electronics

MDPI

Article

AGProto: Adaptive Graph ProtoNet towards Sample Adaption for Few-Shot Malware Classification

Junbo Wang [1], Tongcan Lin [2], Huyu Wu [2] and Peng Wang [3,*]

[1] College of Software Engineering, Sichuan University, Chengdu 610017, China; wangjunbo@stu.scu.edu.cn
[2] College of Computer Science, Sichuan University, Chengdu 610017, China; lintongcan@stu.scu.edu.cn (T.L.); wuhuyu@stu.scu.edu.cn (H.W.)
[3] School of Cyber Science and Engineering, Sichuan University, Chengdu 610207, China
* Correspondence: wpen@scu.edu.cn

Abstract: Traditional malware-classification methods reliant on large pre-labeled datasets falter when encountering new or evolving malware types, particularly when only a few samples are available. And most current models utilize a fixed architecture; however, the characteristics of the various types of malware differ significantly. This discrepancy results in notably inferior classification performance for certain categories or samples with uncommon features, but the threats of these malware samples are of equivalent significance. In this paper, we introduce Adaptive Graph ProtoNet (AGProto), a novel approach for classifying malware in the field of Few-Shot Learning. AGProto leverages Graph Neural Networks (GNNs) to propagate sample features and generate multiple prototypes. It employs an attention mechanism to calculate the relevance of each prototype to individual samples, resulting in a customized prototype for each case. Our approach achieved optimal performance on two few-shot malware classification datasets, surpassing other competitive models with an accuracy improvement of over 2%. In extremely challenging scenarios—specifically, 20-class classification tasks with only five samples per class—our method notably excelled, achieving over 70% accuracy, significantly outperforming existing advanced techniques.

Keywords: few-shot learning; malware classification; graph neural network; sample adaptation; prototype

Citation: Wang, J.; Lin, T.; Wu, H.; Wang, P. AGProto: Adaptive Graph ProtoNet towards Sample Adaption for Few-Shot Malware Classification. *Electronics* **2024**, *13*, 935. https://doi.org/10.3390/electronics13050935

Academic Editor: Stefanos Kollias

Received: 8 January 2024
Revised: 20 February 2024
Accepted: 26 February 2024
Published: 29 February 2024

1. Introduction

In the domain of cybersecurity, the persistent and pervasive impact of malware remains a paramount concern. This category of malicious software, which includes viruses, worms, trojans, backdoors, and rootkits, is designed to compromise or exploit programmable devices, networks, and services. According to Kaspersky's 2023 annual report [1], cybercriminals released an average of 411,000 malicious files daily in 2023, marking a 3% increase from the preceding year. The escalating complexity and sophistication of these threats poses an alarming challenge. Traditional methodologies for malware classification, while moderately effective, are increasingly proving inadequate against the rapid proliferation and evolution of new malware variants, each characterized by unique mechanisms and impacts.

In malware classification, there are two main types [2]: binary classification, which determines if software is malicious or benign (often called malware detection), and multi-classification, which sorts a malware sample into a specific family. This field primarily uses two approaches to analyze malware: utilizing static features and dynamic features. Static features—which represent the malware's binary sequence, often transformed into grayscale images—provide a direct representation of its inherent structure. In contrast, dynamic features, derived from API call sequences through executing binary files in a sandbox, reflect the malware's behavior within a controlled environment. This paper concentrated on the

static-feature approach, employing grayscale images for their simplicity in acquisition and compatibility with advanced computer-vision techniques.

Machine-learning techniques have been the cornerstone of malware classification, including methods like Support Vector Machines (SVM), decision trees [3,4], and Shared Nearest Neighbor (SNN) [5]. These methods have demonstrated moderate success, yet they often fall short in achieving high accuracy levels [6]. Subsequently, the advent of deep-learning techniques, including Convolutional Neural Networks (CNN) [7] and Recurrent Neural Networks (RNN) [8], has signified a substantial progression in malware classification. These models have proven to be highly accurate and effective in identifying and classifying malicious software.

However, despite their successes, deep-learning methods encounter substantial challenges. Firstly, the rapid iteration and emergence of new malware variants demands a swift response that traditional deep-learning techniques struggle to provide. Secondly, the attainment of high accuracy in deep-learning methods relies heavily on the availability of large volumes of high-quality labeled data. In the context of malware, acquiring such labeled data is particularly challenging; it is a task traditionally reserved for experts and is both time-consuming and costly. To surmount these challenges, this paper adopted meta learning and Few-Shot Learning (FSL) [9] methodologies. FSL methods address these challenges by leveraging the ability to learn from a limited number of examples. Considering the unique characteristics of malware, such as its rapid evolution and the scarcity of labeled data, alongside the strengths and limitations of deep-learning models, FSL methods emerge as a promising solution. They adapt to new malware variants swiftly with minimal examples and mitigate the reliance on extensive labeled datasets. Thus, integrating FSL into malware-classification systems could significantly enhance their adaptability, efficiency, and overall performance in the ever-evolving landscape of cybersecurity threats.

Nonetheless, existing Few-Shot Learning methods have inherent limitations. The architecture of most current few-shot models is universally applied across all malware categories and instances, encompassing even those that are yet to be encountered. This uniformity may lead to suboptimal performance, particularly when dealing with categories or instances characterized by rare features. This is attributable to the model's propensity to make trade-offs during the training phase; in pursuit of enhancing overall accuracy, the model might neglect instances with rare features. However, within the realm of cybersecurity, the threat posed by these malware samples is of equivalent significance. Consequently, there is a pressing need for a more robust and dynamically adaptive Few-Shot Learning model for malware classification that can effectively handle diverse and rare samples.

In this paper, we introduce an innovative approach named **A**daptive **G**raph **Proto**Net (**AGProto**), to address the challenges and enhance the performance of malware classification in few-shot scenarios. Unlike traditional Few-Shot Learning methods, such as prototype networks, which typically generate a fixed prototype for each class and classify based on the distance between query samples and those prototypes, our method employs the message-passing mechanism of Graph Neural Networks, allowing the prototypes of each class to adapt dynamically according to the features of the input samples. Moreover, to further enhance the model's robustness and flexibility, we introduce multiple layers of Graph Neural Network, where each layer dynamically generates a set of prototypes for each query sample. After generating multiple prototypes for each sample, we use an attention mechanism to assign different weights to these prototypes, based on the features of the samples. Thus, the final classification decision is based on a set of weighted, dynamically adjusted prototypes, rather than a single static one. This approach enhances the model's adaptability to new samples and the accuracy of classification. In the optimization of the model, to ensure the consistency of predictions generated by different prototypes for query samples, we have incorporated the cosine similarity of each prototype's prediction results into the loss function. Once the model is trained, although its parameters are fixed, it can dynamically adjust the prototypes, based on the features of each new input sample, thereby achieving more precise and robust classification.

To summarize, we make the following contributions:

- **Adaptive Prototyping via Graph Neural Networks**: We introduce a novel approach where class prototypes dynamically adapt through Graph Neural Networks, enhancing representations and robustness by aligning prototypes closely with the features of input samples.
- **Consistency Loss**: We innovatively integrate the cosine similarity of predictions from various prototypes into the loss function, ensuring consistent predictions across different prototypes for a given query sample, thereby improving the model's overall stability and accuracy.
- **Attention-Based Dynamic ProtoNet**: We propose an innovative mechanism that generates multiple weighted prototypes per sample, utilizing an attention framework to dynamically adjust the influence of each prototype based on the sample's unique characteristics, thereby enhancing the model's precision and adaptability.

2. Related Work

2.1. Malware Image

Nataraj et al. [10] initially proposed the innovative concept of converting raw-byte malware PE files into grayscale images, setting a foundational approach for future research in the field. This methodology was further advanced by Vasan et al. [11], who employed an ensemble of CNN architectures to enhance the classification of malware images, demonstrating the effectiveness of deep learning in this domain. Kancherla and Mukkamala [12] expanded upon these techniques by proposing an image-based malware-detection mechanism using Gabor-based features, which contributed to the refinement of feature-extraction methods for malware images. Venkatraman et al. [13] introduced a hybrid deep-learning-image-based analysis for effective malware detection, showcasing the potential of combining various deep-learning techniques for improved accuracy and robustness. Additionally, Vasan et al. [14] further explored the realm of deep learning with their work on fine-tuned Convolutional-Neural-Network architectures, which emphasized the adaptability and precision of CNNs in classifying malware images. Recently, Cui et al. [15] proposed a malicious-code-detection method under 5G HetNets based on a multi-objective RBM model. Their work compared images with different resolutions based on learning-rate values selected by the CLR strategy, contributing to the understanding of how image resolution impacts malware detection.

All these studies collectively represent a trajectory of increasing sophistication and effectiveness in the field of malware-image classification, moving from basic image-conversion techniques to advanced deep-learning models that offer high accuracy and robustness against various types of malware. Collective success also underscores the efficacy of utilizing grayscale images for malware categorization, as demonstrated in Figure 1, which reveals that grayscale images of malware from the same category display similar characteristics.

