DOI: 10.32604/cmes.2022.018492

ARTICLE

Transferable Features from 1D-Convolutional Network for Industrial Malware Classification

Liwei Wang[1,2,3], **Jiankun Sun**[1,2,3], **Xiong Luo**[1,2,3,*] **and Xi Yang**[4]

[1]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, 100083, China

[2]Beijing Key Laboratory of Knowledge Engineering for Materials Science, Beijing, 100083, China

[3]Shunde Graduate School, University of Science and Technology Beijing, Foshan, 528399, China

[4]Beijing Intelligent Logistics System Collaborative Innovation Center, Beijing, 101149, China

[*]Corresponding Author: Xiong Luo. Email: xluo@ustb.edu.cn

Received: 28 July 2021 Accepted: 28 September 2021

ABSTRACT

With the development of information technology, malware threats to the industrial system have become an emergent issue, since various industrial infrastructures have been deeply integrated into our modern works and lives. To identify and classify new malware variants, different types of deep learning models have been widely explored recently. Generally, sufficient data is usually required to achieve a well-trained deep learning classifier with satisfactory generalization ability. However, in current practical applications, an ample supply of data is absent in most specific industrial malware detection scenarios. Transfer learning as an effective approach can be used to alleviate the influence of the small sample size problem. In addition, it can also reuse the knowledge from pre-trained models, which is beneficial to the real-time requirement in industrial malware detection. In this paper, we investigate the transferable features learned by a 1D-convolutional network and evaluate our proposed methods on 6 transfer learning tasks. The experiment results show that 1D-convolutional architecture is effective to learn transferable features for malware classification, and indicate that transferring the first 2 layers of our proposed 1D-convolutional network is the most efficient way to reuse the learned features.

KEYWORDS

Transfer learning; malware classification; sequence data modeling; convolutional network

1 Introduction

In the era of digitalization and intelligence, an increasing number of industrial devices are connected to the Internet and the generated data can be collected and analyzed more efficiently for personalized service and industrial production. Meanwhile, those devices suffer from various cyber-attacks [1]. Malicious attackers can spread malware, such as viruses, Trojan horses, or ransomware, to compromise industrial systems. To avoid the damages from the malware, an extensive exploration about deep neural networks, recurrent neural network (RNN), and convolutional neural network (CNN) for the industrial Internet of Things (IoT) and cyber-physical system security suggests that deep learning is a powerful tool to detect and classify malware [2–6].

Towards the deep learning solutions for the malware detection and classification tasks, the general idea is to train a deep learning model from scratch following the classical supervised training paradigm. However, the small sample size problem in practical applications degrades the performance severely in deep learning. Due to different kinds of heterogeneous devices in industrial systems and the specificity of malware, the small sample size problem often arises when applying deep learning techniques. Following that, transfer learning is a sophisticated paradigm to alleviate such an issue. In recent researches, transfer learning is widely applied to image-based malware analysis [7–10]. The general procedures in such a scenario include: 1) converting every malware instance into a gray-scale or RGB image; 2) pre-training a CNN on a large source dataset, e.g., a ResNet-50 based on ImageNet; 3) transferring the convolutional layers in the pre-trained CNN to a new randomly initialized network, freezing the parameters in the convolutional layers, and fine-tuning the parameters in the fully connected layers on target dataset. Here, the key assumption is that the learned color blobs and Gabor filters from the CNN are also applicable to identify the texture of malware images. It is noted that the differences between natural images and malware images are obvious, which means a semantic gap exists.

To overcome issue of the semantic gap, we select opcode and application programming interface (API) sequence for malware behavior representation. Here, the functionality of various programs in different platforms is a process of executing commands sequentially, which means the opcode or API sequence is a more general and precise behavior representation for malware than image-based representation. Among the data-driven malicious sequence modeling architectures, RNN is a common network structure for modeling sequence data due to its weights sharing mechanism on the time dimension. In the era of big data, processing massive amounts of data has become the norm, and parallel computing is now an essential method to handle it. However, RNN is unable to gain benefit from the graphics processing unit (GPU) with parallel computing ability because of its sequential computation process. Hence, in order to make full use of the parallel computing ability of GPU, a new 1D-convolutional-based sequence data modeling scheme was developed and its excellent performance was validated in language modeling [11,12] and time-series classification subsequently [13]. Based on the discussions above, a transfer learning-enhanced 1D-convolutional network architecture is explored for the specific sequence modeling task in this paper, i.e., malware sequence classification. The contributions of this paper are summarized as follows:

- We propose a 1D-convolutional architecture to perform transfer learning and evaluate 3 convolutional strategies on 6 transfer learning tasks. The experiment results show that the 1D-convolutional network can effectively learn transferable features for malware classification.
- We conduct several experiments to identify the practical way to transfer the parameters of convolutional layer. Moreover, the experiment results show that transfer learning is an effective approach to reduce the training time.

The remainder of this paper is organized as follows. Section 2 overviews some recent works on sequence data modeling. The proposed network architecture is described in Section 3. Our experimental results are presented in Section 4. The conclusion of this paper is stated in Section 5.

2 Related Work

Deep learning and transfer learning are widely explored in sequence data modeling. In this section, for sequence data modeling task, the developed deep learning models, especially the 1D-convolutional networks on malicious sequence data, are overviewed firstly. Then, the transfer learning models on sequence data are analyzed.

