Zhou *et al. Cybersecurity* (2025) 8:69
https://doi.org/10.1186/s42400-025-00358-5

Cybersecurity

RESEARCH Open Access

Androfim: few-shot android malware family detection based on image representation

Fan Zhou[1], Dongxia Wang[2*], Yanhai Xiong[3], Kun Sun[4] and Wenhai Wang[1]

Abstract

Android malware is the major cyber threat to the popular Android platform which may influence millions of end users. To battle against the Android malware, a large number of machine learning methods either based on 1) traditional feature extraction using static and dynamic analysis, or 2) recently proposed image representations, have been developed, and have achieved promising results. However, the vast majority of the existing work rely on a large number of labeled samples which are unfortunately not available for the newly reported Android malware families. This poses a critical challenge to detect such *few-shot Android malware families*. In this paper, we propose a novel few-shot learning approach based on the image representation of an Android application to solve the problem. With an application file converted into an image representation, we preserve all the source code information. We then utilize self-supervised learning to obtain the pre-trained backbone from the *unlabeled* auxiliary data and employ a metric-based few-shot learning method for Android malware classification. Considering the impact of irrelevant information across samples on the family classification, we employ a multi-cropping strategy to capture family label-related information in the images. Extensive experimental results on the popular *CICInvesAndMal2019* dataset confirm the effectiveness of our approach in detecting few-shot Android malware families. We achieve at least 3.16% and 3.7% improvement on 5-way 1-shot and 5-way 5-shot scenarios respectively comparing to state-of-the-art baselines.

Keywords Malware Detection, Android malware family, Few-shot learning, Few-shot malware family

Introduction

Mobile devices connected to the Internet have become ubiquitous. They are changing our daily life in many ways (e.g., socializing, online banking, healthcare, education, daily work) Fan et al. (2021). Android, as one of the most popular mobile operating systems, is the foundational software infrastructure in the mobile market. Unfortunately, Android has also become one of the primary targets of many cyber-attacks due to its openness and popularity Google (2019). For instance, ransomware campaigns use software exploits, flaws, cracked passwords, and other vulnerabilities to invade a system Palmer (2022).

To defend against Android malware, various types of machine learning (ML)-based detection approaches have been studied. Depending on the way of feature extraction to build the detection system, existing ML-based approaches can be roughly divided into those based on static analysis and those based on dynamic analysis. Specifically, the former directly extracts static features from the application source package, such as API calls, the permission and intent messages (Arp et al. 2014; Zhang et al. 2019; Gordon et al. 2015), while the latter requires executing applications in simulators or real devices and extracts dynamic behavioral features at running time (Tripp and Rubin 2014; Xue et al. 2017; Han et al. 2020).

*Correspondence:
Dongxia Wang
dxwang@zju.edu.cn
[1] Zhejiang University, Hangzhou 310000, China
[2] Zhejiang University & ZJU-Hangzhou Global Scientific and Technological Innovation Center, Hangzhou 310000, China
[3] College of William & Mary, Williamsburg 23185, USA
[4] George Mason University, Fairfax 22030, USA

Springer Open

© The Author(s) 2025. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

More recently, researchers have proposed to transform Android applications into image representations (see Fig. 2 for the example) for malware detection. For example, R2-D2 Hsien-De Huang and Kao (2018) transforms the bytecode of *classes.dex* from an Android archive file to RGB color code and stores it as a color image with a fixed size. These color images are used as input to the convolutional neural network (CNN, or ConvNet) for automatic feature extraction and training. Compared to static or dynamic analysis-based approaches, image representations of Android applications are much more cost-efficient for feature extraction. More importantly, such image representations open the door to a large body of advanced image analysis techniques in machine learning, which is potentially helpful in addressing more challenges in Android malware detection.

One such challenge is to detect few-shot Android malware families. Notice that, in general, existing ML-based approaches require a sufficient amount of labeled data to achieve satisfactory detection accuracy. However, for certain Android malware families, the labeled samples may be sparse or even hardly available, e.g., consider those newly emerged malware families. We call them *few-shot Android malware families*. Considering the adaptive behavior of malware attackers, few-shot Android malware families are typically unavoidable. To wait to collect a sufficient amount of labeled samples to train the model is not only costly but also falls behind the attackers.

Noteworthily, the problem of sparse labeled training samples is not exclusive to Android malware family detection. In fact, it has been extensively studied in the image domain. For example, a well-known fact of the human visual system is that it is capable of learning new object categories, with only one or few examples of previously learned categories. This inspired researchers to come up with the idea of few-shot learning Fei-Fei et al. (2006), which aims to classify samples of unseen classes given only a handful of labeled samples (Finn et al. 2017; Snell et al. 2017; Hu et al. 2022). Typically, few-shot learning is realized by learning from a large-scale source task to obtain some transferable knowledge and then generalizing to a small-scale target task Hu et al. (2022). In this paper, we resort to the idea of few-shot learning to detect few-shot Android malware families, based on their image representations to fully embrace its power.

We propose a novel approach for few-shot Android malware family classification. To reduce the cost of feature extraction and better utilize the power of few-shot learning, we design our detection system based on the image representation of Android application packages (APKs). Image representation reserves most of the source code information in an APK. We first employ self-supervision to pre-train the feature extraction backbone on a large number of unlabeled samples, which are usually available in practice. After pre-training, we fine-tune the backbone with the labeled samples (*base* dataset), combining a standard classification loss function and a rotaion auxiliary loss function. The fine-tuned backbone is then used for feature extraction of the few-shot malware dataset (*novel* dataset[1]), which consists of a (labeled) support set and an (unlabeled) query set whose samples are to be classified. Noteworthily, to derive the feature vector of a sample from the support set, we do not directly feed its entire image representation, but make multiple random cropping and take a weighted average of features of the crops. The weight of a crop is determined by its similarity to that in the corresponding position of the query sample to be classified. In so doing, we aim to capture more family label-related information in feature extraction, reducing the influence of irrelevant information in classification. Finally, we complete the classification by a metric-based few-shot learning approach.

In summary, our work makes the following contributions:

- We propose AndroFIM, a novel approach that can effectively and efficiently detect few-shot (or even unknown) families of Android malware. This is important considering the consistently emerging new malware families in the Android domain.
- By using a multiple cropping strategy to retain more family label-related information, AndroFIM effectively reduces the influence of family label-irrelevant information on classification.
- We conduct sufficiently many comparative experiments that demonstrate the superior performance of AndroFIM. We improve the detection accuracy of the benchmark models on 5-way 1-shot and 5-way 5-shot scenarios by at least 3.16% and 3.7% respectively.

Related work

We discuss the most relevant works on Android malware detection, few-shot learning, and also the few-shot Android malware family classification.

Android malware detection

Android malware detection tasks mainly aim to determine whether a given APK is benign software or malware, and sometimes also to determine which specific family a malware APK belongs to (multi-classification

[1] The base dataset is of a much larger scale than the novel dataset, and their label spaces are disjoint.

problem). The existing detection methods can roughly be divided into two types: those based on static analysis and those based on dynamic analysis.

Static analysis based approaches (Arp et al. 2014; Zhang et al. 2019; Gordon et al. 2015) mainly focus on parsing the internal files of an Android APK statically without executing it. DREBIN Arp et al. (2014) uses lightweight features such as permissions, API calls, and intents to distinguish Android malware families. Zhang et al. (2019) first extracts multi-level fingerprints from applications by using n-gram analysis and feature hashing, and then inputs each of its sub-fingerprints to a dedicated online classifier to complete Android malware family classification. DroidSafe Gordon et al. (2015) detects Android malware by analyzing sensitive explicit information flows from sources to sinks. In general, these approaches can be circumvented when an adaptive attacker exploits techniques such as repackaging and obfuscation.

