

Memory Efficient Federated Deep Learning for Intrusion Detection in IoT Networks

Idris Zakariyya^a, Harsha Kalutarage^a and M. Omar Al-Kadri^b

^a*School of Computing, Robert Gordon University, UK*

^b*School of Computing and Digital Technology, Birmingham City University, UK*

Abstract

Deep Neural Networks (DNNs) methods are widely proposed for cyber security monitoring. However, training DNNs requires a lot of computational resources. This restricts direct deployment of DNNs to resource-constrained environments like the Internet of Things (IoT), especially in federated learning settings that train an algorithm across multiple decentralized edge devices. Therefore, this paper proposes a memory efficient method of training a Fully Connected Neural Network (FCNN) for IoT security monitoring in federated learning settings. The model's performance was evaluated against eleven realistic IoT benchmark datasets. Experimental results show that the proposed method can reduce memory requirement by up to 99.46 percentage points when compared to its benchmark counterpart, while maintaining the state-of-the-art accuracy and F1 score.

Keywords

Deep Neural Networks (DNNs), Internet of Things (IoT), Fully Connected Neural Network (FCNN), Memory, Federated Learning, Intrusion Detection

1. Introduction

The Internet of Things (IoT) consists of network-enabled intelligent devices that use embedded systems, such as processors, sensors and communication hardware to collect and exchange data. IoT is an ecosystem use in smart-home, smart-cities, and many intelligent automation systems [1]. However, these devices are increasingly becoming a potential target for various cyber attacks. The Distributed Denial-of-Service (DDoS) attack on Dyn domain name servers in 2016, which used a network of 100,000 odd IoT devices, powered by a virus called Mirai (Linux. Gafgyt), testifies to this [2]. Most IoT devices are made up of limited processors and memory, so security solutions designed for mainstream computing devices cannot be deployed on resource constrained IoT devices. Therefore, security challenges in the IoT must be addressed with resource-efficient and effective mechanisms, ideally in a federated learning manner that supports to overcome data sharing and privacy issues.

Recent intrusion detection research has shown the capabilities of AI techniques, especially Deep Neural Network (DNN), in cyber security monitoring [3]. However, building DNN based detection technique requires a lot of resources during the model training. As IoT devices are


AI-CyberSec 2021: Workshop on Artificial Intelligence and Cyber Security, December 14, 2021, Cambridge, UK

✉ i.zakariyya@rgu.ac.uk (I. Zakariyya); h.kalutarage@rgu.ac.uk (H. Kalutarage); omar.alkadri@bcu.ac.uk (M. O. Al-Kadri)

ORCID 0000-0002-7983-1848 (I. Zakariyya); 0000-0001-6430-9558 (H. Kalutarage); 0000-0002-1146-1860 (M. O. Al-Kadri)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

resource-constrained and distributed in nature, DNN-based cyber security techniques cannot be directly deployed for intrusion detection in IoT networks. In that context, Federated Learning (FL) [4] approach that supports for data privacy may not scale through IoT devices due to their lack of computational resources. To respond to this challenge, we propose an efficient training method for Fully Connected Neural Network (FCNN) for IoT security monitoring, in particular to reduce the memory footprint during the training while maintaining the same or higher level of accuracy than its benchmark counterpart.

For our experiments, we utilize a FCNN along with eleven IoT benchmark datasets to build a memory-efficient DNN (MEDNN) model. The experimental results are encouraging as the resulting MEDNN shows lower memory consumption with better classification performance in both centralized and federated settings against each data set used in our experiments. The federated integration of the model also helps to preserve the privacy of IoT device data during on-device model training.

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

2. Related Work

This section presents related studies concerning deep learning for IoT intrusion detection followed by recent FL techniques applied to IoT security monitoring.

Significant research has been conducted on IoT security monitoring using AI techniques. Most of these methods utilized DNN. Mohammad et al. [5] described the potentiality of DNN for IoT data analysis and classification tasks. Kodali et al. [6] employed DNN, especially FCNN, for classification tasks on resource-limited devices. Shen et al. [7] proposed compact structure-based learning with Convolutional Neural Network (CNN) for an IoT resource-constrained environment. Most of the optimization approach considered the quantization of weights and bias parameters. However, our proposed approach in this paper aims to reduce memory requirements. The method exploits pruning, simulated micro-batching and parameter regularization to optimise the resulting model in terms of memory requirements and accuracy performance. This is useful, especially for the task of distributed learning in a resource-constrained environment.