2.1 Deep Learning for Sequence Data Modeling

For malware classification, opcode and API sequences are usually used for malware behavior representation. HaddadPajouh et al. [14] adopted long short-term memory (LSTM) for ARM-based IoT malware detection based on opcode sequence. Kang et al. [15] and Jha et al. [16] used LSTM to classify different malware families based on API sequence and opcode. To benefit from the parallel computing ability of GPU, the 1D-convolutional network is proposed for sequence data modeling. Shelhamer et al. [17] proposed a fully convolutional network (FCN) architecture for semantic segmentation. Wang et al. [13] modified the original FCN with one-dimensional kernels and replaced the last layer of FCN with fully connected layers and a softmax layer for time series classification. After that, some researchers explored the potential of 1D-convolutional networks for malware detection. Hasegawa et al. [18] used a 1D-convolutional network to detect Android malware. Then, to prevent information leakage from the future into the past, a specific temporal convolutional network (TCN) [19] was utilized to categorize malware into different families in the field of IoT malware classification [20].

2.2 Transfer Learning for Sequence Data Modeling

The major assumption from the classical supervised learning scheme is that training and future data follow the same distribution. However, this assumption fails in some industrial malware detection cases, since malicious training instances from heterogeneous devices may not satisfy the same feature distribution. In addition, the training dataset may easily get out of date in the era of big data [21]. Hence, knowledge transfer will be a useful technique to improve the performance of deep learning models. Fawaz et al. [22] extensively explored the transferable ability of 1D-FCN for time series classification, and their experiment results showed that knowledge transfer is beneficial to performance enhancement. Subsequently, Gao et al. [23] explored a semi-supervised transfer learning technique to alleviate the influence of the small sample size problem.

Meanwhile, there are also some problems in the use of transfer learning. They can be summarized as follows: 1) what to transfer? 2) where to transfer? and 3) how to transfer? Currently, the mainstream transfer learning paradigm is based on pre-training and fine-tuning strategies, as illustrated in Fig. 1, including 1) pre-training a deep learning model on source domain; 2) transferring the learned weights of the first few layers (i.e., what to transfer); 3) freezing the weights of transferred layers and retraining the parameters of the rest on target domain (i.e., how to transfer). In this paper, we follow the general strategy mentioned above, and determine where to transfer the layers pre-trained on the source domain in accordance with the experiment strategy developed in [24].

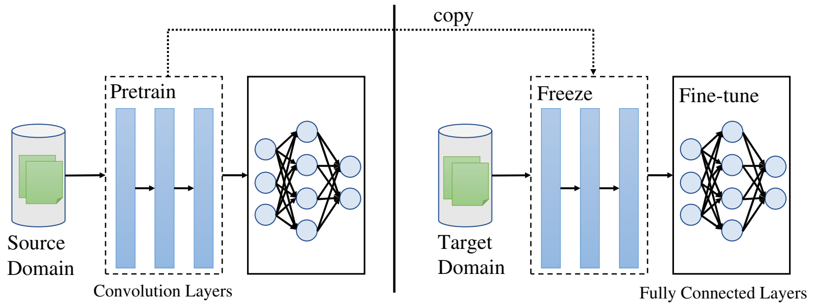

Figure 1: The general transfer learning paradigm using pre-training and fine-tuning strategies

3 Methodology

In this section, we give a formal definition of the transfer learning task and descriptions of data preprocessing steps firstly. Then, the proposed 1D-convolutional network architecture is introduced.

Here, the schematic diagram of our proposed method is illustrated in Fig. 2.

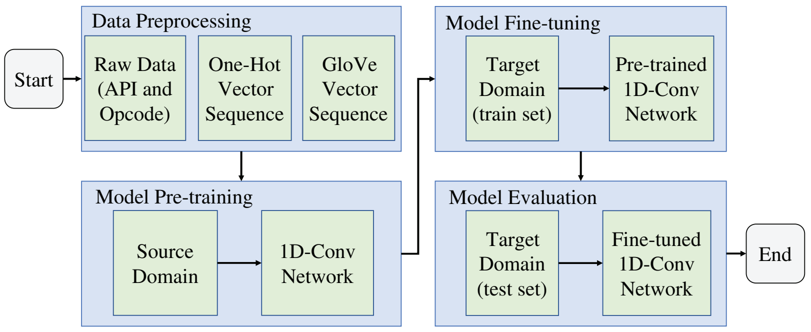

Figure 2: The schematic diagram of our proposed method

3.1 Problem Statement

Domain and task are basic conceptions in transfer learning. A domain D is defined as $D = \{\mathcal{X}, P(X)\}$, where $\mathcal{X}$ is the feature space and $P(X)$ is the marginal distribution of data points. Here, X denotes the sample set. A task T is defined as $T = \{\mathcal{Y}, g\}$, where $\mathcal{Y} = \{1, 2, \cdots, C\}$ is the label space, C denotes the number of categories, and g is the discriminant function that needs to be learned. Here, g is usually a conditional distribution $P(y|\mathbf{x})$, where $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$. Hence, a supervised learning task can be denoted as a tuple $<D, T>$.

Based on the definition above, the transfer learning can be defined as: given a source supervised learning task $<D_s, T_s>$ and a target supervised learning task $<D_t, T_t>$, the transfer learning utilizes the knowledge learned on the source task to improve the performance of discriminant function on the target task. Since opcode and API in different datasets used in this paper are not consistent, which means $D_s \neq D_t$, we adopt the word embedding technique to align the feature space and ensure $\mathcal{X}_s = \mathcal{X}_t$. In addition, the key assumption of this paper is that a 1D-convolutional layer can learn general features of sequence data during the pre-training process, and fine-tuning operation is able to make a 1D-convolutional network learn task-specific features on the basis of the transferred layers.

