Journal of Computer and Communications, 2024, 12, 201-227
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

A Lightweight IoT Malware Detection and Family Classification Method

Changguang Wang[1,2], Ziqi Ma[1], Qingru Li[1,2], Dongmei Zhao[1,2], Fangwei Wang[1,2*]

[1]College of Computer and Cyber Security, Hebei Normal University, Shijiazhuang, China
[2]Key Laboratory of Network and Information Security of Hebei Province, Hebei Normal University, Shijiazhuang, China
Email: *fw_wang@hebtu.edu.cn

How to cite this paper: Wang, C.G., Ma, Z.Q., Li, Q.R., Zhao, D.M. and Wang, F.W. (2024) A Lightweight IoT Malware Detection and Family Classification Method. *Journal of Computer and Communications*, **12**, 201-227.
https://doi.org/10.4236/jcc.2024.124015

Received: March 6, 2024
Accepted: April 27, 2024
Published: April 30, 2024

Abstract

A lightweight malware detection and family classification system for the Internet of Things (IoT) was designed to solve the difficulty of deploying defense models caused by the limited computing and storage resources of IoT devices. By training complex models with IoT software gray-scale images and utilizing the gradient-weighted class-activated mapping technique, the system can identify key codes that influence model decisions. This allows for the reconstruction of gray-scale images to train a lightweight model called LMDNet for malware detection. Additionally, the multi-teacher knowledge distillation method is employed to train KD-LMDNet, which focuses on classifying malware families. The results indicate that the model's identification speed surpasses that of traditional methods by 23.68%. Moreover, the accuracy achieved on the Malimg dataset for family classification is an impressive 99.07%. Furthermore, with a model size of only 0.45M, it appears to be well-suited for the IoT environment. By training complex models using IoT software gray-scale images and utilizing the gradient-weighted class-activated mapping technique, the system can identify key codes that influence model decisions. This allows for the reconstruction of gray-scale images to train a lightweight model called LMDNet for malware detection. Thus, the presented approach can address the challenges associated with malware detection and family classification in IoT devices.

Keywords

IoT Security, Visual Explanations, Multi-Teacher Knowledge Distillation, Lightweight CNN

1. Introduction

The Internet of Things (IoT) is a vast network that integrates diverse informa-

tion sensors with the Internet. It is at the forefront of the global information industry. Nowadays, IoT has been applied in extensive industries, including smart transportation [1], smart homes [2], agricultural production [3], and healthcare [4]. Meanwhile, the security of IoT [5] is attracting more attention.

The landscape of cybersecurity threats, particularly concerning the Internet of Things (IoT), has indeed been evolving significantly over recent years. In 2020, a cybersecurity officer managed to exploit Bluetooth vulnerability and successfully attacked a Tesla Model X in less than two minutes [6]. In 2021, Swiss hackers successfully breached 150,000 Verkada live cameras that are primarily used for monitoring in public places such as schools, hospitals, prisons, and businesses [7]. The 2023 Cyber Threat Landscape report published by SonicWall highlights that the number of IoT malware instances has surpassed 100 million in 2022 [8]. Furthermore, a new attack type called IoT ransomware or R4IoT has emerged, which goes beyond encryption and data leakage. This attack can disrupt industrial production by Programmable Logic Controllers (PLC) and other means, causing significant disruptions in critical infrastructure and operational technology environments. Therefore, the ability to detect unknown malware and identify their families becomes important due to rapid changes in the threat landscape and the emergence of unknown malware.

Malware is commonly defined as software that poses a threat to the security of a user's computer and jeopardizes the user's interests. The interception of unknown malware and its variants presents a formidable challenge. Traditional methods of malware detection and family classification primarily rely on static analysis, dynamic analysis, and hybrid analysis [9]. Malware programs require execution on specific platforms. For IoT devices, the predominant operating system is Linux, which employs executable ELF files (.elf, .o, .so, etc.), similar to Windows executable PE files (.exe, .obj, .dll, etc.). Additionally, IoT devices often utilize various CPU instruction set architectures (such as ARM, MIPS, x86, etc.). The diversity of IoT devices makes it challenging to establish uniform standards for different types of IoT hardware and software. Moreover, the limited computing resources and storage space available in IoT devices make it difficult to set up a dynamic analysis environment suitable for IoT software disassembly and configuration [10]. Furthermore, security schemes designed for Windows systems are not easily implementable on IoT devices [11]. These factors pose significant challenges to the rapid malware detection on IoT devices.

The method based on the raw bytes of malware binaries primarily focuses on the characteristics of the abstract binary data, eliminating the need to address issues arising from platform heterogeneity, such as different opcodes and instruction sets. By avoiding the execution of malware, this approach overcomes the challenge of configuring detection environments that are specific to diverse IoT devices. However, the conversion of malware binaries into gray-scale images which forms the basis of this method, relies on complex models [12] [13] [14]. Consequently, its deployment in the IoT context is impractical. Furthermore, the majority of these methods are currently employed for Windows and Android

malware detection, without validation on IoT malware datasets. Therefore, the issues of security requirements of different platforms and insufficient feature extraction capabilities of small models are crucial.

In this paper, we propose a lightweight malware detection and family classification method based on visual explanations to enhance the accuracy and recognition speed of small models in IoT environments. In the detection, all samples are initially converted into gray-scale images, forming a gray-scale image dataset. Subsequently, ResNet-34 is trained on this dataset. To gain insights into the model's attention during malware detection, we employ the Gradient-weighted class-activated mapping (Grad-CAM) technique. Additionally, we extract and analyze the coordinates of key locations that significantly influence the model's decision-making process. To train the lightweight malware detection networks (LMDNet), we construct a new dataset by transforming only the binary encoding of high-frequency locations into gray-scale images. Experimental results demonstrate that the lightweight model exhibits improved accuracy and recognition speed. For family classification, we utilize VGG-16 and ResNet-34 as the teacher models, while LMDNet serves as the student model. The multi-teacher knowledge distillation method is employed to train the student model on the gray-scale image dataset of malware. Notably, our approach eliminates the need for decompiled files and dynamic analysis, significantly reducing the cost of feature engineering. In contrast to previous deep learning techniques relying solely on gray-scale images of malware, this paper strikes a balance between model accuracy and size.

The main contributions of this paper are as follows:

1) The gradient-weighted class-activated mapping technique was employed to visualize and quantify the high-frequency locations of key codes that have a significant impact on the model's decision-making process. Subsequently, these key location codes were extracted and transformed into grayscale images to train the lightweight malware detection model. This approach addresses the limitations of small models with limited feature extraction capabilities while substantially improving the model's recognition speed.

2) The lightweight model, LMDNet, is designed by incorporating deep separable convolution, channel shuffle, group convolution, and the Efficient Channel Attention (ECA) module, which reduces the number of parameters in the model while improving its accuracy. The model proposed in this paper exhibits a small parameter, compact model size, and faster training and recognition speeds.

3) In the context of malware family classification, a lightweight convolutional neural network is trained using a multi-teacher knowledge distillation approach. This methodology effectively enhances the accuracy of the model in accurately classifying different malware families.

2. Related Work

The primary methods for IoT malware detection and family classification in-

volve static analysis and dynamic analysis. Static analysis methods rely on examining the characteristics of the malware without executing it, while dynamic analysis involves observing the behavior of malware in a controlled environment. Static feature-based analysis methods in IoT malware detection typically utilize both low-level and high-level features. Low-level features include file structure and raw binaries, which can be directly obtained from the malware binaries. High-level features, such as control flow graphs, opcodes, and strings, require disassembly to extract relevant information. Given the diverse CPU architectures and limited computational and storage resources of IoT devices, this paper categorizes the related work into three directions: static high-level feature analysis, static low-level feature analysis, and dynamic analysis.

2.1. Static Features-Based Methods

In static analysis, the malware detection methods rely on extracting the static features such as Operation Codes, Strings, or File Structure to distinguish malicious samples. These characteristics can be divided into two groups: low-level features and high-level features. In particular, the low-level features can be obtained directly from the binary file structure itself, while the high-level features must be extracted by a disassembler.

1) In IoT malware detection, static high-level feature analysis primarily focuses on opcodes, which are individual instructions executed by the CPU and describe the behavior of executable files. Researchers have utilized various approaches in this context. HaddadPajouh *et al.* [15] employed a deep recurrent neural network-based approach that utilizes operand sequences for IoT malware detection. Their method achieved an accuracy of 98.18% on a dataset comprising 270 benign samples and 281 ARM-based IoT malware samples. Darabian *et al.* [16] discovered that certain opcodes were more frequently repeated in malware compared to benign software. They developed a malware detection technique based on counting the number of opcode repetitions in executable files, achieving an accuracy of 99%. Dovom *et al.* [17] transformed the opcodes of a program into a vector space and utilized fuzzy and fast fuzzy pattern tree methods for malware detection. Their approach was tested on an ARM-based IoT dataset, consisting of 1078 benign samples and 128 malware samples, achieving an accuracy of 99.83%.

In summary, it is important to note that different CPU architectures utilize distinct software opcodes and instruction sets. Therefore, static high-level feature analysis methods may not be effective in detecting IoT malware across various architectures.