Recently, researchers from several disciplines explored FL methods from different perspectives. In the field of IoT security monitoring, FL is gaining popularity. Preuveneers et al. [8] explored FL applications for intrusion detection in IoT networks. Lim et al. [9] and Imteaj et al. [10] describes open research problems on FL for resource-constrained IoT devices. Thein et al. [11] utilized FL to detect attacks on industrial IoT devices. Liu et al. [12] conduct a similar investigation by considering sensor reading data. Jiang et al. [13] utilized model pruning for efficient FL training on edge devices. Bonawitz et al. [14] proposed a scalable FL framework for mobile devices to reduce communication overhead. However, none of these proposals considers optimizing FL training to reduce memory consumption on IoT networks using pruning and micro-batching. We address this challenge by optimizing the federated training procedure using raw network traffic datasets from various IoT devices. Then, we proposed a MEDNN FL method

with minimal resource consumption. This method maintains state-of-the-art accuracy while reducing memory consumption.

3. Research Methodology

We propose a framework that manipulates and optimizes an FCNN version of DNN to yield a compact classification model (see Figure 1). We later validated this framework by training the FCNN on IoT benchmark datasets in federated and centralized settings to build MEDNN. This requires evaluating the FCNN regularization to produce a loss function that identifies various parameters relevant to model shrinking. We demonstrate that knowledge of architecture and optimizing parameters is sufficient to produce the MEDNN model. The optimized model can classify malicious activities on IoT networks.

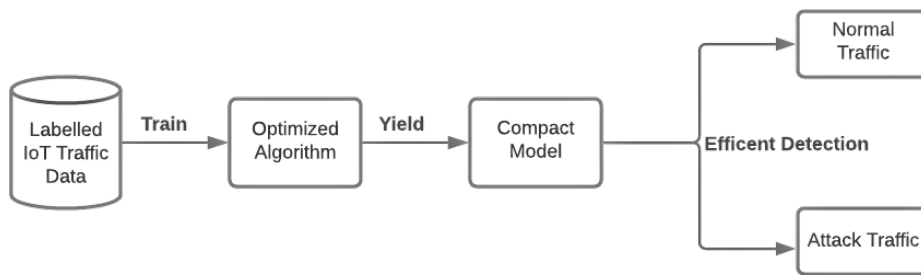


Figure 1: Effective IoT Attack Detection Framework.

3.1. Baseline FCNN Training

A DNN is a neural network containing deep layers of neurons representing the input data. These neurons correspond to computing units. They are capable of transmitting the computational results operated with their activation function and the input. FCNN is a sequential DNN connecting neurons by linking them with their corresponding weights and bias parameters. The weights and biases serve as information storage components. The baseline FCNN model (\mathcal{M}_n) in Algorithm 1 is consist of network topology, activation functions and corresponding values for weights and bias. The weight and bias values settings can minimize the error function $\mathcal{E}_{\mathcal{M}_n}$ evaluated over the labelled training data \mathcal{D}_{tr} . The function BASE in line 1 of Algorithm 1 describes the \mathcal{M}_n training using a gradient descent algorithm with backpropagation [15]. This is determined to minimizes the cost function in Equation 1 and Equation 2 in-order to properly map unseen samples using a function 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 3. This output represents either the benign (1) or the attack (0) traffic instance.