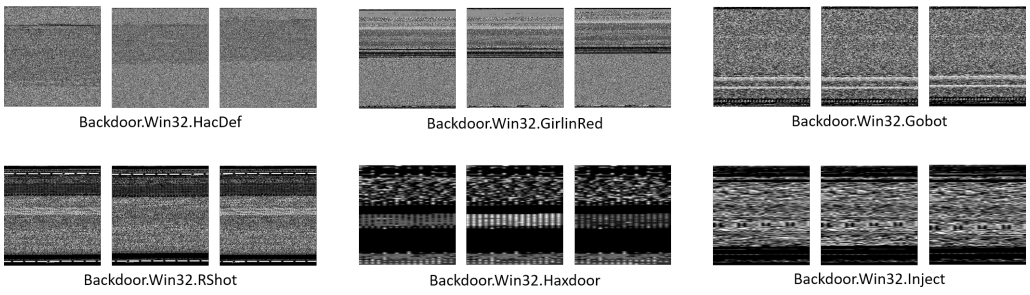

Figure 1. Convert raw-byte malware PE files to gray images, using L. Nataraj's method.

2.2. Few-Shot Learning

Few-Shot Learning (FSL) [9] is a technique aimed at enabling machine-learning models to rapidly learn new tasks from very limited data. In the domain of malware classification,

acquiring a large volume of labeled malware samples is often expensive and time consuming. Few-shot learning seeks to overcome this challenge. The approaches to Few-Shot Learning are primarily divided into two main categories: optimization-based methods and metric-based methods.

Optimization-based methods focus on adjusting the learning algorithm to better adapt to a small amount of data. These methods typically involve modifying the training process of the model so that it can quickly adapt to new tasks after seeing only a few samples. One of the classic examples is Model-Agnostic Meta Learning (MAML) [16], proposed by Chelsea et al. MAML aims to find a good model-parameter initialization so that the model can rapidly adapt to new tasks with a few gradient updates. Variants of MAML [17,18] have also been developed to optimize this initialization by training across multiple tasks, allowing for effective learning of new tasks with minimal steps. Recently, Kang et al. [19] introduced the Geometry-Adaptive Preconditioned gradient descent (GAP), which uses a Geometry-Adaptive Preconditioner to improve the inner-loop optimization, achieving notable results.

Metric-based methods emphasize learning a good distance or similarity metric, enabling the model to effectively classify by comparing the similarity between new samples and known samples. These methods typically involve learning an embedding space where samples from the same class are close to each other, while samples from different classes are farther apart. Representative methods include K-Nearest Neighbors (KNN) and prototypical networks [20]. KNN is a simple yet powerful classifier that classifies by measuring the distance between a test sample and training samples. Prototypical networks operate by learning a prototypical representation of each class—the centroid of all sample embeddings of that class. When classifying a new sample, they calculate the distance between the sample and each class prototype, classifying it as the nearest class. RelationNet [21] learns a distance metric through a neural network on top of a feature-embedding network. Due to the simplicity and efficiency of prototypical networks, many works [22,23] are based on this, including our proposed AGProto.

2.3. GNN in Few-Shot Learning

Initially, Garcia and Bruna [24] laid the groundwork by exploring the potential of GNNs in Few-Shot Learning, proposing a framework that would inspire future research in the field. Building on this foundation, Gidaris and Komodakis [25] introduced a meta model that employs GNN denoising auto-encoders to generate classification weights specifically for Few-Shot Learning tasks, demonstrating the adaptability of GNNs to this new domain. As the field progressed, Zhou et al. [26] developed Meta-GNN, a novel graph-meta-learning framework tailored for few-shot node classification, showcasing the versatility of GNNs in handling the sparse-data scenarios typical of Few-Shot Learning. In 2020, Wang et al. [27] introduced the AMM-GNN, focusing on attribute matching for node classification in Few-Shot Learning, further refining the approach to GNNs in this context. The year 2021 saw Tang et al. [28] propose the Mutual CRF-GNN, emphasizing the critical role of GNN in enhancing methods for Few-Shot Learning and illustrating the continuous evolution of GNN-based strategies. Most recently, Yu et al. [29] presented a hybrid GNN model, combining instance and prototype GNNs, to improve the performance and robustness of Few-Shot Learning systems.

In contrast to the majority of Few-Shot Learning GNNs that rely on label propagation, our AGProto utilizes GNNs to transfer feature information between the support set and the query set, thereby obtaining multiple prototypes customized for the query set. Subsequently, an attention mechanism is introduced to aggregate these multiple prototypes, ultimately increasing the accuracy of few-shot malware classification.

3. Method

3.1. Problem Definition

In this study, we focused on the challenge of malware classification using Few-Shot Learning (FSL). The primary difficulty in this task lay in the requirement for the model to accurately categorize new malware families based on a limited number of samples, which were not encountered during the training phase. To address this, we followed the paradigm of meta learning. Concretely, we divided the dataset into three distinct parts: the meta training set (D_{train}) for model training, the meta validation set (D_{val}) for controlling the training process, and the meta testing set (D_{test}) for evaluating the model's ability to classify new categories. The category labels in these three datasets were entirely different, ensuring that the malware classes faced by the model during testing were novel.

In each training episode, we constructed a specific N-way K-shot classification task sampled from the meta training set (D_{train}). To be specific, we randomly selected N classes from D_{train} to form the class set (C_N). From each of these N classes, we then sampled K instances to create the support set (S_N) and another Q instances to form the query set (Q_N). Thus, the support set $S_N = \{(x_i, y_i)|y_i \in C_N, i = 1, \ldots, N \times K\}$ and the query set $Q_N = \{(x_i, y_i)|y_i \in C_N, i = 1, \ldots, N \times Q\}$, with $S_N \cap Q_N = \emptyset$ to ensure no direct overlap between the training and testing samples within an episode. In each episode, the categories varied, requiring the model to classify samples in the query set (Q) based on the samples provided in the support set (S). This task was designed to simulate the scenario the model would encounter during the actual test phase with the meta test set (D_{test}), where $C_{train} \cap C_{test} = \emptyset$ to ensure no class overlap and maintain the novelty of the test classes.

Following the completion of multiple training episodes, the model was subjected to validation on the meta validation set (D_{val}). This phase enabled the evaluation of the model's efficacy on previously unencountered data, with the selection of the optimally performing model on D_{val} as the ultimate model. To guarantee the exclusivity of the validation classes, it was ensured that $C_{val} \cap (C_{train} \cup C_{test}) = \emptyset$, thereby preventing any class overlap. Subsequently, this model was examined on the meta testing set (D_{test}), to assess its proficiency in classifying novel categories.

During training, we employed a training strategy similar to that of prototype networks [20]. Initially, we constructed N-class prototypes, using the support set S_N. Subsequently, we calculated the similarity between each sample in the query set Q_N and each prototype, which served as the predicted probability for classification. By leveraging the concept of prototypes, we effectively captured the essence of each class, facilitating a more accurate and efficient classification of new and unseen malware families.

3.2. Convert Malware to Image

Building upon the work of Nataraj et al. [10], our approach specifically involved segmenting the binary machine code of malware into bytes (8 bits) and converting each byte into a decimal number (ranging from 0 to 255) to represent the value of a pixel in a grayscale image. Unlike conventional images, the grayscale images of malware that we generated contained only one channel. Drawing from the empirical findings of Nataraj et al., we determined the width of each malware image, as detailed in Table 1. To ensure minimal loss of information, we employed a padding strategy with zeros for the last row of pixels if it did not meet the required length. For efficient training of the Convolutional Neural Network (CNN) model, we resized the images to a standardized dimension of 256 $\times$ 256 pixels and normalized the pixel values.

Table 1. Image width corresponding to converted malware of different sizes.

File Size Range	Image Width
<10 kB	32
10 kB–30 kB	64
30 kB–60 kB	128
60 kB–100 kB	256
100 kB–200 kB	384
200 kB–500 kB	512
500 kB–1000 kB	768
>1000 kB	1024

3.3. Proposed Framework

In this section, we present the framework of our proposed model, AGProto, which is illustrated in Figure 2. The pseudo code of the model is delineated in Algorithm 1. AGProto is structured into four distinct components:

1. **Embedding Module**: This module is responsible for projecting input samples into an initial feature space, denoted as F_0^s for the support set and F_0^q for the query set. Specifically, for a given support-set sample S_N, the feature representation is computed as $F_0^s = f_0(S_N)$, where f_0 represents the embedding function implemented with a CNN4 architecture. The process is analogous for the query set.
2. **Adaptive Graph Modules**: These models are designed to map the samples from both the support and query sets into new feature spaces, represented as F_i^s and F_i^q, respectively. Assuming there are n Adaptive Graph Modules in the framework, the index i ranges from 1 to n.
3. **Prototype Aggregation Layer**: This layer is tasked with obtaining a prototype for each class within every feature space, based on the features of the support set. The prototypes serve as representative points for each class in the feature space.
4. **Attention-Based Dynamic Proto-Layer**: This component adaptively calculates the weights of the prototypes in each feature space based on the features of the query and then aggregates these to output the final class probabilities.