3.2 Feature Space Alignment

To ensure the feature space of different datasets has the same mathematic representation, we utilize the GloVe word embedding algorithm [25] to align the feature space. Here, we treat each API or opcode as a word. The procedure of data preprocessing is illustrated in Fig. 3. The extraction of opcode and API can be implemented through the use of reverse engineering tools or the monitor for the dynamic execution process of malware in a sandbox.

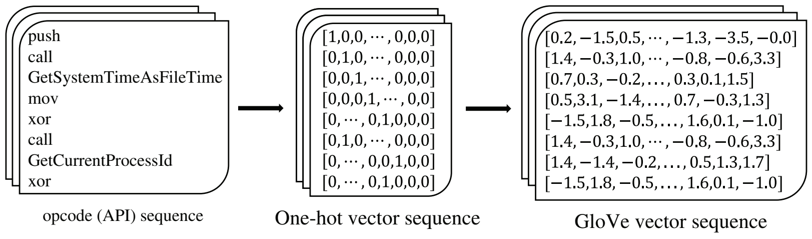

Figure 3: The procedure of data preprocessing

GloVe is a word embedding algorithm that takes advantage of both global statistical information and local context. Before training the word vectors, we need to use a sliding window to compute the word-word co-occurrence matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, where n denotes the number of API (opcode) and $\mathbf{M}_{ij}$ represents occurrence frequency of word i around the word j. Then, the final optimization objective is shown in (1), where $f(\cdot)$ is a weight assignment function in (2), and it is defined in accordance with the similar setting in [25].

$$\text{Loss} = \sum_{1\le i,j\le n} f(\mathbf{M}_{ij})(\mathbf{v}_i\mathbf{w}_j^\top + bv_i + bw_j - \log \mathbf{M}_{ij})^2, \tag{1}$$

$$f(x) = \begin{cases} (x/x_{\max})^{3/4}, & \text{if } x < x_{\max}, \\ 1, & \text{otherwise.} \end{cases} \tag{2}$$

Here, $\mathbf{v}_i \in \mathbb{R}^d$ and $\mathbf{w}_j \in \mathbb{R}^d$ represent the i-th row of $\mathbf{V} \in \mathbb{R}^{n\times d}$ and the j-th row of $\mathbf{W} \in \mathbb{R}^{n\times d}$, respectively, where d denotes the embedding size and $\{\mathbf{V}, \mathbf{W}\}$ are both word embedding matrices, i.e., each row of $\mathbf{V}$ and $\mathbf{W}$ are all word vectors that need to be trained. In addition, $bv_i(1 \le i \le n)$ and $bw_j(1 \le j \le n)$ are both bias terms that need to be updated during training. In practice, the optimization of the loss function (1) is based on the samples with batches, and therefore the word vectors in $\mathbf{V}$ and $\mathbf{W}$ are updated iteratively.

After the convergence of the training process, $(\mathbf{V} + \mathbf{W})$ will be used as the final GloVe word embedding matrix for malware classification. With the assistance of GloVe, all the datasets used in this paper are embedded into the same feature space.

3.3 Network Architecture

In this paper, we redesign the 1D-FCN architecture proposed in [13]. To stabilize the training process, we replace the batch normalization with layer normalization and add shortcut connections to 1D-FCN, which is illustrated in Fig. 4. In this figure, 'ReLU' means the rectified liner unit (ReLU) activation function widely used in neural networks, and it will set negative output to zero and keep non-negative output unchanged. Meanwhile, for multiclass classification task, 'Softmax' is an activation widely used as the last layer to output possibility distribution.

The functionality of basic 1D-convolutional operation is shown in Fig. 5a, and the illustration of 1D-convolution with dilation is shown in Fig. 5b. In Fig. 5, each blue column represents a word vector obtained from GloVe word embedding algorithm, and green columns denote the 1D-kernel. A 1D-kernel acts like a sliding window and travels from left to right to perform convolutional operation. Compared to conventional 1D-kernel, 1D-kernel with dilation can compute larger receptive field, as shown in Fig. 5b. To prevent information leakage from the future into the past, padding zeros with $L-1$ length will be applied before convolution. Here, L denotes the length of 1D-kernel, and such operations are called causal convolution.

