Scientific Journal of Impact Factor (SJIF): 5.71

E-ISSN (O): 2348-4470
P-ISSN (P): 2348-6406

International Journal of Advance Engineering and Research Development

Volume 8,Issue 11, November 2021

A Study on the Deep Learning-based Intrusion Detection Modle for the IoT Network

Sungtaek OH[1], Woong GO[2]

[1,2]*KISA(Korea Internet & Security Agency)*

Abstract —***The escalated growth of the Internet of Things(IoT) has raised the need to detect traffic data in real-time coming from IoT devices and develop autonomous threat analysis technologies. To cope with constantly changing and evolving new security threat, we need to develop advanced detection techniques that provide both real-time cycle-analysis and high-level detection performance. Signature-based detection, a commonly used technique to detect threats, is typically good at detecting known threats, but can't do much to spot the latest changing and evolving security threats. As a way to overcome this limitation, some have suggested semi-supervised learning solutions that use machine learning algorithms to learn traffic from IoT devices only with normal data and then determine whether there are anomaly compared to normal traffic. But the methods are not optimized for detecting IoT traffic anomaly, and are not very useful. Due to different traffic patterns from each household, the traffic anomaly detection system for the IoT devices needs to have a household-specific and, furthermore, a device-specific detection model. Conventional methods require detection systems that minimize the potential challenges because users have to checks anomaly scores from each household and device and set up the threshold setting that determines outliers. Furthermore, existing methods fail to take into account the performance of the routers that will execute the detection model. Generally, the size and complexity of the model correlate positively with detection performance, but the larger and more complex the becomes, the more load on the performance-constrained router device can occur. Accordingly, we need to come up with a methodology that can minimize the load on the router devices while maintaining great detection performance and efficiently handle fast traffic. In this paper, we extract network traffic information communicated by IoT devices from the router to which the devices are connected, and then create vectors based on statistics that can represent unique patterns of network traffic of each device. Subsequently, we propose a method to detect traffic anomaly of each device and identify abnormal network traffic without user intervention.***

Keywords-*Internet of Things; Anomaly Detection; Autoencoder*

I. INTRODUCTION

IoT is a technology that shares data in real time through the Internet by combining information and communication technology with various things. Its main feature is that it allows users to remotely explore various information and control things by connecting commonly-used household appliances to the Internet. The recent development of 5G networks has increased the use of the IoT and relevant markets are rapidly expanding. However, since various IoT devices used in the past are still vulnerable and connected to the Internet, they has been the target of malicious codes such as Mirai. In addition, low-specification IoT products with limited resources have the limits in internal security. It is predicted that attacks on insecure IoT devices will be on the rise, and large-scale infringement cases in the existing ICT environment will shift to the IoT environment in the future and further spread. For users of IoT devices who lack expertise in security, it is hard to even recognize and respond to the infringements. Considering such IoT environment, this paper suggests artificial neural network algorithm-based detection techniques in order to detect and respond to threats associated with diverse, resource-constrained IoT devices by leveraging network packet information of the IoT devices without user intervention.

II. DATA

Due to the absence of authorized actual datasets collected in the IoT environment, we create virtual datasets. For the virtual datasets, we collected network packets by using physical Iot devices. Then we classified and sorted out the stored data by type of use and utilize it as a template. The detection model suggested in this paper uses the header's 5-tuple information except the payload part of the network packet, and the virtually-created dataset consists of time, address information, protocol and packet size.

III. Model

3.1. Feature Extration

We referred to previous studies on network traffic data mining and selected the properties to be extracted by taking into account relevance, feasibility, etc. while most precedent studies have extracted various variables (payload, flag, etc.) by using Wireshark, etc., the models suggested in this paper rather mainly analyzed the features of using 5-tuple and time

information after considering the limits of Raspberry Pi-based smart gateways. Figure 1. is a variable and procedure for extracting features from virtual datasets.