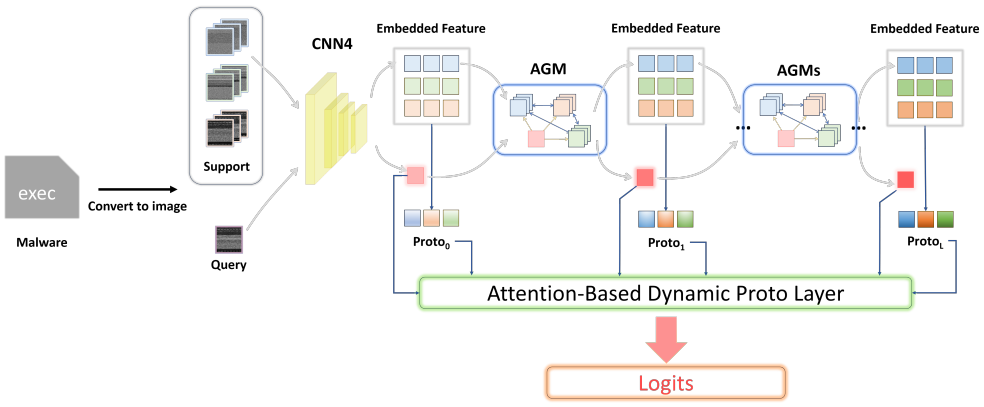

Figure 2. Overview of our AGProto framework in a 3-way-3-shot case. Features extracted by CNN are sequentially fed into several Adaptive Graph Modules and generate several prototypes and corresponding query features. Then, through an Attention-Based Dynamic Proto-Layer (ADPL), the probability of each query output category is calculated.

Within this framework, the architecture of the Adaptive Graph Module (AGM) and the Attention-Based Dynamic Proto-Layer (ADPL) are, respectively, presented in Figures 3 and 4. More details will be discussed in later sections.

Algorithm 1 AGProto for Few-Shot Malware Classification

Input: Support set with N samples S_N, Query set Q, the number of Adaptive Graph Modules (AGMs) n, embedding function facilitated by CNN4 f_0, Adaptive Graph Module (AGM), prototype aggregation layer P, and Attention-Based Dynamic Proto-Layer (ADPL).

Output: Logits representing class probabilities for each sample in the query set.

// Embedding Module
for each sample S in S_N **do**
  $F_0^s \leftarrow f_0(S)$ {Project support-set sample to initial feature space}
end for
for each query Q_i in Q **do**
  $F_0^q \leftarrow f_0(Q_i)$ {Project query set sample to initial feature space}
end for
// Adaptive Graph Modules
for $i = 1$ **to** n **do**
  $(F_i^s, F_i^q) \leftarrow \text{AGM}(F_{i-1}^s, F_{i-1}^q)$ {Adjust the features of the support set based on the information from the query set through GNN}
end for
// Prototype Aggregation Layer
for $i = 0$ **to** n **do**
  $\text{Proto}_i \leftarrow P(F_i^s)$ {Compute prototype for each class in F_i^s}
end for
// Attention-Based Dynamic Proto-Layer
for each query Q_i in Q **do**
  $\text{Logits} \leftarrow \text{ADPL}(\text{Proto}_0, \text{Proto}_1, \ldots, \text{Proto}_n, F_0^q, \ldots, F_n^q)$
  Output Logits for Q_i
end for

3.3.1. Embedding Module

For the embedding module, we opted for the commonly used CNN4 backbone network [30]. This relatively simple Convolutional-Neural-Network structure is specifically designed for few-shot image-classification tasks. While we experimented with slightly more complex architectures, such as ImageNet pre-trained ResNet18 [31], we found that the simpler four-layer convolutional structure of CNN4 yielded better results. We believe the reason is that malware grayscale images contain less information compared to natural images and, thus, a complex network structure might lead to overfitting.

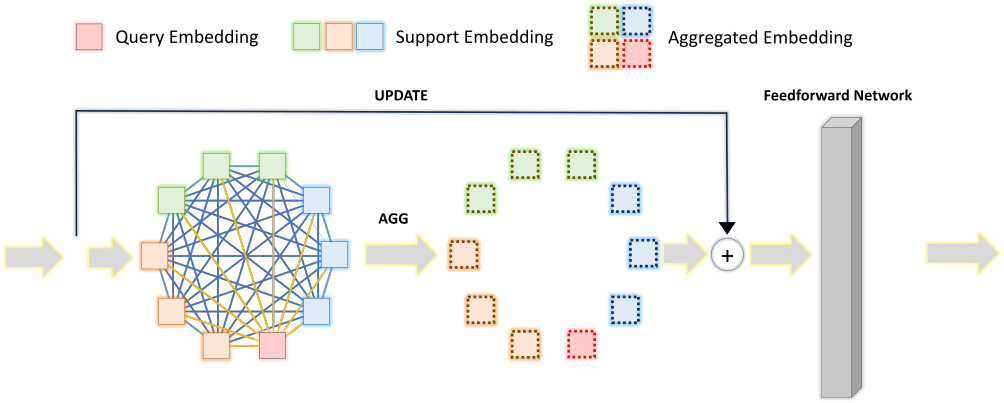

Figure 3. Illustration of Adaptive Graph Module, which intakes the features extracted by CNN and outputs the aggregated features. For the message-passing stage, the blue lines indicate that the information transfer between the support sets is bidirectional, while the yellow lines denote that the information transfer between the query set and the support set is unidirectional, occurring only from the query set to the support set.

Our implementation of CNN4 deviated slightly from the standard version [30]. We set the number of channels in the first convolutional layer to 32 instead of the standard 64. This decision was based on the input being grayscale images, which start with a single channel, as opposed to the three channels of RGB images. Gradually increasing the number of channels allowed for more effective feature extraction while reducing computational load. We also added a residual connection between the last two layers of the CNN, as empirical evidence suggests that this modification stabilizes training and accelerates model convergence. The details of this architecture are illustrated in Figure 5.

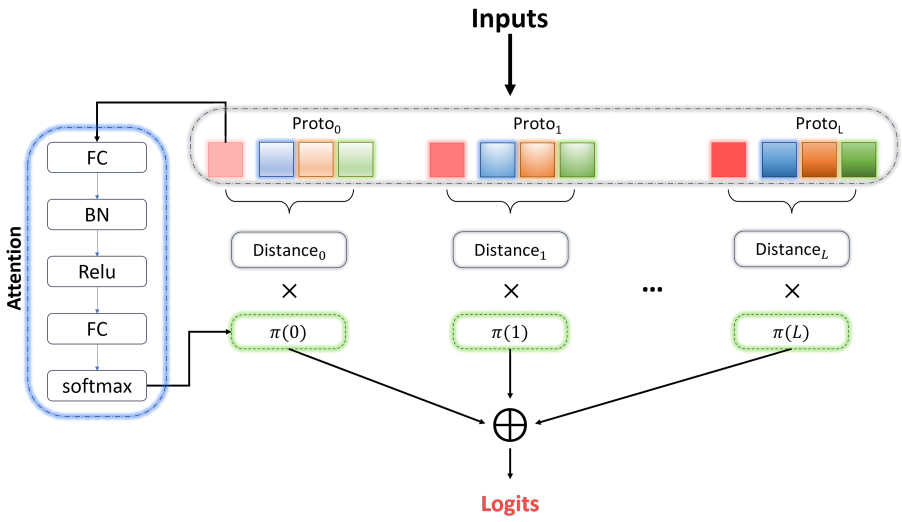

Figure 4. Illustration of Attention-Based Dynamic Proto-Layer, which intakes the prototypes and query features generated by the AGM. It applies an attention mechanism to the weighted distances and outputs the probabilities for each category in the query set.

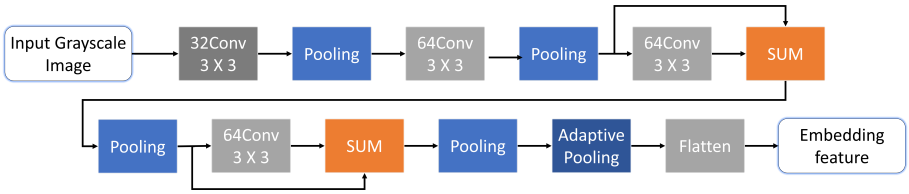

Figure 5. The CNN4-embedding-module architecture comprises sequential conv blocks, each with a convolutional layer, Batch Normalization, and ReLU activation. The first conv block has 32 channels, a stride of 2, while subsequent blocks maintain 64 channels, a stride of 1, padding of 1, and a 3×3 kernel size. Pooling layers are MaxPool2d (kernel_size = 2, stride = 2), except for the final layer, which employs an AdaptiveMaxPool with dimensions (2,4), optimizing feature extraction, and spatial dimensionality reduction.

