An Evolutionary Learning Approach Towards the Open Challenge of IoT Device Identification

Jingfei Bian[1,2], Nan Yu[1,2], Hong Li[1,2], Hongsong Zhu[1,2(✉)], Qiang Wang[1,2], and Limin Sun[1,2]

[1] Beijing Key Laboratory of IOT Information Security Technology, Institute of Information Engineering, CAS, Beijing, China
{bianjingfei,yunan,lihong,zhuhongsong,wangqiang3113,sunlimin}@iie.ac.cn
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

Abstract. Internet of Things (IoT) device identification has become an indispensable prerequisite for secure network management and security policy implementation. However, existing passive device identification methods work under a "closed-world" assumption, failing to take into account the emergence of new and unfamiliar devices in open scenarios. To combat the open-world challenge, we propose a novel evolutionary model which can continuously learn with new device traffic. Our model employs a decoupled architecture suitable for evolutionary learning, which consists of device feature representation and device inference. For device feature representation, an auto-encoder based on metric learning is innovatively introduced to mine latent feature representation of device traffic and form independent compact clusters for each device. For device inference, the nearest class mean (NCM) classification strategy is adopted on the feature representation. In addition, to alleviate the forgetting of old devices during evolutionary learning with new devices, we develop a less-forgetting constraint based on spatial knowledge distillation and impose control on the distribution distance between clusters to reduce inter-class interference. We evaluate our method on the union of three public IoT traffic datasets, in which the accuracy is as high as 87.9% after multi-stage evolutionary learning, outperforming all state-of-the-art methods under diverse experimental settings.

Keywords: IoT device identification · Deep learning · Closed-world · Evolutionary model · NCM · Spatial knowledge distillation

1 Introduction

In recent years, deep learning has been employed to solve increasingly serious network security problems, especially passive IoT device identification [5,9,14,23–26,29], which has shown unlimited potential and achieved remarkable success. However, once the device identification model that performs well in the

F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 20–38, 2023.
https://doi.org/10.1007/978-3-031-25538-0_2

laboratory is deployed in a real open environment, it will fall into the dilemma of performance degradation [28]. One of the main reasons behind the dilemma is that the model is based on a biased assumption during the designing phase, that is, the IoT device types at training time and the IoT device types to be identified are from the same set of predefined types. However, the practical scenarios are complex and changeable, where new and unknown devices will continue to emerge. The knowledge acquired by the static model from the old device traffic will become outdated and unavailable in a short period. Hence, this assumption deviates from reality, which causes the notorious *open-world* [12] problem during the model deployment phase.

To address the problem of the open-world, most of the existing methods [9,13] are based on the idea of Out-of-Distribution for detection. They not only allow the model to identify the known devices but also detects the unknown, which have made some progress. However, the methods merely mark unfamiliar devices as unknown without considering how to further recognize unknown devices. A realistic scenario for IoT device identification is, as time goes by when devices that have never been seen appear on the network, it is more imperative for the model to continuously learn, upgrade and evolve with the new data to adapt to the variations, rather than just to identify as the unknown. Therefore, we intend to solve the open-world problem in the field of IoT device identification from the perspective of model evolution. Our insight is that the fundamental solution for the open-world problem should be to accept new devices, not reject them, just as human beings always have the ability to continuously learn and internalize knowledge in new environments. To achieve the goal, we propose a novel evolutionary model which can continuously learn with new device traffic. And we borrow ideas from class incremental learning [15] and continuous learning [19] in the field of image recognition to overcome the inevitable catastrophic forgetting that arises in evolutionary learning. As far as we know, there is currently no evolutionary learning method in the field of passive network device identification, which is one of the motivations of this paper.

Our Method. We adopt a decoupled architecture for evolutionary learning, divided into representation and device inference. For representation, we design an automatic feature representation scheme for device traffic, which is data-centric. In specific, we propose an auto-encoder mapping algorithm based on metric learning to mine latent features suitable for evolutionary learning and automatically perform feature space layout with intra-class compactness and inter-class separation. Following, for device inference, we make use of the Nearest Class Mean (NCM) classification strategy on the latent representation. Additionally, to alleviate the catastrophic forgetting of old devices during evolutionary learning, we only leverage a tiny number of representative exemplars to ensure the continuity of knowledge in the evolutionary learning process based on the Spatial Knowledge Distillation.

Contributions. In summary, the paper has following main contributions:

– We innovatively propose an evolutionary learning method to overcome the open-world problem of passive IoT device identification. We introduce a

decoupled architecture suitable for evolutionary learning, divided into device feature representation and device inference.
– To alleviate catastrophic forgetting during evolutionary learning, we develop a less-forgetting constraint based on Spatial Knowledge Distillation and introduce metric learning to control the inter-class distribution distance.
– We evaluate our method on the union of three public datasets [3,17,24], with traffic data from 66 different types of devices. The experimental results show that our method outperforms existing methods under diverse settings, with an accuracy of up to 87.9% after multi-stage evolutionary learning.

The remainder of the paper is organized as follows: Sect. 2 details our proposed method. The experiment and evaluation are presented in Sect. 3. Section 4 reviews related work and we discuss and conclude all the work in Sect. 5.

2 Proposed Method

2.1 Motivation and Problem Definition

Due to the rapid development of IoT applications, tens of thousands of new devices, which have never appeared before, emerge in the network every day [12]. Therefore, in practical scenarios, the network devices are usually in a process of dynamic change and continuous increase [28], which brings difficulties for passive IoT device identification. Actually, the main challenge in the open scenario is *the introduction of the new class.*

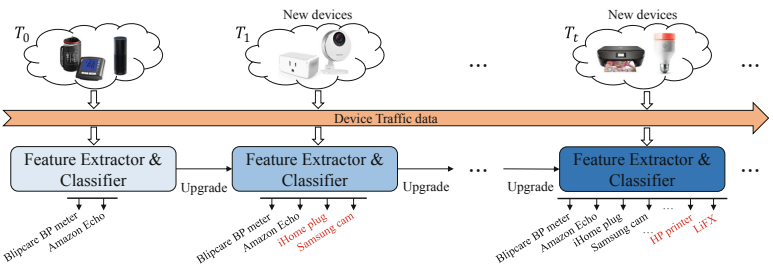


Fig. 1. General scenarios of model evolution.