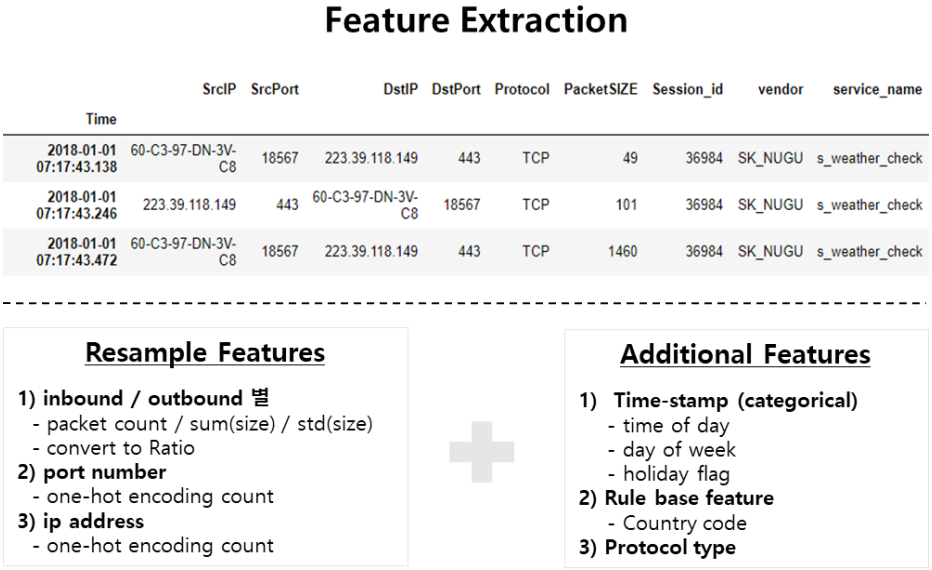

Figure 1. Feature Extraction Process

3.2. Model Comparison

This paper conducted the research by focusing on machine learning techniques that can maintain the performance of a model above a certain level and minimize complexity, learning time, etc. Among them, we selected the algorithms (One-Class SVM, Isolation Forest, and Autoencoder), which are well-suited and high-performance algorithms for high-dimensional problems and most frequently used ones for anomaly detection [1]. Indicators used to evaluate the selected models are model performance, model complexity, and learning time, as shown in the following Figure 2..

Evaluate indicator	Detail List
Model Performance	• Fit • Vanishing Gradient Problem • Noise
Model Complexity	• Number of (Neuron, Layer) • Number of Parameter
Time	• Training time • Convergence Rate

Figure 2. Evaluation Indicator

3.3. Modle Description

3.3.1. One-Class SVM

One-Class SVM is an unsupervised learning version of the existing machine learning algorithm, SVM. This model maps the data into the proper feature space via the kernel function and separates them from the origin with maximum margin. It corresponds to identifying the smallest hypersphere around the normal point [2]. One-Class SVM algorithms perform well even on small datasets and show superior generalization power. This algorithm is best suited to the high-dimensional problem and the time complexity is $O(n^2)$. The hyper-parameters of the algorithm are as shown in Figure 3..

Parameter	Description	Detail List
Kernel	Type of Kernel function	• Linear • Polynomial • rbf
Gamma	Kernel function (polynomial or rbf)	• 'auto': 1/n_features • 'scale': 1/(n_features * X.var())
Nu	Strength of Slack variable	• Float

Figure 3. One-Class SVM hyper-parameter

3.3.2. Isolation Forest

Isolation Forest is a tree-based splitting methodology that isolates all data observations. This defices the distance to the terminal node(leaf node), where a specific object is isolated, as an outlier score, and the shorter the average depth, the higher the outlier score. In this algorithm, if the anomaly data is to be isolated, is has a depth close to the root node. If the normal data is to be isolated, it has the depth close to the terminal node of the tree [3]. The Isolation Forest algorithm showed high performance, ranging from small to large amounts of data. It has a low false alarm rate and the time complexity was also low as O(n). However, when learning this model, its disadvantage is learning poorly in case of containing a high ratio of anomaly data. The hyper-parameters of this algorithm are as shown in Figure 4..

Parameter	Description	Detail List
n_estimators	Number of ensemble base estimator	• Integer
max_samples	Numver of sub-sampling	• Integer
max_features	Number of feature	• Integer
bootstrap	Activation Bootstrap sampling	• Integer

Figure 4. Isolation Forest hyper-parameter

3.3.3. Autoencoder

Autoencoder algorithms show unrivaled excellent performance in many fields based on their strong representation power [4]. This has also been demonstrated in the precedent studies of existing network data-based anomaly detection [5]. Autoencoder is a technique for extracting features of low dimension based on Manifold hypothesis and is used primarily in anomaly detection. For artificial neural networks, autoencoder basically tunes a combination of various parameters and creates a single model. Numerous combinations appear in the relevant procedure and we can experiment and find the optimal parameters to meet the purpose Figure 5..