3.3.2. Adaptive Graph Module

One of the pivotal components of our novel framework is the Adaptive Graph Module (AGM). This module intakes feature embeddings from a support set and a query set, subsequently adjusting the features of the support set based on the information from the query set. The output is the feature embeddings of the support and query sets in an alternative feature space. Essentially, our proposed AGM re-projects the features, enabling the model to calculate the distance or similarity between the query samples and various class prototypes within the channel feature space.

For simplicity, let $F_l = Concat[F_l^s, F_l^q]$, which represents the feature representation of all samples from the support and query sets after the l^{th} layer of AGM. These samples are processed through the next layer of AGM as follows:

$$F_{l+1} = G(F_l) = FFN(F_l + \sigma(F_l^{'})) \quad (1)$$

$$F_l^{'} = AGGREGATE(F_l) = \mathcal{A}F_lW \tag{2}$$

Here, $F_l^{'}$ denotes the aggregated sample information, $\mathcal{A}$ is an adjacency matrix representing the relationships between samples at that layer, W is a projection matrix, σ denotes a non-linear activation function, and FFN stands for FeedForward Network. The AGM module can be dissected into two segments: (1) **Graph Construction**: This part details how we express samples from the support and query sets as a graph; (2) **Message Passing**: We delve into the mechanism of message passing within Graph Neural Networks in the context of our problem. Specifically, to obtain adaptive prototypes, we direct the flow of information from the query-set towards the support-set samples, which allows for dynamic adjustments based on the query.

(1). Graph Construction

For simplicity, we denote $T = N \times (K + Q)$ as the total number of samples encompassing both the support and query sets.

Let $G = (V, E)$ represent a graph constituted by nodes from these sets, where V is the collection of nodes, defined as $V = \{F_l^1, F_l^2, \ldots, F_l^T\}$. Here, F_l^i symbolizes the i^{th} malware sample at the l^{th} AGM layer. For samples drawn from the support set, i ranges from 1 to $N \times K$, and for those from the query set, i ranges from $N\times$ K to T. Given the discussion pertains to samples from a singular layer of the AGM, we simplify our notation by omitting the subscript l, thus reducing it to $V = \{F^1, F^2, \ldots, F^T\}$. E is the set of edges, where for any $i, j \in [1, K + Q], e_{ij} \in E$ delineates the relationship between the i^{th} and j^{th} samples.

However, the semantic relationships between samples remain somewhat nebulous. The subsequent discussion aims to elaborate on the precise construction process of these relationships, enhancing the understanding and analytical capabilities within the graph model.

We define e_{ij} as a measure of the relationship between the i^{th} and j^{th} samples. The underlying premise is that the closer or more relevant the samples are to each other, the larger the value of e_{ij} should be. This conceptualization allows us to capture the intrinsic structure and relationship dynamics within the data. The formula for this relationship measure can be represented as follows:

$$e_{ij} = f_\theta(F^i, F^j) \tag{3}$$

Here, f_θ represents the distance function between different samples, with θ denoting the parameters that can be learned. This function can take various forms, including both parametric and non-parametric expressions. For non-parametric forms, f_θ could be: (1) **Euclidean Distance**: $f_\theta(x, y) = \sum((x - y)^2)$; (2) **Cosine Similarity**: $f_\theta(x, y) = -\frac{x \cdot y}{|x| \cdot |y|}$. For parametric forms, f_θ could be: (3) **Sum**: $f_\theta(x, y) = W^T tanh(x + y)$; (4) **Dot Product**: $f_\theta(x, y) = x^T W y$.

In our constructed sample graph, to enable a more flexible framework that could capture a wider variety of relationships and to allow the model to learn a more nuanced and accurate representation, we employed a parametric distance function f_θ. Furthermore, to maintain the symmetry of distance, ensuring $e_{ij} = e_{ji}$, we incorporated a methodology delineated in [24], which adopts a Multilayer Perceptron (MLP) stacked after the absolute difference between two vectors. This can be mathematically represented as:

$$f_\theta(x, y) = MLP_\theta(|x - y|) \tag{4}$$

Here, θ represents the learnable parameters. By using this architecture, the symmetry of the distance function $f_\theta(x, y) = f_\theta(y, x)$ is satisfied, and the distance property identity $f_\theta(a, a) = 0$ is easily learned. For illustration, we visualize the MLP architecture in Figure 6.

Having established the pairwise distances between samples, we can readily construct the adjacency matrix A, where the proximity of two samples inversely correlates with the strength of their connection. This relationship is mathematically represented as:

$$A = \{e_{ij} | 1 \leq i, j \leq T\} \tag{5}$$

Subsequently, we normalize each row of the adjacency matrix. For this process, we adopt the normalization technique proposed in the Graph Attention Network (GAT) [32], which involves applying a LeakyReLU activation function to the adjacency matrix followed by a softmax computation to determine the attention coefficients α_{ij} for the edges. The formula for this computation is as follows:

$$\alpha_{ij} = \frac{exp(LeakyReLU(e_{ij}))}{\sum_{k=1}^{T} exp(LeakyReLU(e_{ik}))} \tag{6}$$

With the graph constructed, defining its nodes as the collective set of samples from both the support and query sets, and its edges as the relationships between each pair of samples, we now proceed to the message-passing phase. Specifically, we aim to transmit the query set's information to the support set. This process allows for the acquisition of adaptive features that are refined in response to the query set's characteristics.

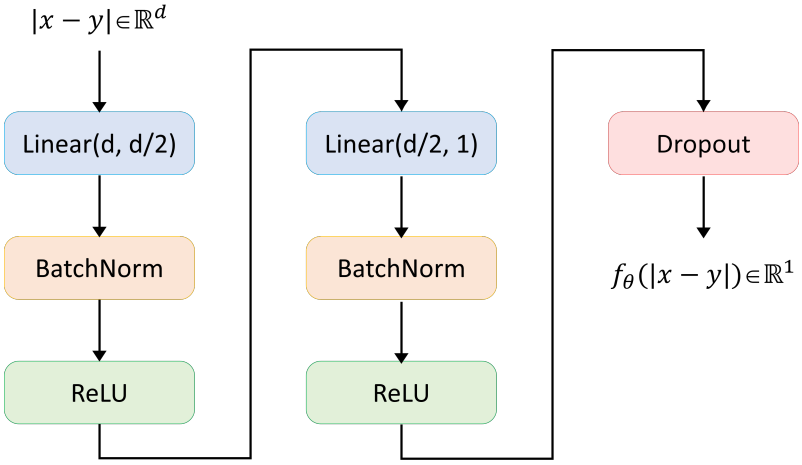

Figure 6. MLP structure of the distance function.

(2). Message Passing

Unlike traditional Graph Neural Networks, the direction of message passing in the Fully Connected graph constructed in the previous section is not bidirectional. Primarily, it is important to note that within the query set, information sharing is not feasible. This is understandable, since each sample in the query set is independent, with no inherent relations between them.

Our objective was to allow the support set to encompass the information from the query-set samples, thereby acquiring adaptive prototypes. This was inspired by [29], which maintained the invariant features of the support set and directed the flow of information towards the query set. However, our goal differed, in that we sought prototypes adapted to the query set, meaning the support set should contain information from the query set. As illustrated in Figure 3, the blue lines between samples represent mutual information transfer, while the yellow lines indicate information flow solely from the query to the support set.

To achieve the directional requirements of the aforementioned information transfer, while ensuring that the sum of each row in the normalized adjacency matrix equaled one, we introduced a mask matrix. The visualization of the mask matrix is shown in Figure 7. With a total sample count of T for the support and query sets, the mask and adjacency matrices were of dimension $T \times T$. The first $N \times K$ rows of the mask were zero, and the subsequent rows were set to $-\infty$ (to become zero after softmax calculation).

$$mask = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \\ -\infty & -\infty & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ -\infty & -\infty & \cdots & -\infty \end{bmatrix}_{T \times T} \Bigg\} N \times K$$

Figure 7. Visualization of the mask matrix.

The entire process can be articulated as:

$$\mathcal{A} = Softmax(LeakyReLU(A + mask)) \tag{7}$$

With the modified adjacency matrix in place, we then proceeded to the standard graph-convolution-aggregation and update processes. The aggregation process involved gathering the information of neighbors, first obtaining the projection of each sample $F_l W$, and then aggregating information according to the adjacency matrix, as shown in Formula (2). The update process involved merging the node's own information with the aggregated neighbor information, where we simply employed a residual connection:

$$F_l^{''} = F_l + \sigma(F_l^{'}) \tag{8}$$

We also experimented with more complex update mechanisms, such as the one used in GAT [32]:

$$F_l^{''} = MLP(Concat[F_l, \sigma(F_l^{'})]) \tag{9}$$

However, the empirical results suggested that the other update mechanisms did not offer a clear advantage. Instead, the simple residual connection model demonstrated more stable convergence.