Dynamic analysis based approaches (Tripp and Rubin 2014; Xue et al. 2017; Han et al. 2020) rely on executing the application in an Android emulator or a device, aiming to detect the loading of dynamic code and record the behavior of the application. BAYESDROID Tripp and Rubin (2014) extract features using multi-level taint tracking, where sensitive information is flagged as a taint source and tracked as the application runs. Malton Xue et al. (2017) provides a comprehensive view of malware's behaviors by conducting multi-layer monitoring and information flow tracking, as well as efficient path exploration. FARM Han et al. (2020) uses three feature transformation techniques to detect three categories of malware. Typically, these approaches are both time-consuming and labor-intensive because they require running the application in a real-world environment.

Different from the above methods, AndroFIM adopts a unique approach by transforming an APK file into an image. With image representation of APKs, AndroidFIM not only expedites traditional feature extraction but also allows for the integration of well-established techniques from the computer vision domain, tapping their potential to support the domain of Android malware detection.

Few-shot learning

Few-shot learning (FSL) is a machine learning technique that aims to identify new, unknown classes with only a small number of labeled samples. This is achieved usually by leveraging the knowledge learned from existing datasets and applying it to the new classes. The existing FSL methods can be roughly split into two clusters: optimization-based methods and metric-based methods.

Optimization-based approaches (Finn et al. 2017; Lee et al. 2019) design optimization-centered algorithms to adapt to novel tasks. MAML Finn et al. (2017) is a general training strategy trying to offer good initial value to fine-tune and accelerate the adaption to novel tasks. MetaOptNet Lee et al. (2019) exploits two properties of the linear classifiers: implicit differentiation of the optimality conditions for convex problems and pairwise formulas for optimal problems.

Metric-based approaches (Vinyals et al. 2016; Snell et al. 2017) learn a feature embedding and then use a distance function to perform the classification. Lots of existing works adopt the training strategy of *meta-learning* where the training set is partitioned in episodes resembling in structure the novel tasks Lazarou et al. (2021). Matching Network Vinyals et al. (2016) introduces attention and memory mechanisms to accelerate learning, and cosine distance is used as the metric. ProtoNet Snell et al. (2017) treats the mean of all the samples as the prototype of a class and uses the Euclidean distance as the metric. More recently, it is recognized that it is more effective to learn an embedding on the entire training set than sampling few-shot training episodes that resemble novel tasks (Tian et al. 2020; Lazarou et al. 2021).

While most of the few-shot learning problems and methods above target computer vision domain, we specifically consider few-shot Android malware family detection task in this work, and also how the idea of few-shot learning can help.

Few-shot android malware family classification

Few-shot Android malware families are those for which there are few labeled samples, e.g., considering those newly developed families. For such families, the existing supervised learning-based classification methods which require a sufficient amount of labeled samples may be infeasible. We identify a few approaches for few-shot Android malware family classification that use the classical few-shot learning models on manually collected features. For example, Bai et al. (2020) proposes a siamese-network-based learning method, which trains an effective Multi-Layer Perceptron (MLP) network for embedding malware applications into a real-valued, continuous vector space. Strictly speaking, this work does not address the few-shot problem but rather focuses on addressing data imbalance. Specifically, the dataset it uses contains both large-scale families and few-shot families. Notably, the model is not equipped to handle families that were not encountered during the training phase. FSMC Ale et al. (2020) achieves few-shot Android malware family classification by using API calls as features to train a mature few-shot learning method - Matching Network Vinyals et al. (2016). However, its performance is limited. Besides, these works above require manual feature extraction which is costly and inefficient.

Furthermore, there also exist studies dealing with few-shot malware families in other domains (Chai et al. 2022; Conti et al. 2022; Rong et al. 2021), not for Android. For example, DPNSA Chai et al. (2022) focuses on Windows malware and is not suitable to be our baseline. Windows and Android are distinct operating systems with unique executable file structures, security mechanisms, and malware attack vectors. Consequently, the methods developed for Windows malware classification are tailored to the specific characteristics and behaviors of Windows-based malware. For instance, DPNSA needs to extract the specific semantic information of the executable file of the Windows platform software to perform feature extraction.

Hence, the problem of classifying few-shot Android malware families remains to be under-explored. We propose AndroFIM, and this new pipeline amalgamates specialized, mature modules. For instance, it integrates self-supervision methods during backbone network pre-training, incorporates rotation loss in fine-tuning, and also employs a novel multi-cropping strategy to retain more family label-related information for classification.

Methodology

In this section, we formally define the problem of detecting few-shot Android malware families with few-shot learning and introduce our approach in detail.

Problem formulation

There is a small set of Android malware samples $\mathbb{D}_{novel}$ that belong to N families (few-shot Android malware families), each of which has only a handful of (K) samples labeled. These labeled samples in $\mathbb{D}_{novel}$ constitute the *support set* $\mathbb{S} : \mathbb{S} \subset \mathbb{D}_{novel}$, $\mathbb{S} = \{(x_i, y_i)\}_{i=1}^{N \times K}$, where x_i denotes the labeled APK and y_i denotes its label. The remaining unlabeled samples form the *query set* (which are to be classified): $\mathbb{Q} : \mathbb{Q} = \{x_j\}_{j=1}^{M}$. $\mathbb{D}_{novel} = \mathbb{S} \cup \mathbb{Q}$. This setting is denoted as a N-way-K-shot problem. The proportion of correctly classified APKs in $\mathbb{Q}$ decides the detection accuracy.

Learning from the support set and then classifying for the query set is the target task of few-shot learning-based classification. However, it is usually challenging to build an ideal classifier based on the insufficient labeled samples in $\mathbb{S}$. The existing few-shot learning methods often try to obtain some transferable knowledge first by learning from another, usually large-scale dataset $\mathbb{D}_{base}$, and then generalize to the target classification task. $\mathbb{D}_{base}$ and $\mathbb{D}_{novel}$ usually have similar data distributions, but their label spaces are disjoint.

To obtain $\mathbb{D}_{base}$, a common practice in the literature is to sample $\mathbb{D}_{base}$ and $\mathbb{D}_{novel}$ from the same dataset Chen et al. (2019). Sometimes, $\mathbb{D}_{base}$ and $\mathbb{D}_{novel}$ are also generated from different domains (Guo et al. 2020; Chen et al. 2019). Whatever the way, $\mathbb{D}_{base}$ is usually required to be labeled in these approaches. Considering that in practice it may not always be easy to obtain a sufficient amount of *labeled* samples for Android malware families, we propose to use the *unlabeled* APKs $\mathbb{D}_{add}$ (which are much more than the labeled ones) in additional, which is relatively easy to obtain. In our training stage, only $\mathbb{D}_{base}$ and $\mathbb{D}_{add}$ are available. In the testing stage, $\mathbb{D}_{novel}$ is used to sample $\mathbb{S}$ and $\mathbb{Q}$ to test how well our few-shot learning model learns from $\mathbb{D}_{base}$ and $\mathbb{D}_{add}$. The size of our $\mathbb{D}_{base}$ is denoted as N_b.

Approach overview

In this section, we explain our few-shot learning-based Android malware family detection approach in detail. As shown in Fig. 1, AndroFIM mainly consists of the following four phases.

- *APK Transformation*: for each given Android application (in the format of APK file), we decompress it to get the *classes.dex* file, which is transformed to a color image sample.
- *Pre-training of Backbone*: we collect a large set of unlabeled samples (D_{add}) from popular open-source datasets, and pre-train the backbone for feature extraction in a self-supervised way.
- *Fine-tuning of Backbone*: we use standard classification loss and auxiliary rotation loss in combination to fine-tune the backbone with a much smaller set of labeled samples (D_{base}).
- *Android Malware Family Classification:* Relying on the fine-tuned backbone, we map all the samples in the support set and query set to the feature space. To reduce the impact of family label-irrelevant information, given a query sample, for each support sample we adopt a multiple cropping strategy and use the weighted average to determine its feature vector. Then we calculate the prototype (as a representative) of each family (referring to ProtoNet Snell et al. 2017) in the support set, the distance to which determines the family of the query sample.