Algorithm 1 Baseline FCNN Training

Input: Labelled data \mathcal{D}_{tr} , Number of iteration \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:     Sample mini-batch  $B = \{(x_1, y_1), \dots, (x_m, y_m)\} \subset \mathcal{D}_{tr}$ 
4:      $F_p(B)$  ▷ forward pass
5:      $\mathcal{E}_i \leftarrow L$  ▷  $L =$  Base loss
6:      $B_p(B)$  ▷ backward pass based on model parameters for  $F_k(B)$ 
7:     Compute gradients for parameters update
8:     Estimate  $m_i$  ▷ Execution memory at epoch  $i$ 
9:      $\mathcal{M}_n =$  Trained model that estimate  $\mathcal{E}_i, m_i$ 
10:  end for
11:  return ( $\mathcal{M}_n, m_i, \mathcal{E}_i$ )
12: end function
```

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

$$L(\hat{Y}^i, Y^i) = -(Y \log \hat{Y} + (1 - Y) \log (1 - \hat{Y})) \quad (2)$$

$$Output = \begin{cases} 0 & \text{if } \hat{Y} \leq t \\ 1 & \text{if } \hat{Y} > t \end{cases} \quad (3)$$

3.2. Memory Efficient MEDNN Training

Training a resource efficient DNN model can be a challenging task [16]. Especially in considerations of model parameters requirements in designing and building the desirable architecture. The complexity of such an approach increases with multidimensional datasets.

To this end, we utilize the baseline \mathcal{M}_n model (a trained FCNN model) to produce the memory efficient version of it (MEDNN). The training procedure described in Algorithm 2 optimizes a function that requires \mathcal{D}_{tr} to return the efficient M_e correspond to the MEDNN model. As described in line 4 in Algorithm 2, the optimization procedure utilized micro-batching [17, 18] for efficient training. To reduce network complexity, we used a penalty [19] (weight elimination) technique with a threshold parameter w_0 as shown in regularized Equation 4. This is a requirement to discover those sets of relevant weights from the irrelevant ones. Particularly in determining the significant and insignificant large weights of the baseline FCNN model. Weights greater than w_0 that yield a complexity cost closer to 1 requires a regularization using the penalty parameter λ . The regularization considers a scenario where the baseline produces a higher error value \mathcal{E}_i as in line 9. For better performance, we utilized the set of parameters to produce a lower error value \mathcal{E}_j . This process can reduce the complexity of the FCNN model while building the MEDNN.

Algorithm 2 Procedure to build MEDNN

Input: Penalty term λ
Output: Efficient Model \mathcal{M}_e

- 1: **function** EFFICIENT($\mathcal{D}_{tr}[\]$) ▷ \mathcal{D}_{tr} in Alg. 1
- 2: **for** $j = 1$ to \mathcal{T} ; **do** ▷ \mathcal{T} in Alg. 1
- 3: Sample mini-batch $K = \{(x_1, y_1), \dots, (x_m, y_m)\} \subset \mathcal{D}_{tr}$
- 4: Sample micro-batch $M = \{(x_1, y_1), \dots, (x_m, y_m)\} \subset K$
- 5: $F_p(M)$ ▷ forward pass
- 6: $\mathcal{E}_j \leftarrow L + \lambda \sum_{j=1}^W \frac{(w_j^2/w_0^2)}{(1+w_j^2/w_0^2)}$ ▷ $L, W, w_0 =$ Loss, total weights, threshold
- 7: $B_p(M)$ ▷ backward pass based on model parameters for $F_k(M)$
- 8: Compute gradients for parameters update
- 9: **if** $(\mathcal{E}_j \leq \mathcal{E}_i)$ **then** ▷ \mathcal{E}_i in Alg. 1
- 10: $\lambda = \lambda + \Delta\lambda$
- 11: Estimate m_j ▷ Execution memory at epoch j
- 12: **if** $m_j \leq m_i$ **then** ▷ m_i in Alg. 1
- 13: $m_{tr} = m_j$ ▷ $m_{tr} =$ Efficient memory footprint
- 14: $\mathcal{M}_e =$ Trained model that estimate \mathcal{E}_j, m_{tr}
- 15: **end if**
- 16: **end if**
- 17: **end for**
- 18: **return** $(\mathcal{M}_e, m_{tr}, \mathcal{E}_j)$
- 19: **end function**

$$R = \lambda \sum_{j=1}^W \frac{(w_j^2/w_0^2)}{(1 + w_j^2/w_0^2)} \quad (4)$$

3.3. MEDNN in Federated Learning

FL is a machine learning approach that supports distributed model training using multiple clients without exposing their training data. This technique updates a shared global model by aggregating each client training output [20]. Building a federated model can be a challenge for resource-constrained IoT devices. With this in mind, we tested the proposed MEDNN in FL settings to see how much memory it can save in model training. Our federated learning approach is less complex, efficient and effective for the task of IoT intrusion detection compared to its benchmark counterpart (see experimental results in Section 5.3).

4. Evaluation