As described in Formula (1), after updating the features of each sample, we enhanced the model's expressiveness and obtained more flexible feature-dimension representation by adding a Feed-Forward Network (FFN) following the graph-convolution operation, achieving the feature embedding of each sample in the new feature space.

3.3.3. Proto Aggregation Layer

In the preceding section, we explored the transformation of sample features F_l from the support and query sets through an AGM into F_{l+1}. Assuming our model comprised L AGM layers, in addition to the initial features extracted via a CNN4 architecture, we obtained $L + 1$ sets of sample features for both the support and query sets, denoted as $F = \{F_0, F_1, \ldots, F_L\}$. Each set, F_l, contained features from the support (F_l^s) and query (F_l^q) sets.

The proto aggregate layer focuses on computing the prototype of each class within the support-set sample features. We adhered to the most classical prototype calculation method [20], utilizing the mean of all features within the same class in the support set as the prototype for that class. The computation proceeds as follows:

Prototype Calculation: For each class c within the support set at layer l, the prototype P_l^c is computed as the mean of all feature vectors belonging to that class. Mathematically, this is represented as:

$$P_l^c = \frac{1}{N_c} \sum_{f \in F_l^s(c)} f \tag{10}$$

Here, P_l^c is the prototype for class c at layer l, N_c is the number of samples in class c, and $F_l^s(c)$ represents the set of all feature vectors in the support set at layer l that belong to class c. This formula ensures that the prototype is the centroid of the features in the class, effectively summarizing the class's overall position in the feature space.

For each set of sample features, we computed prototypes using the aforementioned method, ultimately obtaining $L + 1$ sets of prototypes, denoted as $P = \{P_0, P_1, \ldots, P_L\}$.

3.3.4. Attention-Based Dynamic Proto-Layer

In this section, we introduce an innovative **A**ttention-Based **D**ynamic **P**roto-**L**ayer (ADPL), which marries the principles of prototype networks with the dynamic-convolution methodology. Upon traversing the proto aggregate layer, we acquire $L + 1$ prototype sets denoted by P. Inspired by the dynamic-convolution concept proposed by [33], ADPL leverages an attention mechanism to adeptly distribute weights across these prototypes, contingent upon the query-set embeddings' nuances.

As shown in Figure 4, the ADPL mainly consists of three parts:

Distance Computation: For each prototype set P_i, where there are C classes of prototypes denoted as $P_i^1, P_i^2, \ldots, P_i^C$,we calculate the Euclidean distance between the query embedding F_i^q and each class's prototype. This metric, denoted as $Distance_i^q$, assesses how closely each prototype corresponds to the query instance. It forms the foundational metric for subsequent attention-weight allocation, influencing the model's decision-making process in classification or retrieval tasks. The distance for the i^{th}prototype set is computed as follows:

$$Distance_i^q = \left[\sum (F_i^q - P_i^j)^2\right]_{j=1}^{C} \tag{11}$$

Here, $Distance_i^q$ represents the Euclidean distance for the i^{th} prototype set to the query sample F_i^q; F^q is the query-embedding vector; P_i^c is the c^{th} class prototype in the i^{th} set; and C is the total number of classes.

Attention Mechanism: We introduce an attention mechanism where the input is the embedding from the first feature space (the 0^{th} space) of the query set. This mechanism comprises a Fully Connected (FC) layer succeeded by Batch Normalization (BN) and Rectified Linear Unit (ReLU) activation. A subsequent FC layer projects the dimensions to align with the number of prototypes, and a softmax function is applied, to yield the attention weights $\pi(i)$ for each prototype. To address the issue of the softmax layer's output labels approximating one-hot encoding or, in other words, to prevent the model from excessively focusing on prototypes specific to certain groups, we use a large temperature in softmax to flatten attention as follows:

$$\pi_i = \frac{\exp(o_i/\tau)}{\sum_j \exp(o_j/\tau)} \tag{12}$$

where o_i is the output of the last FC layer in the attention branch (see Figure 4). The attention weights reflect the importance or relevance of each prototype in relation to the query.

Weighted Summation: In the final step, a weighted summation is performed, where the Euclidean distances are multiplied by their corresponding attention weights. Since a smaller distance indicates that the query set sample is closer to the class, the predicted probability should be higher; therefore, the logits need to be negated. The formula for the weighted summation is expressed as:

$$Logits = -\sum_{i=0}^{L} \pi(i) \times Distance_i \tag{13}$$

The logits are the aggregated result of this process and represent the final predictive output of the model.

By integrating the attention mechanism, the ADPL allows our model to dynamically emphasize the most pertinent prototypes and provides a more nuanced and context-aware method for prototype weighting. This ensures that our model remains robust and efficient, even when faced with the challenge of learning from a limited number of samples.

3.4. Optimization

This section introduces the optimization objectives (loss functions) of our model. After processing through the Attention-Based Dynamic Proto-Layer (ADPL), the obtained logits represent a C-dimensional vector, where each dimension corresponds to the probability that the query sample belongs to a respective class. Cross-entropy is a common choice for the loss function in classification problems; thus, we adopted it as part of our loss function. Specifically, we first applied softmax to convert the logits into probabilities and then computed the cross-entropy loss. The loss for comparing the predicted class of the query-set samples with the true labels can be expressed as:

$$Prob(k) = \frac{e^{Logits[k]}}{\sum_{j=1}^{C} e^{Logits[j]}} \tag{14}$$

$$L1 = -\sum_{k=1}^{C} y_k \log(Prob(k)) \tag{15}$$

Here, $Prob(k)$ is the probability that the query sample belongs to class k, and C is the total number of classes; y_k is the true label of the sample, which is 1 for the correct class and 0 for all others. The loss is calculated across all classes, but only the term for the true class will contribute to the sum.

Additionally, to ensure consistency in the prediction scores for each query across all feature spaces, we also incorporated a consistency loss as part of our loss function. We initially experimented with KL divergence but found that the logarithmic operations involved led to instability and even to gradient explosion during model training. Therefore, we opted to measure consistency using cosine similarity between the prediction scores of models in different feature spaces. To simplify the calculations, we only computed the consistency between adjacent layers, reducing the computational burden from $\binom{2}{L+1}$ to L. Specifically, we calculated the consistency of the distances between the query-set samples and the prototypes across adjacent layers, mathematically expressed as:

$$L2 = \frac{1}{|Q|} \sum_{i=1}^{L} \sum_{q \in Q} (1 - sim(Distance_i, Distance_{i-1})) \tag{16}$$

$$sim(x, y) = \frac{x \cdot y}{|x| \cdot |y|} \tag{17}$$

Here, Q represents the set of query-set samples, and cos_sim calculates the cosine similarity, which has a range of $[-1,1]$. A higher value indicates greater similarity. To ensure that the loss was positive and decreased as similarity increased, we constructed the formula as above. Finally, our optimization objective was the sum of the two loss values:

$$L = L1 + \lambda L2 \tag{18}$$

Here, λ is a hyperparameter representing the weight of the consistency loss.

4. Results

4.1. Datasets

In our research, we evaluated our method on two few-shot malicious-software datasets, LargePE [22] and VirusShare [23], both of which contain the binary code of malware and corresponding classes. LargePE comprises 100, 58, and 50 classes for training, validation, and testing, respectively, with each category containing 20 samples. Similarly, VirusShare contains 87, 20, and 20 classes for training, validation, and testing phases, with each class also having 20 samples.

Unlike traditional datasets in the computer-vision domain, labels in malware datasets are relatively complex to obtain and often require expert knowledge. The samples in these datasets were first scanned through the professional analysis website VirusTotal [34], to obtain detection reports. These reports included many Anti-Virus (AV) labels provided by third-party companies. Subsequently, we utilized AVClass [35] to standardize the nomenclature of these third-party labels, to serve as the final class labels.

As both datasets consist solely of raw malicious-software binary codes, we employed the method mentioned in Section 3.2 to extract the grayscale images of the malware. This process resulted in obtaining images with a single channel, with dimensions of 256 $\times$ 256 pixels.

4.2. Baseline Models

Three types of baselines were chosen for comparisons:

1. Classic machine-learning methods
   Gist+KNN [36]: A classic machine-learning method for few-shot malware classification that uses Gist descriptors to capture the spatial structure of an image as features for the K-Nearest Neighbors algorithm.
   Pixel+KNN: This method directly utilizes the normalized pixel values of images as features for the K-Nearest Neighbors algorithm.
2. Optimization-Based Methods
   MAML [16]: Model-Agnostic Meta Learning is designed to rapidly adapt to new tasks with minimal data. It prepares the model with an initialization that is sensitive to changes in the task, allowing for quick adaptation in just a few gradient updates.
   GAP [19]: This approach enhances the inner-loop optimization in meta learning by using a Geometry-Adaptive Preconditioner.
3. Metric-Based Methods
   ProtoNet [20]: Prototypical networks learn a metric space in which classification can be performed by computing distances to prototype representations of each class.
   RelationNet [21]: Employs a learnable relation module to compare query and few-shot support images.
   ConvProtoNet [22]: This model enhances the standard ProtoNet by incorporating a convolutional induction module.
   Dynamic Conv [33] **+ ProtoNet** [20]: This model replaces the convolution blocks of each layer of the backbone network in ProtoNet with the dynamic convolution proposed by Chen et al. [33].