Parameter	Description	Detail List
Hidden layer	width & depth	• Integer
optimizer	algorithme	• GD • RMSprop • ...
Activation function	Function for adding non-linerirty to hidden nodes	• Sigmoid / Relu / tanh ...

Figure 5. Autoencoder hyper-parameter

IV. Experiment

4.1. Data Description

We experimented by creating virtual data and set the period of creating the data to 12 months. This study also set the period of creating training data to 9 months, and the validation data (test) to 3 months. Training data was created only with normal data, and we included abnormal behaviors to generate verification data. The aim of the anomaly detection model proposed in this paper is to detect outliers by learning only normal data. The scenario to create virtual datasets is as shown in Figure 6.. We experiment on the server since there are so many tuning spaces that it takes too much time to perform all the experiments from Raspberry Pi.

	Time	Action	Maximum number of trials
Week	7:05~8:00	Anomaly Behavior	10
	7:15~7:25	Weather Check	5
	7:30~7:45	Web Surfing	2
	18:30~20:00	Streaming Music	10min
	19:00~19:30	Item Order	1
	20:30~20:45	Web Surfing	5
Weekend	8:05~9:00	Anomaly Behavior	3
	11:30~11:45	News Check	2
	12:00~13:45	Item Order	3
	19:30~19:20	Item Order	1
	20:45~22:00	Web Surfing	5

Figure 6. Generate Virtual Dataset Configuration

4.2. Experiment Description

We extracted feature information by re-sampling generated packets in 5 seconds. The variables and details used are as shown in Figure 7.. Figure 7. Shows the model and evaluation index used in the experiment.

- Extract Features: resampling in 5 seconds

Feature	Detail info.
Main Features	• Inbound packet count / sum(size) / std(size) • outbound packet count / sum(size) / std(size) • Ratio of inbound and outbound
Source / Destination IP address(one-hot)	• Known IP • Unknown IP
Source / Destination port(one-hot)	• Well-Known port • Unknown port

- Models & Evaluation indicators

모델		One-Class SVM	Isolation Forest	Autoencoder
Library		sklearn	sklearn	tensorflow
variable		• Type of Features • Type of Kernel • Size of nu • Size of gamma	• Type of Features • n_estimators • Whether or not the bootstrap • Size of max_features	• Type of Features • learning rate • Optimizer • activation function • Initializer • hidden_layer size
evaluation indicators	Anomaly Detecting Performance	Recall / Precision / Accuracy / Etc.		
	Model Performance	Model Size / Training Time / Prediction Time		

Figure 7. Feature & Evaluation Indicator

4.3. Experiment Result

After checking the experimental environment Figure 8., data Figure 8., and model-specific evaluation results Figure 8., we found that One-Class SVM using rbf kernels and Autoencoder models expressed great performance. Due to difficult linear separation from the distribution of the data, it seems that using a model that can reflect non-linear features may secure better performance in anomaly detection.

1. Autoencoder Training: P: 13,449 Test P: 742 N: 1197

Parameters						Result						
Features	Learning Rate	Optimizer	Activation	Hidden layer	Initializer	TP	FP	FN	TN	Train_time	Predict_time	Model_size
All	0.0001	GD	Sigmoid	64	Xavier	736	0	6	1197	6.1s	0.15s	99KB
	0.005	GD	Sigmoid	64	Xavier	737	0	5	1196	5.8s	0.10s	120KB

2. One-Class SVM

Parameters				Result						
Features	Kernel	nu	gamma	TP	FP	FN	TN	Train_time	Predict_time	Model_size
All	poly	0.0001	auto	736	128	6	1069	0.003s	0.001s	2.1KB
	rbf	0.01	auto	742	9	0	1188	0.003s	0.001s	34KB
Main_only	rbf	0.01	auto	737	7	5	1190	0.01s	0.002s	4.3KB

3. Isolation Forest

Parameters				Result						
Features	n_estimators	bootstrap	max_features	TP	FP	FN	TN	Train_time	Predict_time	Model_size
All	500	TRUE	1	737	131	5	1066	0.79s	0.26s	1,908KB
	1000	FALSE	1	736	133	6	1064	1.614s	0.52s	4,012KB
Main_only	500	FALSE	10	742	120	0	1077	0.85s	0.27s	3,578KB

※ Environment : 2.4GHz 20core(CPU), 263GB(Ram), Quadro4000(GPU)

Figure 8. Summary

4.4. Discussion

4.4.1. One-Class SVM

Among the SVMs, when using the rbf kernel, it showed the greatest performance. The learning process was completed earlier than expected, but we cannot ignore the time complexity above $O(n^2)$. One-Class SVM must include existing data when retraining, making it difficult to estimate the time required.