2) Static low-level feature-based analysis methods in IoT malware detection encompass the ELF file header-based approach and the gray image-based approach. Notable studies have explored these methods and achieved varying levels of accuracy. Shahzad *et al.* [18] extracted 383 features from ELF file headers for malware family classification, attaining a 99% accuracy rate on a dataset con-

sisting of 709 malicious samples. Bai *et al.* [19] utilized information extracted from the ELF file symbol table for IoT malware classification, achieving an accuracy of 98%. The gray image-based approach, initially proposed by Nataraj *et al.* [20], involved converting binary files into grayscale images and extracting gist texture features for malware classification. Su *et al.* [21] proposed a lightweight solution utilizing shallow convolutional neural networks to detect IoT malware. Both malware and benign samples were classified by inputting grayscale images into the network. However, the shallow convolutional neural network in their study had limited model learning capability, resulting in an accuracy of only 81.8% in classifying benign software and two types of IoT malware. Karanja *et al.* [22] employed Haralick image texture features of malware gray-scale images in combination with machine learning methods for IoT malware classification. They achieved an accuracy of 95% using a random forest classifier. However, their dataset was relatively small, with only 133 Gafygt class samples and 125 Mirai class samples. Wang *et al.* [12] proposed a lightweight attention module called DEAM to enhance the application of the channel attention model in malware detection. Their method, combined with DenseNet, improved malware detection by focusing on malware features. However, their method was specifically applied to a Windows malware dataset, making it challenging to extrapolate its performance to heterogeneous malware datasets. Yuan *et al.* [23] introduced a method for IoT malware classification using lightweight convolutional neural networks and multidimensional Markov images. This approach demonstrated an average accuracy higher than 95% on an IoT malware dataset. However, the multidimensional Markov image used in their method is a three-channel image of 256 × 256 pixels, which is larger than the size of many IoT software, resulting in redundant information.

Static low-level feature-based analysis does not require consideration of different opcodes and instruction sets, making it applicable to various platforms. However, shallow convolutional neural networks and traditional machine learning methods tend to have lower accuracy. On the other hand, complex models that achieve high accuracy often rely on significant computational and storage resources, making them challenging to employ in the IoT context.

2.2. Dynamic Features-Based Methods

Dynamic code analysis, as a part of code debugging, involves analyzing the behavior of an application during its execution. This approach allows for thorough testing of the program under various scenarios, eliminating the need to create artificial inputs or situations that may introduce unexpected errors. By scrutinizing the program's behavior at runtime, dynamic code analysis can identify unforeseen issues and un-cover required functionalities that may not have been apparent during the design stage. While it is impossible to account for all potential scenarios, dynamic code analysis is a standard procedure that reduces testing costs and time while facilitating maintenance activities.

Chen *et al.* [24] developed an automated virtual machine-based analysis system to collect IoT malware behaviors, including API calls and system calls. Multiple API calls were aggregated into a family behavior graph for further analysis. Jeon *et al.* [25] analyzed IoT malware on a cloud platform, extracting data on behaviors related to memory, network, virtual file system, processes, and system calls. The behavioral data was then transformed into IoT malware behavioral images and classified using convolutional neural networks. However, this approach relies on the cloud platform and cannot be directly deployed on IoT devices. There are several sandboxing tools available for IoT malware analysis [26] [27] [28]. These automated sandbox analysis tools allow malware to be executed in a controlled and isolated environment, generating malware analysis reports automatically. However, the challenge with sandboxing is the need to simulate the underlying environment in which IoT malware operates. Currently, there are no sandbox analysis tools that can support all CPU architectures of IoT samples.

While dynamic analysis methods tend to be more accurate, they require the development of specific analysis tools for different platforms as IoT devices run on different CPU architectures. Dynamic analysis also consumes more computing re-sources and storage space compared to static analysis. Additionally, dynamic analysis is time-consuming, which can be considered a disadvantage.

3. Proposed Method

To solve the problem of low accuracy and recognition speed of small models in IoT malware detection and family classification, we propose a lightweight malware detection and family classification method for IoT based on visual explanations. The general framework of our proposed method is illustrated in **Figure 1**.

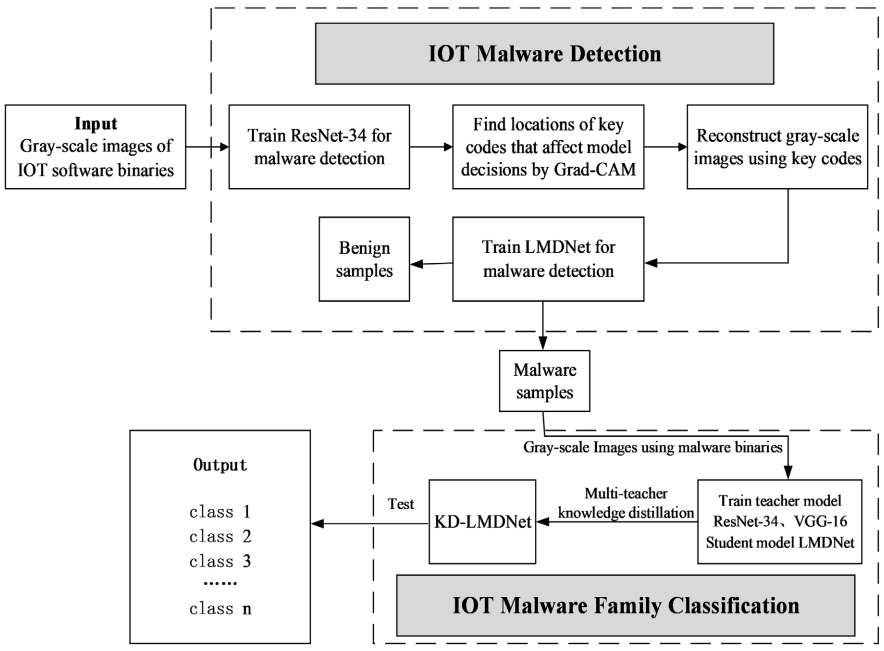

Figure 1. The framework of malware detection and family classification for IoT.

In this section, we will provide a detailed introduction to our proposed model. In malware detection, it is often not necessary to traverse all the information of the software to determine if it has malicious functions. By focusing on local and key information, we can improve the efficiency of detection. We start by converting IoT software into gray-scale images. A large-scale model is trained on this dataset, and we utilize the gradient-weighted class-activated mapping method to visualize and count the high-frequency locations of key codes that influence the model's decisions. We then intercept these key location codes and reconstruct grayscale images to train the light-weight model. This approach addresses the limited feature extraction capabilities of small models, thereby enhancing their accuracy and speed in malware detection and classification.

In malware family classification, preserving as much original information about the malware as possible is crucial for improving classification accuracy. In our method, we convert IoT malware binary files into grayscale images, which can be executed on different CPU architectures. By employing the technique of multi-teacher knowledge distillation, we train a student model called LMDNet to enhance its accuracy in malware family classification.

3.1. Targeting Malicious Code Locations by Gard-CAM

Malware detection poses challenges in identifying the location of malicious code solely through human analysis because not all code within malware exhibits malicious functions, and a significant portion of the malware consists of benign information. Therefore, isolating the malicious code portion and providing it to the model for learning, we can enhance the accuracy, training speed, and recognition speed of the model. In this paper, we convert IoT software into grayscale images for malware detection. These images are used to train a ResNet-34 model. We employ the Grad-CAM [29] technique to visualize and count the regions that contribute to the decisions made by the convolutional neural network. This visualization and counting process helps us identify the regions of interest in the malware images. We then transform the local binary code, which contains the focused information, into grayscale images to create a new dataset. The process is depicted in **Figure 2**.

Grad-CAM is a technique for interpreting decisions in convolutional neural net-works. It is possible to visualize the results of decisions in a convolutional neural network as to which regions of the image contributed to the decision. To generate a Grad-CAM, the first step is to perform a forward pass of the input image through the CNN and obtain the prediction probabilities for different classes. Then, the gradients of the predicted class with respect to the feature maps of the last convolutional layer are computed using back propagation. These gradients represent the importance of each feature map in the final prediction. Next, the global average pooling is applied to the gradients, resulting in a weighted combination of the gradients for each feature map. This step helps to aggregate the importance of different spatial locations within each feature map.

After obtaining the weighted combination, a heat map is generated by linearly combining the feature maps with their corresponding weights. This heat map represents the saliency or relevance of each pixel in the input image to the predicted class. By overlaying the heat map on the original input image, Grad-CAM produces a visual explanation of the decision-making process of the CNN. The regions in the image that are highlighted by the heat map indicate the areas that strongly contribute to the network's prediction. This helps us to understand the decision-making process and the basis of the model. This is shown in **Figure 3**. In malware image classification, the network is first forward propagated to obtain the feature layer A and the model prediction y. If we want to look at the regions where the model is particularly interested in benign samples, we first derive the logits of the model for benign samples by training, y^c. The backward propagation is then performed on y^c to obtain the gradient information A' at the feature layer A. By calculating the weights that determine the influence of each channel in the feature layer A on the final decision, a weighted summation

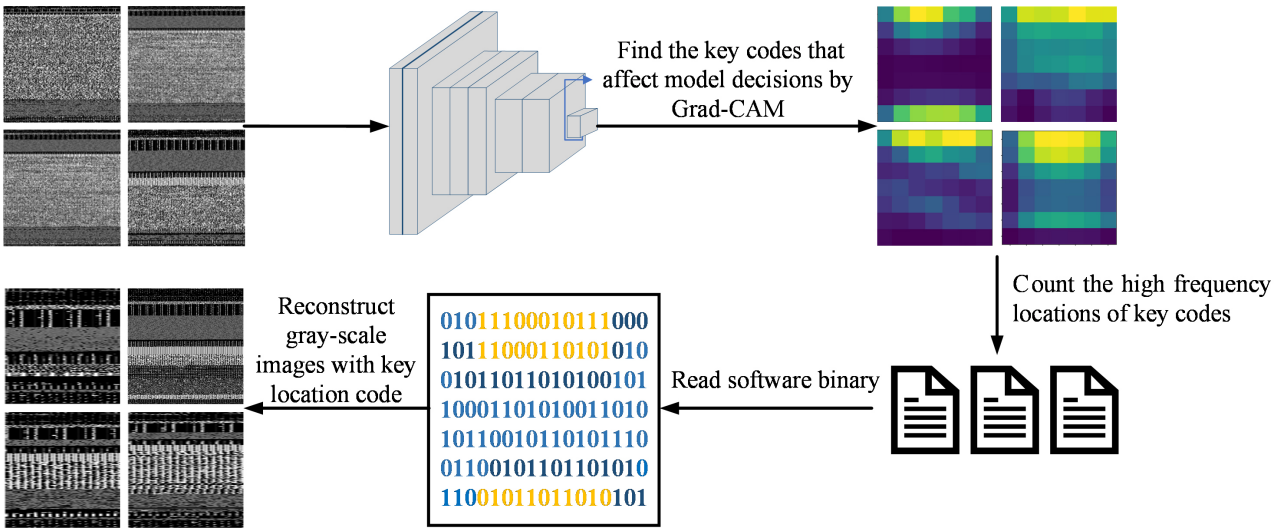

Figure 2. Targeting malicious code locations by Gard-CAM and converting important location codes to grey-scale images.