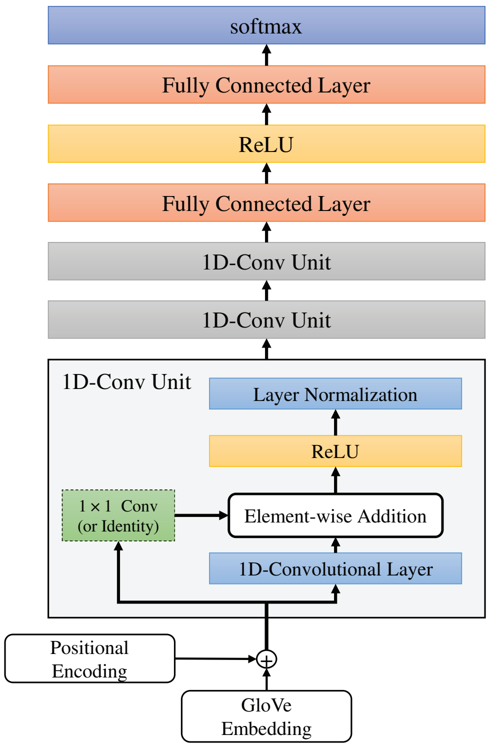

Figure 4: The architecture of our proposed 1D-convolutional network

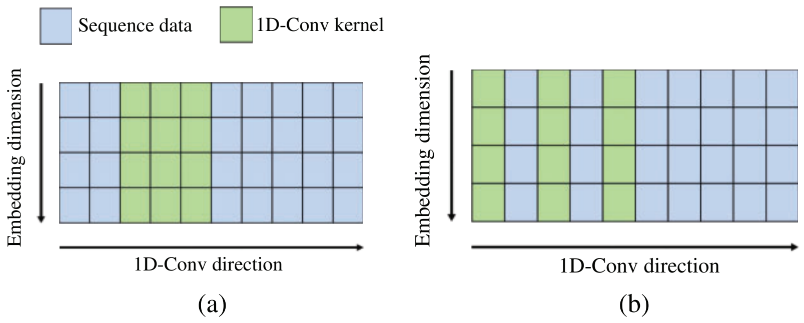

Figure 5: Illustration of 1D-convolution. (a) 1D-Conv without dilation (b) 1D-Conv with dilation

With the architecture we proposed, we evaluate 3 different convolutional strategies: 1) FCN [13] (conventional 1D-convolution as shown in Fig. 5a; 2) TCN [19] (1D-convolution with dilation as shown in Fig. 5b, where the dilation is not a constant but expansion with the exponential rate, i.e., the dilation factors of the convolutional layers are $2^0, 2^1, 2^2, \ldots$, sequentially; 3) gated convolutional network (GCN) (gated 1D-convolution which was first proposed in [11]).

Since different convolutional strategies share the same network architecture in our experiment settings, we select the gated convolutional layer (GCL) as our baseline convolutional strategy to tune the hyper-parameters. The GCL can be expressed by (3):

$$\mathrm{GCL}(X) = (X * K) \odot \mathrm{sigmoid}(X * M), \tag{3}$$

where X is the input sequence data, K and M are two different 1D-convolutional kernels with the same shape, '$*$' is the convolution operator, and '$\odot$' denotes the element-wise multiplication.

Shortcut Connection During the training process of a deep learning model, to solve the gradient vanishing problem, the shortcut connection was proposed in [26]. It is effective and concise shown in (4):

$$\mathbf{z} = \mathrm{Activate}(F(\mathbf{x}) + \mathbf{x}), \tag{4}$$

where $\mathbf{x}$ denotes the input data, $F(\cdot)$ represents a layer in a deep learning model, $\mathrm{Activate}(\cdot)$ denotes the activation function, and $\mathbf{z}$ denotes the output of the shortcut connection. According to the shortcut connection, the low-level feature can pass into inner layers, and the fusion of low-level and high-level features is implemented implicitly. In our experiment settings, the number of input channels in the first convolutional layer may be inconsistent with the number of output channels. Hence, a 1×1 convolutional kernel is applied to adjust the shape of input sequence data to make sure element-wise addition operation is feasible.

Layer Normalization Compared with batch normalization, the layer normalization (LN) is helpful to stabilize the training process, and it is more suitable for sequence data and independent of batch size [27], since its computation is limited to one single instance. The computation of LN is defined in (5):

$$\mathrm{LN}(\mathbf{x}) = \frac{\mathbf{x} - \mathbb{E}[\mathbf{x}]}{\sqrt{\mathrm{Var}[\mathbf{x}] + \varepsilon}} \odot \gamma + \beta, \tag{5}$$

where $\mathbf{x}$ is the input data that needs to be normalized, $\mathbb{E}[\cdot]$ represents the operation of calculating average value of $\mathbf{x}$, $\mathrm{Var}[\cdot]$ denotes the operation of computing variance of $\mathbf{x}$, ε is a small constant that is used to avoid zero division error, and it is usually set to 10^{-5}. Meanwhile, γ and β are the affine coefficients to be learned during training, and they are usually initialized to $\mathbf{1}$ and $\mathbf{0}$, respectively.

Positional Encoding To enrich the input sequence representation, positional encoding, which can provide the current positional information for the 1D convolution, is introduced in the proposed model. The utilized positional encoding can be formalized by (6) [28]:

$$\mathrm{Positional\_Encoding}(q,k) = \begin{cases} \sin(q/10000^{k/d}), & \text{if } k \text{ is even,} \\ \cos(q/10000^{(k-1)/d}), & \text{if } k \text{ is odd,} \end{cases} \tag{6}$$

where q denotes the q-position in the sequence, d is the positional embedding size, k denotes the k-th entry of the positional vector. In addition, its visualization is illustrated in Fig. 6. Each row in Fig. 6 represents a position vector, and as time step increases, the changes in the high-dimensional component of the position vector become more significant. In this paper, the addition operation between the GloVe vector and positional vector is applied before the 1D-convolutional network.

