentation [32]. The utilized documentation enables the development of generic adversarial attacks methods using NumPy. Each employed attacking technique except the semantic utilized an epsilon value scale within [0,1] incremented by 0.1. These scaling values determined the proportion of perturbation data generated. For the PGD, an infinity norm value was used with 40 iterations. This method enables the production of a new set of testing data that can be evaluated against the baseline FCNN and REDNN models.

Table 1: Topology and distribution of normal and attack for each device data.

Device	Normal	Attack	Inputs	Outputs	Topology
Danmini Doorbell	49,548	968,750	115	1	128-128-128-128
Ecobee Thermostat	13,113	822,763	115	1	32-64-64-16
Ennio Doorbell	39,100	316,400	115	1	64-128-128-64
Philips B120N10	175,240	923,437	115	1	128-128-128-128
Provision PT-737E	62,154	766,106	115	1	128-128-128-128
Provision PT-838	98,514	729,862	115	1	128-128-128-128
Samsung SNH-1011-N	52,150	323,072	115	1	128-128-128-128
SimpleHome XCS-1002-WHT	46,585	816,471	115	1	128-128-128-128
SimpleHome XCS-1003-WHT	19,528	831,298	115	1	128-128-128-128
Kitsune	121,621	642,516	115	1	128-128-128-128
Wustl	6,566,438	471,545	6	1	128-128-128-128

Algorithm 3 FGSM Perturbation procedure

Input: $\mathcal{X}, \mathcal{Y}, \epsilon, \mathcal{M}$, Data, label, epsilon, model
Output: Perturbed data $\mathcal{X}'$

for each ϵ **do**
  $F_k(\mathcal{M})$ ▷ Model $\mathcal{M}$ forward propagation
  $B_K(\mathcal{M}, \mathcal{Y})$ ▷ Model $\mathcal{M}$ backward propagation for gradient update (g)
  $\mathcal{X}' = \mathcal{X} + sign(\epsilon * g)$
  $\mathcal{X}' = clip(\mathcal{X}', [0, 1])$
end for

$$\theta = X^{fgsm} - X^{o} \quad \text{and} \quad X^{pgd} = X^{o} + \theta \tag{6}$$

Another perturbation procedure considered in this work is the data poisoning attacks in Algorithm 4. In this scenario, the data is poisoned by randomly flipping the labels. The flipping procedure considers label modification for the attacks (0s) and benign (1s) samples. This is the all labels modification technique that changes 1s to 0s and 0s to 1s. It is a non-targeted form of attacking method that concentrates on both classes. The rationale is to mislead the model by bringing its accuracy value down to 0%. We generate this form of attack by considering the training dataset. The data samples are randomized to have a fair proportion of the attack and benign samples. All labels of the randomized samples are flipped based on the specified poisoning proportion. For simplicity, we consider the rate to be from 0% - 50% by 5% increment.

Algorithm 4 Label modification perturbation procedure

Input: $\mathcal{X}, \mathcal{Y}, n, p$, Data, label, data length, percent
Output: Poisoned data $\{\mathcal{X}', \mathcal{Y}'\}$

for $t = 1$ *to* n; **do**
  if $t \in (1, p * n)$ **then** ▷ Random samples selection
    $y_t = 1 - y_t$ ▷ Labels 0 and 1 modification
    $\mathcal{Y}' = \{(x_t, y_t)\}, t = 1 \ldots n$ ▷ Integrating label
  end if
end for
return $\{\mathcal{X}', \mathcal{Y}'\}$

5 RESULTS AND DISCUSSION

This section discusses the experimental results. It details the evaluation comparison of the REDNN and optimized FCNN models across datasets.

5.1 REDNN Model Robustness

In Table 2, comparison results between the performance of the REDNN and FCNN models against tested adversarial samples are presented. In most cases, the REDNN provides higher test accuracy in detecting adversarial perturbation. Using the SimpleHome XCS-1002-WHT device data, PGD reduces the accuracy performance of the FCNN and REDNN by 2.6 and 1.94 percentage points, respectively. With the Kitsune dataset, PGD affects the accuracy of the FCNN by 13.64 percentage points, while that of the REDNN by 3.91 percentage points only. These results demonstrate the capability of REDNN in resisting the most successful PGD perturbation technique. The semantic attack is the weakest that shows no reduction in accuracy with each model. The reason can be altering the original samples by negation may not affect the accuracy with network traffic data rather than image format data [39]. For the noise attack, the REDNN provides better accuracy in many instances. The regularized REDNN model can accurately detect several adversarial attacks in IoT networks environments better than its counterparts. These results suggest that an effective parameters optimization of an in-depth DNN model can influence a robust detection of adversarial attacks in the context of IoT networks. As demonstrated, training with perturbed samples is not a requirement for effective and efficient detection in IoT network environment.