This section describes benchmark datasets and the evaluation procedure used to build the MEDNN and FCNN techniques in centralized and federated learning settings.

4.1. Utilized Datasets

The N-BaIoT dataset consists of various raw subsets data instances from many commercial IoT devices (see Table 1). Each device contains data samples of attacks and benign network traffic flows [21]. These devices are either infected by BASHLITE or Mirai attacks with some benign instances. The overall dataset serves as a benchmark for the proposal of IoT intrusion detection methods. We consider device subsets data of the N-BaIoT to train and test our models. The distribution of the benign and attack samples for each subset of the data show its unbalanced nature. Each device subset data consists of 115 features vector.

Table 1

Distribution of benign and attack instances for each device of the N-BaIoT.

Device	Benign instance	Attack instance
Danmini Doorbell	49 548	968 750
Ecobee Thermostat	13 113	822 763
Philips B120N10	175 240	923 437
Provision PT-737E	62 154	766 106
Provision PT-838	98 514	729 862
Samsung SNH-1011-N	52 150	323 072
SimpleHome XCS7-1002-WHT	46 585	816 471
SimpleHome XCS7-1003-WHT	19 528	831 298

Kitsune dataset contains multiple traffic captured on an IoT network setting [22]. A subset of this data employed to evaluate our models has 764,137 instances of Mirai and regular traffic. This dataset has 115 features with a normal distribution of 121,621 raw traffics data.

IoT-DDoS consists of various captured traffics representing the DDoS botnet attacks and some portion of regular traffic [23]. We consider 79,035 benign data and 398,391 attack data samples for empirical model evaluation.

WUSTL consists of multiple flows of traffic from an emulated SCADA system [24]. The dataset can be used to investigate the feasibility of ML algorithms in detecting various attacks. The raw data consists of 7,037,983 data samples. For experimental purposes, the distribution of 471,545 attacks and 6,566,438 normal instances was considered.

4.2. Data Preprocessing

The choice of utilized datasets allows efficient model training for investigations purposes. The classes in these datasets are unbalanced, making them suitable for IoT security monitoring. Employed datasets are categorized 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, \dots, x_n , within a dataset, the normalization is performed using the formula in Equation 5. The description x_i' , represents the normalized value of the i th 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.

$$x_i' = \frac{x_i - \min_{x_i}}{\max_{x_i} - \min_{x_i}} \quad (5)$$

4.3. Experimental Setup

We profile the memory usage for each model training procedure using the integrated memory usage [25]. 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. For models analytics, the Spyder scientific Integrated Development Environment (IDE) [26] was used to store the model for each dataset. At training, parameters remain constant to enable a fair comparison. This applied to the baseline FCNN model and optimized MEDNN. The code used for this study can be accessible at [27].

4.4. Implementation Details

FCNN and MEDNN Models. For building the sequential FCNN and MEDNN with each dataset, we used the scientific NumPy python module [28]. Each sequential model consists of an input layer, three hidden layers, and an output layer. Regarding the eight device subset data of N-BaIoT, the topology used consists of 83 neurons in the first and last hidden layer, with 128 neurons in the second hidden layer (83-128-83). The network architecture used with the kitsune dataset consists of 83 neurons in the first and third hidden layers, with 141 neurons in the second hidden layer (83-141-83). For each implementation of these mentioned models topology, the input layer has 115 neurons representing the number of data features, while the output has one neuron.

The network architecture used with the Wustl dataset has three hidden layers with 26 neurons each (26-26-26), while the input and output layers have 6 and 1 neurons, respectively. The model topology used against the IoT-DDoS dataset consists of 20 neurons in each of the three hidden layers (20-20-20), while the input and output layer has 12 and 1 neurons.