We consider that the fundamental strategy to solve the open-world problem is to make the model continuously evolve during the life cycle to adapt to the transformation of the network. We regard the introduction of each batch of new devices as an evolution task. As shown in Fig. 1, in the open scenarios, the unknown number of tasks with previously unseen devices arrive at the model sequentially, denoted as $T_0, T_1, \cdots, T_t, \cdots$. The new traffic data contained in the task T_t at time point t-th is $X_t = \{x_i^t, y_i^t\}_{i=0}^{N_t}$, where, x_i^t and $y_i^t \in C_t$ is i-th device traffic sample and its label respectively. N_t is the number of samples and C_t represents the disjoint label set at time point t-th, where $C_t \wedge C_k =$

$\emptyset, \forall k \in \{0, 1, \cdots, t-1\}$. The model of the current stage can only be upgraded and evolved based on the currently visible new device data set and the model of the previous stage. And it would be evaluated on the test sets Z_t, whose label set is the union of all the encountered classes $\bigcup_{j=0}^{t} C_j$.

Our problem scope is how to effectively continuously evolve the model when traffic data for new devices is available or only a few traffic data is available. A naive idea is to use the latest traffic data to fine-tune the model of the previous task without much consideration when the new data arrives. However, the approach will lead to serious catastrophic forgetting [4]. Because the model will pay too much attention to the latest task, and the performance on the old tasks will drop significantly. The other simplest way is to store all the data that has appeared, and every time a new task comes, retrain a new model from scratch using all the data. However, this scheme greatly wastes storage and computing resources and even involves privacy protection issues, which is not feasible in practical applications. The above two ideas are extreme choices between consumption and performance, and also show the *main difficulty* of model evolution: how to avoid forgetting and achieve better performance while reducing the consumption of memory and computing resources.

2.2 Overall Framework

In this section, we present the overall framework of the method, as shown in Fig. 2. Following, we begin by describing our analysis and findings, which underlie our design.

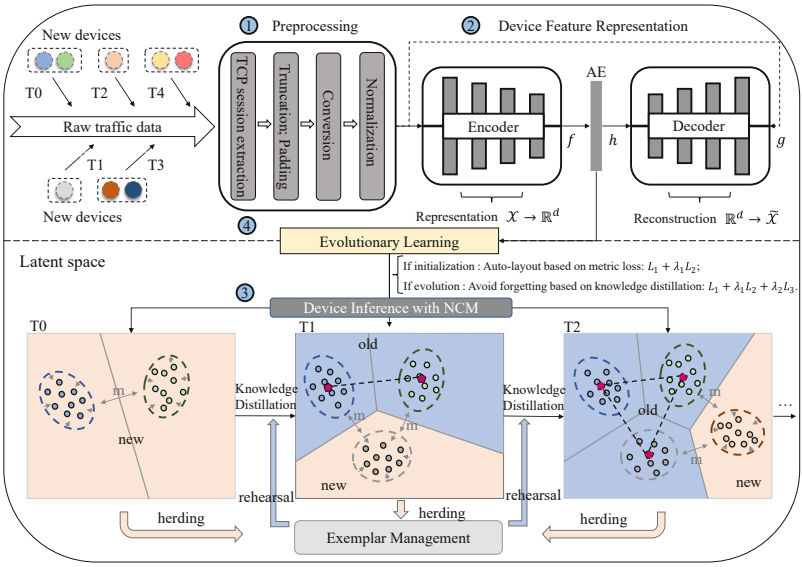
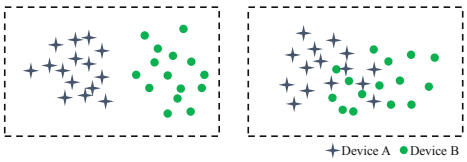


Fig. 2. The framework of our method.

1. *Feature representation of device traffic.* Most IoT device identification methods [23–26] rely on handcrafted features. But for evolutionary learning, it is impossible to expect current handcrafted features to be useful for modeling new device behaviors in the future. Therefore, it is necessary to design a traffic feature representation method suitable for evolutionary learning that directly relies on the data itself rather than hand-designed.
2. *Defects of traditional network architecture.* The usual classification neural network can be interpreted as a feature extractor followed by a classifier. For evolutionary learning, a universal feature extractor is crucial. If the extracted features only serve the current task, when a new device arrives, the parameters of the extractor will be drastically modified to regain new features favorable for the new task, which will exacerbate forgetting. In addition, the usual classifier is a linear fully-connected layer with as many softmax output nodes as classes observed so far. Rebuffi has shown in [20] that linear classification layers can become unstable and uncontrolled during evolutionary learning. Moreover, when learning to identify new devices, it is necessary to adjust and increase the output units of the network structure, which is cumbersome and inconvenient. In summary, traditional task-centric feature extraction methods and linear fully-connected classification are detrimental to evolutionary learning.
3. *Inter-class confusion and interference.* Figure 3 presents our findings from experiments on the IoT device dataset of UNSW [24]. The features of the well-trained devices form respective distribution regions, and there is no overlap between the features of base classes. However, with the introduction of the new device, the features of the old devices are confused and overlapped. We argue that the confusion between features of the old classes directly leads to catastrophic forgetting. We will explain later how to use metric learning to control the inter-class separation to reduce confusion and interference.