4.3. Experiment Setup

For the machine-learning models, we set $k = 3$ for the K-Nearest Neighbors (KNN) model, to ensure a degree of generalization and prevent overfitting. We used the Euclidean distance as a metric. If there were multiple neighbors with the same class label, we selected the label of the nearest neighbor as the model's output. For the Gist+KNN model, we employed the same Gabor filters settings as in [22], using orientations of 0, $\pi/4$, $\pi/2$, and π, and scales of 3, 5, 7, and 9 to construct 16 Gabor filters, segmenting the input image into 64 blocks and the results into a 1024-dimensional feature vector. For the deep-learning models, to ensure fair comparison, we employed the CNN4 structure mentioned in Section 3.3.1 as our backbone for all the models except ConvProto [22], which followed the original paper's settings. The last layer of our backbone had 64 channels, followed by a max-pooling layer of shape (2, 4), resulting in a 512-dimensional feature embedding. For the optimization-based Few-Shot Learning methods MAML and GAP, both the inner-loop and outer-loop learning rates were set to 1×10^{-4}, with other settings remaining consistent.

We employed a three-layered Adaptive-Graph-Module architecture. Each layer incorporated a learnable distance function, as depicted in Figure 6. The FeedForward Network within each layer progressively mapped the feature space from 512 dimensions to 256, 128,

and, finally, 64 dimensions. This gradual reduction was designed to refine the focus of the model as it delved deeper into the network. Concurrently, the attention temperature was uniformly decreased from 30 to 1 across the layers [33]. For the loss function, we opted for a hyperparameter setting of $\lambda = 1$.

Our dataset was divided into three parts: meta training sets, meta validation sets, and meta testing sets. The meta training set was used for updating the model parameters, the meta validation set for controlling the training process, and the meta testing set for evaluating model performance. In one training epoch, the model would randomly select N classes from the meta training set, with K samples per class as the support set, and the remaining samples as the query set. During meta validation, after every 100 training epochs on the meta training set, validation was conducted for 100 epochs on the validation set. The model with the best performance on the validation set was selected for final evaluation on the test set. For simplicity, we focused solely on the model's accuracy.

Each malware sample's grayscale image was of shape 256×256, with augmentation including random cropping to 224 and horizontal flipping. We used the Adam optimizer with an initial learning rate of 1×10^{-4}, decaying the learning rate by $\times 0.5$ every 10,000 epochs, with a weight decay of 0.01, training for a total of 50,000 epochs. All experiments were run on 4 RTX 2080Ti GPUs and the code was implemented by PyTorch.

4.4. Experiment Results

We report results for the N-way K-shot classification on LargePE and VirusShare as, respectively, shown in Tables 2 and 3, with a 95% confidence interval. The number of classes N is in [5, 20], and the number of support-set samples, K, is in [5, 10], constituting a total of four tasks, as is common in most Few-Shot Learning experiments. In nearly all configurations, AGProto significantly outperformed all the baseline models, achieving commendable accuracy even in the challenging 20-way-5-shot tasks. Due to the constraints of the graphics-card memory capacity, the results for the optimization-based methods (MAML and GAP) in all 20-way settings are not reported. Even when compared to well-performing models (such as ProtoNet, ConvProto, and MAML), AGProto consistently showed an increase of around 2% in accuracy, and for the more demanding 20-way tasks, the accuracy improvement of AGProto exceeded 3% or more. Against other somewhat weaker models, AGProto demonstrated an advantage exceeding 10%.

To validate the generalization ability of our model, we also performed cross-validation between the two datasets. Specifically, we tested the model trained on the LargePE dataset, using data from VirusShare and, similarly, we tested the model trained on VirusShare with data from LargePE. The experimental results are presented in Table 4. It was observed that once AGProto was well trained it could achieve impressive results on other datasets without the need for retraining or fine tuning. It is noteworthy that when evaluated on the same dataset, the performance of AGProto, trained with external data, was comparable to that of the baseline methods, further underscoring the robustness and efficacy of our approach.

To further underscore the effectiveness of AGProto, we visualized the confusion matrix for the 5-way-5-shot task on the VirusShare dataset, as shown in Figure 8. Clearly, AGProto accurately classified the vast majority of malware samples. The confusion matrix reveals not only high true positive rates but also minimal misclassifications, indicating a robust discriminatory capability even among closely related malware classes. This highlights AGProto's nuanced understanding of subtle feature differences, crucial in the precise identification of various malware types.

4.5. Performance Analysis

In the subsequent analysis, we dissected the performance metrics of our proposed model, following the experimental outcomes presented. Our evaluation was benchmarked against established metric-based models, as detailed in Section 4.4, critically assessing our model's efficiency along several axes: the quantity of parameters, model dimensions, training and inference speeds, and memory consumption. The comparison was deliber-

ately confined to metric-based approaches for Few-Shot Learning, such as those exemplified by optimization-based methods like MAML. This choice stemmed from the inherent differences in training strategies between the two paradigms. Specifically, optimization-based methods, due to their reliance on the backbone network for parameter quantity and their intensive use of inner-loop gradient updates within a single step, typically exhibit slower training and inference times as well as higher memory demands than metric-based methods.

Our analytical framework centered on a 5-way-5-shot task, shedding light on the operational efficiency of our model. This methodology facilitated a balanced review of computational demands in relation to performance outcomes, as illustrated in the comparative results with baseline models shown in Table 5.

Table 2. Testing accuracy of all models on LargePE Dataset. 95% confidence intervals are attached after the mean accuracy, and the best accuracy of each column is in bold. Results marked with an asterisk (*) are derived from [22].

Model/Task	5-Way-5-Shot	5-Way-10-Shot	20-Way-5-Shot	20-Way-10-Shot
Gist+KNN3 *	69.07 ± 2.81	75.18 ± 2.53	55.51 ± 1.60	61.82 ± 1.58
pixel KNN3 *	63.60 ± 0.94	67.39 ± 0.96	42.49 ± 0.58	45.22 ± 0.42
MAML	83.11 ± 0.02	86.20 ± 0.02	-	-
GAP	83.19 ± 0.03	85.96 ± 0.03	-	-
ProtoNet *	79.57 ± 0.22	82.63 ± 0.20	63.31 ± 0.12	65.00 ± 0.11
RelationNet *	74.98 ± 0.23	77.28 ± 0.23	53.13 ± 0.12	57.48 ± 0.13
ConvProtoNet *	83.34 ± 0.13	86.63 ± 0.13	68.56 ± 0.08	71.38 ± 0.08
Dynamic Conv	82.59 ± 0.20	84.72 ± 0.19	69.27 ± 0.11	72.25 ± 0.12
Ours	**86.08 ± 0.18**	**89.22 ± 0.16**	**72.09 ± 0.11**	**75.03 ± 0.10**

Table 3. Testing accuracy of all models on the VirusShare Dataset, with 95% confidence intervals attached after the mean accuracy. The best accuracy of each column is in bold.

Model/Task	5-Way-5-Shot	5-Way-10-Shot	20-Way-5-Shot	20-Way-10-Shot
Gist+KNN3	74.30 ± 0.22	80.60 ± 0.19	61.12 ± 0.06	68.36 ± 0.05
pixel KNN3	63.59 ± 0.27	69.28 ± 0.24	49.79 ± 0.08	55.57 ± 0.06
MAML	83.32 ± 0.02	85.06 ± 0.02	-	-
GAP	83.07 ± 0.02	86.20 ± 0.02	-	-
ProtoNet	81.78 ± 0.20	84.24 ± 0.17	67.60 ± 0.05	71.01 ± 0.06
RelationNet	78.25 ± 0.22	80.14 ± 0.19	65.29 ± 0.04	67.03 ± 0.04
ConvProtoNet	83.39 ± 0.19	85.87 ± 0.18	67.52 ± 0.06	71.52 ± 0.06
Dynamic Conv	83.40 ± 0.18	86.34 ± 0.17	68.43 ± 0.06	72.25 ± 0.06
Ours	**86.15 ± 0.19**	**88.80 ± 0.17**	**71.13 ± 0.05**	**74.72 ± 0.05**

Table 4. Cross-validation results demonstrating model performance on different tasks. For "LargePE + AGProto", the test data were sourced from VirusShare, while for "VirusShare + AGProto", the test data were sourced from LargePE.