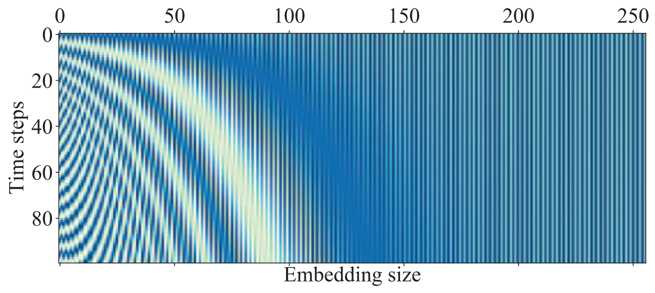

Figure 6: Visualization of positional encoding

4 Experimental Results and Discussion

In our experiments, we choose Python 3.8, PyTorch 1.7 and CUDA 11.0 to build the convolutional network architecture. The proposed model is tested in a computer that runs Ubuntu and consists of NVIDIA 1080Ti, Intel(R) Xeon(R) CPU E5-2673 v3 and 64 GB memory.

4.1 Dataset

There are two Windows platform datasets, including a bigger one, i.e., the Microsoft Malware Classification Challenge (BIG 2015) dataset from Kaggle competition [29], and a smaller one, i.e., the Windows portable executable (PE) dataset from Allan et al. [30]. To make full use of the diversity of the samples and the different malware families in the original datasets, we construct different transfer learning tasks to evaluate our proposed models more comprehensively. Hence, these two datasets are reorganized to conduct the experiments. After the data reorganization, the former bigger dataset is comprised of sequential opcode and API function names, and the latter one is comprised of API sequences. To extensively explore the performance of our proposed architecture, the former Kaggle dataset is split into 3 sub-datasets $\{A, B, C\}$, and the latter PE dataset is split into 2 sub-datasets $\{D, E\}$. The statistical information about the sub-datasets $\{A, B, C, D, E\}$ is presented in Table 1. In the second column of Table 1, each malware family in each dataset is assigned an index. The number of instances in different malware families is listed in Table 1, and their corresponding categories are also listed in the bracket. Additionally, the number of opcodes (APIs) in each dataset is listed in the third column. In addition, each sub-dataset is split into training set and test set according to the proportion 80% and 20%, respectively.

Table 1: Statistical information about the sub-datasets

Dataset	Categories and the corresponding indices			Opcode (API)
	0	1	2	
A	1513 (Ramnit)	2470 (Lollipop)	1168 (Obfuscator.ACY)	1070

(Continued)

Table 1 (Continued)

Dataset	Categories and the corresponding indices			Opcode (API)
	0	1	2	
B	2936 (Kelihos_ver3)	1168 (Obfuscator.ACY)	1012 (Gatak)	680
C	446 (Vundo)	294 (Tracur)	387 (Kelihos_ver1)	439
D	100 (benign)	100 (Backdoor)	80 (Virus)	179
E	80 (Virus)	100 (Worm)	172 (Trojan)	225

4.2 Parameters Selection

Here, GloVe word embedding algorithm is applied before the training process of the proposed architecture, and the hyper-parameters for the GloVe algorithm are listed in Table 2. Towards the proposed architecture, the key hyper-parameters, including output channels, kernel size in the convolutional layers, and the hidden dimension in the fully connected layer are determined by cross-validation. In addition, other hyper-parameters listed in Table 3 are selected by our engineering experience motivated by [13–15].

Table 2: Hyper-parameters for GloVe algorithm

Parameter	Value
Epochs	20
Batch size	16
Learning rate	10^{-3}
$x_{\max}$	100
Sliding window size	11
Embedding size	256

Table 3: Hyper-parameters for transfer learning

Parameter	Value
Input channels	256
Output channels	64
Kernel size	7
Learning rate	10^{-3}
Batch size	16
Pre-training epochs	20
Fine-tuning epochs	40
Cross-validation folds	5
Hidden dimension of fully connected layer	64

The cross-validation results, i.e., accuracy (mean $\pm$ std) on training set of A, are shown in Fig. 7. Firstly, the key hyper-parameters in the 1D-convolutional layer, i.e., kernel size and the number of output channels, are tuned via grid search strategy. In Fig. 7a, the horizontal axis denotes the kernel size and the vertical axis denotes the output channels. By observing the cross-validation performance, we select the best combination of output channel and kernel size, then 7 and 64 are chosen as kernel size and output channels, respectively. Moreover, in the fully connected layer, the hyper-parameter, i.e., hidden dimension, is tuned with fixed kernel size and output channels from the previous results. In addition, according to the cross-validation results, as shown in Fig. 7b, we select the hidden dimension value that achieves the highest accuracy, then 64 is selected as the hidden dimension. It should be noted that different convolutional networks used here have the same architecture. Therefore, the same hyper-parameter settings can be applied for all different networks on the same dataset. Here, we select GCN as our baseline model to tune hyper-parameters. When all the hyper-parameters are determined, the parameters of 1D-convolution network are initialized to random values, and then they are updated by Adam optimizer.