Robustness against number of epoch. Table 3 shows the effect of epoch variation on model robustness against the XCS-1003 dataset. At ten epochs, the adversarial accuracy of both models reduces, especially the baseline FCNN. Particularly with the PGD attacks that reduce the accuracy of the FCNN and REDNN to 69.65 and 77.43 percentage points. At 100 epoch iteration, the resilience of each model against perturbation attacks is better. Their accuracy value is more than 97 percent. This is useful as the optimized model can save more resources while thwarting adversarial attacks.

Robustness with clipped perturbation samples. Table 4 compares models' performance with clipped and non-clipped adversarial samples against randomly chosen datasets. The performance of detecting FGSM and random noise is better with the clipped

Table 2: Models performance comparisons across datasets.

Dataset	Model	Clean set acc (%)	FGSM set acc (%)	PGD set acc (%)	Noise set acc (%)	Semantic set acc (%)
Danmini Doorbell	FCNN	95.11	95.05	93.99	95.11	95.11
	REDNN	95.11	95.10	94.57	95.11	95.11
Ecobee Thermostat	FCNN	93.36	93.36	92.97	93.36	93.36
	REDNN	93.36	93.36	93.36	93.36	93.36
Ennio Doorbell	FCNN	88.94	88.90	88.90	88.94	88.94
	REDNN	88.94	88.89	88.89	88.94	88.94
Philips B120N10	FCNN	84.08	83.27	80.81	84.05	84.08
	REDNN	84.08	84.07	83.54	84.08	84.08
Provision PT-838	FCNN	88.07	87.19	84.75	88.06	88.07
	REDNN	88.07	87.83	86.70	88.07	88.07
Provsision PT-737E	FCNN	92.52	92.49	91.10	91.57	92.52
	REDNN	92.52	92.51	91.47	91.87	92.52
Samsung SNH-1011-N	FCNN	86.07	85.33	83.12	86.05	86.07
	REDNN	86.07	85.94	85.32	86.06	86.07
SimpleHome XCS-1002-WHT	FCNN	94.65	94.65	92.05	94.65	94.65
	REDNN	94.65	94.65	92.71	94.65	94.65
SimpleHome XCS-1003-WHT	FCNN	97.73	97.69	97.24	97.73	97.73
	REDNN	97.73	97.70	97.28	97.73	97.73
Kitsune	FCNN	84.09	78.27	70.45	81.00	84.09
	REDNN	84.09	83.52	80.18	84.02	84.09
Wustl	FCNN	94.26	94.26	94.26	94.26	94.26
	REDNN	94.26	94.26	94.26	94.26	94.26

Table 3: Effect of number of epoch against models performance with XCS-1003 dataset.

Epoch	Model	Clean set acc (%)	FGSM set acc (%)	PGD set acc (%)	Noise set acc (%)
10	FCNN	97.73	79.51	69.65	89.52
	REDNN	97.73	86.70	77.43	89.79
20	FCNN	97.73	86.35	77.07	93.86
	REDNN	97.73	86.70	77.43	94.08
40	FCNN	97.73	93.74	86.66	97.08
	REDNN	97.73	94.19	87.10	97.17
60	FCNN	97.73	96.48	90.72	97.63
	REDNN	97.73	96.84	92.09	97.69
80	FCNN	97.73	97.48	94.82	97.72
	REDNN	97.73	97.53	95.34	97.73
100	FCNN	97.73	97.69	97.24	97.73
	REDNN	97.73	97.70	97.29	97.73

procedure. The REDNN demonstrates better performance than its baseline counterpart. For thwarting the non-clipped FGSM adversarial samples of XCS-1003 device data, REDNN and FCNN accuracy decreased by 0.41 and 0.45 percentage points, respectively. With the same procedure to detect the random noise attacks against the Kitsune data, the accuracy of the FCNN and REDNN decreased by 4.86 and 0.93 percentage points. These results demonstrate the resilience nature of the REDNN with clipped and non-clipped adversarial samples.