Model/Task	5-Way-5-Shot	5-Way-10-Shot	20-Way-5-Shot	20-Way-10-Shot
LargePE + AGProto	82.13 ± 0.19	85.04 ± 0.18	66.34 ± 0.04	68.86 ± 0.05
VirusShare + AGProto	80.65 ± 0.25	83.10 ± 0.22	65.93 ± 0.13	68.12 ± 0.13

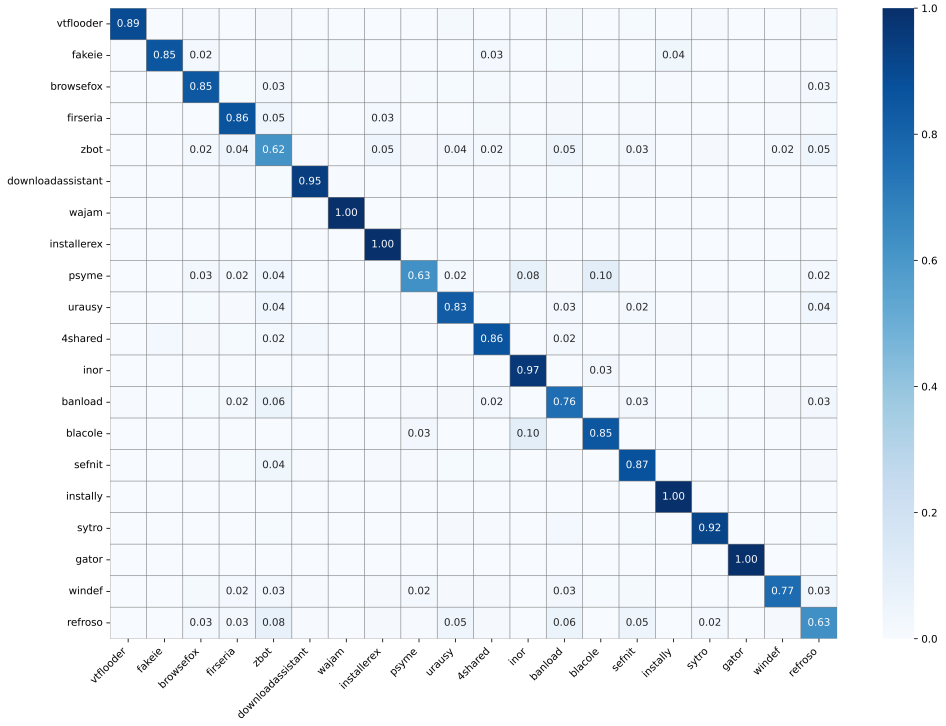

Figure 8. Confusion matrix of the LargePE datasets for 5-shot 5-way problem. Weights less than 0.02 are omitted.

Table 5. Comparative analysis for a 5-way-5-shot task, with parameter quantity and model size derived using PyTorch tools, and speed and memory usage measured using a consistent 2080Ti GPU across experiments.

Model	Parameters	Model Size	Training Speed	Inference Speed	Memory Usage
ProtoNet	95.49 k	390.3 KB	38.21 it/s	99.78 it/s	2163 MB
RelationNet	143.18 k	601.59 KB	37.34 it/s	97.66 it/s	2219 MB
ConvProtoNet	106.4 k	439.6 KB	37.83 it/s	98.57 it/s	2345 MB
Dynamic Conv	373.66 k	1.50 MB	28.64 it/s	82.46 it/s	2027 MB
AGProto (Ours)	852.61 k	6.65 MB	27.37 it/s	80.75 it/s	2415 MB

The results indicate that, although our model exhibited a higher parameter count and model size than the other baseline models, its 6 MB size remained within an acceptable range and was easily deployable. Despite a slower training speed compared to the baselines, the inference speed was comparable, underscoring the practical utility of our model. Considering the disparity in parameter quantity and the marginal difference in inference speed, our model demonstrated efficient GPU utilization. Our model's parameter count was more than eight times that of ProtoNet, yet the inference speed was slowed by less than 20%, with a significant improvement in accuracy. Memory consumption across these models showed no significant variation.

5. Discussion

5.1. Why AGProto Works

To elucidate why our AGProto network demonstrated superior performance over other Few-Shot Learning models, a meticulous analysis of the results garnered by AGProto was conducted. To be specific, for an N-way K-shot task, we meticulously chronicled the correctness of each query-set sample's original category for every task. This comprehensive data compilation enabled us to compare the accuracy distribution for each category across the dataset. Our methodology was grounded in two pivotal considerations: the inherent

variability in features within samples of the same category and the random nature of the support-set sample selection, both of which could significantly impact the outcomes. Furthermore, the pronounced feature differences between distinct categories also played a critical role, where the selection of $N-1$ categories to compare against a fixed one could skew the accuracy of that category. AGProto was conceived with the ambition of fostering a network capable of adapting to the sample's unique features, autonomously adjusting its output in response to the data's intrinsic characteristics. Therefore, the task described above was very suitable for comparing the differences between AGProto and other Few-Shot Learning models. We chose the classic ProtoNet as comparison and visualized the tasks described above in box plots, as shown in Figure 9.

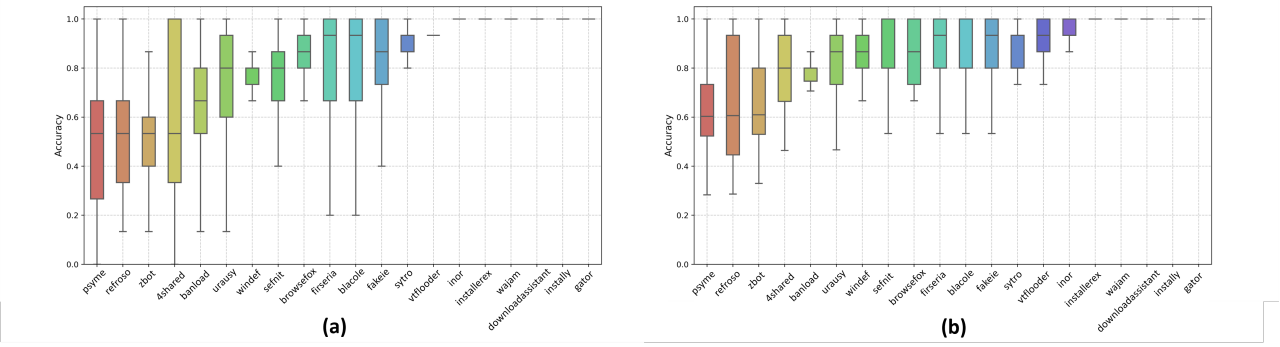

Figure 9. The comparative analysis of (**a**) ProtoNet and (**b**) AGProto, using box plots to display the distribution of accuracy across different categories.

Upon inspecting the results visualized in the box plots, we can clearly observe that compared to the box plots of ProtoNet, AGProto's box plots have two very noticeable advantages: overall accuracy improvement and constricted InterQuartile Range.

Overall Accuracy Improvement: It is evident that AGProto catalyzed an elevation in the median accuracy across categories when compared to ProtoNet. For some categories that are difficult to identify, the improvement was obvious. It can be seen from here that for categories such as zbot, psyme, refroso, etc., AGProto improved the accuracy by about 7% compared to ProtoNet. Such a holistic enhancement is suggestive of AGProto's robustness and its capacity to consistently elevate performance across diverse scenarios, which is a cornerstone of Few-Shot Learning models' utility.

Constricted Interquartile Range: The constriction of the InterQuartile Range (IQR) in AGProto's box plot compared to that of ProtoNet's is particularly telling. This narrowed IQR signifies a more stable and consistent accuracy across various tasks, with a reduction in the dispersion of data points. The AGProto model's ability to adapt leads to a performance that is not only higher on average but also more predictable and reliable, with fewer instances of erratic outliers. The marked consistency, as evidenced by the diminished range of the middle 50% of data points, underscores AGProto's competency in handling the inherent variability within Few-Shot Learning tasks.

In summary, the visual data from the box plots unequivocally illustrate that AGProto not only enhances the overall accuracy but also brings about a constricted InterQuartile Range, manifesting a more consistent and reliable performance. The results validate our hypothesis that our designed Adaptive Graph ProtoNet tuning itself to the features of the samples can indeed achieve superior performance.

5.2. Future Work

We suggest several potential avenues for extending our work, aimed at advancing research in few-shot malware classification. Firstly, traditional approaches generally assume an equal number of samples per class; however, this is seldom the case in real-world scenarios where some malware types are more prevalent than others. This leads to a model bias towards recognizing common malware classes while neglecting rarer ones. To mitigate

this, future studies could explore adaptive techniques that adjust the learning process based on class frequency, thereby ensuring a more balanced and representative model.

Additionally, our experimental results indicate that optimization-based Few-Shot Learning methods, such MAML [16] and GAP [19], achieve commendable accuracy, trailing our proposed AGProto by less than 3% in 5-way-5-shot and 5-way-10-shot tasks. These methods fundamentally learn an embedding model, which is the same as our AGProto, and append a classifier layer. Integrating these optimization-based approaches could potentially enhance performance, warranting further exploration.