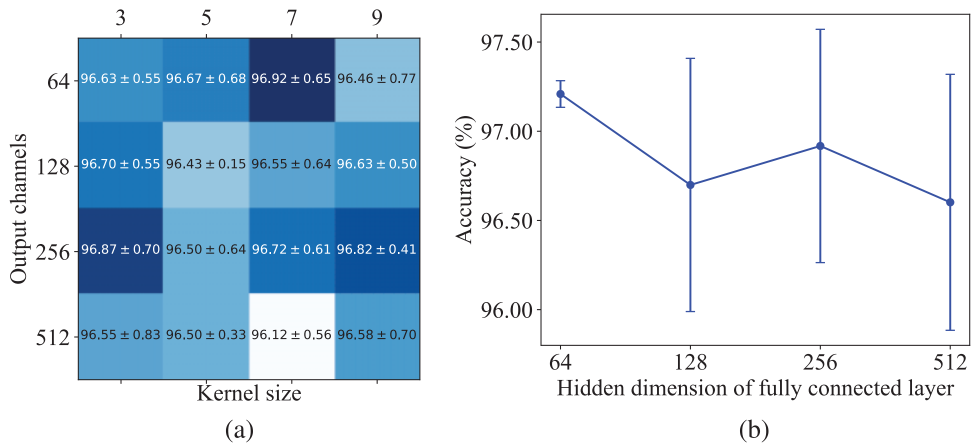

Figure 7: Cross-validation results (mean ± std%) for hyper-parameters tuning

4.3 Results and Discussion

To evaluate the performance of our proposed 1D-convolutional architecture comprehensively, GCN, TCN and FCN are selected as comparative algorithms due to their validated excellent performance in many other tasks [11,19,22]. To identify where to transfer the learned weights of the 1D-convolution network, we evaluate GCN on 4 transfer learning tasks: $A \rightarrow A$, $A \rightarrow B$, $B \rightarrow B$, and $B \rightarrow A$. In Figs. 8a and 8d, we show the accuracy (mean ± std) changes on the test set with the increase of the transferred layers for the GCN structure. The baseline A and B represent the accuracy on test set of A and B from the GCN without transferred layers. Meanwhile, AnB means the first n layers of a pre-trained GCN on dataset A are transferred to the corresponding first n layers in an untrained model and the remaining layers are fine-tuned on B. According to Fig. 8a, the fine-tuning is able to recover the accuracy no matter how many layers are transferred for AnA task, which means fragile co-adaptation phenomenon does not exist in this transfer learning task. In addition, the performance of transfer learning is dropped with the increase of the transferred layers for AnB task, which is caused by task-specific features learned by the model. Similarly, BnB and BnA hold the same conclusion, as shown in Fig. 8d. Furthermore, as shown in Figs. 8b, 8c, 8e, and 8f, the experimental results of TCN and FCN also hold the same conclusion. Additionally, in order to find out how much time can be reduced by transfer learning, the training time of the fine-tuned model and the pre-trained one is recorded, and then the ratio between them is calculated, as illustrated in Fig. 9. The results demonstrate that transferring one convolutional layer is able to reduce the training time by over 10% and transferring all 3 convolutional layers can reduce over 30% of the time. Moreover, the GCN is the most effective model to reduce training time.

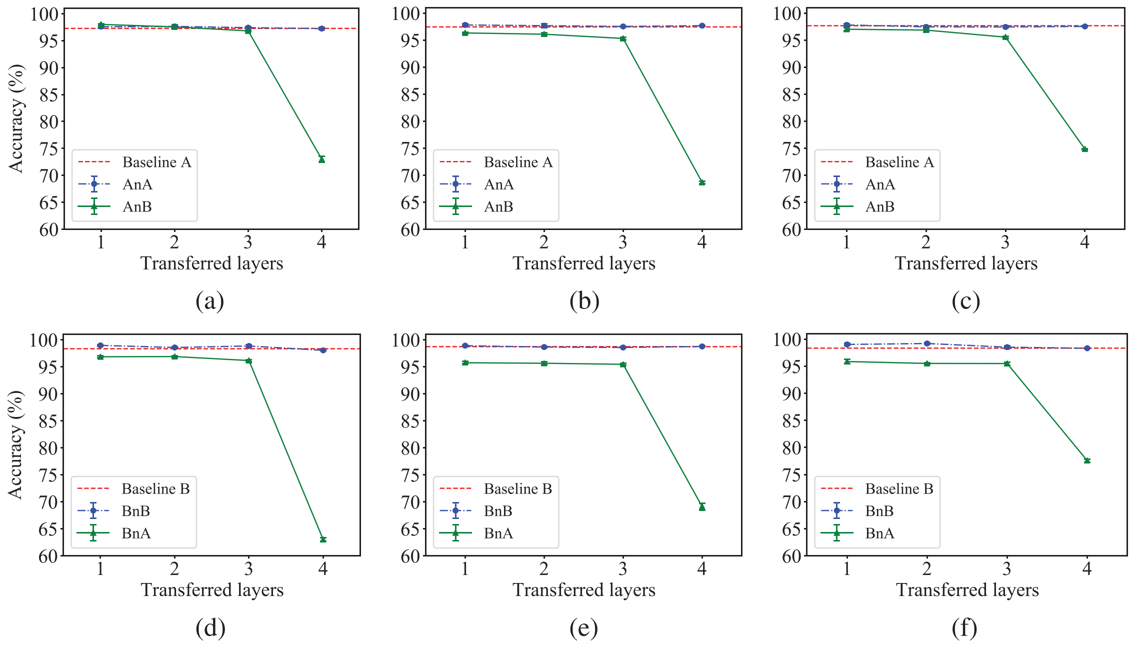

Figure 8: Accuracy with different transferred layers. (a) *AnX* tasks using GCN. (b) *AnX* tasks using TCN. (c) *AnX* tasks using FCN. (d) *BnX* tasks using GCN. (e) *BnX* tasks using TCN. (f) *BnX* tasks using FCN