4.4.2. Isolation Forest

The smaller the size of the tree is, the greater the variations in the result value will be. Isolation Forest showed a relatively higher ratio of false positive than other models, which is contrary to previously-known results. If the distribution of anomaly data is similar to that of normal data, the downside is unsatisfactory performance. In situations where the normal-anomaly ratio of verification data is almost the same, the normal-anomaly ratio in case of learning could be different and may not be properly predicted.

4.4.3. Autoencoder

Autoencoder showed high performance in the majority of evaluation criteria. The need for feature engineering has been reduced compared to existing machine learning algorithms such as One-Class SVM and Isolation Forest. Autoencoder trains only new data additionally even when re-learning, so it is very useful at the re-learning stage. However, since the complexity of the model is higher than that of the other two models, it is likely to result in increased inference time or over-fitting.

4.4.4. Raspberry PI Experiment

The behavioral environment of the detection model proposed in this paper is the Raspberry Pi model. As a result of our validation test by training representative models of Raspberry Pi in the relevant environment, there was no difference in anomaly detection performance results. The results of the learning and detection time comparisons of the models were also similar to that of experiments conducted on the server. For One-Class SVM algorithms, the quadratic programming optimization methodology requires application of other optimization algorithms because memory errors hamper learning.

V. DETECTING

5.1. Base Model

Based on the experimental results, we conducted a study to apply a lightweight methodology to autoencoder, which showed good performance. The researchers of this paper established the hypothesis to enable lightweights that can reduce learning, inference time, and complexity while maintaining the highest performance possible. We also explored many methodologies and the latest lightweight ML methodologies suggested for effective learning. And then we tuned the parameters of existing neural networks by reinterpreting them from the aspect of lightweight. Furthermore, the researchers found ways in which the structure of the model itself can be light-weighted.

5.2. Parameters

5.2.1. Network Strucure

When optimizing the number of layers and neurons, we can effectively reduce the size and complexity of the model. It can also reduce parameters by eliminating useless variables by extracting proper features. We can also solve sparse dimensionality issue that occurs in case of one-hot encoding IP address and port information among 5-tuple information by using embedded multidimensional vectors, which enables us to extract significant feature information and utilize it in the model.

5.2.2. Weight Initialization

In the output distribution of randomly-initialized neurons, the variance increases as the number of inputs increases. To mitigate, we can normalize the variance of each neuron's output to 1 by scaling its weight vector by the square root of its number of inputs. This can empirically helps with the vanishing gradient problem, improving the convergence rate. Xavier initialization is used with sigmoid activation functions, and ReLU exploding gradients can be applied effectively with He initialization.

5.2.3. Pre-Processing

Zero-centered normalization (scale to [-1,1]) is applied. It effectively enhances the convergence rate when using Sigmoid and ReLU as an activation function. Dimensionality can be reduced when applied with Principal Component Analysis(PCA). By removing the correlation between input variables, we can reduce parameters.

5.2.5. Activation Function

A Select the optimal activation function by using several activation functions. The sigmoid function is the simplest and basic activation function. Tanh has zero-centered function effects while ReLU is less susceptible to vanishing gradients and makes the convergence faster.

5.2.6. Optimizer

The three optimizers, RMS Prop, Adam, and Nesterov Momentum, are applied: the followings are advantages of each optimizer. RMS Prop automatically adjusts the learning rate per parameter, improving AdaGrad's shortcomings through moving average, which reduces the learning rate unconditionally as much as it is updated. Adam is an optimizer that combines the benefits of AdaGrad and RMS Prop and is known to perform best along with SGD+Nesterov Momentum. Nesterov Momentum Figure 9. has a fast converge rate effect, performing excellently when used with SGD.