Moreover, we observed that the choice of backbone network significantly impacts performance. While some pre-trained models from the computer-vision domain, such as ResNet and ViT, did not exhibit superior performance on malware datasets, designing convolutional backbone networks tailored to the characteristics of malware grayscale images may yield improvements. For instance, considering that malware grayscale images typically contain less information than natural images and have more densely packed lateral information (adjacent bits), simpler network architectures or asymmetric convolutional kernels might be more effective. Such specialized networks would cater to the unique data structure and distribution found in malware images, potentially leading to more accurate and robust classification results.

Lastly, the issue of using adversarial attacks to bypass model detection has always been a focal point in the field of malware classification. However, adversarial attacks targeting multi-classification problems are currently underexplored. While bypassing malware-detection models through adversarial attacks is undoubtedly valuable for simulating the process by which attackers disguise malicious code, we posit that attacking malware-classification models holds significant research value as well. Such efforts can aid in understanding the relationships among different malware families. This approach not only addresses an existing research gap but also contributes to the development of more sophisticated and secure malware-classification systems.

6. Conclusions

In this research, we introduced AGProto, an innovative approach designed to tackle the few-shot malware-classification challenge. This method ingeniously harnesses the power of Graph Neural Networks for effective message passing, thereby generating multiple prototypes that encapsulate the query-set samples' information. These prototypes are then dynamically weighted through an attention mechanism, considering each sample's distinctive features. Consequently, AGProto tailors a customized prototype for each sample, from which classifications are derived using Euclidean distances.

Our empirical studies manifested that AGProto achieves the highest accuracy on two distinct few-shot malware datasets. Further explorations revealed that not only does AG-Proto exhibit commendable generalization and robustness, but it also significantly enhances the classification accuracy of particularly challenging categories. This performance aligns with the initial aspiration behind AGProto's creation: to foster a network that inherently adapts to the unique characteristics of each sample, autonomously modifying its responses based on the inherent traits of the data. Such adaptability and precision underline AG-Proto's potential as a formidable model in the realm of malware classification, particularly in scenarios constrained by limited data availability.

Author Contributions: Conceptualization, J.W. and P.W.; methodology, software, validation, formal analysis: J.W.; investigation, J.W., T.L. and H.W.; resources, P.W.; data curation, J.W.; writing—original draft preparation, J.W.; writing—review and editing, P.W., T.L. and H.W.; visualization, J.W.; supervision, P.W.; funding acquisition, P.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the Key R&D projects of the Sichuan Science and Technology Plan (2022YFG0323); in part by the Key R&D projects of the Chengdu Science and

Technology Plan (2022-YF05-00451-SN) and in part by the Key R&D projects of Sichuan Science and Technology Plan (2023YFG0101).

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Kaspersky. Kaspersky Security Bulletin 2023. Available online: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2023/11/28102415/KSB_statistics_2023_en.pdf (accessed on 7 January 2024).
2. Yan, S.; Ren, J.; Wang, W.; Sun, L.; Zhang, W.; Yu, Q. A Survey of Adversarial Attack and Defense Methods for Malware Classification in Cyber Security. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 467–496. [CrossRef]
3. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
4. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
5. Liu, L.; Wang, B.S.; Yu, B.; Zhong, Q.X. Automatic malware classification and new malware detection using machine learning. *Front. Inf. Technol. Electron. Eng.* **2017**, *18*, 1336–1347. [CrossRef]
6. Cakir, B.; Dogdu, E. Malware classification using deep learning methods. In Proceedings of the ACMSE 2018 Conference, Richmond, KY, USA, 29–31 March 2018; pp. 1–5.
7. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.D.; Wang, Y.; Iqbal, F. Malware classification with deep convolutional neural networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018; pp. 1–5.
8. Pascanu, R.; Stokes, J.W.; Sanossian, H.; Marinescu, M.; Thomas, A. Malware classification with recurrent networks. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 19–24 April 2015; pp. 1916–1920.
9. Li, F.; Fergus, R.; Perona, P. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.* **2006**, *28*, 594–611.
10. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, Pittsburgh, PA, USA, 20 July 2011; pp. 1–7.
11. Vasan, D.; Alazab, M.; Wassan, S.; Safaei, B.; Zheng, Q. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Comput. Secur.* **2020**, *92*, 101748. [CrossRef]
12. Kancherla, K.; Mukkamala, S. Image visualization based malware detection. In Proceedings of the 2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), Singapore, 16–19 April 2013; pp. 40–44.
13. Venkatraman, S.; Alazab, M.; Vinayakumar, R. A hybrid deep learning image-based analysis for effective malware detection. *J. Inf. Secur. Appl.* **2019**, *47*, 377–389. [CrossRef]
14. Vasan, D.; Alazab, M.; Wassan, S.; Naeem, H.; Safaei, B.; Zheng, Q. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Comput. Netw.* **2020**, *171*, 107138. [CrossRef]
15. Cui, Z.; Zhao, Y.; Cao, Y.; Cai, X.; Zhang, W.; Chen, J. Malicious code detection under 5G HetNets based on a multi-objective RBM model. *IEEE Netw.* **2021**, *35*, 82–87. [CrossRef]
16. Finn, C.; Abbeel, P.; Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In Proceedings of the International Conference on Machine Learning. PMLR, Sydney, Australia, 6–11 August 2017; pp. 1126–1135.
17. Nichol, A.; Achiam, J.; Schulman, J. On first-order meta-learning algorithms. *arXiv* **2018**, arXiv:1803.02999.
18. Rajeswaran, A.; Finn, C.; Kakade, S.M.; Levine, S. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2019; Volume 32.
19. Kang, S.; Hwang, D.; Eo, M.; Kim, T.; Rhee, W. Meta-Learning with a Geometry-Adaptive Preconditioner. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vancouver, BC, Canada, 17–24 June 2023; pp. 16080–16090.
20. Snell, J.; Swersky, K.; Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
21. Sung, F.; Yang, Y.; Zhang, L.; Xiang, T.; Torr, P.H.; Hospedales, T.M. Learning to compare: Relation network for few-shot learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 1199–1208.
22. Tang, Z.; Wang, P.; Wang, J. ConvProtoNet: Deep prototype induction towards better class representation for few-shot malware classification. *Appl. Sci.* **2020**, *10*, 2847. [CrossRef]
23. Wang, P.; Tang, Z.; Wang, J. A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Comput. Secur.* **2021**, *106*, 102273. [CrossRef]
24. Garcia, V.; Bruna, J. Few-shot learning with graph neural networks. *arXiv* **2017**, arXiv:1711.04043.
25. Gidaris, S.; Komodakis, N. Generating classification weights with gnn denoising autoencoders for few-shot learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 21–30.

26. Zhou, F.; Cao, C.; Zhang, K.; Trajcevski, G.; Zhong, T.; Geng, J. Meta-gnn: On few-shot node classification in graph meta-learning. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Beijing, China, 3–7 November 2019; pp. 2357–2360.
27. Wang, N.; Luo, M.; Ding, K.; Zhang, L.; Li, J.; Zheng, Q. Graph few-shot learning with attribute matching. In Proceedings of the 29th ACM International Conference on Information & Knowledge Management, Online, 19–23 October 2020; pp. 1545–1554.
28. Tang, S.; Chen, D.; Bai, L.; Liu, K.; Ge, Y.; Ouyang, W. Mutual crf-gnn for few-shot learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 2329–2339.
29. Yu, T.; He, S.; Song, Y.Z.; Xiang, T. Hybrid graph neural networks for few-shot learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Virtual, 22 February–1 March 2022; Volume 36, pp. 3179–3187.
30. Vinyals, O.; Blundell, C.; Lillicrap, T.; Wierstra, D. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: Red Hook, NY, USA, 2016; Volume 29.
31. He, K.; Zhang, X.; Ren, S.; Sun, J. Identity mappings in deep residual networks. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part IV 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 630–645.
32. Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2017**, arXiv:1710.10903.
33. Chen, Y.; Dai, X.; Liu, M.; Chen, D.; Yuan, L.; Liu, Z. Dynamic convolution: Attention over convolution kernels. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 11030–11039.
34. Total, V. Virustotal-Free Online Virus, Malware and Url Scanner. Available online: https://www.virustotal.com/en (accessed on 7 January 2024).
35. Sebastián, M.; Rivera, R.; Kotzias, P.; Caballero, J. Avclass: A tool for massive malware labeling. In Proceedings of the Research in Attacks, Intrusions, and Defenses: 19th International Symposium, RAID 2016, Paris, France, 19–21 September 2016; Proceedings 19; Springer: Berlin/Heidelberg, Germany, 2016; pp. 230–253.
36. Yajamanam, S.; Selvin, V.R.S.; Di Troia, F.; Stamp, M. Deep Learning versus Gist Descriptors for Image-based Malware Classification. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018), Madeira, Portugal, 22–24 January 2018; pp. 553–561.