These topology architectures are the requirement for the task of binary classification. The setting considers meant to minimize training computations while increasing the performance metrics. These architectures settings are identical for evaluating the baseline FCNN and the proposed MEDNN model. The only difference during the training would be FCNN used Algorithm 1, while MEDNN utilized Algorithm 2. This indicates that significant memory reduction was due to the optimization procedure in Algorithm 2.

For training each model, a mini-batch gradient descent was used. The weight and bias parameters are initialized randomly within $[0,1]$. The baseline and optimized training procedure utilized $lr = 0.001$. We used 0.01 values for λ , $\Delta\lambda$ and threshold w_0 [29] with 4 micro-batches to build the MEDNN model. The activation function considered in the fully connected layers is relu with sigmoid in the output layer. Models are trained in 128 batches within the 100 epochs for accuracy to converge. Parameters and hyperparameters were choosing based on grid search. Binary cross entropy was utilized for calculating loss function. See Figure 2a and Figure 2b for the learning process using the chosen epoch for the optimized and baseline training procedure. The optimized training algorithm provides better training accuracy even with fewer iterations than its baseline counterpart.

Low Precision 16-bit Implementation. In Numpy, training with 16-bit floating precision (FP16) requires calling the `.float16()` method on all model parameters and input data. We consider FP16 while training the baseline FCNN and in obtaining the efficient MEDNN model.

FL Setup. For the FL experimental settings, we used PyTorch version 1.4.0 [30] and PySyft version 0.2.9 [31]. Pysyft framework simplifies the creation of virtual workers. These workers

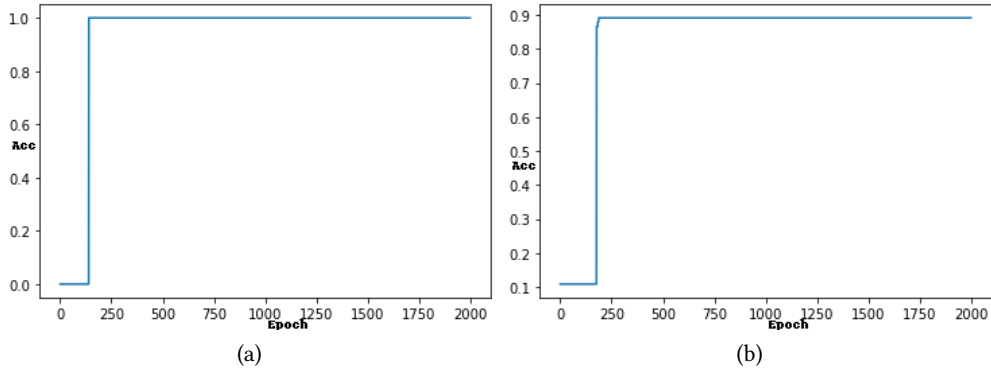


Figure 2: Effect of epoch with accuracy for (a) optimized and (b) baseline training with Danmini Doorbel data

emulate real virtual machines and can run as a separate process within the same python program. Our federation training procedure utilized three virtual workers representing clients and a coordinating worker. As we utilized Federated averaging (FedAvg), a Stochastic Gradient Descent (SGD) was used to optimize each model. Federated models are trained in 128 batches within four epochs in 30 workers iterations. After the clients model training is complete, average weights values are sent to the coordinating worker. This worker aggregates those weights to update the global model.

5. Results and Discussion

This section discusses the experimental results. It details the evaluation comparison of the optimized MEDNN and baseline FCNN models in centralized and federated settings across datasets.

5.1. MEDNN Model Training (Centralized Manner)

With 11 IoT data sets, we first examined the memory requirements for training FCNN and MEDNN models in a centralised manner. Table 2 presents the memory profile in MB across each dataset. The optimized MEDNN model training requires a lower memory. It reduces the memory requirements of training with Philips B120N10 by 97.60 percentage points and achieves a higher classification accuracy of 84.10 percentage points than its baseline counterpart. These results show the regularization advantage [32, 33] on accuracy with certain datasets. It indicates the less complexity, faster learning capability and better performance behaviour of the optimized model. These resources minimization make it a better choice for IoT security monitoring.

5.2. Low Precision 16-bit Training of MEDNN

Training with reduced precision has become the de facto technique for increasing the energy efficiency of deep learning hardware [34]. Therefore we investigated the memory efficiency of

Table 2

Model training (Centralized): Comparison of training memory consumption between MEDNN and FCNN.

Dataset	Model	Memory (MB)	Memory Reduction (%)	Test Accuracy (%)
Danmini Doorbell	FCNN	4.039	N/A	95.11
	MEDNN	0.098	97.57	95.11
Ecobee Thermostat	FCNN	4.125	N/A	1.50
	MEDNN	0.121	97.07	93.32
Philips B120N10	FCNN	4.082	N/A	15.92
	MEDNN	0.098	97.60	84.10
Provision PT-838	FCNN	3.438	N/A	13.97
	MEDNN	0.098	97.15	88.08
Provision PT-737E	FCNN	3.980	N/A	92.52
	MEDNN	0.098	97.54	92.52
Samsung SNH-1011-N	FCNN	3.789	N/A	13.94
	MEDNN	1.199	68.36	86.07
SimpleHome XCS7-1002-WHT	FCNN	3.680	N/A	94.65
	MEDNN	0.102	97.23	94.65
SimpleHome XCS7-1003-WHT	FCNN	3.969	N/A	97.72
	MEDNN	0.102	97.43	97.72
Kitsune	FCNN	3.656	N/A	84.09
	MEDNN	0.102	97.21	84.09
IoT-DDoS	FCNN	1.738	N/A	83.34
	MEDNN	0.047	97.30	83.34
Wustl	FCNN	1.516	N/A	94.26
	MEDNN	0.063	95.84	94.26