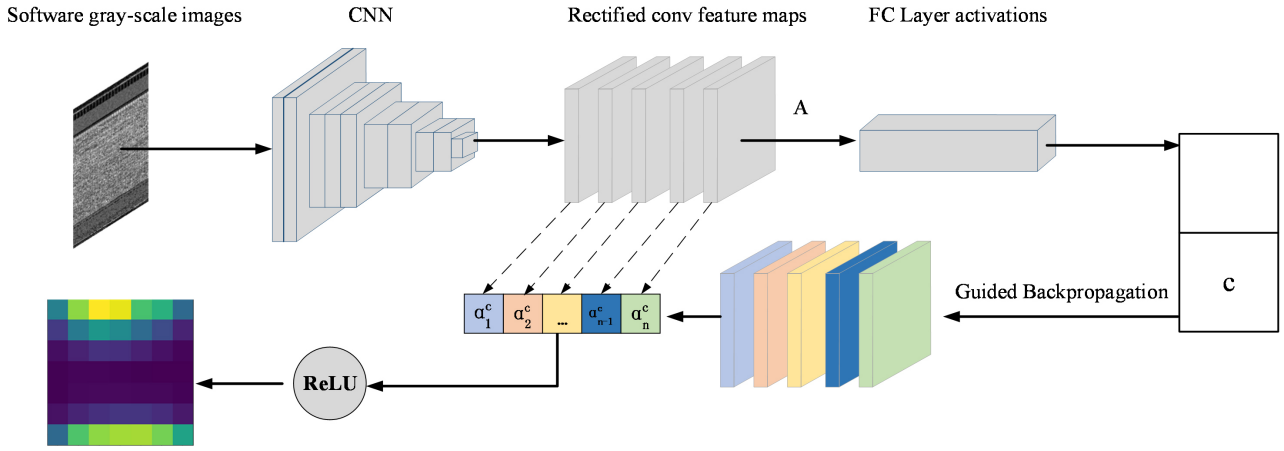

Figure 3. Generating heatmaps by Gard-CAM.

is computed. The corresponding class activation map for each category is obtained using ReLU activation. The process of gradient-weighted class activation mapping can be represented as follows:

$$L_{\text{Grad-GAM}}^{C} = \text{ReLU}\left(\sum_{k} \alpha_k^c A^k\right) \tag{1}$$

$$\alpha_k^c = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{2}$$

A represents a specific feature layer. In convolutional neural networks, high-level semantic features are extracted by deep convolutional layers [30]. These deep convolutional layers also contain information in both the spatial dimensions (length and width). Hence, in this study, we selected the feature layer output from the last convolutional layer of the model. k represents the kth channel in feature layer A; c rep-resents a specific class; A^k represents the data of channel k in feature layer A; α_k^c represents the weights applied to A^k; y^c represents the logits of the network for class c. These logits refer to the predicted probability for class c without being passed through the Softmax activation function; A_{ij}^k represents the data at position (i, j) in channel k of feature layer A. Z represents the size of the feature layer.

In this paper, we focus on extracting key positions within the feature layer A, which is obtained from the last convolutional layer output of the ResNet-34 model. The feature map size of layer A is 7 × 7. By applying the Grad-CAM technique, we generate a class activation map that is also a 7 × 7 two-dimensional array. Due to the highly abstract nature of gray-scale images, it is difficult to visually assess the magnitude of the impact of heatmaps on model decisions. Therefore, we conducted extensive experiments on natural image classification datasets. Through experiments, we identified positions with heatmap values greater than 0.8 as critical locations influencing model decisions. As an example, **Figure 4** illustrates the significant regions of interest when the model recognizes malware. The portion of the heatmap with values greater than 0.8 corresponds to the location of the malware. Specifically, we count the frequency of the 49 positions in the array where the element values exceed a threshold of 0.8. These positions correspond to regions in the grayscale image that have a strong impact on the model's decision-making process. After counting the frequency of occurrence, we sort the 49 coordinate points based on the frequency of their associated key codes. This sorting process allows us to identify the key positions that have a higher frequency of occurrence and therefore have a more significant influence on the model's decision. By focusing on these key positions, we can effectively extract the essential features from the malware images and improve the accuracy and performance of the lightweight model for malware detection and classification.

3.2. Design of Lightweight Convolutional Neural Network (LMDNet)

The design of LMDNet relies on deepwise separable convolution and group

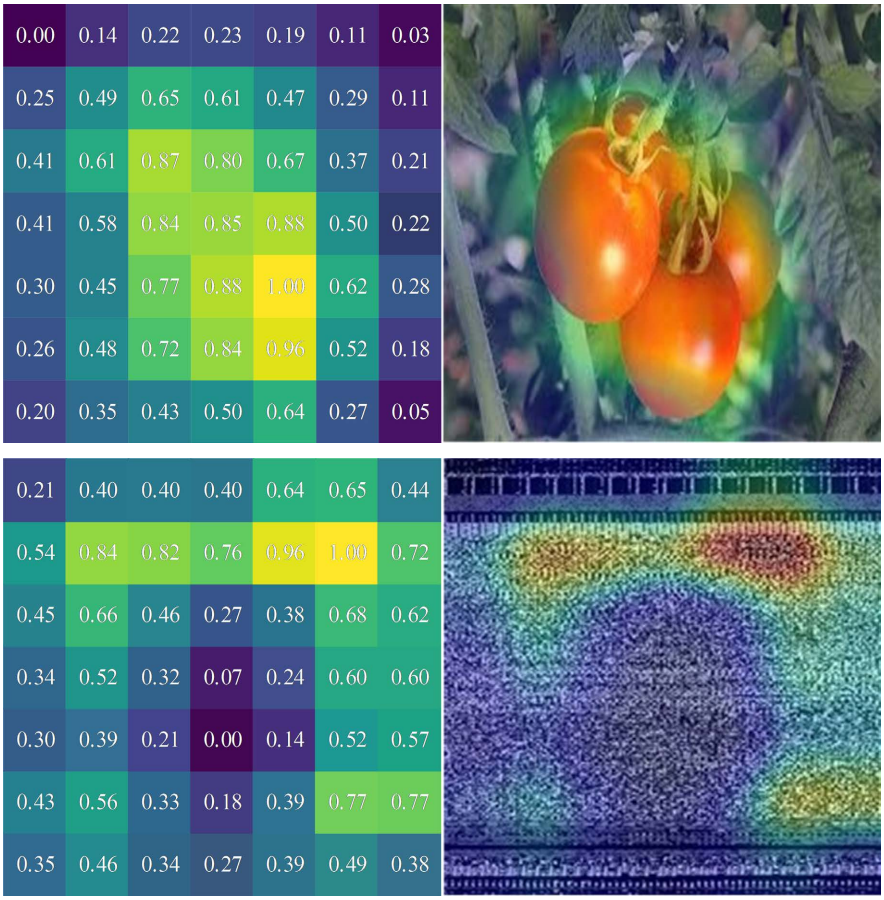

Figure 4. The decision basis for malware recognition using Grad-CAM.

convolution to reduce the number of floating-point operations and the number of parameters in the model. Channel shuffle and ECA module are used to enhance the model's channel feature learning capability.

3.2.1. Deepwise Separable Convolution and Group Convolution

Deepwise separable convolution consists of two processes: depthwise convolution and pointwise convolution. Depthwise convolution involves convolving the input features with the convolutional kernel on a per-channel basis to obtain spatial information. Subsequently, the output features of the depthwise convolution are used as input features for the next layer, where pointwise convolution is performed. Point-wise convolution employs a 1 × 1 convolutional kernel to convolve the output features of the depthwise convolution to obtain channel information. The decomposition of standard convolution into depthwise convolution and pointwise convolution is shown in **Figure 5**. $D_k \times D_k$ is the convolution kernel size, and M and N are the number of input and output channels.

Number of parameters and floating point operations for standard convolution with depthwise separable convolution:

$$P_1 = D_k \times D_k \times M \times N \tag{3}$$

$$C_1 = D_k \times D_k \times M \times N \times D_F \times D_F \tag{4}$$

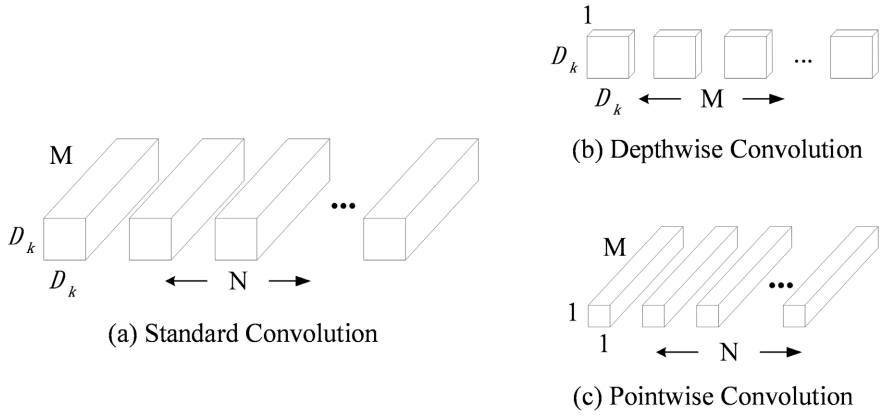

Figure 5. Standard convolution and depthwise separable convolution.

$$P_2 = D_k \times D_k \times M + M \times N \tag{5}$$

$$C_2 = D_k \times D_k \times M \times D_F \times D_F + M \times N \times D_F \times D_F \tag{6}$$

$$\frac{P_2}{P_1} = \frac{1}{N} + \frac{1}{D_k^2} \tag{7}$$

$$\frac{C_2}{C_1} = \frac{1}{N} + \frac{1}{D_k^2} \tag{8}$$

In the formula, P_1 and C_1 denote the number of parameters and floating point operations for standard convolution, P_2 and C_2 denote the number of parameters and floating point operations for depth-separable convolution, and $D_F \times D_F$ means the feature map size. Equation (7) and (8) show that the number of parameters and floating-point operations of the depth-separable convolution is $\frac{1}{N} + \frac{1}{D_k^2}$ of that of the standard convolution. In this paper, the size of the convolution kernel for the depth-wise convolution operation is 3 × 3, so the number of deepwise separable convolution parameters and the number of floating-point operations is about 1/9 of the number of standard convolution parameters.