Robustness against model variation. Table 5 presents the performance of REDNN and FCNN using three hidden layer architectures. Across each tested dataset, REDNN resists adversarial attacks better than its counterparts. As tested against the Danmini Doorbell dataset, PGD attacks lower the accuracy of the FCNN and REDNN by 9.18 and 7.23 percentage points, respectively. With the optimized four hidden layer model, accuracy reduction is by 1.12 and 0.54 percentage points for the FCNN and REDNN model (see Table 2). These results demonstrate that the four hidden layer networks can enable better attacks detection. As a result, a neural network model with few hidden layers may not stand robust against adversarial attacks. As demonstrated, REDNN is more appropriate for detecting adversarial perturbations in an IoT network environment.

Figures 1(a) and 1(b) show the impact of epsilon value against variational architecture in detecting PGD adversarial attack. In Figure 1(a), the FCNN and REDNN used three hidden layers. In Figure 1(b), both models utilized four hidden layers to detect PGD perturbations. In both scenarios, test set accuracy decrease with the epsilon value of 1.0. The REDNN model appears to provide an incremental accuracy value with 0.6 and 2.0 percentage points with three and four hidden layers while detecting the PGD attacks. The reason may be removing a hidden layer of a model may affect performance accuracy. In our case, each hidden layer has 128 neuron values.

Figures 2(a) and 2(b) present the impacts of reducing the second hidden layer neuron of each model by 50% and 25% against resilience using the Kitsune dataset. In each setting, REDNN provides better detection accuracy against adversarial samples. As depicted

Table 4: Effect of clipping samples against perturbations method.

Dataset	Procedure	Model	Clean set acc (%)	FGSM set acc (%)	PGD set acc (%)	Noise set acc (%)
XCS-1003	Clipped	FCNN	97.73	97.69	97.24	97.73
		REDNN	97.73	97.70	97.29	97.73
	Non-clip	FCNN	97.73	97.24	97.24	97.56
		REDNN	97.73	97.29	97.29	97.58
Danmini Doorbell	Clipped	FCNN	95.11	95.05	93.99	95.11
		REDNN	95.11	95.10	94.57	95.10
	Non-clip	FCNN	95.11	93.99	93.99	94.79
		REDNN	95.11	94.57	94.57	94.98
Kitsune	Clip	FCNN	84.09	78.27	70.45	80.67
		REDNN	84.09	83.52	80.18	83.84
	Non clip	FCNN	84.09	70.45	70.45	75.81
		REDNN	84.09	80.18	80.18	82.91

Table 5: Variational models perturbations evaluations across datasets.

Dataset	Model	Clean set acc (%)	FGSM set acc (%)	PGD set acc (%)	Noise set acc (%)
Danmini Doorbell	FCNN	95.11	91.43	85.93	93.78
	REDNN	95.11	92.93	87.88	94.45
Provsision PT-737E	FCNN	92.52	90.31	86.31	91.61
	REDNN	92.52	90.81	87.20	91.91
SimpleHome XCS-1002-WHT	FCNN	94.65	92.48	87.87	93.54
	REDNN	94.65	93.21	89.02	93.99
SimpleHome XCS-1003-WHT	FCNN	97.73	96.51	92.20	96.98
	REDNN	97.73	96.62	92.33	97.03
Kitsune	FCNN	84.09	75.73	70.02	81.72
	REDNN	84.09	81.56	77.65	83.88

in Figure 2(a), lowering hidden neurons values affect accuracy. It reduces FCNN and REDNN accuracy by 14.66 and 0.42 percentage points. For detecting PGD attacks using the 25% reduced neurons shown in Figure 2(b), FCNN and REDNN accuracy reduces by 24.52 and 5.26 percentage points, respectively. These results suggest that a significant reduction of hidden neurons affects models resilience against adversarial samples. In each scenario, REDNN stands to be more robust with topology variation than its counterparts. As a result, proper architecture selection can influence an efficient and effective identification of adversarial samples.