the proposed MEDNN with low precision implementation. Table 3 presents training memory usage while integrating the FP16 precision. Across each dataset, memory consumption was reduced by the complete training iterations. Regarding the Philips data, the reduction is 43.63 and 80.61 percentage points with the baseline and optimized training process, respectively. With the same data, the accuracy increased by 68.18 percentage points using the optimized method. The results suggest that FP16 operations can influence memory reduction using the optimized training method. It demonstrated that FP16 integration does not influence MEDNN accuracy reduction in most cases. It can reduce the FCNN classification accuracy across some datasets. As a result, the regularized MEDNN can maintain a better accuracy with FP16 computations.

5.3. MEDNN Model Training (Decentralized Manner)

The results in Table 4 are for the implemented FL method with baseline (FCNN) and its optimized model (MEDNN). These results compared the training memory requirements and accuracy across each dataset. In federated training, the MEDNN model requires lower memory across all datasets. It saves 99.46 percentage points of memory while training the SimpleHome XCS-

Table 3

Model training (Low precision): Comparison of training memory consumption between MEDNN and FCNN.

Dataset	Model	Memory (MB)	Memory Reduction (%)	Test Accuracy (%)
Danmini Doorbell	FCNN	3.125	N/A	4.90
	MEDNN	0.285	90.88	95.12
Ecobee Thermostat	FCNN	2.313	N/A	1.50
	MEDNN	0.023	99.01	93.32
Philips B120N10	FCNN	2.301	N/A	15.92
	MEDNN	0.019	99.17	84.10
Provision PT-838	FCNN	1.961	N/A	11.93
	MEDNN	0.113	94.24	88.07
Provision PT-737E	FCNN	2.343	N/A	9.60
	MEDNN	0.098	95.82	92.52
Samsung SNH-1011-N	FCNN	3.039	N/A	13.94
	MEDNN	0.148	95.13	86.07
SimpleHome XCS7-1002-WHT	FCNN	1.949	N/A	6.30
	MEDNN	0.055	97.18	94.63
SimpleHome XCS7-1003-WHT	FCNN	1.988	N/A	97.67
	MEDNN	0.027	98.64	97.70
Kitsune	FCNN	2.813	N/A	84.09
	MEDNN	0.160	94.31	84.09
IoT-DDoS	FCNN	0.227	N/A	83.34
	MEDNN	0.027	88.11	83.34
Wustl	FCNN	0.199	N/A	94.26
	MEDNN	0.012	93.97	94.26

1003-WHT dataset. Across all tested datasets, the classification accuracy is not degraded by the proposed method. This result demonstrates the advantage of the optimized model in building an efficient federated training method, and the usefulness of the proposed method for effective attack detection on resource-constrained devices.

We investigated the effect of the proposed method in federated learning using all the datasets. However, due to the space constraint, we only present the result of the SimpleHome XCS-71003-WHT device data to show the significant memory reduction (see Table 5). In addition to the significant memory reduction by the MEDNN model, it outperforms the FCNN model with low precision 16-bit implementation. As shown in the table, FP16 integration reduces the accuracy of the FCNN by 0.05 percentage points while reducing that of the MEDNN by only 0.02 percentage points, respectively. In centralized and federated training procedures, both models demonstrate equal accuracy performance. These results suggest the significance of our optimized model compared with its benchmark counterpart. It indicates that the proposed method is efficient and effective for on-device training in a distributed manner.

Table 4

Model training (Fedarated): Comparison of training memory consumption between MEDNN and FCNN.

Dataset	Model	Memory (MB)	Memory Reduction	Test Accuracy (%)
Danmini Doorbell	FCNN	8.637	N/A	95.11
	MEDNN	0.402	95.34	95.11
Ecobee Thermostat	FCNN	9.426	N/A	93.35
	MEDNN	0.070	99.26	93.35
Philips B120N10	FCNN	9.695	N/A	84.08
	MEDNN	0.465	95.20	84.08
Provision PT-838	FCNN	10.26	N/A	88.07
	MEDNN	0.418	95.93	88.07
Provision PT-737E	FCNN	10.23	N/A	92.52
	MEDNN	0.117	98.86	92.52
Samsung SNH-1011-N	FCNN	12.27	N/A	86.07
	MEDNN	4.195	65.80	86.07
SimpleHome XCS7-1002-WHT	FCNN	9.598	N/A	94.65
	MEDNN	0.117	98.78	94.65
SimpleHome XCS7-1003-WHT	FCNN	10.15	N/A	97.72
	MEDNN	0.055	99.46	97.72
Kitsune	FCNN	8.223	N/A	84.09
	MEDNN	0.156	98.10	84.09
IoT-DDoS	FCNN	6.492	N/A	83.34
	MEDNN	0.804	87.62	83.34
Wustl	FCNN	5.867	N/A	94.26
	MEDNN	1.102	81.22	94.26

Table 5

Performance comparisons against training procedure.

Procedure	Model	Memory (MB)	Accuracy (%)
Centralized	FCNN	3.969	97.72
	MEDNN	0.102	97.72
Low precision (FP16)	FCNN	1.988	97.67
	MEDNN	0.027	97.70
Decentralized (FedAvg)	FCNN	10.15	97.72
	MEDNN	0.055	97.72