3.2.2. Channel Shuffle and ECA Module

Group convolution divides the input feature map into G groups by channel. The convolution kernel performs the convolution operation on the input features of the same group only, and then concatenates the output results of each group to obtain the final output features. Number of covariates for group convolution:

$$P_3 = D_k \times D_k \times \frac{M}{G} \times N \tag{9}$$

where P_3 means the number of parameters of the grouped convolution. Group convolution has only $1/G$ of the number of parameters of standard convolution, so grouped convolution has some lightweighting effect. The idea of grouped convolution has its roots in LeNet-5 [31] and AlexNet [32]. Because of the limitation of early GPU storage, splitting the model and training it through two

GPUs can solve this problem.

In group convolution, the convolutional kernel operates only on the inputs within its group, resulting in relatively isolated feature information between different groups. This isolation leads to the inability to capture information from all input features in the output feature maps. To address this issue, we introduce channel shuffle, a technique that reorganizes feature information across different groups, allowing for better integration among the groups without increasing computational complexity. The detailed processes of grouped convolution and channel shuffle are illustrated in Figure 6.

In 2020 Wang [33] *et al.* proposed an efficient channel attention (ECA) module, which can effectively capture the information of cross-channel interactions and achieve the effect of channel feature enhancement. As depicted in Figure 7, the Efficient Channel Attention (ECA) module operates by performing a one-dimensional local convolution operation among the original channel data, enabling the fusion of local channel information. Subsequently, an appropriate activation function is applied. This approach effectively mitigates the computational and parameter increase issues encountered with the fully connected layers employed in SENet [34]. The ECA module, as a lightweight and readily applicable component, significantly enhances network performance while ensuring a lightweight design.

The output of the ECA module does not alter the size of the feature maps. During the learning process of channel features, a 1 × 1 convolutional kernel is employed, where the length of the convolutional kernel remains fixed for the

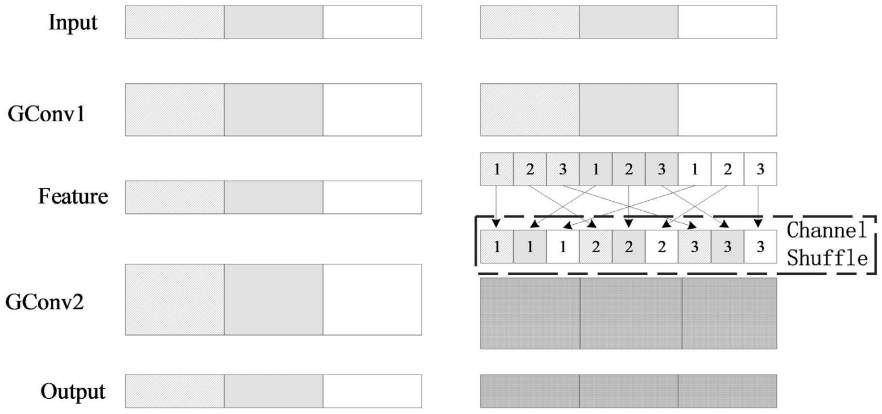

Figure 6. Group convolution and channel shuffle.

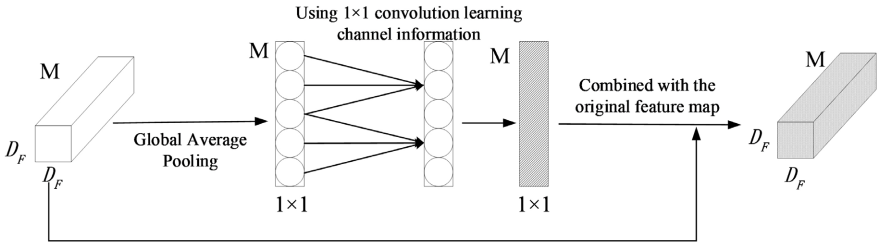

Figure 7. Efficient Channel Attention (ECA) module.

high and low-dimensional channels, while the size of the convolutional kernel for the remaining channels is set to 1 × 1 L. The value of L adapts and changes based on the number of input feature map channels.

$$L = \varphi(M) = \left| \frac{\log_2(M)}{a} + \frac{b}{a} \right|_{odd} \tag{10}$$

In the publicity, odd means that L can only take odd numbers. *a* and *b* are used to adjust the ratio of the number of channels M to the length L of the 1 × 1 convolution kernel.

3.2.3. The Overall Structure of LMDNet

The basic unit and structure of the proposed lightweight student network, LMDNet, is shown in **Figure 8**.

In LMDNet, the input grayscale images derived from malicious binary files are treated as three-channel images instead of single-channel. This means that the gray-scale image is replicated across all three channels to create a three-channel representation. This approach has been experimentally shown to improve the classification accuracy. The decision to treat grayscale images as three-channel images is motivated by previous research studies in different domains, such as the work of Kalash *et al.* [35] and medical image classification [36]. These studies have observed that reading grayscale images as three-channel images can result in better classification performance. In the initial stage of

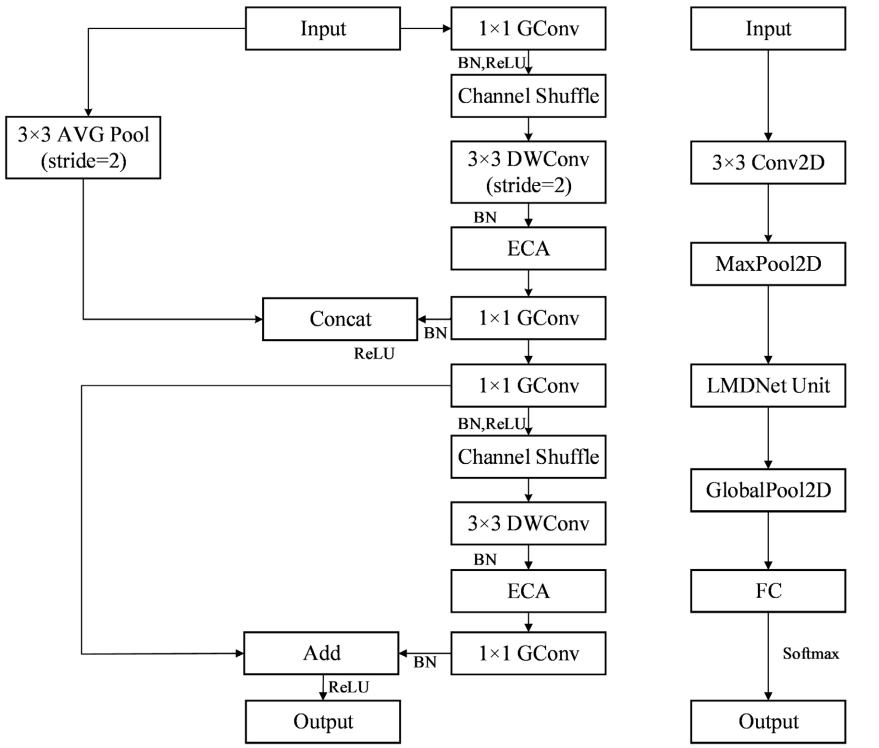

Figure 8. The basic unit and structure of LMDNet.

LMDNet, we perform a 3 × 3 convolution followed by max pooling. This helps introduce initial non-linear feature transformations and gradually increasing receptive fields, providing richer input features for subsequent depth-wise separable convolution modules. The LMDNet basic unit is then used for feature extraction. Initially, the feature maps are downsampled and divided into multiple groups using grouped convolution. Channel shuffling is applied to alternate the feature maps along the channel dimension, enhancing interaction between the groups. The feature maps are then processed using depth-wise separable convolution to combine local spatial and channel information, and an ECA (Efficient Channel Attention) module is employed to enhance long-range dependencies between channels. Finally, a 1 × 1 grouped convolution is used for dimensionality increase. In the experiments, the addition of channel shuffling after the last grouped convolution has only a slight impact on the results, so it is not included after the grouped convolution. The first half of the basic unit performs downsampling, while the second half is responsible for feature extraction. Specifically, the output channel numbers of the three basic units are set as 120, 240, and 480. The fully connected layers in the network are adjusted accordingly based on the number of classes required for the classification task.

3.3. Malware Image Classification Using Multi-Teacher Knowledge Distillation

In traditional malware image classification studies, complex models exhibit good performance and generalization capabilities. However, they often come with a large number of parameters, requiring significant storage and computational resources. This limitation hinders the direct application of traditional malware detection and family classification methods in the IoT environment.

Knowledge distillation provides a solution by transferring the knowledge from a large teacher model to a lightweight student model. This approach compensates for the accuracy limitations of the student model caused by its smaller network size, resulting in improved performance. By leveraging knowledge distillation, the student model can benefit from the expertise of the larger model while maintaining a smaller footprint.