APK transformation

An Android application is packaged into an APK, and the *classes.dex* file is the core of an APK, which refers to a binary container for the code and the associated data Mahdavifar et al. (2022). There have been some studies that transform the *classes.dex* file into an image for Android malware detection (Xiao and Yang 2019; Hsien-De Huang and Kao 2018). We follow the idea proposed by Xiao and Yang (2019) to visualize

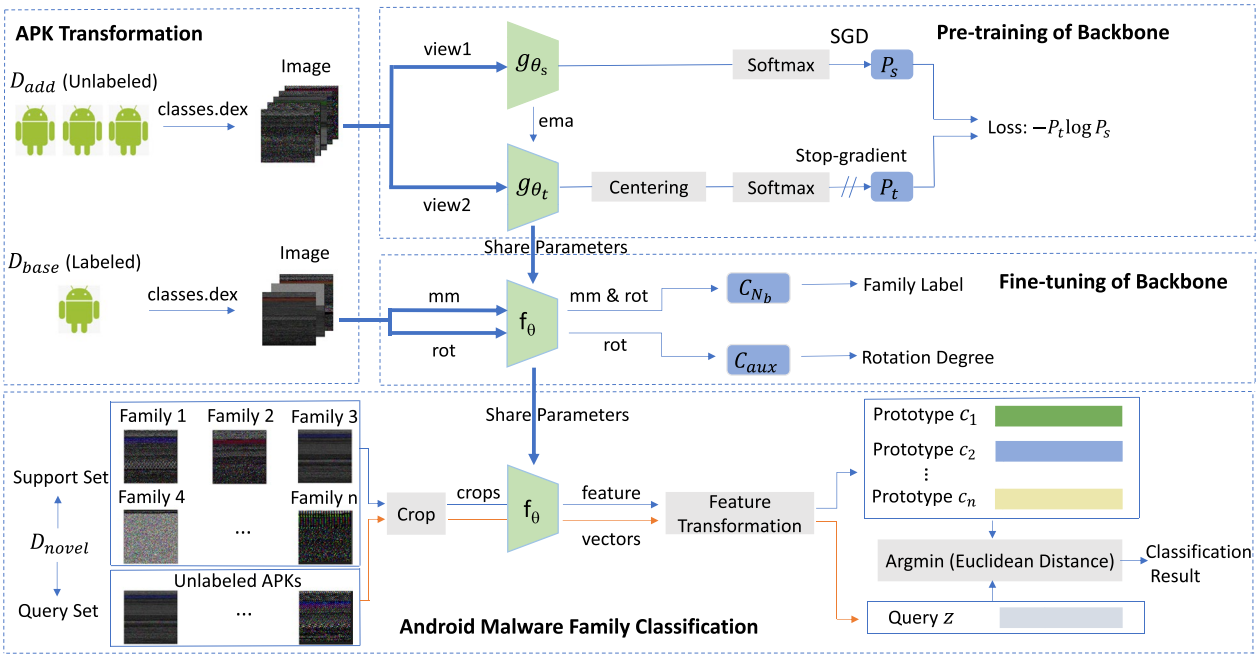


Fig. 1 The framework of AndroFIM

the Android bytecode as a color image. Using Android malware images can reduce the need of domain expert knowledge and costs less. Specifically, we get the *classes.dex* file by unpacking the APK first. Then we read this bytecode file in hexadecimal format since the range of 2 hexadecimal digits is from 00 to FF, which corresponds to 0 to 255 in decimal. By transforming 3 consecutive 2 hexadecimal digits into 3 values for each of the RGB channels, we can transform the sequence of bytecodes into a color matrix, from where an image file is generated.

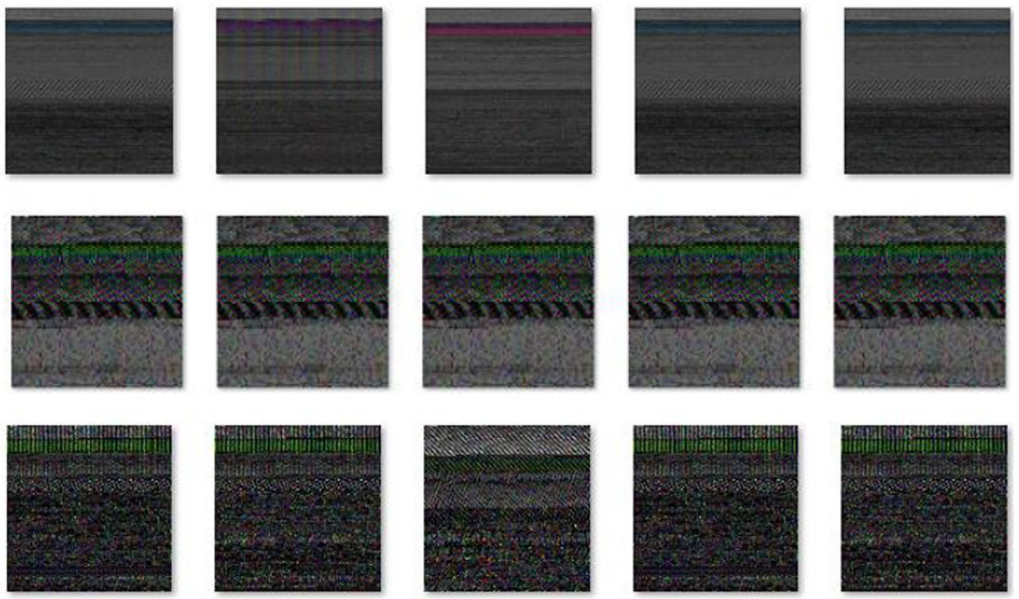


Fig. 2 Transformed images obtained from five samples from the AndroidDefender, Ewind, and fakeAV malware families respectively (Top row to bottom row)

Some transforming results are shown in Fig. 2. It can be seen that the texture of images transformed from the same malware family tends to be similar, and perceptible differences exist between image samples transformed from different families. Note that later we use $D_{add}, D_{base}, D_{novel}, S, Q$ to represent the set of transformed images of APK samples in the set $\mathbb{D}_{add}, \mathbb{D}_{base}, \mathbb{D}_{novel}, \mathbb{S}, \mathbb{Q}$ respectively.

Pre-training of backbone

Usually, we may be faced with insufficient labeled data to directly train a well-performing backbone. We notice that a large number of unlabeled Android malware samples are readily available in practice, which is much cheaper to get than the labeled samples. We introduce a well-established self-supervised approach DINO Caron et al. (2021) (instead of common supervised learning solutions like CNN) to pre-train the backbone with these unlabeled samples. By pre-training, we narrow the distance between samples of the same class (family) in the feature space, which facilitates the subsequent classification. As our input image size could vary with different APK sizes, DINO can minimize the loss of semantic information in the image resizing process by identifying the global and local consistency of the input image when training the model. Specifically, given an APK image $x \in D_{add}$, we generate a set (V) of different views of the image (Refer to Caron et al. (2020)), containing two global views, x_{g_1} and x_{g_2}, and several (Default is 8) local views. We define two networks with the same structure, the student network g_{θ_s} and the teacher network g_{θ_t}. We send the global views to the teacher network and all the views in V to the student network and then train the student network to match the output of the teacher network. Denote the probabilistic outputs of the two networks as P_s and P_t respectively. $P \in \{P_s, P_t\}$ is obtained by normalizing the network with a softmax function (Refer to Caron et al. (2021)). The function is then used to convert the output of the network into a probability distribution. The i^{th} output of the network with x as the input is:

$$P(x)^i = \frac{exp(g_\theta(x)^i/\tau)}{\sum_{k=1}^{K} exp(g_\theta(x)^k/\tau)}, \tag{1}$$

where $\tau > 0$ is a temperature parameter that controls the sharpness of the output distribution Caron et al. (2021). The output of the teacher network is centered with a mean computed over the batch. We match the two outputs P_s and P_t by minimizing the cross-entropy loss and learn the parameter θ_s with stochastic gradient descent (SGD):

$$\min_{\theta_s} \sum_{x \in x_{g_1}, x_{g_2}} \sum_{x' \in V, x' \neq x} H(P_t(x), P_s(x')), \tag{2}$$

where $H(a, b) = -a \log b$. We adopt the parameter update strategy of DINO, and the teacher network is updated with parameters via exponential moving average (EMA) (He et al. 2020; Caron et al. 2021). We may further improve the power of the backbone by adding more unlabeled data than labeled ones.