5.4. Model Performances

Table 6 describes the federated model performance evaluated by test set accuracy, precision, recall and harmonic mean on randomly chosen datasets. As the chosen IoT datasets are often unbalanced, test accuracy alone would not be a sufficient metric to measure the performance in

security applications. Instead, the F1 score that corresponds to the harmonic mean of precision and recall is more appropriate. It considers accuracy for each class sample. Employed metrics utilized the True Positive (TP), False Positive (FP), True Negative (TN), False Negative (FN). Accuracy, precision, recall and F1 score are defined in Equation 6, 7, 8 and 9. In each scenario, the optimized MEDNN model maintains similar detection performance across all metrics. The performance metrics result presented in Table 6 remained identical for models trained in centralized settings against each dataset. In each case, accuracy, precision, recall and F1-score remained similar. The results indicate that the utilized number of virtual workers nodes in the federated settings had a minor influence on model performance. This behaviour indicates the lightweight advantage and effectiveness of MEDNN in detecting IoT attacks with good F1-score performance.

Table 6
Testing performance comparisons across datasets.

Dataset	Model	Accuracy (%)	Precision	Recall	F1-Score
IoT-DDoS	FCNN	83.34	0.8334	1.0000	0.9091
	MEDNN	83.34	0.8334	1.0000	0.9091
Wustl	FCNN	94.26	0.9426	1.0000	0.9705
	MEDNN	94.26	0.9426	1.0000	0.9705

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (6)$$

$$Precision = \frac{TP}{TP + FP} \quad (7)$$

$$Recall = \frac{TP}{TP + FN} \quad (8)$$

$$F1score = 2 \times \frac{Recall \times Precision}{Recall + Precision} \quad (9)$$

6. Conclusion

This paper investigated the possibility of reducing memory consumption during DNN training, intending to use DNN-based security solutions in resource-constrained environments. Using FCNN, we proposed a memory-efficient MEDNN for the effective detection of cyber attacks on IoT devices. The effectiveness of MEDNN was tested using eleven IoT benchmark datasets in both centralized and federated learning manners. Experimental results showed that the proposed MEDNN can outperform its benchmark counterparts for memory efficiency and accuracy performance, especially with federated learning. This could be because many clients are involved in training in a federation and thus the cumulative savings are higher than with centralized training on a single node. In addition, the aggregation of models in federated

training can influence faster learning compared with centralized training. However, these initial experimental results are encouraging and warrant further investigation, particularly consideration of more computational nodes in a virtual and realistic federated environment. Therefore, in future, we plan to deploy the model in a real IoT network and examine its capabilities to detect IoT attacks in near real-time in a federated learning setting. In addition, we plan to investigate the impact of adversarial attacks on the proposed MEDNN.

Acknowledgments

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

References

- [1] B. Dong, Q. Shi, Y. Yang, F. Wen, Z. Zhang, C. Lee, Technology evolution from self-powered sensors to aiot enabled smart homes, *Nano Energy* (2020) 105414.
- [2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the mirai botnet, in: 26th {USENIX} security symposium ({USENIX} Security 17), 2017, pp. 1093–1110.
- [3] I. V. Kotenko, I. Saenko, A. Branitskiy, Applying big data processing and machine learning methods for mobile internet of things security monitoring., *J. Internet Serv. Inf. Secur.* 8 (2018) 54–63.
- [4] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, *arXiv preprint arXiv:1610.05492* (2016).
- [5] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for iot big data and streaming analytics: A survey, *IEEE Communications Surveys & Tutorials* 20 (2018) 2923–2960.
- [6] S. Kodali, P. Hansen, N. Mulholland, P. Whatmough, D. Brooks, G.-Y. Wei, Applications of deep neural networks for ultra low power iot, in: 2017 IEEE International Conference on Computer Design (ICCD), IEEE, 2017, pp. 589–592.
- [7] S. Shen, R. Li, Z. Zhao, Q. Liu, J. Liang, H. Zhang, Efficient deep structure learning for resource-limited iot devices, in: GLOBECOM 2020-2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1–6.