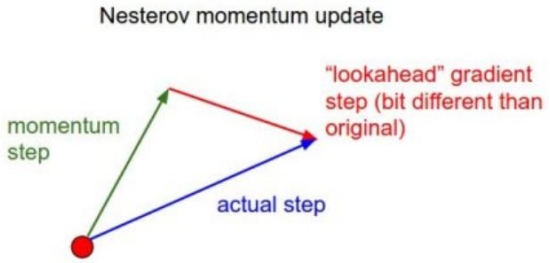

Figure 9. Nesterov Momentum

5.2.7. Mini-Batch

In learning, we can leverage the features of learning in a more stable way than SGD by using Mini-Batch with increased convergence speed over Batch. However, if the size of Mini-Batch is too large, it is prone over-fitting, so we need to be careful.

5.2.8. Drop out

As a regularization technique, applying drop out reduces the impacts of noise. In learning, we can keep a neuron active with a certain probability P or deactivate the neurons with a probability 1-p. In this lightweight test, we use inverted drop out, which performs the scaling when learning and remains the forward pass untouched in case of validation.

5.2.9. Batch Normalization

Batch Normalization tackles internal covariate shift problem, thereby enabling to use faster learning rate and making initialization and activation function less important. It also allows a faster convergence rate. Batch normalization also enhances robustness and reduces the impact of randomness.

5.2.10. Pruning

As the artificial neural network structure becomes more complex and larger, it may exhibit very powerful performance, but also consume lots of memory and computational resources. The anomaly detection model proposed in this paper used pruning techniques to minimize such issues because it operates on resource-constrained Raspberry Pi. After learning, this model removes all connections with low weight value, converting to low-density sparse networks, and then re-trains. Through this process, it allows us to maintain high performance while removing a number of unaffected parameters [6]. The application techniques are as follow: add a binary mask layer before all the weight layers. Arrange layer-specific weights after forward propagation. The smallest weight of the individual layer is masked and retrained, where the masked weight values are not updated during the back propagation. Repeat the above process until it reaches the established sparsity value, and the learning phase ends upon its arrival. The sparsity value (=s) is determined by Figure 10. [7].

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for} \quad t \in \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + n\Delta t\}$$

Figure 10. Sparsity value

5.3. Experiment

5.3.1. Description

The test is conducted based on the same variables and the same evaluation criteria as the previous test performed in Section 4. We conducted the test with new datasets by adding noise to existing datasets. For the lightweight test, we combined parameters such as Figure 11. and derived the 1st test results. After checking the corresponding results, we selected high-performance models and conducted the final test in the Raspberry Pi environment.

Features	Learning Rate
Learning Rate (3)	0.001, 0.005, 0.0001
Optimizer (5)	GD, RMSprop, Adam, Momentum, Nesterov
Activation Function (3)	Tanh, Relu, Sigmoid
Initializer (3)	He, Xavier, Default
Hidden Layer (6)	2-Layer : 128_64, 64_32, 32_16 1- Layer : 64, 32, 16
Embedding layer (4)	No, 5, 20, 50
Batch size (3)	64, 128, 256

※ **Environment : 2.4GHz 20core(CPU), 263GB(Ram), Quadro4000(GPU)**

Figure 11. Parameter combination for lightweight experiments

5.3.2. Anomaly Detection Experiment Result

Based on the results of the first experiment, we selected a combination of parameters and then conducted the experiment on Raspberry Pi. There was little difference in the results of anomaly detection performance between models, and the results of comparing the models' learning-time and inference time also turned out to have good performance similar to conventional experiments Figure 12..

P: 799 N: 1461

Features	Learning Rate	Optimizer	Hidden layer	Embedding layer	activation	Initializer	TP	FP	FN	TN	Train_time	Predict_time	Model_size
All	0.005	Momentum	128_64	20	Relu	he	795	0	4	1461	112s	0.44s	377KB
	0.001	Nesterov	128_64	20	tanh	Xavier	795	0	4	1461	114s	0.46s	377KB
	0.005	Momentum	128_64	5	tanh	Xavier	795	0	4	1461	108s	0.46s	289KB
	0.001	Nesterov	128_64	5	tanh	Xavier	795	0	4	1461	109s	0.49s	289KB
	0.005	Nesterov	128_64	5	Relu	he	795	0	4	1461	106s	0.45s	289KB

※ Raspberry PI Environment : 1.4GHz 4core(CPU), 1GB (Ram)

Figure 12. Experiment Result (Raspberry PI)

5.3.3. Results of Applying Lightweight Techniques (Fast Convergence & Generalization)

It is possible to learn effectively by applying various learning techniques of autoencoder. Prime examples are shown in Figure 13., Figure 14..