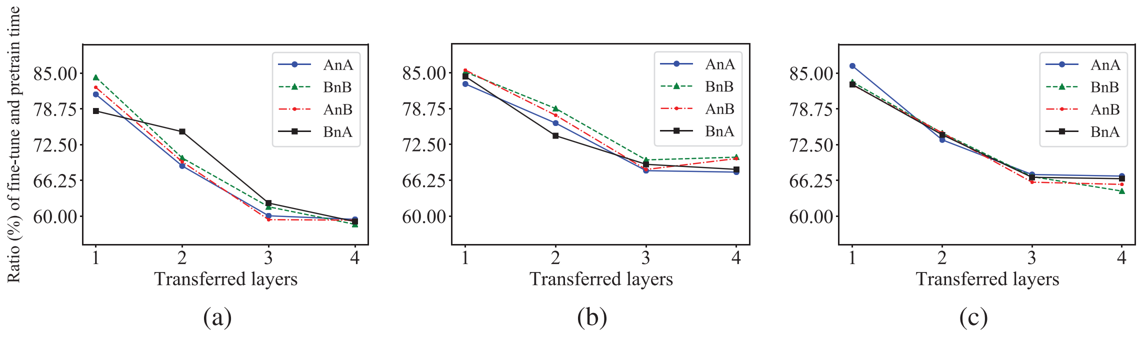

Figure 9: Ratio with different transferred layers. (a) GCN. (b) TCN. (c) FCN

We evaluate 3 different 1D-convolutional strategies on 6 transfer learning tasks, as shown in Table 4. Here, 'XCN-*n*' means transferring the first *n* layers of XCN, and each row in Table 4 demonstrates experimental results of XCN-*n* on different transfer tasks. For a fair comparison, TCN and FCN have the same hyper-parameters as GCN. From Table 4, in most cases, transferring the first 2 convolutional layers is better than transferring the first 3 layers directly, since the top convolutional layer is more task-specific than previous layers. However, the improvement is slight or even negative when the number of training instances in the target domain is small (i.e., D and E).

Table 4: Accuracy (mean ± std) with different transfer tasks on test set

Model	$A \to C$(%)	$A \to D$(%)	$A \to E$(%)	$B \to C$(%)	$B \to D$(%)	$B \to E$(%)
GCN-2	95.51 ± 0.58	100.0 ± 0.00	92.96 ± 1.78	96.65 ± 0.60	100.0 ± 0.00	91.55 ± 0.00
GCN-3	94.10 ± 0.82	100.0 ± 0.00	92.11 ± 2.11	95.86 ± 0.60	100.0 ± 0.00	91.55 ± 0.00
TCN-2	93.22 ± 0.45	98.93 ± 0.87	92.68 ± 1.38	96.04 ± 0.84	96.79 ± 1.34	88.73 ± 1.78
TCN-3	92.69 ± 0.35	98.57 ± 0.71	90.42 ± 1.64	94.80 ± 0.85	98.21 ± 0.00	89.30 ± 2.11
FCN-2	92.51 ± 1.15	100.0 ± 0.00	91.55 ± 0.00	95.33 ± 0.45	100.0 ± 0.00	89.86 ± 1.64
FCN-3	90.31 ± 0.56	100.0 ± 0.00	91.55 ± 0.00	94.63 ± 0.51	100.0 ± 0.00	89.30 ± 1.13

The other conclusion revealed by Table 4 is that the source domain, which contains more behavior representation, is more capable of improving the performance of transfer learning tasks. For example, the source domain A contains 1070 opcodes (APIs) and B contains 680 opcodes (APIs). Hence, the performance of $A \to D$ and $A \to E$ is better than (or equal to) the performance of $B \to D$ and $B \to E$. In addition, the accuracy of $A \to C$ is less than that of $B \to C$, since B and C both contain Kelihos malware as shown in Table 1, which means B and C are more similar.

5 Conclusion

In this paper, we propose a 1D-convolutional architecture for malware classification, and then evaluate the architecture via 3 convolutional strategies on 6 transfer learning tasks. Our experimental results verify that the transfer learning technique is an effective approach to reuse the learned knowledge from other datasets, and transferring the first 2 convolutional layers is a better choice for this malware classification task. Moreover, the experimental results also demonstrate the time reduction in the training process. For future work, our proposed method is expected to be evaluated on other extensive tasks, and domain adaption techniques will be further investigated for performance improvement.

Funding Statement: This work was supported in part by the National Natural Science Foundation of China under Grants U1836106 and 81961138010, in part by the Beijing Natural Science Foundation under Grants 19L2029 and M21032, in part by the Scientific and Technological Innovation Foundation of Foshan under Grants BK20BF010 and BK21BF001, in part by the Scientific and Technological Innovation Foundation of Shunde Graduate School, USTB, under Grant BK19BF006, and in part by the Fundamental Research Funds for the University of Science and Technology Beijing under Grant FRF-BD-19-012A.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Lee, J., Kim, J., Seo, J. (2019). Cyber attack scenarios on smart city and their ripple effects. *Proceedings of the International Conference on Platform Technology and Service*, pp. 1–5. Jeju, Korea, IEEE.