The concept of knowledge distillation was formally introduced by Hinton *et al.* in 2015. In the training process of knowledge distillation, the student model learns from the output of the teacher model at the Softmax layer. By introducing a distillation temperature, denoted as T, soft labels are generated. These soft labels are then used to train the student network, allowing the network to learn additional knowledge by exposing the information related to non-correct class probabilities more thoroughly.

In this paper, we employ a multi-teacher knowledge distillation approach to enhance the effectiveness of distillation by integrating predictions from multiple teachers. The efficacy of multi-teacher knowledge distillation has been demonstrated in various studies [37] [38] [39]. The process of multi-teacher knowledge

distillation can be represented by the following formula:

$$p_i^T = \frac{\exp\left(\frac{v_i}{T}\right)}{\sum_k^N \exp\left(\frac{v_k}{T}\right)} \tag{11}$$

$$q_i^T = \frac{\exp\left(\frac{z_i}{T}\right)}{S_k^N \exp\left(\frac{z_k}{T}\right)} \tag{12}$$

$$L_{soft} = -\sum_i^N p_i^T \log\left(q_i^T\right) \tag{13}$$

$$L_{mkd} = \sum_{i=1}^m Q_i \times L_{soft} \tag{14}$$

where represents the logits of the teacher model, z_i represents the logits of the student model, p_i^T and q_i^T refer to the softened outputs of the teacher and student models, respectively, under the temperature T. L_{soft} represents the cross-entropy between the student model and a single teacher model's softened labels at temperature T. N denotes the total number of labels. Q_i represents the weights assigned to different teacher models during the knowledge distillation process. L_{mkd} represents the cross-entropy between the student model and multiple teacher models with different weights, resulting in softened labels at temperature T.

The overall loss of knowledge distillation can be divided into two components: distillation loss and student loss. The distillation loss is partially derived from the cross-entropy loss between the student network's output and the soft labels generated by the teacher network using the temperature T. The student loss, on the other hand, is computed using the cross-entropy loss between the student network's output and the true labels. The total loss is a weighted sum of these two losses.

Specifically, the process of knowledge distillation involves two aspects. Firstly, the student model is trained to fit the soft label information generated by the teacher network, enabling the student network to learn underlying semantic information and capture the experience of the teacher network. Secondly, the student network is trained with the cross-entropy loss using the true hard labels, allowing it to understand the differences in real data. The total loss is obtained by combining these two losses with appropriate weights.

The cross-entropy loss between the Softmax output of the student model, under the condition of introducing the temperature parameter T = 1, and the true labels forms the second part of the overall loss function, L_{hard} .

$$L_{hard} = -\sum_j^N c_j \log\left(q_j^1\right) \tag{15}$$

$$q_j^1 = \frac{\exp\left(z_i\right)}{\sum_k^N \exp\left(z_k\right)} \tag{16}$$

Knowledge distillation utilizes a weighted combination of soft and hard label cross-entropy loss functions to train the student model. The parameter Q represents the proportion of distillation loss. The overall weighted function can be expressed as:

$$Loss = Q \times L_{mkd} + 1 - Q \times L_{hard} \tag{17}$$

The proposed LMDNet structure in this paper is simple and has fewer trainable parameters, resulting in limited accuracy. Therefore, this paper adopts a multi-teacher knowledge distillation approach to enhance the accuracy of LMDNet. LMDNet is used as the student model, while ResNet-34 and VGG-16 are employed as the teacher models. The multi-teacher knowledge distillation structure used in this paper is illustrated in Figure 9.

4. Experimental Results and Analysis

4.1. Data Preprocessing

This paper utilizes three datasets: the IoT malware detection dataset, the IoT malware family classification dataset, and the Malimg dataset. As shown in Table 1, the Malimg dataset consists of 9339 malicious samples, distributed among 25 malicious families. The IoT malware detection dataset comprises 11,499 benign and malicious samples across various architectures such as MIPS, x86, SUPERH, etc., with an equal number of samples for each category. The IoT malware family classification dataset consists of 11,499 samples from three malicious families, selected from the detection dataset. Further details can be found in Table 2 and Table 3.

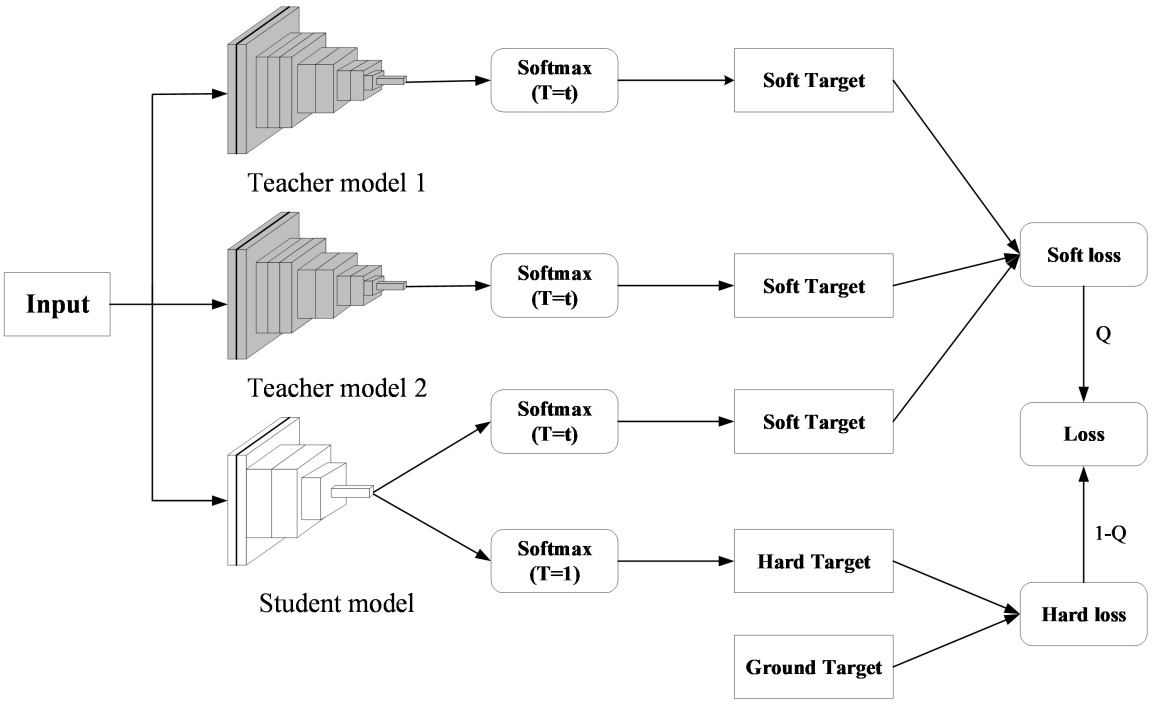

Figure 9. Multi-teacher knowledge distillation framework.

Table 1. MalImg: distribution of samples.

No.	Family	Number of samples
1	Adialer. C	122
2	Agent. FYI	116
3	Allaple. A	2949
4	Allaple. L	1591
5	Alueron. gen!J	198
6	Autorun. K	106
7	C2LOP. gen!g	200
8	C2LOP. P	146
9	Dialplatform. B	177
10	Dontovo. A	162
11	Fakerean	381
12	Instantaccess	431
13	Lolyda. AA1	213
14	Lolyda. AA2	184
15	Lolyda. AA3	123
16	Lolyda. AT	159
17	Malex. gen!J	136
18	Obfuscator. AD	142
19	Rbot!gen	158
20	Skintrim. N	80
21	Swizzor. gen!E	128
22	Swizzor. gen!I	132
23	VB. AT	408
24	Wintrim. BX	97
25	Yuner. A	800
Total		9339

Table 2. Sample distribution of IoT malware detection dataset.

No.	Family	Number of samples
1	Benign samples	11,499
2	Malware	11,499
Total		22,998

Table 3. Sample distribution of IoT malware family classification dataset.

No.	Family	Number of samples
1	Gafgyt	1848
2	Generic	1651
3	Mirai	8000
Total		11,499

The samples in the IoT malware dataset used in this paper are derived from the real captures of IoT honeypots, namely IoTPOT [40] and X-Pot [41], spanning from 2016 to 2020. This dataset is the first publicly released dataset for IoT malware [22], although it is not openly available and requires permission from the authors for its usage. The benign software samples were collected from projects on GitHub [42]. As the IoT software dataset does not provide data labels, this paper employs VirusTotal [43] to query and label the IoT software samples. The detailed process is illustrated in **Figure 10**.

4.2. Experimental Environment

The experiments were conducted on the Kaggle cloud platform using the following equipment specifications: Ubuntu 20.04.4 system, Intel(R) Xeon(R) CPU @ 2.20 GHz, and NVIDIA Tesla P100 PCIe 16GB. The programming language used was Python 3.7, and the neural network framework PyTorch was utilized for creating and training the neural networks.

Regarding the key parameters of the neural network, the settings were as follows: epochs = 50, batch size = 128, optimizer = Adam, and learning rate = 0.001. Additionally, the values of a = 2, b = 1, G = 3, and T = 2 were used, and Q = 0.5 represented the weights assigned to different teacher models in the knowledge distillation process based on their accuracy. The three datasets were split into training and testing sets in a 7:3 ratio.

4.3. Evaluation Indicators

This study evaluates the performance of the models using widely employed metrics such as accuracy, precision, recall, and F1 score. Additionally, to better align with the IoT application environment, parameters like the number of parameters, model size, training time per epoch, and inference time per image are introduced as evaluation metrics. Model size encompasses not only the number of parameters but also includes information about the network architecture, optimizer details, and other relevant factors. These metrics provide a comprehensive assessment of the model's efficiency and suitability for IoT applications.