In Figure 3, we measured the results generated by poisoning the training data samples. A label flipping attack was integrated on each model using the kitsune and PT-737E device centred dataset. Each model performs better with the modified data of the kitsune data (see Figure 3(a)). They can thwart label poisoning attacks with 10% - 30% mislabelled training data as supported by the architecture of each model. Unlike Dunn et al. [10] that utilized 5% - 30% poisoned IoT data, we investigated further. With a 40% poisoning rate, REDNN significantly outperforms the FCNN model. For the PT-737E dataset result in Figure 3(b), the accuracy of the FCNN reduces significantly with 50% poisoning data. This behaviour may be due to the regularized property of the REDNN model [25]. Because of this, slighter changes in the training data may not affect the REDNN model. As demonstrated, the FCNN model is more

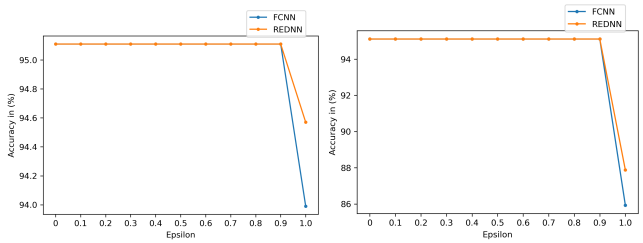
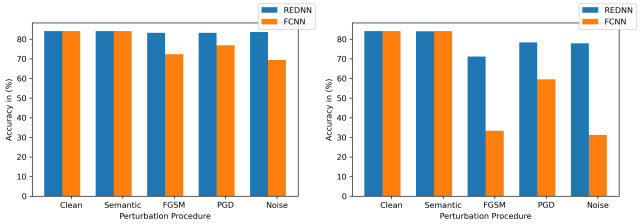


Figure 1: PGD test accuracy changes with epsilon for (a) four and (b) three hidden layers architecture against the Danmini Doorbell dataset.

sensitive to the 40% and 50% poisoning of the Kitsune and PT-737E datasets.

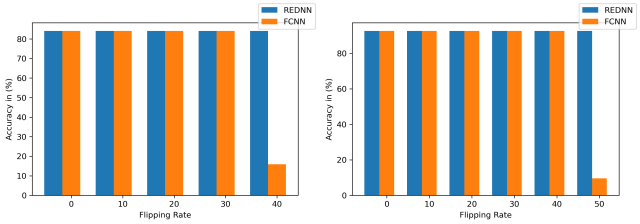
5.2 REDNN Model Resource Usage

Figures 4(a) and 4(b) present the execution time in seconds and CPU performance ($1/execution\ time$) for training each model. As demonstrated, the REDNN model is more efficient with better CPU performance against each dataset. This suggests its less computational expensive nature. The resources saving capabilities make



(a) (b)

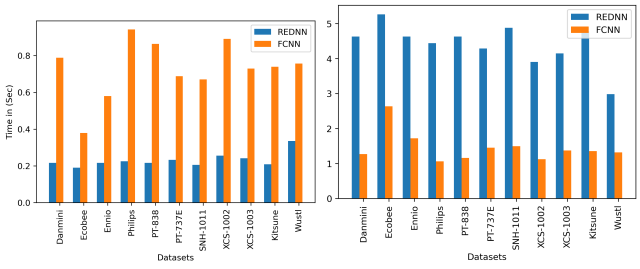
Figure 2: Accuracy changes with reduce hidden neuron by (a) 50% and (b) 25% against the Kitsune dataset.



(a) (b)

Figure 3: Accuracy changes with label flip against (a) Kitsune and (b) PT-737E dataset.

it an appropriate model for intrusion detection in IoT network environments.



(a) (b)

Figure 4: Model training (a) execution time and (b) CPU performance against utilized datasets.

We present measured testing results of the eleven IoT datasets runs with the FCNN and REDNN models in Table 6. In each case, the testing memory consumption of the datasets is profiled in MB accordingly. As demonstrated, the REDNN model can operate with a lower memory footprint. It can process the Wustl and Philips B120N10 datasets with 86.08 times and 2.16 times memory reduction than FCNN. These resources minimization make it a better choice for IoT security monitoring. It indicates its less complexity and faster detection capability.

5.3 REDNN Model Performance Comparison

The descriptions in Table 7 compare the performance evaluation of REDNN against the standard python technology frameworks. At training, REDNN demonstrates better memory footprint and

Table 6: Testing memory footprint.