(a) Initial(T_0) (b) Evolutionary(T_1)

Fig. 3. Feature distribution of old classes. On the basis of the well-trained model for classifying device A and device B, we use finetuning to evolutionary learn novel device and observe the feature space distribution. 3(a) Initially, the feature distribution of the base class is well separated. 3(b) During evolutionary learning, the feature distributions of the old classes are confused and overlapped.

2.3 Preprocessing

Generally, we can capture the traffic data by deploying sniffers on the gateway or the router. In the application scenarios, the devices will generate different traffic according to the special services. The service-related traffic behavior provides various device features, which facilitate IoT device identification [12]. However, most existing feature extraction methods rely on handcrafting. They are designed from closed scenarios where all training data is known. In the open world, it is impossible to anticipate which crafted features are useful to unknown devices. Therefore, hand-crafted methods are not suitable for evolutionary learning.

In contrast to methods that rely on hand-crafted features, we would like to make the model automatically mine latent features from traffic data. We find that when a device provides services, it always produces the natural sequences of TCP packets that are highly related to its services. In contrast, due to the connectionless nature of the UDP protocol, the network behavior based on UDP is less distinguishable than TCP, such as the largest number of UDP network services, NTP and DNS, the inadequacies of which have been discussed in [9]. Hence we consider using the TCP session as the basic unit to present traffic. In specific, we split a TCP session by the network five-tuple (source IP, source port, protocol, destination IP, destination port) and the session establishment or teardown flag. However, due to uncontrollable reasons such as network congestion, captured packets may be out of order, lost, or repeated, so we correct or ignore them according to the five-tuples, sequence numbers and timestamp intervals.

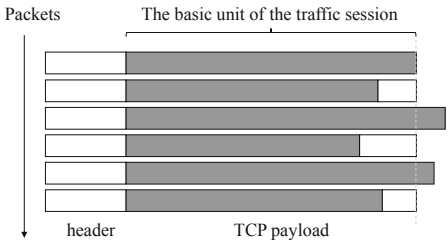


Fig. 4. The basic unit of TCP session.

Then, we extract the TCP payload of all sessions and drop the headers to avoid the model over-fitting with IP and MAC addresses [18]. A TCP session includes the whole process of network communication between a pair of device subjects, the dominant and responder. We arrange all communications of a session in a vertical request-response sequence, forming a 2D matrix. It is observed that most TCP sessions are short and compact, almost ending the session within the first 16 interactions. Therefore, we intercept the first 16 packets, keep the first 256 bytes of each payload, and pad any shortfalls with zeros so that we can express the TCP session with a smaller size. Additionally, for encrypted traffic of SSL/TLS, the first 16 packets always contain the entire process of key exchange and key negotiation, which is beneficial for the identification of encrypted traffic. Before the training data is fed into the neural network, the data will be

converted from each byte to decimal and normalized. As shown in Fig. 5, we randomly sample 4 sessions of 6 devices from the public dataset UNSW [24] and visualize them using grayscale maps. It is found that behavioral patterns differed significantly between devices.

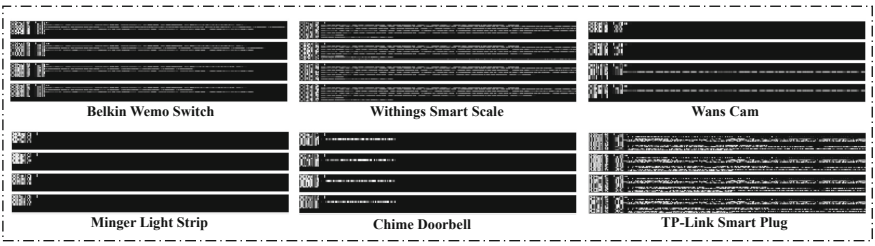


Fig. 5. Grayscale visualization of preprocessed data.

2.4 Device Feature Representation Learning

The analysis at the beginning of this section shows the limitations of traditional architectures. Thus we abandon the structural form of a task-centric extractor followed by the fully connected classifier, and creatively employ the stacked auto-encoder to learn latent representations for devices. Figure 2 shows the architecture of our method. We value the good representation ability of the auto-encoder. Given the inputs $X \in \mathcal{X}$ and features $h \in \mathbb{R}^d$, the auto-encoder can be divided into two parts: encoder $f : \mathcal{X} \to \mathbb{R}^d$ and decoder $g : \mathbb{R}^d \to \mathcal{X}$, which solves the mapping to minimize the *reconstruction loss* between the inputs and outputs:

$$L_1 = \frac{1}{N} \sum_{i=1}^{N} \parallel x_i - g[f(x_i)] \parallel_2^2 . \tag{1}$$

Here, encoder f maps the original inputs X to latent space features $h = f(X)$ and decoder g reconstructs h back to X, $\tilde{X} = g(h)$. The output h of the encoder is named encoded feature or encoded embedding. We design the dimension d of h to be far smaller than the input, forcing the f to capture the most prominent and representative features. Therefore, the captured features are good representations, as they do not serve a specific task, but work to characterize and express all latent information.

However, the auto-encoder is not well compatible with traditional fully connected classification layers, and auto-encoders cannot perform classification tasks independently. Hence, we introduce metric learning [21] into the representation of auto-encoder to automatically achieve independent compact spatial distributions. Specifically, we combine ideas from auto-encoding and metric learning to produce a compact, representative, and class-separable embedding space:

$$L_2 = \frac{1}{N} \sum_{i=1}^{N} \max \left(\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + m, 0 \right) . \tag{2}$$

Equation (2) refers to the triplet loss [21], which takes a triple of samples (x_i^a, x_i^p, x_i^n) and enforce the x_i^a (*anchor sample*) of a certain device to be closer to all x_i^p (*positive sample*) of the same device than to x_i^n (*negative sample*) of any other device in the feature space. Here, $\|f(x_i^a) - f(x_i^p)\|_2^2$ represents the Euclidean distance metric between the anchor and the positive, and $\|f(x_i^a) - f(x_i^n)\|_2^2$ represents the Euclidean distance metric between the anchor and the negative. m is the minimum interval between positive and negative samples relative to the anchor sample. As illustrated in the latent space in Fig. 2, The loss function intents to ensure that any traffic samples from different devices are sufficiently far apart.