4.4. Experimental Results

In the detection process, the IoT software is first transformed into grayscale

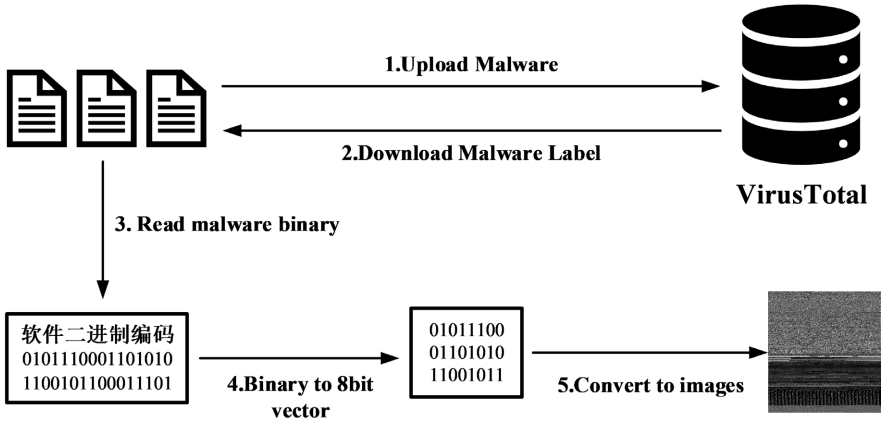

Figure 10. Acquisition of sample grayscale image.

images. ResNet-34 is then trained on this dataset, and gradient-weighted class activation mapping is used to generate square heatmaps of size 7 × 7, representing the key locations influencing the model's decision. The heatmaps are analyzed to identify the "important" positions where the heatmap values are greater than 0.8. These positions are then mapped back to the corresponding regions in the original software structure.

Through statistical analysis, it is determined that the grayscale images of the software contribute significantly to the detection task, ranging from 2.04% to 28.57% and from 87.75% to 97.95%. Therefore, in the next experiments, only the portions of the binary code are converted into gray-scale images.

Table 4 shows the experimental results using LMDNet on the initial and optimized datasets for IoT malware detection. It can be observed that LMDNet achieves a 0.61% improvement in accuracy and a 23.68% increase in recognition speed, demonstrating the effectiveness of the proposed method.

In the task of malicious software family classification, the use of the multi-teacher knowledge distillation method effectively improves the accuracy of lightweight models in the classification task. To better select the teacher models, we carefully observed the specific classification performance of each model and plotted the distribution of classification accuracies for each model on the malimg dataset. The results are shown in Figure 11.

From Figure 11, it can be observed that VGG-16 and ResNet-34 have relatively high accuracies, and each of them excels in different categories. This is beneficial for the student model to learn from their respective strengths. In the C2LOP.P, Swizzor.gen!E, and Swizzor.gen!I malware families, ResNet-34 achieves higher accuracies, while VGG-16 performs better in the C2LOP.gen!g family. On the other hand, AlexNet and LMDNet have relatively lower accuracies, with both models achieving 0 accuracy in the classification task of the Autorun.K family. Considering all these factors, this paper selects VGG-16 and ResNet-34 as the teacher models, as they are well-performing convolutional neural network models with promising performance.

Table 4. IoT malware detection dataset test results.

Model	Acc	Pre	Rec	F1	Params	Model size	Time of one epoch	Time of inference	Dataset
LMDNet	0.9831	0.9835	0.9831	0.9831	0.0852M	0.39M	69.09s	0.0038s	initial
LMDNet	0.9892	0.9894	0.9893	0.9893	0.0852M	0.39M	54.13s	0.0029s	upgrade

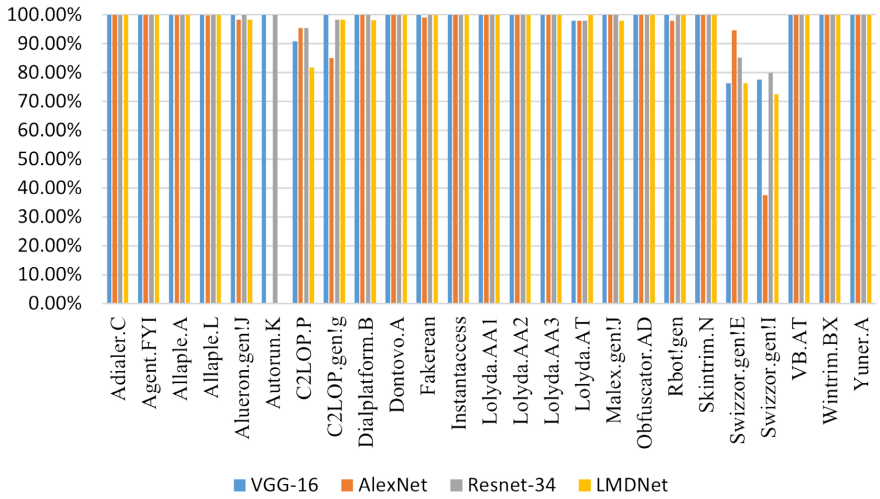

Figure 11. Accuracy of each model for each category in the malimg dataset.

The experimental results for malicious software family classification are shown in Table 5 and Table 6. In these tables, KD-LMDNet represents the student model after multi-teacher knowledge distillation. Although the student model in this paper is small, its initial accuracy is not low. After undergoing multi-teacher knowledge distillation, the accuracy in "weak" classification categories shows a significant improvement, leading to an overall improvement in the model's accuracy.

As shown in Figure 12, in the classification task of the Autorun.K family, the accuracy of the student model LMDNet is 0. However, by learning from the two teachers, KD-LMDNet achieves the same level of accuracy as the two teachers, reaching 100% accuracy in this category.

In addition, to validate the effectiveness of multi-teacher knowledge distillation, we compared the accuracy of the multi-teacher knowledge distillation method with the accuracy obtained using single-teacher knowledge distillation methods. The experimental results using single teachers, VGG, and ResNet-34, are shown in Figure 13 and Figure 14. In the figures, T-VGG represents the student model distilled using only VGG-16 as the teacher, and T-RES represents the student model distilled using only ResNet-34 as the teacher. In the malimg dataset, many classes are relatively easy for the student model, and the student model achieves a perfect score before the "apprenticeship learning" process. To make the experimental results clearer, we selected only 8 difficult LMDNet classification families, such as Alueron.gen!J, for demonstration.

Table 5. Malimg dataset family classification test results.

Model	Acc	Pre	Rec	F1	Params	Model size	Time of one epoch	Time of inference
ResNet-34	0.9935	0.9850	0.9847	0.9849	21.2975M	85.34M	54.64s	0.0066s
VGG-16	0.9910	0.9767	0.9760	0.9762	134.3630M	537.47M	86.48s	0.0071s
LMDNet	0.9804	0.9361	0.9397	0.9375	0.0963M	0.45M	36.46s	0.0051s
KD-LMDNet	0.9925	0.9814	0.9801	0.9799	0.0963M	0.45M	54.81s	0.0051s

Table 6. IoT malware family classification dataset test results.

Model	Acc	Pre	Rec	F1	Params	Model size	Time of one epoch	Time of inference
ResNet-34	0.9666	0.9566	0.9428	0.9495	21.2862M	85.30M	49.03s	0.0443s
VGG-16	0.9797	0.9756	0.9609	0.9681	134.2728M	537.11M	89.75s	0.0059s
LMDNet	0.9579	0.9396	0.9333	0.9363	0.0867M	0.40M	35.39s	0.0037s
KD-LMDNet	0.9620	0.9421	0.9439	0.9429	0.0867M	0.40M	58.72s	0.0037s

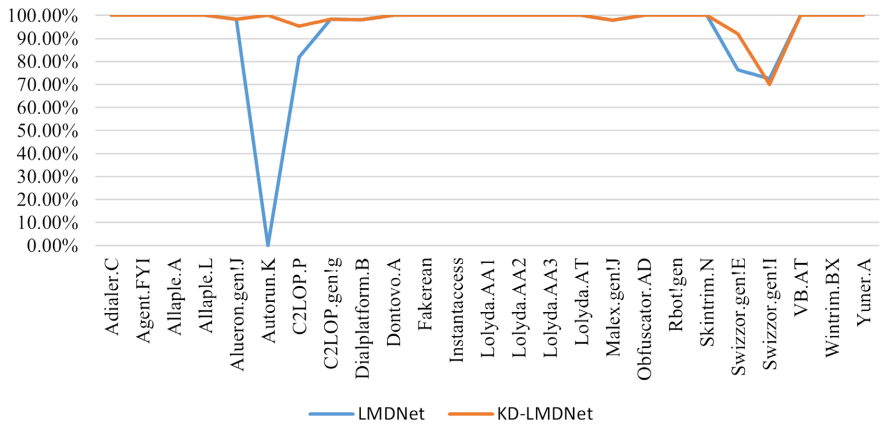

Figure 12. Accuracy improvement of KD-LMDNet in malimg.

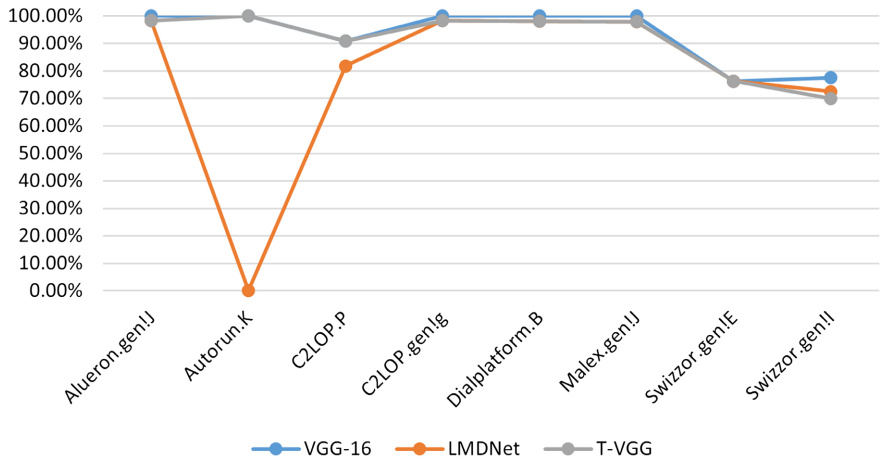

Figure 13. Single-teacher VGG knowledge distillation.