2. Al-Garadi, M. A., Mohamed, A., Al-Ali, A. K., Du, X. J., Ali, I. et al. (2020). A survey of machine and deep learning methods for Internet of Things (IoT) security. *IEEE Communications Surveys & Tutorials, 22(3),* 1646–1685. DOI 10.1109/COMST.2020.2988293.
3. Ma, H., Tian, J., Qiu, K., Lo, D., Gao, D. et al. (2021). Deep-learning-based app sensitive behavior surveillance for android powered cyber-physical systems. *IEEE Transactions on Industrial Informatics, 17(8),* 5840–5850. DOI 10.1109/TII.2020.3038745.
4. Chen, M., Li, Y., Luo, X., Wang, W., Wang, L. et al. (2019). A novel human activity recognition scheme for smart health using multilayer extreme learning machine. *IEEE Internet of Things Journal, 6(2),* 1410–1418. DOI 10.1109/JIOT.2018.2856241.
5. Luo, X., Sun, J., Wang, L., Wang, W., Zhao, W. et al. (2018). Short-term wind speed forecasting via stacked extreme learning machine with generalized correntropy. *IEEE Transactions on Industrial Informatics, 14(11),* 4963–4971. DOI 10.1109/TII.2018.2854549.
6. Luo, X., Li, J., Chen, M., Yang, X., Li, X. (2021). Ophthalmic disease detection via deep learning with a novel mixture loss function. *IEEE Journal of Biomedical and Health Informatics, 25(9),* 3332–3339. DOI 10.1109/JBHI.2021.3083605.
7. Rezende, E., Ruppert, G., Carvalho, T., Ramos, F., de Geus, P. (2017). Malicious software classification using transfer learning of resNet-50 deep neural network. *Proceedings of the 16th IEEE International Conference on Machine Learning and Applications*, pp. 1011–1014. Cancun, Mexico, IEEE.
8. Bhodia, N., Prajapati, P., Di Troia, F., Stamp, M. (2019). Transfer learning for image-based malware classification. *Proceedings of the 5th International Conference on Information Systems Security and Privacy*, pp. 719–726. Prague, Czech Republic, SciTePress.
9. Nahmias, D., Cohen, A., Nissim, N., Elovici, Y. (2020). Deep feature transfer learning for trusted and automated malware signature generation in private cloud environments. *Neural Networks, 124,* 243–257. DOI 10.1016/j.neunet.2020.01.003.
10. Sudhakar, Kumar, S. (2021). MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things. *Future Generation Computer Systems, 125,* 334–351. DOI 10.1016/j.future.2021.06.029.
11. Dauphin, Y. N., Fan, A., Auli, M., Grangier, D. (2017). Language modeling with gated convolutional networks. *Proceedings of the 34th International Conference on Machine Learning*, pp. 933–941. Sydney, Australia, IMLS.
12. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *Proceedings of the 34th International Conference on Machine Learning*, pp. 2029–2042. Sydney, Australia, IMLS.
13. Wang, Z., Yan, W., Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. *Proceedings of the International Joint Conference on Neural Networks*, pp. 1578–1585. Anchorage, AK, USA, IEEE.
14. HaddadPajouh, H., Dehghantanha, A., Khayami, R., Choo, K. K. R. (2018). A deep recurrent neural network based approach for internet of things malware threat hunting. *Future Generation Computer Systems, 85,* 88–96. DOI 10.1016/j.future.2018.03.007.
15. Kang, J., Jang, S., Li, S., Jeong, Y. S., Sung, Y. (2019). Long short-term memory-based malware classification method for information security. *Computers & Electrical Engineering, 77,* 366–375. DOI 10.1016/j.compeleceng.2019.06.014.
16. Jha, S., Prashar, D., Long, H. V., Taniar, D. (2020). Recurrent neural network for detecting malware. *Computers & Security, 99,* 102037. DOI 10.1016/j.cose.2020.102037.
17. Shelhamer, E., Long, J., Darrell, T. (2017). Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(4),* 640–651. DOI 10.1109/TPAMI.2016.2572683.
18. Hasegawa, C., Iyatomi, H. (2018). One-dimensional convolutional neural networks for android malware detection. *Proceedings of the IEEE 14th International Colloquium on Signal Processing & Its Applications*, pp. 99–102. Penang, Malaysia, IEEE.

19. Bai, S., Kolter, J. Z., Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271.
20. Sun, J., Luo, X., Gao, H., Wang, W., Gao, Y. et al. (2020). Categorizing malware via a word2vec-based temporal convolutional network scheme. *Journal of Cloud Computing, 9(1),* 53. DOI 10.1186/s13677-020-00200-y.
21. Pan, S. J., Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering, 22(10),* 1345–1359. DOI 10.1109/TKDE.2009.191.
22. Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., Muller, P. A. (2018). Transfer learning for time series classification. *Proceedings of the IEEE International Conference on Big Data*, pp. 1367–1376. Seattle, WA, USA, IEEE.
23. Gao, X., Hu, C., Shan, C., Liu, B., Niu, Z. et al. (2020). Malware classification for the cloud via semi-supervised transfer learning. *Journal of Information Security and Applications, 55,* 102661. DOI 10.1016/j.jisa.2020.102661.
24. Yosinski, J., Clune, J., Bengio, Y., Lipson, H. (2014). How transferable are features in deep neural networks? *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pp. 3320–3328. Montreal, QC, Canada, NIPSF.
25. Pennington, J., Socher, R., Manning, C. (2014). Glove: Global vectors for word representation. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543. Doha, Qatar, ACL.
26. He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778. Las Vegas, NV, USA, IEEE.
27. Ba, J. L., Kiros, J. R., Hinton, G. E. (2016). Layer normalization. arXiv:1607.06450.
28. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 5999–6009. Long Beach, CA, USA, NIPSF.
29. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M. (2018). Microsoft malware classification challenge. arXiv:1802.10135.
30. Allan, N., Ngubiri, J. (2019). Windows PE API calls for malicious and benigin programs. DOI 10.13140/RG.2.2.14417.68960.