Dataset	Model	Mem (MB)	Mem save (%)	Test acc (%)
Danmini Doorbell	FCNN	3.742	N/A	95.11
	REDNN	1.555	58.44	95.11
Ecobee Thermostat	FCNN	2.804	N/A	93.36
	REDNN	1.277	54.46	93.36
Ennio Doorbell	FCNN	2.410	N/A	88.94
	REDNN	0.539	77.63	88.94
Philips B120N10	FCNN	3.738	N/A	84.08
	REDNN	1.731	53.71	84.08
Provision PT-838	FCNN	3.031	N/A	88.07
	REDNN	1.266	58.23	88.07
Provision PT-737E	FCNN	3.008	N/A	92.52
	REDNN	1.285	57.28	92.52
Samsung SNH-1011-N	FCNN	2.598	N/A	86.07
	REDNN	0.582	77.60	86.07
SimpleHome XCS-1002	FCNN	3.004	N/A	94.65
	REDNN	1.320	56.06	94.65
SimpleHome XCS-1003	FCNN	3.145	N/A	97.72
	REDNN	1.305	58.51	97.72
Kitsune	FCNN	2.726	N/A	84.09
	REDNN	1.168	57.15	84.09
Wustl	FCNN	491.6	N/A	94.26
	REDNN	5.711	98.84	94.26

time savings. It saves 99.85 and 99.99 percentage points of training time and memory footprint than the baseline model trained with Keras. As compared with the converted FCNN TFLite model, the REDNN demonstrates better memory usage. The reason can be the TFLite model inherits the default Keras parameters during the model conversion. As a result, the conversion produces a lighter version of the Keras model. However, the quantized optimized TFLite model consumes fewer resources. It required 0.111 seconds of training execution times and 0.004 MB of training memory footprint. This is due to the computations with tf.float16 low precision [28] at training while optimizing the converted TFLite model [43]. However, in some cases, low precision can cause numerical issues [28]. This can leads to accuracy performance degradation with some datasets [46]. Because of that we implement each framework in 32 bits and compare their performance in Table 7. This is useful to investigate the resource savings without the low precision integration. As presented, the significant training resource-saving of the optimized REDNN model can be useful for on-device learning.

Table 7: Training performance evaluation across frameworks with Provision PT-737E dataset.

Procedure	Train time (sec)	Train mem (MB)	Test set acc (%)
FCNN-Keras	145.650	518.082	92.52
FCNN-TFLite	1.772	61.672	92.52
FCNN-Numpy	0.631	2.805	92.52
REDNN-Numpy	0.216	0.023	92.52

In Table 8, we show the testing resources (memory and time) consumption by each model against utilized frameworks. Regarding the processing times, the NumPy implementation is faster. REDNN can process IoT data efficiently at a slight rate than the baseline FCNN model runs on the same framework. The TFLite model is more efficient than the Keras model but slower than Numpy (FCNN and REDNN) models. Overall, REDNN saves 4.318, 69.81 and 80.55 percentage points of processing times than FCNN, TFLite and Keras models, respectively.

For the memory consumption in column (Test mem), REDNN demonstrates better savings. The reduction is by 78.91, 80.11 and 98.51 percentage points of memory footprint than FCNN, TFLite and Keras models, respectively. The TFLite more resources consumption is due to the to and fro data type conversion during prediction [43]. The conversion can increase the execution time and memory [19] as demonstrated in Table 8. The higher resources (memory and time) consumption of the TFLite at the testing stage is a limitation for effective IoT attacks detection. REDNN that demonstrate minimal testing resources consumption stands appropriate for IoT security monitoring.

Table 8: Testing resource consumption across frameworks with Provision PT-737E dataset.

Procedure	Test time (sec)	Test mem (MB)	Test set acc (%)
FCNN-Keras	6.494	84.92	92.52
FCNN-TFLite	4.184	6.383	92.52
FCNN-Numpy	1.320	6.016	92.52
REDNN-Numpy	1.263	1.269	92.52

Table 9 presents measured results that compare the performance of the REDNN model against state-of-the-art techniques using the PT-737E dataset. The REDNN model demonstrates better memory and time savings. At training, it saves more than 99.99 and 99.80 percentage points of execution time and memory footprint than the SVM model. For testing, the reduction is 99.91 and 99.17 percentage points of processing time and memory against the SVM model. The testing memory reduction remains promising across all tested models. This is good as the proposed REDNN can provide effective and efficient detection using minimal memory consumption.