As shown in Fig. 2, combining with Eq. (1) and Eq. (2), encoder f can map the traffic to a latent space with tight clusters classes of different devices, which could be used for identification. The process is like we employ an encoder to assign the most appropriate and compact feature distribution for each device. Our representation learning method is not task-centric. The metric-based encoding algorithm can extract common latent features, which solves **the problem of traditional feature extractors**. In addition, we can intervene in the spatial layout by controlling the parameter m, which we will discuss in subsequent experiments to address **the problem of confusion and interference**. It is worth mentioning that, during evolutionary learning, we do not need to add neurons in the output layer of a deep network model to accommodate new devices, but can easily expand new devices in the latent representation space.

Algorithm 1: Device Inference

Input: Devices data set $X = \{x_0, \cdots, x_{N-1}\}$; devices class $Y = \{y_1, \cdots, y_K\}$; the number of samples in each class subset is $N^1, \cdots, N^K$; the test set $(\widehat{x}_j, \widehat{y}_j) \in Z$, where $j = 0, \cdots, \widehat{N} - 1$, $\widehat{N}$ is the total number of test samples; The encoder f.

Output: The inferred device class y.

for class $k = 1$ to K **do**
  $\mu_k = \frac{1}{N^k} \sum_{i=0}^{N^k-1} f(x_i^k)$; `// The mean of the class k`
end
`// Device Inference`
for $j = 0$ to $\widehat{N}$ **do**
  for $k = 1$ to K **do**
    $d_j^k = \|\widehat{x}_j - \mu_k\|_2^2$
  end
  $y_j^* = \underset{k=1,2,\cdots,K}{\arg\min}\ d_j^k$; `// NCM Classification Strategy`
end

2.5 Device Inference

In the device feature representation, we propose an auto-encoder mapping method based on metric learning to automatically perform feature space layout with intra-class compactness and inter-class separation. Following, for device

inference, we properly make use of the nearest class mean (NCM) classification strategy [20]. As described in Algorithm 1, we first use an auto-encoder to map the traffic data of all devices into the feature space and then obtain the mean feature vector for each device. Finally, we assign class labels based on the smallest distance from the mean vector in the feature space. Figure 6 visualizes the main idea of our device inference method. The entire feature space can be regarded as a multidimensional Voronoi Diagram. The embeddings of traffic samples for each class form corresponding regions, called Voronoi cells, consisting of all points that are closer to the mean vector of the class than any other classes. The inference of the device type is to determine in which cell it is mapped.

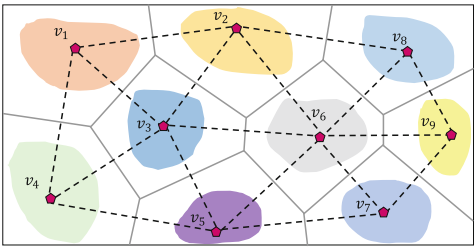


Fig. 6. Device inference with NCM. Combining metric learning and NCM strategy, the feature space is naturally formed similar to the Voronoi Diagram. Take the Voronoi diagram in 2-D space as an example, where $v_1, \cdots, v_9$ are the mean vectors of classes.

2.6 Evolutionary Learning

Algorithm 2: Exemplar Management

Input: Devices data set $X = \{x_0, \cdots, x_{N-1}\}$; devices class $Y = \{y_1, \cdots, y_K\}$; the number of samples in each class subset is $N^1, \cdots, N^K$; the encoder f; the size of exemplar set S.

Output: The exemplar set P.

for class $k = 1$ to K **do**
  $\mu_k = \frac{1}{N^k} \sum_{i=0}^{N^k-1} f(x_i^k)$; // The mean of the class k
  $P^k = \{\}$; // Initialization
  for $j = 0$ to $S - 1$ **do**
    $p_j \leftarrow \underset{x \in X^k \setminus P^k}{\arg\min} \left\| \mu_k - \frac{1}{j} \left[f(x) + \sum_{z=1}^{j-1} f(p_z) \right] \right\|$
    Add p_j to P^k
  end
  $P \leftarrow \left(P^1, P^2, \cdots, P^K\right)$; // Add to the set P
end

Representative Exemplar Management. We have designed a model architecture suitable for evolutionary learning. However, during continuous learning with new devices, it is also necessary to overcome the inevitable forgetting of old devices. To help the model acquire knowledge from old classes, we use an exemplar manager to manage the most representative samples from old data for knowledge replay to avoid forgetting. When a set of new devices is added to the current model, we select a subset of the most representative samples from these classes and store them. In our work, we consider a manager with minimal storage compared to the original dataset, and select representative samples for each class to manage when new tasks arrive. For the selection of typical samples, we introduce the method of herding, which is detailed in Algorithm 2. When the device samples are added to the exemplar set, the algorithm guarantees that the average feature vector of the exemplar set most closely approximates the average feature vector of the overall sample set.

Algorithm 3: Evolutionary Learning

Input: A series of tasks arriving by time $T_0, \cdots, T_t$ and the training data of each task $X_0, \cdots, X_t$; the test tasks $\mathcal{Z}_0, \cdots, \mathcal{Z}_t$; the model f; the size of exemplar set S.

Output: Evolutionary model f^*; device identification results Y.