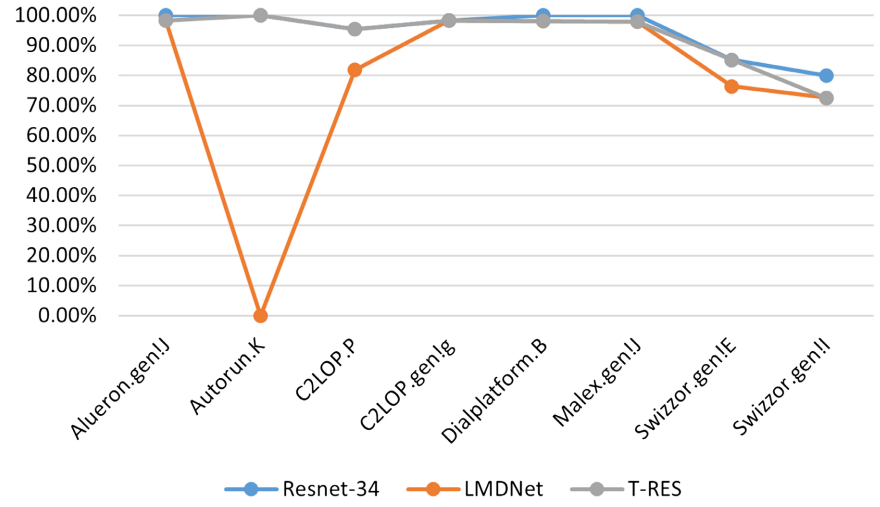

Figure 14. Single-teacher ResNet-34 knowledge distillation.

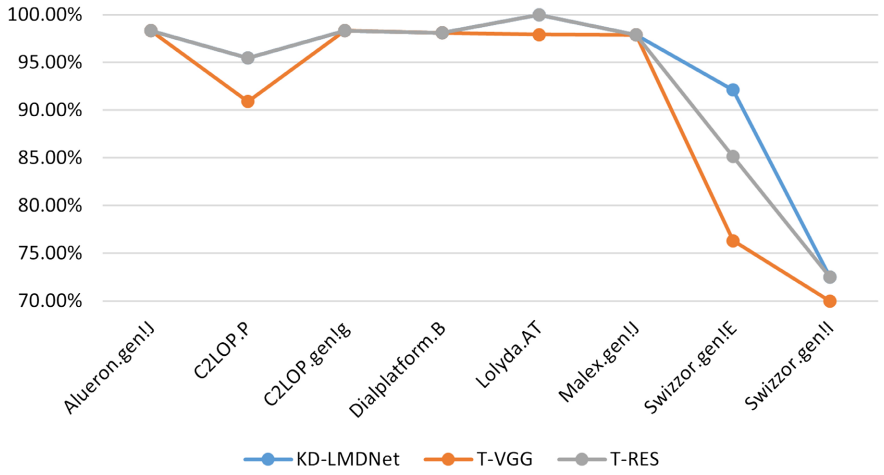

Figure 15. Comparison of accuracy between multi-teacher knowledge distillation and single-teacher knowledge distillation.

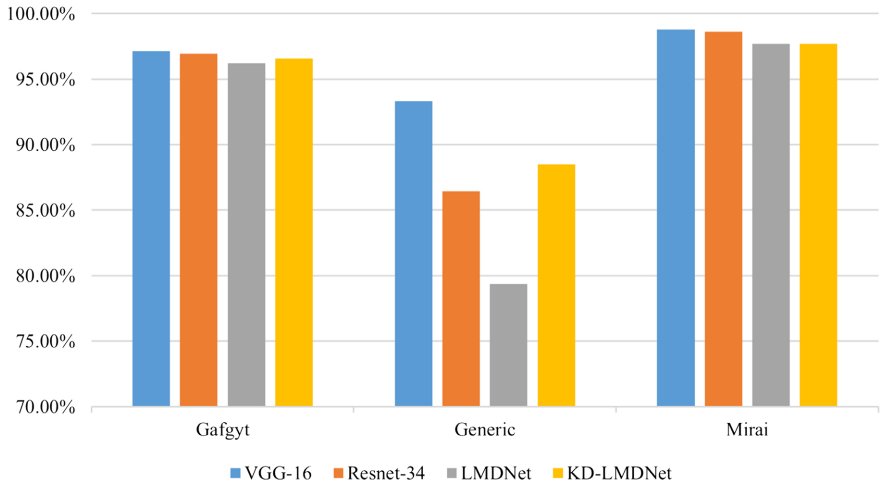

Figure 16. Accuracy of each model for each category in the IoT malware family classification dataset.

Table 7. Comparison of the detection results on the Malimg dataset.

Method	Acc	Pre	Rec	F1
DEAM + DenseNet [12]	0.985	0.969	0.966	0.967
IMCFN [44]	0.9827	0.9825	0.9819	0.9820
[45]	0.9450	0.9460	0.9450	0.9455
VGG19 [46]	0.9872	0.9761	0.9875	0.9661
Ours	0.9907	0.9749	0.9752	0.9749

The results of the comparison of the accuracy of single-teacher knowledge distillation and multi-teacher knowledge distillation are shown in **Figure 15**. The performance of the student model KD-LMDNet trained by multi-teacher knowledge distillation in each family classification is not lower than T-VGG and T-RES has the highest accuracy.

In the task of malicious software family classification in the IoT domain, KD-LMDNet, the student model after knowledge distillation, shows a significant improvement in accuracy compared to LMDNet in the Generic family classification category. The results are shown in **Figure 16**.

4.5. Comparison with the State-of-the-Art Malware Classification Methods

To validate the effectiveness of the proposed method, it was compared with traditional machine learning methods and deep learning methods based on malicious software images. The results are shown in **Table 7**.

From **Table 7**, it can be observed that the proposed method achieves higher accuracy compared to existing literature, and the model has fewer parameters and a smaller size. Traditional machine learning methods mostly rely on extracting features based on texture similarity, which requires significant computational resources to extract complex texture features from malicious software images, making it less efficient. Deep learning methods based on malicious software images often require deploying large neural networks, which are challenging to apply in the resource-constrained IoT domain with limited computational and storage resources. In contrast, the proposed model is simple and efficient. Through the use of multi-teacher knowledge distillation, the accuracy is further improved, achieving high accuracy with a small number of trainable parameters.

5. Conclusions

This paper introduces a lightweight malware detection and family classification method for IoT based on visual explanations. The proposed method offers several key advantages, including a small model size, high accuracy, fast recognition, and compatibility with multiple platforms. In malware detection, a gradient-weighted class activation mapping technique is employed to identify the crucial locations recognized by the convolutional neural network. Only these key

locations are considered for detection, thereby improving the efficiency of the model. The design of LMDNet incorporates lightweight convolutional neural network design techniques and incorporates the ECA module for simplicity and efficiency. In family classification, the student models are trained using multi-teacher knowledge distillation, resulting in improved accuracy rates. In comparison to methods relying on static low-level features, the models proposed in this paper demonstrate higher accuracy and smaller size, making them suitable for the IoT environment.

Although the model presented in this paper performs well in known malware recognition, it suffers from overfitting when faced with unknown malware. In future work, we will consider combining the model proposed in this paper with few-shot learning techniques to build a more robust detection system that achieves accurate detection of unknown malware samples. This will help address the increasingly complex and evolving malware threats in the IoT domain.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Gaikwad, N.B., Khare, S.K., Satpute, N., *et al.* (2022) Hardware Implementation of High-Performance Classifiers for Edge Gateway of Smart Automobile. 2022 1*st International Conference on the Paradigm Shifts in Communication, Embedded Systems, Machine Learning and Signal Processing* (*PCEMS*), Nagpur, 6-7 May 2022, 74-77. https://doi.org/10.1109/PCEMS55161.2022.9808049

[2] Casillo, M., Colace, F., Gupta, B.B., *et al.* (2022) A Situation Awareness Approach for Smart Home Management. 2021 *International Seminar on Machine Learning, Optimization, and Data Science* (*ISMODE*), Jakarta, 29-30 January 2022, 260-265. https://doi.org/10.1109/ISMODE53584.2022.9742901

[3] Rokade, A. and Singh, M. (2021) Analysis of Precise Green House Management System Using Machine Learning Based Internet of Things (IoT) for Smart Farming. 2021 2*nd International Conference on Smart Electronics and Communication* (*ICOSEC*), Trichy, 7-9 October 2021, 21-28. https://doi.org/10.1109/ICOSEC51865.2021.9591962

[4] Subrahmannian, A. and Behera, S.K. (2022) Chipless RFID Sensors for IoT-Based Healthcare Applications: A Review of State of the Art. *IEEE Transactions on Instrumentation and Measurement*, **71**, 1-20. https://doi.org/10.1109/TIM.2022.3180422

[5] HaddadPajouh, H., Dehghantanha, A., Parizi, R.M., *et al.* (2021) A Survey on Internet of Things Security: Requirements, Challenges, and Solutions. *Internet of Things*, **14**, Article 100129. https://doi.org/10.1016/j.iot.2019.100129

[6] Greenberg, A. (2020) This Bluetooth Attack Can Steal a Tesla Model X in Minutes. https://www.wired.com/story/tesla-model-x-hack-bluetooth

[7] Yousuf, M.J., Kanwal, N., Ansari, M.S., *et al.* (2022) Deep Learning Based Human Detection in Privacy-Preserved Surveillance Videos. 35*th International BCS Human-Computer Interaction Conference* (*HCI*2022), Keele, 11-13 July 2022, 1-7.

https://doi.org/10.14236/ewic/HCI2022.33

[8] Sonicwall, J. (2023) 2023 SonicWall Cyber Threat Report. SonicWall, Milpitas.