Table 9: Performance evaluation comparison on Provision PT-737E dataset.

Model	Train time (sec)	Test time (sec)	Train mem (MB)	Test mem (MB)	Test set acc (%)
SVM	10045.545	1382.825	251.102	152.977	92.52
GB	360.247	0.619	14.730	3.316	92.58
AdaBoost	344.685	7.213	2.777	2.293	92.47
FCNN	0.631	1.320	2.805	1.320	92.52
REDNN	0.216	1.263	0.023	1.269	92.52

In addition to REDNN significant savings capability, it is more resilient against random noise attacks than each compared model (see Table 10). We later examined the effects of poisoning 50% of the training data with label modification (see Poisoned label column). The poisoning affects the robustness of the FCNN, SVM, GB and Adaboost model by lowering their accuracy to 9.55, 7.48, 10.01 and 11.05 percentage points, respectively. As demonstrated, REDNN indicates better resistance against labelled poisoned attacks. The results suggest that a stable and less complex model can defeat label poisoning attacks. It further demonstrates the effectiveness and lightweight nature of the REDNN model. As a result, it stands appropriate for the task of IoT security monitoring.

Table 10: Performance evaluation comparison with Provision PT-737E dataset.

Model	Clean set acc (%)	Noise set acc (%)	Poisoned label acc (%)
SVM	92.52	70.89	7.48
GB	92.58	61.91	10.01
Adaboost	92.47	53.31	11.05
FCNN	92.52	91.57	9.55
REDNN	92.52	91.87	92.52

Table 11 presents the model performance evaluated by test set accuracy, precision, recall and harmonic score (F1) while investigating the effects of FP16 integration on model resilience. Since the IoT datasets we considered are often in-balanced, test accuracy alone is not enough as a performance metric. Instead, precision considers the proportion of samples that are relevant within a predicted class and the F1 score that corresponds to the harmonic mean of precision and recall are more appropriate. The employed metrics utilized the True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN). The accuracy, precision, recall and F1 score are defined in Equation 7, 8, 9 and 10.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{7}$$

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

$$F1score = 2 \times \frac{Recall \times Precision}{Recall + Precision} \tag{10}$$

The integrated FP16 low precision affects the robustness of the FCNN model, especially in defeating random noise attacks. As demonstrated, REDNN indicates better resilience in thwarting each adversarial attack. The results suggest that FP16 integration had a minor influence on the robustness of the REDNN model. Because of this, REDDN is a more effective and robust IoT security monitoring technique than its baseline counterparts.

Table 11: Model resilience evaluation with kitsune dataset.

Attacks	Model	Accuracy (%)	Precision	Recall	F1-Score
FGSM	FCNN	83.60	0.8408	0.9744	0.9027
	REDNN	84.09	0.8409	1.0000	0.9136
PGD	FCNN	82.34	0.8408	0.9744	0.9027
	REDNN	84.09	0.8409	1.0000	0.9136
Noise	FCNN	76.67	0.8412	0.8906	0.8652
	REDNN	83.73	0.8411	0.9944	0.9113

6 CONCLUSION

This work served to evaluate the feasibility of using AI techniques in IoT security monitoring in a resource-efficient manner. In this context, we introduced a procedure to obtain a robust, efficient and effective method (REDNN) by exploiting the training algorithm of the model. In building the REDNN, we utilized FCNN and proposed a model that defeats adversarial attacks and maintains better performance for crafted features of the IoT networks traffic. We demonstrated its performance through empirical evaluation using eleven datasets. The technique can accurately detect adversarial attacks from each IoT device's dataset. It demonstrates more efficient performance than its counterpart in terms of robustness and resource consumption. The training and testing resource-saving results for each device model and accurate detection of adversarial samples depend on the topology settings and effective parameters optimization. The results suggest that utilizing perturbed traffic features at training is not a requirement for resisting adversarial attacks. It further shows the potentiality of the DNN algorithm in providing a robust and efficient solution for IoT network intrusion detection tasks. This is useful in finding an optimal model for different datasets in IoT security monitoring or any other classification problem using DNN in general. Especially in attracting future research for model consideration near the task of real-time IoT security monitoring. An essential approach to this would be to enhance this study's results to investigate more challenging ML techniques and CNN variants of the DNN methods. In particular, the exploration and consideration of the impact of adversarial perturbations in the unsupervised scenario. In addition, we plan further research to exploit REDNN learning capability with non-IoT datasets to benefit from the training and testing resource-saving advantage of the optimized algorithm. This can be useful for testing the generalization ability of REDNN for on-device learning using real IoT resources-constraints devices.