Preprocess raw traffic data from network devices;

`// Initial Stage`

if the first task T_0 arrives **then**

  $\mathcal{D}_0 \leftarrow \{(x, y) : x \in X_0\}$;

  Run training with loss function:

$$f_0 \leftarrow \arg\min_f \; L_1 + \lambda_1 L_2, \tag{3}$$

  that L_1 serves as Reconstruction Loss and L_2 serves as Metric Loss;

  $P_0 \leftarrow ExemplarManage(\mathcal{D}_0, S)$;

  $Y_0 \leftarrow DeviceInference(\mathcal{Z}_0, f_0)$;

end

`// Evolutionary Stage`

while task $T_t \in \{T_1,\ T_2,\ T_3, \cdots\}$ arrive **do**

  $\mathcal{D}_t \leftarrow \{(x, y) : x \in X_t\} \cup \bigcup_{j=0}^{t-1} \{(x, y) : x \in P_j\}$;

  Knowledge replay with less consumption:

$$f_t \leftarrow \arg\min_f \; L_1 + \lambda_1 L_2 + \lambda_2 L_3, \tag{4}$$

  $P_t \leftarrow ExemplarManage(\mathcal{D}_t, S)$; `// get the exemplars set`

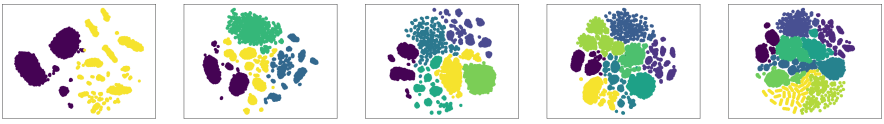
  $Y_t \leftarrow DeviceInference(\mathcal{Z}_t, f_t)$;

end

Spatial Knowledge Distillation. As described in Algorithm 3, evolutionary learning is divided into two stages: initialization and evolution. In general, the initial stage is the same as traditional learning, which learns to identify all IoT devices in T_0 by solving the constraint (3). The evolutionary stage is following the initial stage. As a sequence of tasks $T_0, T_1, \cdots, T_t$ arrive, the model needs to be re-learn based on the model obtained in the previous stage to continuously enhance the classification ability. During the training process for new tasks, we need to avoid using all the old training traffic data used in historical tasks, but not overly impair the ability to recognize old devices. So we introduce the method of *knowledge distillation* [7] into the stage of evolutionary training to alleviate the problem of catastrophic forgetting:

$$L_3 = \sum_{x_i \in \mathcal{D}_t} \delta_{x_i \in (P_0, P_2, \cdots, P_{t-1})} \| f_t(x_i) - f_{t-1}(x_i) \|_2^2, \tag{5}$$

where the term is named Spatial Knowledge Distillation, δ indicates true when $x_i \in (P_0, P_2, \cdots, P_{t-1})$. We regard the model of the previous stage as the teacher model and the model of the new stage as the student model. Then we transfer the knowledge from the model of the previous task stage T_t to the next task stage T_{t+1} with the constraint (5). In addition to learning the old knowledge, the student model needs to undertake more work. It not only needs to learn the knowledge of the teacher model in T_t but also to learn new knowledge in the new task T_{t+1}. The process of dynamic evolution is visualized in Fig. 7.



(a) 2. (b) 4. (c) 6. (d) 8. (e) 10.

Fig. 7. Visualization of dynamic evolution.

3 Experiments and Results

3.1 Experimental Setups and Datasets

Our experiments are based on three public datasets: IoT Traffic Traces of UNSW [24], Sentinel [17], and LSIF [3]. The detailed setup of the dataset and experiments will be described as follows.

Traffic Traces of UNSW. The dataset is open-source traffic data collected in 2018 by researchers at the UNSW in the process of studying IoT assets and tracking the behavior of IoT devices, recording daily traffic data for 3 month period. The dataset includes smart cameras, plugs, sensors, health monitors, etc., with a total of 30 network devices.

IoT Sentinel. The dataset is network device data generated by Miettinen [17] in the research on managing security and privacy risks posed by insecure IoT devices. The devices are mainly a representative set of consumer-oriented IoT devices, with 23 device data covering the most common types of devices such as smart lighting, home automation, and household appliances.

LSIF. Charyyev et al. [3] set up an experimental platform that automatically collects network traffic. The data set contains data on Internet devices produced by different manufacturers, which recorded network devices such as smart plugs, smart light bulbs, doorbells, and cameras. This dataset collects network traffic generated from experiments over approximately 20 d.

Experimental Setups. We choose the popular 18-layer ResNet as the underlying CNN network for all methods for comparative experiments. During the experiments, we use the Adam optimizer with which the learning rate starts at 0.001, the decay coefficient is set to 0.96, and the batch size is 256. In our method, the hyper-parameters λ_1 and λ_2 in the loss function are both set to 1 and the dimension of the encoding layer is set to 100. For the dataset, in order to perform multi-stage evolutionary learning, we merge all datasets together, remove the data of all duplicate devices, and then integrate the extracted TCP session data into a large dataset containing 66 different classes for experiments. We set the number of downsampling to 4000 and split the training and test sets with a ratio of 8:2. The number of exemplars per class is set to 50, which only accounts for a very small proportion of the original data, around 0.01. In addition, parameters such as task capacity c, number of exemplars per class n, and boundary distance m will be discussed in detail in the later sections. In the experiments, we aim to answer the following research questions:

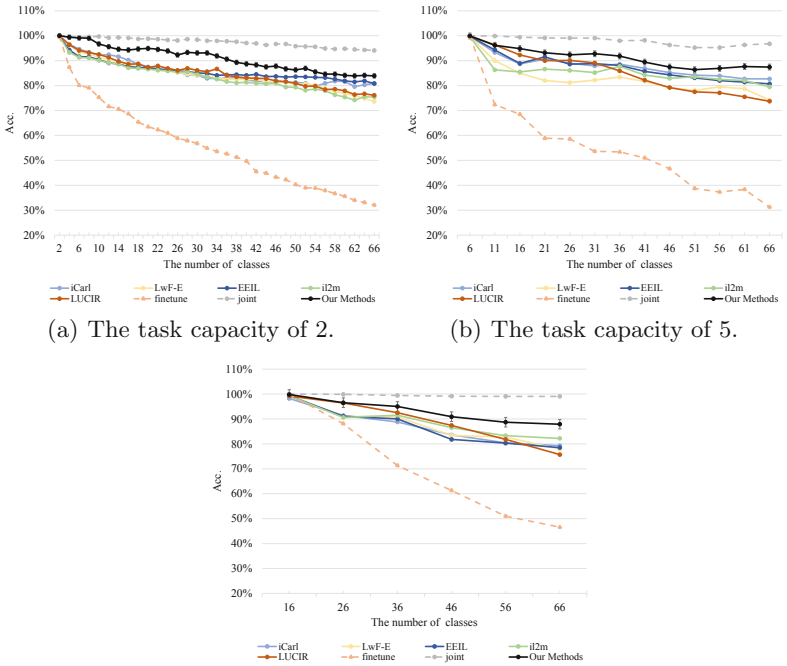
- **Q1** - How about the accuracy of our proposed method after multi-stage evolutionary learning?
- **Q2** - How well do various methods perform in evolutionary learning for the anti-forgetting of old devices?
- **Q3** - How do key parameters affect evolutionary learning in experiments and how do we make trade-offs in performance, computational resources, and memory?

3.2 Accuracy Evaluation (Q1)

In this evaluation, our main purpose is to test the accuracy of our method after multi-stage evolutionary learning. We conduct comparative experiments with various state-of-the-art methods [15], including LwF-M [11], iCarl [20], LUCIR [8], EEIL [2] and il2m [1]. Note that the LwF-M here is an improvement of the original LwF method, adding the method of representative exemplars to improve the accuracy. In addition, we compare our method with the naive **finetune** method and **joint** learning method. Finetune is a naive idea and refers to fine-tuning models directly as new tasks arrive without any knowledge-preserving effort, which is considered as one of the baselines for models in evolutionary

learning. The joint learning method represents storing all historical traffic data. Every time a new task is trained, a new model is reconstructed from scratch using all the data, which wastes huge computing resources and storage. Although the joint learning method is not available in practical scenarios, it can be regarded as the highest upper bound.

We set up a fair comparison experiment basis, all methods perform multiple comparisons at task capacities of 2, 5, and 10, respectively, where the task capacity refers to the number of new devices included in each task. The comparison results are shown in Fig. 8. We summarize the results as follows:



(a) The task capacity of 2.

(b) The task capacity of 5.

(c) The task capacity of 10.

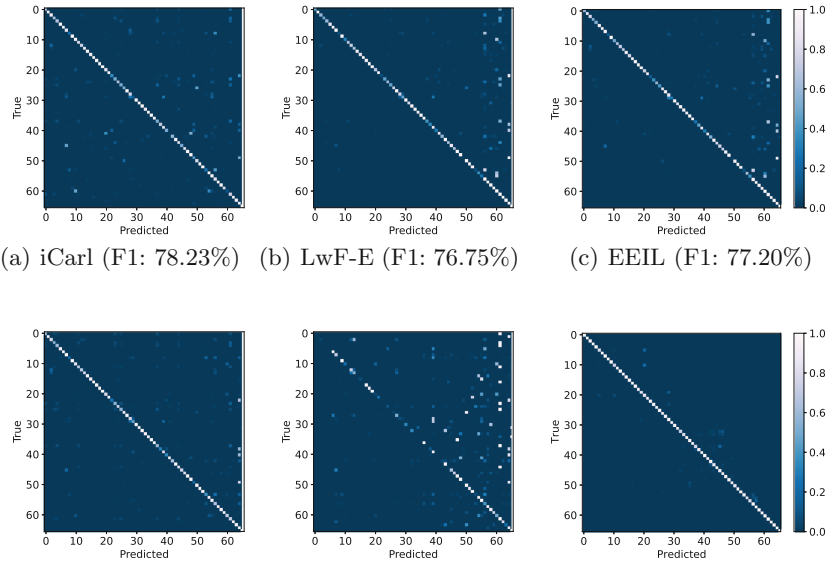
Fig. 8. Accuracy evaluation.

– Our method significantly outperforms state-of-the-art methods under diverse experimental settings and is closest to the joint method. Furthermore, the memory consumption of our model is only 0.045, 0.09, and 0.165 of the joint method, respectively. Specifically, for the experiments with task capacities of 2, 5, and 10, we end up with an accuracy of $83.94 \pm 0.50\%$, $87.35 \pm 0.50\%$, and $87.96 \pm 0.50\%$, while the finetune is just 32.53%, 31.24%, and 46.61%. In addition to our method, il2m performs better in fewer task stages with an accuracy of 75.43%, 79.54%, and 82.21%. EEIL performs better in more task stages on tasks with an accuracy of 80.82%, 80.73%, and 78.51%.

– Since the total number of devices is 66, the smaller the task capacity, the more stages for evolutionary learning. Comparing the three experimental results, it can be found that as the number of task stages increases evolutionary learning becomes more difficult.

3.3 Anti-forgetting Evaluation (Q2)

As shown in Fig. 9, in the experiment with a task capacity of 10, we use the confusion matrix and F1 score to compare and analyze the forgetting that occurs in the evolutionary learning process of various methods. The vertical axis represents the correct device type, and the horizontal axis represents the predicted device type. LUCIR produces the worst forgetting problem. Its F1 score is only 75.65%. Additionally, other methods such as EEIL and LwF-E show an excessive preference for the latest learned devices. In contrast, our method performs well on both old and new devices, producing a better confusion matrice where the activations are mostly distributed at the diagonal. The F1 score of our method is 88.31%. In conclusion, our method effectively alleviates the forgetting of old devices during evolutionary learning.