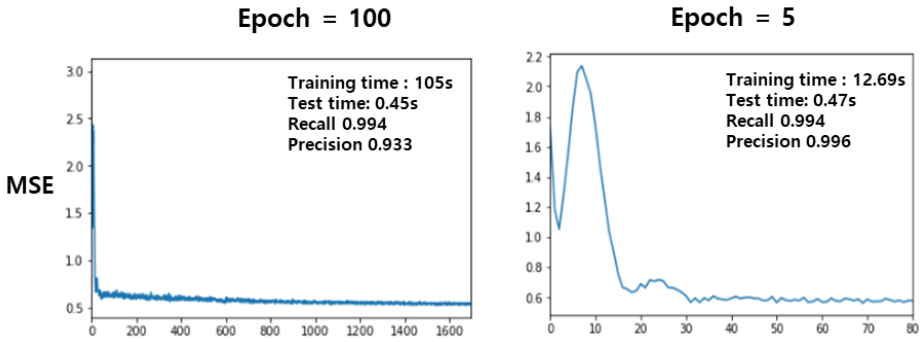

Figure 13. Early Stopping Effects

Reduced unnecessary epochs can reduce learning time. It can also enhance the generalization power.

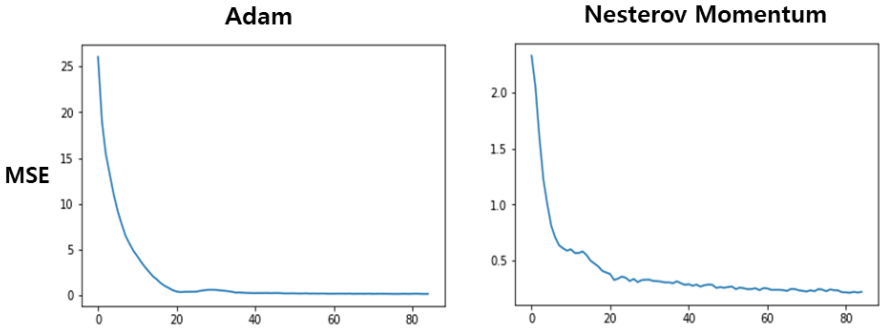

Figure 14. Optimizer Effect

The momentum algorithm allows fast convergence rate.

5.3.4. Results of Applying Lightweight Techniques (Pruning)

This study produced the results of maintaining similar accuracy, using less weight, and reducing the prediction time Figure 15.. After applying pruning to existing results, we found that the number of weights used decreased by 90% while the prediction time doubled.

구분	TP	FP	FN	TN	Recall	Precision	사용된 weight 개수	Train_time	Predict_time	Model_size
No pruning	795	4	1	1460	0.998	0.994	18688	97s	0.37s	234KB
Pruning	795	4	9	1452	0.988	0.994	**1920** (Reduce 90%)	181s	**0.18s** (Reduce 200%)	430KB

Figure 15. Lightweight Experiment Result (Pruning)

5.4. Discussion

This pruning research results showed that DL generally has a better interpretation of multi-dimensional spaces than ML. Pruning took a little longer to learn but was effective in reducing inference time with similar performance. Early-stopping techniques can be applied to avoid overfitting and increase the generalization power. Additionally, momentum-based optimizer is effective to increase the convergence rate. Furthermore, embedding can be helpful when adding IP and Port

information into variables, and we confirmed that certain variables do not directly affect the performance of the model. In order to achieve lighter model weight, it is necessary to eliminate such useless variables. Using PCA as a preprocessing did not have much impact. In this experiment, batch normalization did not bring about big impact, but as it has achieved high performance thanks to stabilized learning in many fields, we need to consider the technique in future experiments based on real data.