[9] Liang, G., Bai, L., Pang, J., *et al.* (2021) A Malware Detection Method Based on Hybrid Learning. *Acta Electronica Sinica*, **49**, 286-291.

[10] Ngo, Q.D., Nguyen, H.T., Le, V.H., *et al.* (2020) A Survey of IoT Malware and Detection Methods Based on Static Features. *ICT Express*, **6**, 280-286. https://doi.org/10.1016/j.icte.2020.04.005

[11] Madan, S., Sofat, S. and Bansal, D. (2022) Tools and Techniques for Collection and Analysis of Internet-of-Things Malware: A Systematic State-of-Art Review. *Journal of King Saud University- Computer and Information Sciences*, **34**, 9867-9888. https://doi.org/10.1016/j.jksuci.2021.12.016

[12] Wang, C., Zhao, Z., Wang, F., *et al.* (2021) A Novel Malware Detection and Family Classification Scheme for IoT Based on DEAM and DenseNet. *Security and Communication Networks*, **2021**, Article ID: 6658842. https://doi.org/10.1155/2021/6658842

[13] Aslan, Ö. and Yilmaz, A.A. (2021) A New Malware Classification Framework Based on Deep Learning Algorithms. *IEEE Access*, **9**, 87936-87951. https://doi.org/10.1109/ACCESS.2021.3089586

[14] Nisa, M., Shah, J.H., Kanwal, S., *et al.* (2020) Hybrid Malware Classification Method Using Segmentation-Based Fractal Texture Analysis and Deep Convolution Neural Network Features. *Applied Sciences*, **10**, Article 4966. https://doi.org/10.3390/app10144966

[15] HaddadPajouh, H., Dehghantanha, A., Khayami, R., *et al.* (2018) A Deep Recurrent Neural Network Based Approach for Internet of Things Malware Threat Hunting. *Future Generation Computer Systems*, **85**, 88-96. https://doi.org/10.1016/j.future.2018.03.007

[16] Darabian, H., Dehghantanha, A., Hashemi, S., *et al.* (2020) An Opcode-Based Technique for Polymorphic Internet of Things Malware Detection. *Concurrency and Computation: Practice and Experience*, **32**, e5173. https://doi.org/10.1002/cpe.5173

[17] Dovom, E.M., Azmoodeh, A., Dehghantanha, A., *et al.* (2019) Fuzzy Pattern Tree for Edge Malware Detection and Categorization in IoT. *Journal of Systems Architecture*, **97**, 1-7. https://doi.org/10.1016/j.sysarc.2019.01.017

[18] Shahzad, F. and Farooq, M. (2012) ELF-Miner: Using Structural Knowledge and Data Mining Methods to Detect New (Linux) Malicious Executables. *Knowledge and Information Systems*, **30**, 589-612. https://link.springer.com/article/10.1007/s10115-011-0393-5 https://doi.org/10.1007/s10115-011-0393-5

[19] Bai, J., Yang, Y., Mu, S., *et al.* (2013) Malware Detection through Mining Symbol Table of Linux Executables. *Information Technology Journal*, **12**, 380-384. https://doi.org/10.3923/itj.2013.380.384

[20] Nataraj, L., Karthikeyan, S., Jacob, G., *et al.* (2011) Malware Images: Visualization and Automatic Classification. *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, Pittsburgh, 20 July 2011, 1-7. https://doi.org/10.1145/2016904.2016908

[21] Su, J., Vasconcellos, D.V., Prasad, S., *et al.* (2018) Lightweight Classification of IoT Malware Based on Image Recognition. 2018 *IEEE 42nd Annual Computer Software and Applications Conference* (*COMPSAC*), Tokyo, 23-27 July 2018, 664-669. https://doi.org/10.1109/COMPSAC.2018.10315

[22] Karanja, E.M., Masupe, S. and Jeffrey, M.G. (2020) Analysis of Internet of Things Malware Using Image Texture Features and Machine Learning Techniques. *Internet of Things*, **9**, Article 100153. https://doi.org/10.1016/j.iot.2019.100153

[23] Yuan, B., Wang, J., Wu, P., *et al.* (2021) IoT Malware Classification Based on Lightweight Convolutional Neural Networks. *IEEE Internet of Things Journal*, **9**, 3770-3783. https://doi.org/10.1109/JIOT.2021.3100063

[24] Chen, C.Y. and Hsiao, S.W. (2019) IoT Malware Dynamic Analysis Profiling System and Family Behavior Analysis. 2019 *IEEE International Conference on Big Data* (*Big Data*), Los Angeles, 9-12 December 2019, 6013-6015. https://doi.org/10.1109/BigData47090.2019.9005981

[25] Jeon, J., Park, J.H. and Jeong, Y.S. (2020) Dynamic Analysis for IoT Malware Detection with Convolution Neural Network Model. *IEEE Access*, **8**, 96899-96911. https://doi.org/10.1109/ACCESS.2020.2995887

[26] Monnappa, K. (2015) Automating Linux Malware Analysis Using Limon Sandbox. Black Hat Europe. https://www.blackhat.com/docs/asia-16/materials/arsenal/asia-16-KA-Limon-wp.pdf

[27] HaboMalHunter: Habo Malware Analysis System. https://habo.qq.com

[28] Falcon Sandbox: Automated Malware Analysis Tool. https://www.crowdstrike.com

[29] Selvaraju, R.R., Cogswell, M., Das, A., *et al.* (2017) Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. *Proceedings of the* 2017 *IEEE International Conference on Computer Vision*, Venice, 22-29 October 2017, 618-626. https://doi.org/10.1109/ICCV.2017.74

[30] Zeiler, M.D. and Fergus, R. (2014) Visualizing and Understanding Convolutional Networks. *Computer Vision—ECCV* 2014: 13*th European Conference*, Zurich, 6-12 September 2014, 818-833. https://doi.org/10.1007/978-3-319-10590-1_53

[31] LeCun, Y., Bottou, L., Bengio, Y., *et al.* (1998) Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, **86**, 2278-2324. https://doi.org/10.1109/5.726791

[32] Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2017) Imagenet Classification with Deep Convolutional Neural Networks. *Communications of the ACM*, **60**, 84-90. https://doi.org/10.1145/3065386

[33] Wang, Q., Wu, B., Zhu, P., *et al.* (2020) ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks. 2020 *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), Seattle, 13-19 June 2020, 11534-11542. https://doi.org/10.1109/CVPR42600.2020.01155

[34] Hu, J., Shen, L. and Sun, G. (2018) Squeeze-and-Excitation Networks. 2018 *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, 18-23 June 2018, 7132-7141. https://doi.org/10.1109/CVPR.2018.00745

[35] Kalash, M., Rochan, M., Mohammed, N., *et al.* (2018) Malware Classification with Deep Convolutional Neural Networks. 2018 9*th IFIP International Conference on New Technologies, Mobility and Security* (*NTMS*), Paris, 26-28 February 2018, 1-5. https://doi.org/10.1109/NTMS.2018.8328749

[36] Sharma, H., Jain, J.S., Bansal, P., *et al.* (2020) Feature Extraction and Classification of Chest X-Ray Images Using CNN to Detect Pneumonia. 2020 10*th International Conference on Cloud Computing, Data Science & Engineering* (*Confluence*), Noida, 29-31 January 2020, 227-231. https://doi.org/10.1109/Confluence47617.2020.9057809

[37] You, S., Xu, C., Xu, C., *et al.* (2017) Learning from Multiple Teacher Networks.

Proceedings of the 23*rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, 13-17 August 2017, 1285-1294. https://doi.org/10.1145/3097983.3098135

[38] Fukuda, T., Suzuki, M., Kurata, G., *et al.* (2017) Efficient Knowledge Distillation from an Ensemble of Teachers. *Proceedings of the* 18*th Annual Conference of the International Speech Communication Association*, Stockholm, 20-24 August 2017, 3697-3701. https://doi.org/10.21437/interspeech.2017-614

[39] Wu, M.C., Chiu, C.T. and Wu, K.H. (2019) Multi-Teacher Knowledge Distillation for Compressed Video Action Recognition on Deep Neural Networks. 2019 *IEEE International Conference on Acoustics*, *Speech and Signal Processing* (*ICASSP*), Brighton, 12-17 May 2019, 2202-2206. https://doi.org/10.1109/ICASSP.2019.8682450

[40] Pa, Y.M.P., Suzuki, S., Yoshioka, K., *et al.* (2015) IoTPOT: Analysing the Rise of IoT Compromises. 9*th USENIX Workshop on Offensive Technologies* (*WOOT* 15), Washington, D.C., 10-11 August 2015, 1-9. https://www.usenix.org/conference/woot15/workshop-program/presentation/pa

[41] Kato, S., Tanabe, R., Yoshioka, K., *et al.* (2021) Adaptive Observation of Emerging Cyber Attacks Targeting Various IoT Devices. 2021 *IFIP/IEEE International Symposium on Integrated Network Management* (*IM*), Bordeaux, 17-21 May 2021, 143-151. https://ieeexplore.ieee.org/abstract/document/9464004

[42] https://github.com/azmoodeh/

[43] https://www.virustotal.com/

[44] Vasan, D., Alazab, M., Wassan, S., *et al.* (2020) IMCFN: Image-Based Malware Classification Using Fine-Tuned Convolutional Neural Network Architecture. *Computer Networks*, **171**, Article 107138. https://doi.org/10.1016/j.comnet.2020.107138

[45] Cui, Z., Xue, F., Cai, X., *et al.* (2018) Detection of Malicious Code Variants Based on Deep Learning. *IEEE Transactions on Industrial Informatics*, **14**, 3187-3196. https://doi.org/10.1109/TII.2018.2822680

[46] Çayir, A., Ünal, U. and Dağ, H. (2021) Random CapsNet Forest Model for Imbalanced Malware Type Classification Task. *Computers & Security*, **102**, Article 102133. https://doi.org/10.1016/j.cose.2020.102133