(a) iCarl (F1: 78.23%) (b) LwF-E (F1: 76.75%) (c) EEIL (F1: 77.20%)

(d) il2m (F1: 81.68%) (e) LUCIR (F1: 75.65%) (f) Our Method (F1: 88.31%)

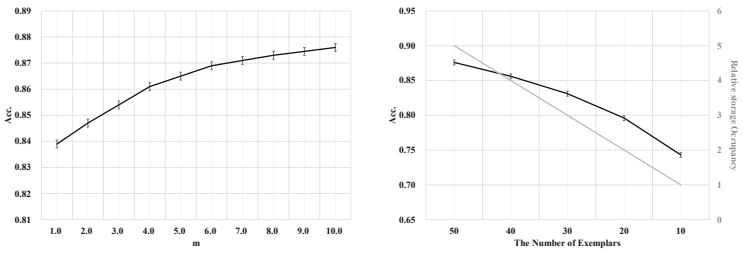
Fig. 9. Confusion matrix and F1.

3.4 Sensitivity Analysis (Q3)

As mentioned earlier, the farther the boundaries of different classes in the feature space are, the less likely they are to interfere and confuse each other. Therefore,

we discuss the contribution of the parameter m to the accuracy of evolutionary learning. In this experiment, the task capacity is set to 10, and other hyper-parameters are set to the default configuration. As shown in Fig. 10(a), it can be found that when the m becomes larger and larger, the final accuracy rate also becomes higher. When m is 1, the accuracy rate can reach about 84%, and when m is 10, the accuracy rate can reach about 88%. This result proves our findings, and also shows that the greater the spatial distance, the better the effect of evolutionary learning in the process of evolutionary learning.

As shown in Fig. 10(b), the experiments investigate the effect of the number of exemplars in the exemplar manager for evolutionary learning. It is found that as the number of exemplars decreases, the accuracy also decreases, which is the same result as we expected. Because the larger the number of exemplars, the more knowledge of old IoT devices can be represented, and therefore more knowledge can be retained with the distillation loss. However, we have to take into account that as the number of samples increases, so does the memory footprint, so ultimately we have to strike the right balance between performance and memory.



(a) The analysis of m. (b) The analysis of exemplars.

Fig. 10. Sensitivity analysis.

4 Related Work

4.1 Identification of Network Devices

In order to better manage the network space, it is necessary to implement appropriate security policies for different devices, but the premise is to accurately detect and identify network devices [22,31].

The specific feature-based approaches focus on discovering various useful features for device identification. PingPong [26] introduces an efficient way to automatically extract packet-level signatures from traffic. Then, Wan et al. [27] propose an improved method for PingPong. Sivanathan proposes statistical properties of more than 20 IoT devices in his research work [25]. Then, in his recent research works [23,24], it is found that using the Naive Bayes classifier on the activity period, port number, domain name, and cipher suite can achieve better performance in classifying 28 commercial IoT devices. Feng [6] proposes a

method to automatically generate rule features and annotate IoT devices. In addition, deep learning technology is also gradually applied in device identification. Meidan et al. [16] design a multi-stage meta classifier for IoT devices. [14,29] shows that the best IoT device classification results can be obtained by combining CNN and RNN. Fan [5] proposes a semi-supervised model based on CNN and multi-task learning.

Although the above methods have made some progress, when the processed device is not in the training set, the above methods will encounter the critical open-world problem. Yu [30] proposes an identification scheme with good scalability, but the limitation is that it depends on the traffic of the network connection stage. Recently, Hu [9] proposes out-of-distribution (OOD) with EVT to detect unknown devices to improve generalization.

4.2 Class Incremental Learning

Human learning mechanisms are different from machine learning models. Humans can continue to learn new knowledge without forgetting, but machine learning models usually only perform well on the latest learning tasks. Once they continue to learn new knowledge, they will forget the previous [19]. Therefore, it has drawn attention to class incremental learning [15].

The *regularization-based* method aims to constrain the optimization direction of the model in the new task and minimize the interference caused by the old task. Kirkpatrick [10] introduces an additional regularization term that consolidates important parameters from the previous task. Then, LwF [11] is the first to use the idea of knowledge distillation to preserve knowledge from past tasks. The idea based on *bias correction* aims to address the problem of task bias. Castro [2] proposes an efficient bias correction-based end-to-end incremental learning method. EEIL [8] develops a unified processing framework with three components to reduce the impact of imbalance. Belouadah et al. [1] propose an incremental learning approach that utilizes finetuning and a dual memory mechanism il2m to reduce the negative effects of catastrophic forgetting. The *rehearsal-based method* is considered to be the most promising method, aiming at replaying some key knowledge of old tasks when learning new tasks, just like reviewing old knowledge when learning new knowledge. iCaRL [20] combines playback and distillation loss to transfer knowledge, which inspires the method proposed in this paper.

5 Conclusion and Future Work

This work is devoted to solving the open-world problem of IoT device recognition from the perspective of model evolution. We propose a novel evolutionary model which can continuously learn with new device traffic. In specific, we represent the traffic rationally and introduce a metric learning-based auto-encoder to mine latent features of device traffic. Then we utilize the NCM classification strategy for device inference. We find that the less-forgetting constraint based

on knowledge replay and the independent compact spatial distribution can cope with the catastrophic forgetting of old devices in evolutionary learning. During evolutionary learning, our method does not need to store all the device data and only requires very few representative samples to achieve high performance. The results of comparative experiments show that our method outperforms the state-of-the-art incremental learning methods. In future work, we will study how to use the few-shot learning method to reduce the workload of manual annotation, thereby improving the efficiency of model evolutionary learning.