7 ACKNOWLEDGMENT

This work was supported by the Petroleum Technology Development Fund (PTDF), Nigeria.

REFERENCES

[1] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. 2018. State-of-the-art in artificial neural network applications: A survey. *Heliyon* 4, 11 (2018), e00938.

[2] Charu C Aggarwal et al. 2018. Neural networks and deep learning. *Springer* 10 (2018), 978–3.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*. 1093–1110.

[4] Anish Athalye, Nicholas Carlini, and David Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 274–283. https://proceedings.mlr.press/v80/athalye18a.html

[5] Suman Sankar Bhunia and Mohan Gurusamy. 2017. Dynamic attack detection and mitigation in IoT using SDN. In *2017 27th International telecommunication networks and applications conference (ITNAC)*. IEEE, 1–6.

[6] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. 2014. Poisoning behavioral malware clustering. In *Proceedings of the 2014 workshop on artificial intelligent and security workshop*. 27–36.

[7] Anna Bosman, Andries Engelbrecht, and Mardé Helbig. 2018. Fitness landscape analysis of weight-elimination neural networks. *Neural Processing Letters* 48, 1 (2018), 353–373.

[8] Yves Chauvin and David E Rumelhart. 2013. *Backpropagation: theory, architectures, and applications*. Psychology press.

[9] Bowei Dong, Qiongfeng Shi, Yanqin Yang, Feng Wen, Zixuan Zhang, and Chengkuo Lee. 2021. Technology evolution from self-powered sensors to AIoT enabled smart homes. *Nano Energy* 79 (2021), 105414. https://doi.org/10.1016/j.nanoen.2020.105414

[10] Corey Dunn, Nour Moustafa, and Benjamin Turnbull. 2020. Robustness evaluations of sustainable machine learning models against data poisoning attacks in the internet of things. *Sustainability* 12, 16 (2020), 6434.

[11] Mohamed Faisal Elrawy, Ali Ismail Awad, and Hesham FA Hamed. 2018. Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing* 7, 1 (2018), 1–20.

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

[13] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning both Weights and Connections for Efficient Neural Networks. arXiv:1506.02626 [cs.NE]

[14] Hossein Hosseini, Baicen Xiao, Mayoore Jaiswal, and Radha Poovendran. 2017. On the limitation of convolutional neural networks in recognizing negative images. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 352–358. https://doi.org/10.1109/ICMLA.2017.0-136

[15] Jenalea Howell. [n. d.]. Number of Connected IoT Devices Will Surge to 125 Billion by 2030, IHS Markit Says. Retrieved March 03, 2019 from https://news.ihsmarkit.com/prviewer/release_only/slug/number-connected-iot-devices-will-surge-125-billion-2030-ihs-markit-says

[16] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. 2019. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems* 32 (2019), 103–112.

[17] Forrest Iandola and Kurt Keutzer. 2017. Keynote: small neural nets are beautiful: enabling embedded systems with small deep-neural-network architectures. In *2017 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 1–10.

[18] Hidenori Ide and Takio Kurita. 2017. Improvement of learning for CNN with ReLU activation by sparse regularization. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2684–2691.

[19] intel. 2020. Choose Precision. Retrieved Aug 10, 2020 from https://software.intel.com/content/www/us/en/develop/articles/should-i-choose-fp16-or-fp32-for-my-deep-learning-model.html

[20] Robert Johansson. 2018. *Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib*. Apress.

[21] Sreela Kodali, Patrick Hansen, Niamh Mulholland, Paul Whatmough, David Brooks, and Gu-Yeon Wei. 2017. Applications of deep neural networks for ultra low power IoT. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 589–592.

[22] Brent Komer, James Bergstra, and Chris Eliasmith. 2019. Hyperopt-sklearn. In *Automated Machine Learning*. Springer, Cham, 97–111.