Fine-tuning of backbone

After the pre-training stage, g_{θ_t} has stronger generalization ability than g_{θ_s} Hinton et al. (2015). We use the parameters of g_{θ_t} as the backbone f_θ parameters. Inspired by Bendou et al. (2022), Mangla et al. (2020), we further combine rotation and manifold mixup Verma et al. (2019) to fine-tune our backbone, using a two-step forward-backward pass at each step Bendou et al. (2022). Combining the regularization techniques of Manifold Mixup and rotation loss during the training process enables robust semantic feature learning, resulting in a significant improvement in few-shot classification Mangla et al. (2020). We add two classifiers after f_θ: a standard N_b-way linear classifier C_{N_b} to retrieve the category to which the image belongs (Bendou et al. 2022; Mangla et al. 2020), and an auxiliary 4-way linear classifier C_{aux} to retrieve the rotation angle of the image.

In the first step, we feed the images of APKs $(x_1, y_1), (x_2, y_2) \in D_{base}$ into the backbone f_θ, the feature representation of layer l would be $f_\theta^l(x_1)$ and $f_\theta^l(x_2)$, then the classification loss function after combining manifold-mixup is:

$$L_{mm} = \mathbb{E}_{(x_1,y_1),(x_2,y_2)\in D_{base}}[L(\mathrm{Mix}_\lambda(\mathrm{f}_\theta^{\mathrm{l}}(\mathrm{x}_1), \mathrm{f}_\theta^{\mathrm{l}}(\mathrm{x}_2)), \mathrm{Mix}_\lambda(\mathrm{y}_1, \mathrm{y}_2))], \tag{3}$$

where

$$\mathrm{Mix}_\lambda(\mathrm{a}, \mathrm{b}) = \lambda \cdot \mathrm{a} + (1 - \lambda) \cdot \mathrm{b} \tag{4}$$

The mixing coefficient λ is sampled from a beta distribution and L is the standard cross-entropy loss.

In the second step, we first rotate the image x by r degrees, where $r \in C_R = \{0, 90, 180, 270\}$. Suppose that x^r is the rotated image from x, we feed it into the two classifiers. For the standard classifier, the loss is given by:

$$L_{\text{family}} = \mathbb{E}_{(x,y)\in D_{\text{base}}, r\in C_R}[L(x^r, y)]. \tag{5}$$

For the auxiliary classifier, the loss is given by:

$$L_{\text{rot}} = \frac{1}{|C_R|} \sum_{x\in D_{\text{base}}} \sum_{r\in C_R} L(C_{\text{aux}}(f_\theta(x^r)), r), \tag{6}$$

where $|C_R|$ denotes the cardinality of C_R and L is the standard cross-entropy loss. The total loss of the second step $L_2 = L_{family} + L_{rot}$.

We fine-tune the backbone parameters θ by minimizing the loss, i.e., L_{mm} in the first step and L_2 in the second step, with stochastic gradient descent (SGD) or adaptive moment estimation (Adam) Bendou et al. (2022). The choice of optimizer depends on the current learning rate. The parameters of the backbone f_θ get determined after the fine-tuning phase, which will be used in the later phase.

Android malware family classification

With the fine-tuned backbone f_θ, we can now map all the image samples in D_{novel} to the feature space. In few-shot scenarios where the number of labeled samples is limited, it is important to use sample information effectively. Image representation of an APK contains most of the source code information, some of which may be critical for distinguishing a malware family while some others may be irrelevant, or could even be misleading. Using the feature of the entire image representation of a sample for classification, which is common in the existing approaches (Hsien-De Huang and Kao 2018; Xiao and Yang 2019), cannot reduce the influence of this irrelevant information. We want to propose a more effective way by capturing and placing more weight on the information that is more related to family labels.

Such information is usually difficult to justify given a single image sample but can get reflected by comparing samples within a same family, where there exist similarities. For an image sample s in the support set S, we make multiple random crops and compare all its crops to those from the corresponding positions of the given query sample q. Based on the similarity, each crop in s is assigned a weight and the average decides the sample's

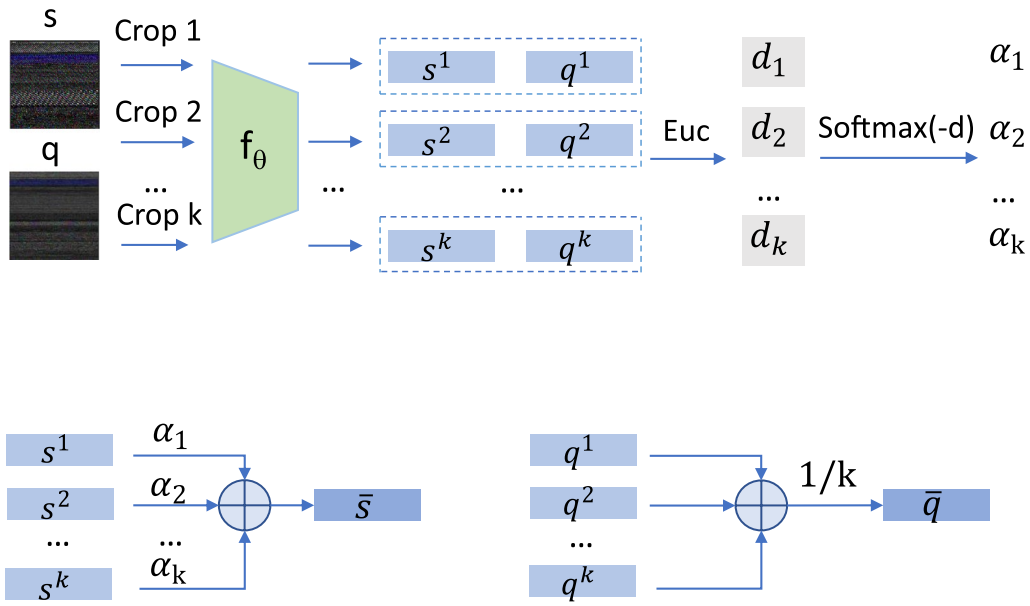


Fig. 3 The support set sample, represented by *s*, and the query sample, represented by *q*, are cropped identically *k* times. The weight of each crop for *s* is determined by its similarity to the corresponding crop of *q*. The feature vector of the query sample is obtained by taking the average of all the crops

feature vector. In so doing, those regions that are more similar, meaning they may more likely represent family label-related information, would be given more weights in the extracted feature of s. For cropping, considering the consecutive semantic information in the *classes.dex* file, we choose to crop consecutive rows and reshape them into squares to facilitate input into the backbone as shown in Algorithm 1.