VI. FUTURE WORK

Our experiments confirmed that autoencoder depends on the MSE threshold in terms of anomaly detection performance. This makes the test difficult by setting each threshold value for anomaly detection in the process of creating an anomaly detection model for each IoT device. To solve this issue, we will further study how to map the learned features to latent spaces with clear boundaries between normal and anomalous behaviors. Additionally, autoencoder has a weak point that is vulnerable to noise of learning data and may be also vulnerable to noise in learning with less data [8] To this end, we can conduct a future study on how to initialize partial weights of local anomaly detection models or to receive massive volumes of pre-collected learning data by leveraging the encoder weight of autoencoder, which was learned as global data from Cloud. Figure 16..

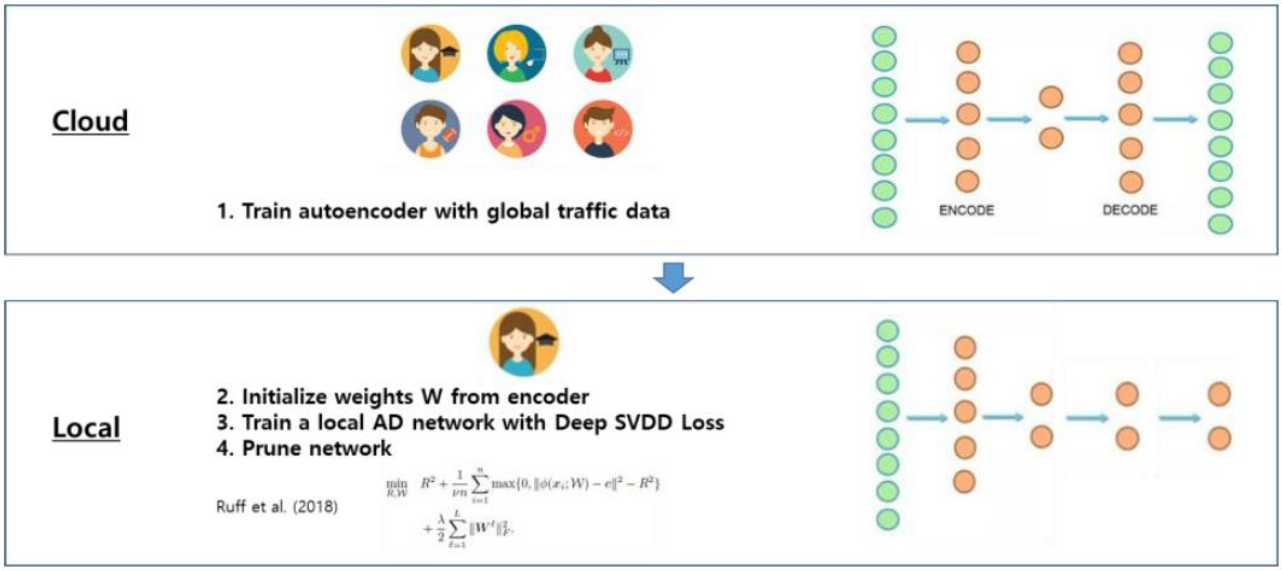

Figure 16. Transfer Learning Example (Solve Cold Start Problem)

ACKNOWLEDGEMENT

This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government (MSIT)
(No.2018-0-00232, Cloud-based IoT Threat Autonomic Analysis and Response Technology).

REFERENCES

[1] Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., & Robinson, S. (2017, March). Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In Workshops at the Thirty-First AAAI Conference on Artificial Intelligence.

[2] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., & Platt, J. C. (2000). Support vector method for novelty detection. In Advances in neural information processing systems (pp. 582-588).

[3] Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008, December). Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE.

[4] Chalapathy, R., &Chawla, S. (2019). Deep Learning for Anomaly Detection: A Survey. arXiv preprint arXiv:1901.03407

[5] Meidan, Y., Bohadana, M., Mathov, Y., Mirsky, Y., Shabtai, A., Breitenbacher, D., &Elovici, Y.(2018). N-BaIoT-Network-based Detection of IoT Botnet Attacks Using Deep Autoencoders. IEEE Pervasive Computing, 17(3), 12-22

[6] Han, S., Mao, H., & Dally, W. J. (2015). Deepcompression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149

[7] Zhu, M., & Gupta, S. (2017). To prune, or not to prune: exploring the efficacy of pruning for model compression. In Proc. NIPS Workshop on Machine Learning of Phones and other Consumer Devices.

[8] Zhou, C., & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 665-674). ACM.