[23] Igor V Kotenko, Igor Saenko, and Alexander Branitskiy. 2018. Applying Big Data Processing and Machine Learning Methods for Mobile Internet of Things Security Monitoring. *J. Internet Serv. Inf. Secur.* 8, 3 (2018), 54–63.

[24] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2017. Adversarial Machine Learning at Scale. arXiv:1611.01236 [cs.CV]

[25] Jake Lever, Martin Krzywinski, and Naomi Altman. 2016. Points of significance: Regularization. *Nature methods* 13, 10 (2016), 803–805.

[26] Manuel Lopez-Martin, Belen Carro, and Antonio Sanchez-Esguevillas. 2020. IoT type-of-traffic forecasting method based on gradient boosting neural networks. *Future Generation Computer Systems* 105 (2020), 331–345.

[27] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. 2018. N-BaIoT—Network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing* 17, 3 (2018), 12–22.

[28] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2018. Mixed Precision Training. In *International Conference on Learning Representations*.

[29] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. arXiv:1802.09089 [cs.CR]

[30] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. 2018. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials* 20, 4 (2018), 2923–2960.

[31] Yosuke Oyama, Tal Ben-Nun, Torsten Hoefler, and Satoshi Matsuoka. 2018. Accelerating deep learning frameworks with micro-batches. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 402–412.

[32] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke,

Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, Rujun Long, and Patrick McDaniel. 2018. Technical Report on the CleverHans v2.1.0 Adversarial Examples Library. arXiv:1610.00768 [cs.LG]

[33] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.

[34] Fabian Pedregosa. 2019. Memory-profiler: a module for monitoring memory usage of a Python program. Retrieved March 10, 2021 from https://github.com/pythonprofilers/memory_profiler

[35] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[36] Nikolaos Pitropakis, Emmanouil Panaousis, Thanassis Giannetsos, Eleftherios Anastasiadis, and George Loukas. 2019. A taxonomy and survey of attacks against machine learning. *Computer Science Review* 34 (2019), 100199.

[37] Max Pumperla. [n. d.]. Hyperas: Hyperopt: A very simple and convenient wrapper for hyperparameter optimization. Retrieved October 10, 2021 from https://github.com/maxpumperla/hyperas

[38] Shibo Shen, Rongpeng Li, Zhifeng Zhao, Qing Liu, Jing Liang, and Honggang Zhang. 2020. Efficient Deep Structure Learning for Resource-Limited IoT Devices. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 1–6.

[39] Octavian Suciu, Scott E Coull, and Jeffrey Johns. 2019. Exploring adversarial examples in malware detection. In *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 8–14.

[40] Pedro Miguel Sánchez Sánchez, José María Jorquera Valero, Alberto Huertas Celdrán, Gérôme Bovet, Manuel Gil Pérez, and Gregorio Martínez Pérez. 2021. A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets. *IEEE Communications Surveys Tutorials* 23, 2 (2021), 1048–1077. https://doi.org/10.1109/COMST.2021.3064259

[41] Dan Tang, Liu Tang, Rui Dai, Jingwen Chen, Xiong Li, and Joel JPC Rodrigues. 2020. MF-Adaboost: LDoS attack detection based on multi-features and improved Adaboost. *Future Generation Computer Systems* 106 (2020), 347–359.

[42] Marcio Andrey Teixeira, Tara Salman, Maede Zolanvari, Raj Jain, Nader Meskin, and Mohammed Samaka. 2018. SCADA system testbed for cybersecurity research using machine learning approach. *Future Internet* 10, 8 (2018), 76.

[43] TensorFlow. 2022. float16 Quantization. Retrieved Feb 13, 2022 from https://www.tensorflow.org/lite/performance/post_training_float16_quant

[44] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2020. Ensemble Adversarial Training: Attacks and Defenses. arXiv:1705.07204 [stat.ML]

[45] Idris Zakariyya. 2021. Resource Efficient IoT DNNs Algorithm. Retrieved November 12, 2021 from https://github.com/izakariyya/R_DNN_IoT

[46] Jiawei Zhao, Steve Dai, Rangharajan Venkatesan, Ming-Yu Liu, Brucek Khailany, Bill Dally, and Anima Anandkumar. 2021. Low-Precision Training in Logarithmic Number System using Multiplicative Weight Update. *CoRR* abs/2106.13914 (2021). arXiv:2106.13914 https://arxiv.org/abs/2106.13914