Algorithm 1 Cropping algorithm

Input: S: Support set, Q: Query set, k: number of crops, b: size of crops
Output: $features_set_S$: the set of feature vectors for support images in S, $features_set_Q$: the set of feature vectors for query images in Q

backbone: Trained feature extraction network f_θ
generate_positions: This function generates random crop positions for a given image size, crop size, and the number of crops. Specifically, the function randomly selects n cropping positions within the width of the image, each of which is some consecutive rows whose size can be reshaped to b^2, and returns them as a list.
crop: This function crops a region with a specified size and position from a given input image. Specifically, the function crops a number of consecutive rows at a fixed position, then reshapes them into squares of size b and returns the result.
foreach q *in* Q **do**
    $pos \leftarrow$ generate_positions(q.shape, b, k);
    $crops_q \leftarrow$ [crop(q, b, pos) for p in pos];
    $features_q \leftarrow$ mean(backbone($crops_q$));
    $s_set_single_query \leftarrow$ [];
    foreach s *in* S **do**
        $crops_s \leftarrow$ [crop(s, $crop_size$, pos) for p in pos];
        $features_s \leftarrow$ backbone($crops_s$);
        $distances \leftarrow$ [euc($features_q[i]$, $features_s[i]$) for i in range(k)];
        $weights \leftarrow$ softmax(-array($distances$));
        $vector_s \leftarrow$ average($features_s$, weights=$weights$);
        $s_set_single_query$.append($vector_s$);
    end
    $features_set_S$.append($s_set_single_query$);
    $features_set_Q$.append($vector_q$);
end
return $features_set_S$, $features_set_Q$

As shown in Fig. 3, given samples $s \in S$ and $q \in Q$. We first apply k crops of the same positions to both the images and then send all the resulting crops to the backbone f_θ. This yields a set of vectors $\{s^1, s^2, \ldots, s^k\}$ for s, and a corresponding set $\{q^1, q^2, \ldots, q^k\}$ for q, where s^i and q^i denote the vectors obtained by the i-th crop of s and q, respectively. Let $d_j = ||s^j - q^j||_2$, and $\bar{s}$ denote the feature vector of s. Then:

$$\bar{s} = \sum_{j=1}^{k} \alpha_j s^j \tag{7}$$

, where α_j is defined as:

$$\alpha_j = \frac{exp(d_j)}{\sum_{v=1}^{k} exp(d_v)} \tag{8}$$

, Denote $\overline{q}$ the feature vector of the q, $\overline{q} = \frac{1}{k}\sum_{j=1}^{k} q^j$. More details are summarized as pseudo-code in Algorithm 1.

With different query samples, the feature vector of a support sample varies. Given a query sample q_j, the feature vectors of all the samples in S form a set denoted as $\overline{S^j}$. And for a family (class) $S_i : S_i \subseteq S$, we denote the set of the feature vectors of all its samples as $\overline{S_i^j} : \overline{S_i^j} \subseteq \overline{S^j}$.

Before the classification step, we apply two types of feature transformations as in Wang et al. (2019), Bendou et al. (2022) on feature vectors of all the samples in D_{novel}, including the query sample and the support samples. Given a query q_j, denote z^* the average of all the feature vectors in the set $\{\overline{S^j}, \overline{q_j}\}$. For an arbitrary feature vector $z : z \in \{\overline{S^j}, \overline{q_j}\}$, the operations are centering and L_2 normalization: $z_{CH} = \frac{z - z^*}{||z - z^*||_2}$. Centering does not change the Euclidean distance between individual eigenvectors but becomes effective if combined with L_2 normalization. These two feature transformations are straightforward but empirically have a positive impact on few-shot learners Wang et al. (2019).

Then for classification, we follow the method in ProtoNet Snell et al. (2017), which has been shown to be highly effective in the few-shot scenario. The basic idea is to represent each class (family) in the support set with a prototype and determine the class of a query sample based on its distance to all the prototypes. The prototype $\overline{c_i}$ of a family S_i, given the query q_j, is simply defined as the mean of the feature vectors of all its samples:

$$\overline{c_i} = \frac{1}{|S_i|} \sum_{z \in \overline{S_i^j}} z \tag{9}$$

Then the query sample q_j is classified into the family whose prototype is the closest:

$$C(q_j, [\overline{c_1}, \overline{c_2}, ...\overline{c_N}]) = \arg\min_i ||z - \overline{c_i}||_2 \tag{10}$$

Evaluation

In this section, we present the experimental results regarding the effectiveness of AndroFIM in detecting few-shot Android malware families. We also delve deeper into the rationality behind our multiple-cropping strategy and analyze the effect of hyper-parameters.

Experiment setup

Dataset

Given a sample, we need to decompile it to obtain its underlying *classes.dex* file and then convert it into an image. This requires the original APK files to be provided in the datasets we use. As far as we know, only the *Drebin* dataset Arp et al. (2014), *CICMalDroid 2020* Mahdavifar et al. (2022) and *CICInvesAndMal2019* Taheri et al. (2019) meet this condition. Other commonly used datasets like *AMD* Li et al. (2017) is no longer publicly available. Similarly, the link to download the *Faldroid* dataset Fan et al. (2018) does not provide access to the APK samples. In addition, some datasets (Damshenas et al. 2015; Lashkari et al. 2017) that can be obtained after feature extraction do not meet the condition either since we can only get the processed *.csv* files. Compared to *Drebin*, *CICMalDroid 2020* was released more recently, and we chose it to pre-train the backbone. It has sufficiently large sample size to fulfill the training requirements. *CIC-InvesAndMal2019* was chosen as the test phase dataset because its APKs are available and also its families fit the characteristics of few-shot families well: Each family has much fewer samples compared to that of *CICMalDroid 2020*. *CIC-InvesAndMal2019* has been utilized in other paper Ale et al. (2020) dealing with few-shot Android malware detection.

Specifically, our datasets consist of two parts. The unlabeled data D_{add} used in the pre-training phase is from *CICMalDroid 2020* Mahdavifar et al. (2022), which has a total of 17,341 APKs, including 12703 malware of different categories (1514 Adware, 2468 Banking trojans, 4817 SMS fraud, 3904 Riskware) and 4638 benign software. Note that the category information is not used in our self-supervision-based pre-training. The APKs for the fine-tuning phase and classification phase, i.e., D_{base} and D_{novel}, are from *CIC-InvesAndMal2019* Taheri et al. (2019), which contains 426 malware with family information (contains 42 unique malware families) and 5065 benign software. We consider those malware families with no less than 10 APKs, resulting in 35 malware families in total. We randomly shrink each of the 35 malware families such that there are only 10 APKs left, refer to Ale et al. (2020). Then we follow the regular dataset classification ratio Chai et al. (2022), and divide them into a training set, a validation set and a test set. Finally, there are 22 families in the training set (i.e. D_{base}), 6 families in the validation set, and 7 families in the test set (i.e. D_{novel}).

Implementation details

For the backbone, we several well-known models commonly used in the Android field, i.e., CONV4 Ganesh et al. (2017), LSTM Xiao et al. (2019) and ResNet-12 Bendou et al. (2022) respectively. CONV4 has fewer parameters and is the most classic Convolutional Neural Network (CNN) model in the few-shot learning domain (Vinyals et al. 2016; Snell et al. 2017). ResNet-12 makes a good balance between model complexity and performance (Bendou et al. 2022; Snell et al. 2017; Boudiaf et al. 2020). The images transformed from the

classes.dex file have sequence features and context structure information, regarding which LSTM has great advantages. LSTM can remember important information from earlier parts of the sequence. For the pre-training phase, we use data augmentation with color jitters, Gaussian blur, and solarization Grill et al. (2020) and multi-crop Caron et al. (2021) refer to the default settings Caron et al. (2021). We train with the SGD optimizer and a batch size of 64. The learning rate is set 0.03 and decayed with a cosine schedule Loshchilov and Hutter (2016). The backbone is pre-trained for 100 epochs. For the fine-tuning phase, we resize the input images into 200×200 pixels. We use a cosine-annealing scheduler Loshchilov and Hutter (2016), which updates the learning rate as follows. When the learning rate is less than zero, the Adam optimizer is used; otherwise, the SGD optimizer is used instead. We freeze the parameters of the first convolutional layer at the fine-tuning phase. The backbone is fine-tuned for 40 epochs, and we evaluate the few-shot learning performance after 30 epochs. We also adopt the dropout mechanism for the fine-tuning phase, and the dropout parameter is set to 0.1. For the classification phase, we set the number of crops k to 20, and crop size b to 180×180 pixels (such choices are based on the hyperparameter experiments in the next section). For each task, we set $N = 5$ and $K = \{1, 5\}$, i.e., the number of classes is 5, and the number of samples per class is 1 or 5, which are commonly used settings for few-shot learning Jiang et al. (2020). we use $q = 5$ query samples per family and results are averaged over 10,000 tasks. PyTorch Paszke et al. (2017) and NVIDIA 3090 GPUs are used throughout our experiments.