Acknowledgement. This work was supported by the National Key Research and Development Program of China (Grant No.2018YFB0803402), the Young Scientists Fund of the National Natural Science Foundation of China (Grant No.61702504) and the Industrial Internet Innovation and Development Project (Grant No.KFZ0120200004).

References

1. Belouadah, E., Popescu, A.: Il2m: Class incremental learning with dual memory. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 583–592 (2019)
2. Castro, F.M., Marín-Jiménez, M.J., Guil, N., Schmid, C., Alahari, K.: End-to-end incremental learning. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11216, pp. 241–257. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01258-8_15
3. Charyyev, B., Gunes, M.H.: Iot traffic flow identification using locality sensitive hashes. In: ICC 2020–2020 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2020)
4. Delange, M., et al.: A continual learning survey: Defying forgetting in classification tasks. IEEE Trans. Pattern Anal. Mach. Intell. (2021)
5. Fan, L., et al.: An iot device identification method based on semi-supervised learning. In: 2020 16th International Conference on Network and Service Management (CNSM), pp. 1–7. IEEE (2020)
6. Feng, X., Li, Q., Wang, H., Sun, L.: Acquisitional rule-based engine for discovering {Internet-of-Things} devices. In: 27th USENIX Security Symposium (USENIX Security 18), pp. 327–341 (2018)
7. Hinton, G., Vinyals, O., Dean, J., et al.: Distilling the knowledge in a neural network, vol. 2(7). arXiv preprint arXiv:1503.02531 (2015)
8. Hou, S., Pan, X., Loy, C.C., Wang, Z., Lin, D.: Learning a unified classifier incrementally via rebalancing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 831–839 (2019)
9. Hu, X., Li, H., Shi, Z., Yu, N., Zhu, H., Sun, L.: A robust IoT device identification method with unknown traffic detection. In: Liu, Z., Wu, F., Das, S.K. (eds.) WASA 2021. LNCS, vol. 12937, pp. 190–202. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85928-2_15
10. Kirkpatrick, J., et al.: Overcoming catastrophic forgetting in neural networks. Proc. Natl. Acad. Sci. **114**(13), 3521–3526 (2017)
11. Li, Z., Hoiem, D.: Learning without forgetting. IEEE Trans. Pattern Anal. Mach. Intell. **40**(12), 2935–2947 (2017)

12. Liu, Y., Wang, J., Li, J., Niu, S., Song, H.: Machine learning for the detection and identification of internet of things (iot) devices: A survey. arXiv preprint arXiv:2101.10181 (2021)
13. Liu, Z., Cai, L., Zhao, L., Yu, A., Meng, D.: Towards open world traffic classification. In: Gao, D., Li, Q., Guan, X., Liao, X. (eds.) ICICS 2021. LNCS, vol. 12918, pp. 331–347. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86890-1_19
14. Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., Lloret, J.: Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access **5**, 18042–18050 (2017)
15. Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A.D., van de Weijer, J.: Class-incremental learning: survey and performance evaluation on image classification. arXiv preprint arXiv:2010.15277 (2020)
16. Meidan, Y., et al.: Profiliot: a machine learning approach for iot device identification based on network traffic analysis. In: Proceedings of the Symposium On Applied Computing, pp. 506–509 (2017)
17. Miettinen, M., Marchal, S., Hafeez, I., Asokan, N., Sadeghi, A.R., Tarkoma, S.: Iot sentinel: Automated device-type identification for security enforcement in iot. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 2177–2184. IEEE (2017)
18. Ortiz, J., Crawford, C., Le, F.: Devicemien: network device behavior modeling for identifying unknown iot devices. In: Proceedings of the International Conference on Internet of Things Design and Implementation, pp. 106–117 (2019)
19. Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S.: Continual lifelong learning with neural networks: A review. Neural Netw. **113**, 54–71 (2019)
20. Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H.: icarl: Incremental classifier and representation learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2001–2010 (2017)
21. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE Conference On Computer Vision And Pattern Recognition, pp. 815–823 (2015)
22. Shahid, M.R., Blanc, G., Zhang, Z., Debar, H.: Iot devices recognition through network traffic analysis. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 5187–5192. IEEE (2018)
23. Sivanathan, A.: Iot behavioral monitoring via network traffic analysis. arXiv preprint arXiv:2001.10632 (2020)
24. Sivanathan, A., et al.: Classifying iot devices in smart environments using network traffic characteristics. IEEE Trans. Mob. Comput. **18**(8), 1745–1759 (2018)
25. Sivanathan, A., et al.: Characterizing and classifying iot traffic in smart cities and campuses. In: 2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp. 559–564. IEEE (2017)
26. Trimananda, R., Varmarken, J., Markopoulou, A., Demsky, B.: Pingpong: Packet-level signatures for smart home device events. arXiv preprint arXiv:1907.11797 (2019)
27. Wan, Y., Xu, K., Wang, F., Xue, G.: Iotathena: Unveiling iot device activities from network traffic. IEEE Trans. Wireless Commun. **21**(1), 651–664 (2021)
28. Yang, L., et al.: {CADE}: Detecting and explaining concept drift samples for security applications. In: 30th {USENIX} Security Symposium ({USENIX} Security 2021) (2021)

29. Yin, F., Yang, L., Wang, Y., Dai, J.: Iot etei: End-to-end iot device identification method. In: 2021 IEEE Conference on Dependable and Secure Computing (DSC), pp. 1–8. IEEE (2021)
30. Yu, L., Liu, T., Zhou, Z., Zhu, Y., Liu, Q., Tan, J.: Wdmti: wireless device manufacturer and type identification using hierarchical dirichlet process. In: 2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), pp. 19–27. IEEE (2018)
31. Yu, L., Luo, B., Ma, J., Zhou, Z., Liu, Q.: You are what you broadcast: Identification of mobile and {IoT} devices from (public){WiFi}. In: 29th USENIX security symposium (USENIX security 2020). pp. 55–72 (2020)