- [8] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, E. Ilie-Zudor, Chained anomaly detection models for federated learning: An intrusion detection case study, *Applied Sciences* 8 (2018) 2663.
- [9] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, C. Miao, Federated learning in mobile edge networks: A comprehensive survey, *IEEE Communications Surveys & Tutorials* 22 (2020) 2031–2063.
- [10] A. Imteaj, U. Thakker, S. Wang, J. Li, M. H. Amini, A survey on federated learning for resource-constrained iot devices, *IEEE Internet of Things Journal* (2021).
- [11] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, A.-R. Sadeghi, Diot: A federated self-learning anomaly detection system for iot, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2019, pp. 756–767.

- [12] Y. Liu, N. Kumar, Z. Xiong, W. Y. B. Lim, J. Kang, D. Niyato, Communication-efficient federated learning for anomaly detection in industrial internet of things, in: GLOBECOM 2020-2020 IEEE Global Communications Conference, IEEE, 2020, pp. 1–6.
- [13] Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, L. Tassiulas, Model pruning enables efficient federated learning on edge devices, arXiv preprint arXiv:1909.12326 (2019).
- [14] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, et al., Towards federated learning at scale: System design, arXiv preprint arXiv:1902.01046 (2019).
- [15] Y. Chauvin, D. E. Rumelhart, Backpropagation: theory, architectures, and applications, Psychology press, 2013.
- [16] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, H. Arshad, State-of-the-art in artificial neural network applications: A survey, Heliyon 4 (2018) e00938.
- [17] Y. Oyama, T. Ben-Nun, T. Hoefler, S. Matsuoka, Accelerating deep learning frameworks with micro-batches, in: 2018 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2018, pp. 402–412.
- [18] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, et al., Gpipe: Efficient training of giant neural networks using pipeline parallelism, Advances in neural information processing systems 32 (2019) 103–112.
- [19] S. Han, J. Pool, J. Tran, W. J. Dally, Learning both weights and connections for efficient neural networks, arXiv preprint arXiv:1506.02626 (2015).
- [20] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017, pp. 1273–1282.
- [21] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, Y. Elovici, N-baiot—network-based detection of iot botnet attacks using deep autoencoders, IEEE Pervasive Computing 17 (2018) 12–22.
- [22] Y. Mirsky, T. Doitshman, Y. Elovici, A. Shabtai, Kitsune: an ensemble of autoencoders for online network intrusion detection, arXiv preprint arXiv:1802.09089 (2018).
- [23] M. Siddharth, IoT-DDoS dataset, 2020. URL: <https://www.kaggle.com/siddharthm1698/ddos-botnet-attack-on-iot-devices>, accessed: 2021-02-10.
- [24] M. A. Teixeira, T. Salman, M. Zolanvari, R. Jain, N. Meskin, M. Samaka, Scada system testbed for cybersecurity research using machine learning approach, Future Internet 10 (2018) 76.
- [25] F. Pedregosa, P. Gervais, Memory profiler (python), Python Software Foundation, <https://pypi.org/project/memory-profiler/>. Accessed March 25 (2019).
- [26] P. Raybaut, Spyder-documentation, Available online at: pythonhosted.org (2009).
- [27] I. Zakariyya, Memory efficient federated algorithm., 2021. URL: https://github.com/izakariyya/Robust_DNN_IoT.
- [28] R. Johansson, Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib, Apress, 2018.
- [29] A. Bosman, A. Engelbrecht, M. Helbig, Fitness landscape analysis of weight-elimination neural networks, Neural Processing Letters 48 (2018) 353–373.
- [30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin,

- N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, *Advances in neural information processing systems* 32 (2019) 8026–8037.
- [31] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, J. Passerat-Palmbach, A generic framework for privacy preserving deep learning, *arXiv preprint arXiv:1811.04017* (2018).
- [32] D. Krueger, R. Memisevic, Regularizing rnns by stabilizing activations, *arXiv preprint arXiv:1511.08400* (2015).
- [33] J. Lever, M. Krzywinski, N. Altman, Points of significance: Regularization, *Nature methods* 13 (2016) 803–805.
- [34] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, K. Gopalakrishnan, Ultra-low precision 4-bit training of deep neural networks, *Advances in Neural Information Processing Systems* 33 (2020).