Table 1 The 5-way 1-shot, and 5-shot classification accuracy (%) compared with some classical machine learning methods (C) and few-shot learning methods. For few-shot learning methods, we list the type they belong to, i.e., optimization-based methods (O) and metric-based methods (M)

Model	Type	5-way 1-shot	5-way 5-shot
Classical machine learning methods			
Intents + Permissions +KNN	C	63.57%	71.42%
API Calls + NetworkFlows + KNN	C	54.63%	63.32%
Color image + KNN	C	59.42%	65.64%
Few-shot learning methods			
MAML (CONV4)	O	61.28%	68.06%
MetaOptNet (ResNet12)	O	61.98%	77.32%
Transductive-finetuning (ResNet12)	O	67.64%	76.62%
LoCoOp (ViT-B/16)	O	74.92%	80.54%
FSMC (CONV4)	M	59.21%	67.82%
Matching Network (ResNet12)	M	62.41%	72.63%
Prototypical Network (ResNet12)	M	70.86%	79.55%
Relation Network (ResNet12)	M	64.52%	75.17%
TIM (ResNet12)	M	74.92%	80.01%
EASY (ResNet12)	M	75.77%	80.76%
PUTM (ResNet18)	M	76.47%	81.42%
AndroFIM (CONV4)	M	73.89%	79.17%
AndroFIM (LSTM)	M	74.78%	81.46%
AndroFIM (ResNet12)	M	79.63%	85.12%

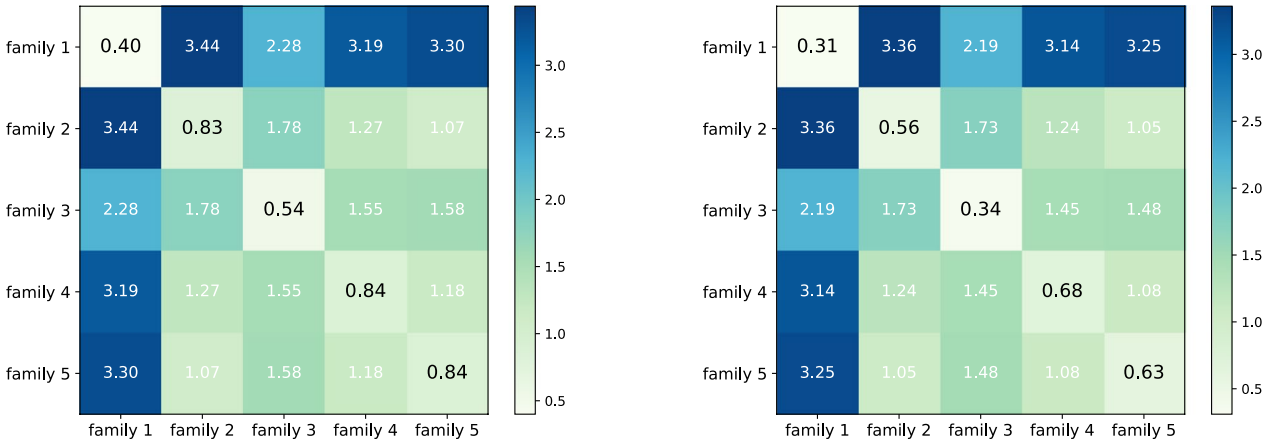
Baselines

Many existing machine learning-based Android malware detection methods require a large amount of data for training to achieve high performance, and to the best of our knowledge, few-shot Android malware family detection has not yet been extensively explored. We formulate a couple of baselines in this regard to compare against AndroFIM: (1) classic machine learning methods, Some non-parametric machine learning methods, like K-Nearest Neighbors (KNN) algorithm Taheri et al. (2019), are not dependent on a large amount of data for training, which means they may be suitable for the few-shot scenarios; (2) FSMC, FSMC achieves few-shot Android malware family detection by using API calls as features to train a Matching Network Vinyals et al. (2016; 3) few-shot learning methods that target at other domains like image domain as baselines, which are categorized into optimization-based methods (MAML Finn et al. 2017, MetaOptNet Lee et al. 2019, Transductive-finetuning Dhillon et al. 2019, LoCoOp Miyai et al. 2023 etc.) and metric-based methods (FSMC Ale et al. 2020, Matching network Vinyals et al. 2016, Prototypical Network Snell et al. 2017, Relation network Sung et al. 2018, TIM Boudiaf et al. 2020, and EASY Bendou et al. 2022, PUTM Tian et al. 2023 etc.). Regarding feature extraction, both FSMC and the classical machine learning methods require manual feature extraction, while AndroFIM and the other few-shot learning methods use images as input and train the model in an end-to-end manner, automatically learning the feature information from the samples.

Experiment results and analasis

Effectiveness

We compare AndroFIM to other baseline methods and report the average accuracy of 5-way 1-shot and 5-way 5-shot settings, which are very commonly used scenarios in few-shot learning. Three models are tried to pre-train the backbone. From Table 1, we can observe that when ResNet12 is used, AndroFIM achieve 79.63 and 85.12% accuracy in 5-way 1-shot and 5-way 5-shot, respectively. The accuracy achieved with CONV4 or LSTM as the backbone is not promising. We conjecture the reason is that relatively low number of



(a) Before crop (b) After crop

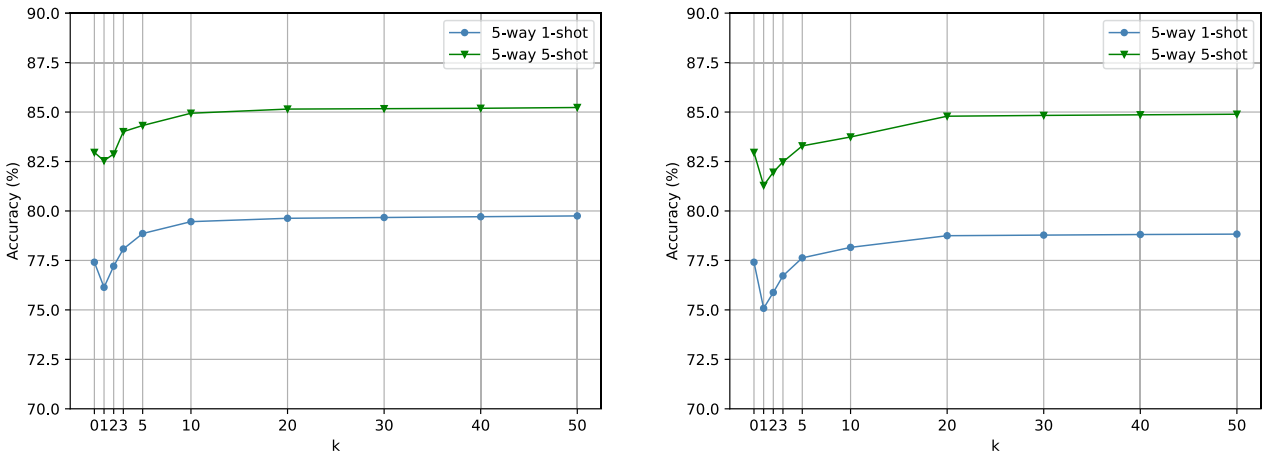
Fig. 4 Average intra-class sample distance and average inter-class sample distance before and after crop

parameters in CONV4 results in a slightly insufficient feature extraction capability. Additionally, using LSTM to convert images into pixel sequences may cause information loss, as pixel sequences cannot capture the hierarchical and semantic information in images. We thus choose to use ResNet-12 as the default backbone in AndroFIM for subsequent analysis.

As shown in Tabel 1, AndroFIM has achieved the highest detection accuracy among all the methods. Compared with the best classic machine learning-based methods, our method has an average accuracy increase of 16.06% in the 5-way 1-shot scenario and increase of 13.70% in the 5-way 5-shot scenario. Compared with the best few-shot learning methods, AndroFIM improves the average accuracy by 3.16% in the 5-way 1-shot scenario and by 3.7% in the 5-way 5-shot scenario.

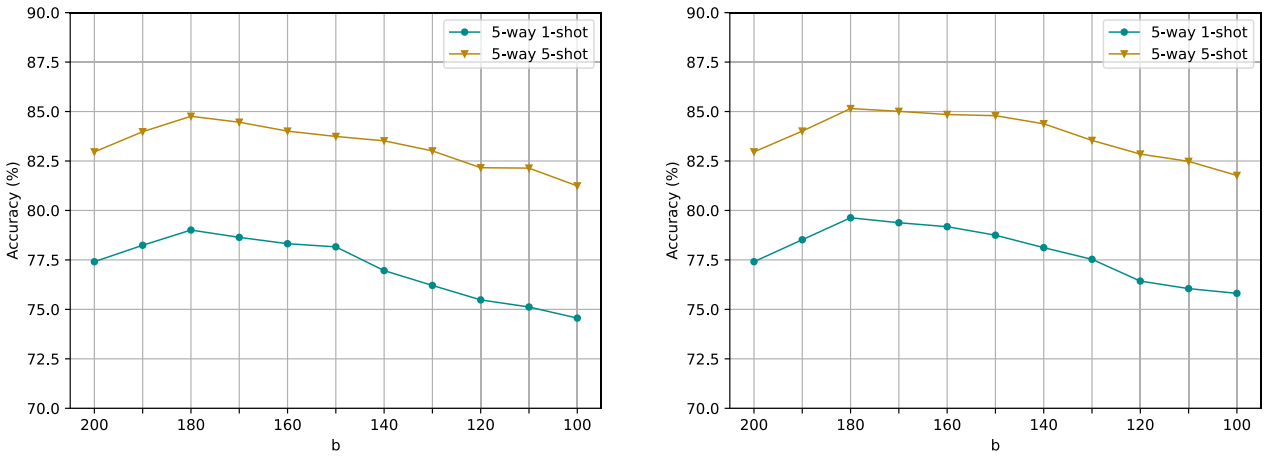
Analysis of multiple cropping strategy

It is commonly perceived that classification performance can be improved by minimizing the distance between samples of the same family and maximizing the distance between samples of different families in the feature space. Following this principle, we verify the effectiveness of our multiple cropping strategy. We select 5 families



(a) crop size $b = 180$ (b) crop size $b = 150$

Fig. 5 Impact of crop number



(a) crop number $k = 10$ (b) crop number $k = 20$

Fig. 6 Impact of crop size

Table 2 The average test phase running time before and after cropping of AndroFIM under different experimental settings

Experimental setting	5-way 1-shot	5-way 5-shot
With cropping	6.54 s	8.08 s
Without cropping	5.52 s	6.83 s

(AndroidDefender, fakeAV, Fakemart, Fakenotify, Zsone, and later for brevity, we directly use family index 1–5 to refer to them). Given two image samples, we crop their same regions and use the smallest distance among all the crop pairs to represent their distance. We then calculate the average distance between samples in the same family (i.e., intra-class distance) and the average distance between samples of different families (i.e., inter-class distance) before and after cropping, as shown in Fig. 4.

We observe that after cropping, both the average intra-class sample distance and the average inter-class sample distance have decreased, which is not surprising as the distance calculated with cropping captures more the similarity between samples. More importantly, we found that the average intra-class sample distance decreased more than the inter-class sample distance. This is because when a part of an image from the same family is cropped and the distance is calculated multiple times, the cropped part with the minimum distance is likely to be a common family feature, which may result in a greater reduction in distance. However, for images from different families, there are significant differences between them, and it is unlikely to find similarities even with cropping, resulting in less reduction in the distance between them. In summary, our proposed multiple cropping strategy effectively captures family label-related information, resulting in a significant decrease in the intra-class distance and accordingly an improvement in the detection performance.

Influence of hyper-parameters k, b

Usually, the number of crops k and their size b can affect the accuracy and efficiency of the detection. We design the hyperparametric experiment to analyze the effect of these two hyperparameters. The result is shown in Fig. 5. $k = 0$ means that there is no cropping.

As shown in Fig. 5, the model's accuracy improves as the number of crops k increases. However, a small drop in performance is observed when switching from a global ($k = 0$) reshape to small crop mode ($k = 1, 2$), especially at small crop size b due to randomness stronger. This decline can be attributed to the possibility of crops missing similar family information, which can have a negative impact on the model's accuracy. Also, the computation time for generating crops also increases linearly. We suggest choosing k in the range of [15,25], within which both high accuracy and efficiency can be balanced. To discuss the impact of the size of crops as shown in Fig. 6, we set $k = 10, 20$. The original image size is 200×200. As crop size b decreases, the overall detection accuracy increases first and then decreases. This suggests that if the crop size is too large, it may not effectively capture the key information of the image, while if the crop size is too small, it may result in the loss of important information. Additionally, when the value of k is 20, the decrease in model performance caused by excessively small crop sizes becomes less steep. This aligns with our previous analysis, which suggests that excessively small crop sizes are

more prone to random variation and that a greater number of crops can help compensate for this performance loss. In this case, we suggest choosing b in the range of [175, 185] to strive for a balance between crop size and amount of information.

In addition, the application of the multiple cropping strategy, although it improves performance, also leads to an increase in overhead. We computed the running time of the algorithm test phase before and after cropping when $k = 20$ and $b = 180$, as presented in Table 2. It can be observed that the average running time before and after cropping has increased by 1−1.5 s, and the algorithm efficiency has decreased by approximately 20%. Nevertheless, we think that this reduction in efficiency can be acceptable when the recognition accuracy is significantly enhanced.

Limitations and future work

Though effective, there is still room to improve AndroFIM. First, transforming all source code to images does save a lot of cost compared with where manual feature extraction is used, but it also reduces the interpretability of the model. Second, compared to directly feeding the D_{novel} into the backbone to obtain the feature vector, AndroFIM incorporates cropping, which enhances the model's effectiveness in capturing more important information of samples. However, this requires the model to calculate the feature vectors of the support samples for each query, which inevitably increases the computational burden.

In the future, to address these issues, we suggest to employ semantic analysis of the underlying source code to generate more interpretable images and also to explore the use of other lightweight networks to optimize the efficiency of the model.

Conclusion

In this paper, we propose AndroFIM, a novel pipeline for few-shot Android malware family detection. We pretrain the backbone with unlabeled APKs through self-supervised learning and fine-tune it using a standard splitting loss and an auxiliary rotation loss to enhance generalization. To address the few-shot problem, we use a metric-based few-shot learning method and propose a multiple cropping strategy to crop continuous rows of the malware image before feature extraction. By comparing their similarity after feature extraction to the query, we determine their weight. Feature vectors of all the samples of a family are averaged to derive its prototype, and the distance to which decides the class of the query sample thus we complete the classification by a metric-based few-shot learning approach. Experimental results show that AndroFIM outperforms the existing methods in solving the few-shot Android malware family detection problem.

Acknowledgements
There is no any third person/ organisation to acknowledge.

Authors' contributions
All the authors read and approved the final manuscript.

Funding
This work was supported by National Natural Science Foundation of China (NSFC) under grant 62106223 and Provincial Key R&D Program of Zhejiang(2020C01038).

Availability of data and materials
The datasets analysed during the current study are available on the site Canadian Institute for Cybersecurity, CICMalDroid 2020 (https://www.unb.ca/cic/datasets/maldroid-2020.html) and CIC-InvesAndMal2019 (https://www.unb.ca/cic/datasets/invesandmal2019.html).

Declarations

Competing interests
The authors declare no competing interests.

Received: 8 February 2024 Accepted: 6 January 2025
Published online: 16 September 2025

References

Ale L, Li L, Kar D, et al (2020) Few-shot learning to classify android malwares. In: 2020 IEEE 5th International Conference on Signal and Image Processing (ICSIP), IEEE, pp 1001–1007

Arp D, Spreitzenbarth M, Hubner M, et al (2014) Drebin: Effective and explainable detection of android malware in your pocket. In: NDSS, pp 23–26

Bai Y, Xing Z, Li X, et al (2020) Unsuccessful story about few shot malware family classification and siamese network to the rescue. In: 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), IEEE, pp 1560–1571

Bendou Y, Hu Y, Lafargue R, et al (2022) Easy: Ensemble augmented-shot y-shaped learning: State-of-the-art few-shot classification with simple ingredients. arXiv preprint arXiv:2201.09699

Boudiaf M, Ziko I, Rony J et al (2020) Information maximization for few-shot learning. Adv Neural Inf Process Syst 33:2445–2457

Caron M, Misra I, Mairal J et al (2020) Unsupervised learning of visual features by contrasting cluster assignments. Adv Neural Inf Process Syst 33:9912–9924

Caron M, Touvron H, Misra I, et al (2021) Emerging properties in self-supervised vision transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 9650–9660

Chai Y, Du L, Qiu J et al (2022) Dynamic prototype network based on sample adaptation for few-shot malware detection. IEEE Trans Knowl Data Eng 35(5):4754–4766

Chen WY, Liu YC, Kira Z, et al (2019) A closer look at few-shot classification. arXiv preprint arXiv:1904.04232

Conti M, Khandhar S, Vinod P (2022) A few-shot malware classification approach for unknown family recognition using malware feature visualization. Comput Secur 122:102887

Damshenas M, Dehghantanha A, Choo KKR et al (2015) M0droid: an android behavioral-based malware detection model. J Inf Privacy Secur 11(3):141–157

Dhillon GS, Chaudhari P, Ravichandran A, et al (2019) A baseline for few-shot image classification. In: International Conference on Learning Representations

Fan M, Liu J, Luo X et al (2018) Android malware familial classification and representative sample selection via frequent subgraph analysis. IEEE Trans Inf Forensics Secur 13(8):1890–1905
Fan Y, Ju M, Hou S, et al (2021) Heterogeneous temporal graph transformer: An intelligent system for evolving android malware detection. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp 2831–2839
Fei-Fei L, Fergus R, Perona P (2006) One-shot learning of object categories. IEEE Trans Pattern Anal Mach Intell 28(4):594–611
Finn C, Abbeel P, Levine S (2017) Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning, PMLR, pp 1126–1135
Ganesh M, Pednekar P, Prabhuswamy P, et al (2017) Cnn-based android malware detection. In: 2017 International Conference on Software Security and Assurance (ICSSA), IEEE, pp 60–65
Google (2019) Android security & privacy 2018 year in review. https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf
Gordon MI, Kim D, Perkins JH, et al (2015) Information flow analysis of android applications in droidsafe. In: NDSS, p 110
Grill JB, Strub F, Altché F et al (2020) Bootstrap your own latent-a new approach to self-supervised learning. Adv Neural Inf Process Syst 33:21271–21284
Guo Y, Codella NC, Karlinsky L, et al (2020) A broader study of cross-domain few-shot learning. In: European Conference on Computer Vision, Springer, pp 124–141
Han Q, Subrahmanian V, Xiong Y (2020) Android malware detection via (somewhat) robust irreversible feature transformations. IEEE Trans Inf Forensics Secur 15:3511–3525
He K, Fan H, Wu Y, et al (2020) Momentum contrast for unsupervised visual representation learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 9729–9738
Hinton G, Vinyals O, Dean J (2015) Distilling the knowledge in a neural network. stat 1050:9
Hsien-De Huang T, Kao HY (2018) R2-d2: Color-inspired convolutional neural network (CNN)-based android malware detections. In: 2018 IEEE International Conference on Big Data (Big Data), IEEE, pp 2633–2642
Hu SX, Li D, Stühmer J, et al (2022) Pushing the limits of simple pipelines for few-shot learning: External data and fine-tuning make a difference. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 9068–9077
Jiang Z, Kang B, Zhou K, et al (2020) Few-shot classification via adaptive attention. arXiv preprint arXiv:2008.02465
Lashkari AH, Kadir AFA, Gonzalez H, et al (2017) Towards a network-based framework for android malware detection and characterization. In: 2017 15th Annual Conference on Privacy, Security and Trust (PST), IEEE, pp 233–23309
Lazarou M, Stathaki T, Avrithis Y (2021) Iterative label cleaning for transductive and semi-supervised few-shot learning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 8751–8760
Lee K, Maji S, Ravichandran A, et al (2019) Meta-learning with differentiable convex optimization. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp 10657–10665
Li Y, Jang J, Hu X, et al (2017) Android malware clustering through malicious payload mining. In: Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings, Springer, pp 192–214
Loshchilov I, Hutter F (2016) Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983
Mahdavifar S, Alhadidi D, Ghorbani A et al (2022) Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder. J Netw Syst Manag 30(1):1–34
Mangla P, Kumari N, Sinha A, et al (2020) Charting the right manifold: Manifold mixup for few-shot learning. In: Proceedings of the IEEE/CVF winter conference on applications of computer vision, pp 2218–2227
Miyai A, Yu Q, Irie G, et al (2023) Locoop: Few-shot out-of-distribution detection via prompt learning. arXiv preprint arXiv:2306.01293
Palmer D (2022) What is ransomware? everything you need to know about one of the biggest menaces on the web. https://www.zdnet.com/article/ransomware-an-executive-guide-to-one-of-t he-biggest-menaces-on-the-web/, accessed on August 1, 2022
Paszke A, Gross S, Chintala S, et al (2017) Automatic differentiation in pytorch
Rong C, Gou G, Hou C, et al (2021) Umvd-fsl: Unseen malware variants detection using few-shot learning. In: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, pp 1–8
Snell J, Swersky K, Zemel R (2017) Prototypical networks for few-shot learning. Advances in Neural Information Processing Systems 30
Sung F, Yang Y, Zhang L, et al (2018) Learning to compare: Relation network for few-shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 1199–1208
Taheri L, Kadir AFA, Lashkari AH (2019) Extensible android malware detection and family classification using network-flows and api-calls. In: 2019 International Carnahan Conference on Security Technology (ICCST), IEEE, pp 1–8
Tian L, Feng J, Chai X, et al (2023) Prototypes-oriented transductive few-shot learning with conditional transport. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 16317–16326
Tian Y, Wang Y, Krishnan D, et al (2020) Rethinking few-shot image classification: a good embedding is all you need? In: European Conference on Computer Vision, Springer, pp 266–282
Tripp O, Rubin J (2014) A bayesian approach to privacy enforcement in smartphones. In: 23rd USENIX Security Symposium (USENIX Security 14), pp 175–190
Verma V, Lamb A, Beckham C, et al (2019) Manifold mixup: Better representations by interpolating hidden states. In: International Conference on Machine Learning, PMLR, pp 6438–6447
Vinyals O, Blundell C, Lillicrap T, et al (2016) Matching networks for one shot learning. Advances in Neural Information Processing Systems 29
Wang Y, Chao WL, Weinberger KQ, et al (2019) Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. arXiv preprint arXiv:1911.04623
Xiao X, Zhang S, Mercaldo F et al (2019) Android malware detection based on system call sequences and LSTM. Multimed Tools Appl 78:3979–3999
Xiao X, Yang S (2019) An image-inspired and cnn-based android malware detection approach. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, pp 1259–1261
Xue L, Zhou Y, Chen T, et al (2017) Malton: Towards {On-Device} {Non-Invasive} mobile malware analysis for {ART}. In: 26th USENIX Security Symposium (USENIX Security 17), pp 289–306
Zhang L, Thing VL, Cheng Y (2019) A scalable and extensible framework for android malware detection and family attribution. Comput Secur 80:120–133

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.