Research on Data Drift and Class Imbalance in Android Malware Detection

Zhen Liu[1], Ruoyu Wang[2(✉)], Bitao Peng[1], Changji Wang[1], and Qingqing Gan[1]

[1] School of Information Science and Technology, Guangdong University of Foreign Studies, Guangzhou 510006, China

[2] Information and Network Engineering Research Center, South China University of Technology, Guangzhou 510041, China

rywang@scut.edu.cn

Abstract. In the Android ecosystem, malware detection is still a nontrivial task. Existing works have recently applied convolution neural networks (CNNs) for detecting Android malwares. However, data drift and class imbalance are still open problems in this field. The distribution of malware data may vary significantly if data are represented by unstable features, leading to data drift problems. The model may not be able to effectively detect malwares on the future data. In addition, the class imbalance may degrade a model on identifying a specific type of malwares with fewer training samples. To handle both of the two problems, this paper presents a new Android malware detection framework. Specifically, we devise a data distribution-aware feature learning framework for learning features with a stable distribution to handle data drift. We further devise a new loss function for CNN to handle the class imbalance problem. Using our loss function, this model can reinforcement learn the minority class samples and hard samples. The experimental results on the real datasets revealed that our method outperforms existing works for Android malware detection on the datasets with data drift and class imbalance problems.

Keywords: Android malware detection · data drift · class imbalance · feature learning · CNN

1 Introduction

According to statistics published by statcounter [1], Android owned a 71.96% mobile OS market in November 2022. The popular usage of Android mobile devices [2] also attracts threats from mobile malwares [3, 4]. These threats include stealing user privacy information, automatically triggering deductions, controlling the system, etc. [4], which causes massive harm to mobile users. The AV-TEST Institute reported more than 90 million new malicious apps in 2022 [6].

To protect mobile apps from being attacked, various techniques have been proposed for detecting mobile malwares [5]. Static analysis methods [5] mainly obtain string

A. Zaslavsky et al. (Eds.): MobiQuitous 2023, LNICST 593, pp. 429–444, 2024.
https://doi.org/10.1007/978-3-031-63989-0_22

information (such as permission, opcode, API calls, etc.) from the decompiled dex and manifest.xml. Reverse engineering techniques are usually used for decompiling. Dynamic analysis methods [7] collect data (such as network traffic) while running apps in an isolation environment. Static analysis and dynamic analysis can also be used together for malware detection [5].

Recent works have focused on utilizing CNN for training an Android malware detection model [8]. For example, Rahali et al. [9] utilized a feature learning and a CNN for training a malware detection model. The string features are transformed into images as the input of CNN. However, the app data distribution would be changed with the upgrading of apps or changing in attack behaviors when the data are represented by unstable features. The trained model may become ineffective on the samples with data shift. In addition, there are a few samples from some malware families (i.e., minority classes). The model trained with the objective of obtaining high overall accuracy may obtain high accuracy for classifying the majority class samples but low accuracy for classifying the minority class samples. This means that Android malware detection also faces the class imbalance problem. To address both of the two problems, this paper proposes a new Android malware detection framework. Our main contributions are as below.

(1) Our proposed android malware detection method is based on a data distribution-aware feature learning and a margin loss (MDFM). It mainly includes feature learning and model training stages.
(2) In the feature learning stage, we propose a feature learning algorithm named Data Distribution Aware Feature Learning (DDAFL). It learns the features with low data distribution discrepancy among the app samples and with high discriminations among classes on the target samples.
(3) In the training stage, we propose a new loss function named M-LDAM by modulating the LDAM loss function [13]. It aims at reinforcement learning the hard samples and minority class samples.
(4) To evaluate the performance of MDFM, we carry out multiple experiments on real datasets. Results show that MDFM performs better in most cases, especially on multiclass datasets with unstable data distributions.

The remainder of this paper is organized as follows. Section 2 overviews related works in Android malware detection field. Section 3 presents our proposed method for detecting Android malwares. Section 4 analyzes the results of Android malware detection. This paper is concluded in Sect. 5.

2 Related Work

2.1 Related Work on CNN-Based Malware Detection Methods

In the field of CNN-based Android malware detection, some studies convert [10, 26] the opcodes from the binary codes of Apks into images and then utilize CNNs to train a malware detection model. Ren et al. [27] proposed to apply the raw bytecodes of Apks as the input of deep neural networks, including CNN and RNN (recurrent neural network). CNN-based and RNN-based methods achieve approximately 93.4% and 95.8% accuracy respectively. Rong et al. [28] proposed an Android malware detection

method named TransNet for handling the data shift. It is based on a deep transfer learning algorithm and applies transferable normalization in their deep neural network. A classification accuracy of 95.89% and an Fscore of 96.09% are obtained by their model on two public datasets. Vasan et al. [10] proposed a method named IMCFN, in which CNN is pretrained on images from ImageNet, and then the pretrained model is finetuned by malware samples. The binary codes of Apks are converted into color images, which will be used for finetuning the pretrained model. IMCFN obtains 97.35% accuracy on IoT-android samples.

Recently, DIDroid [9] was presented for malware detection. It firstly selects features using an extremely randomized tree classifier and then converts the vectors represented by selected features into 2D gray images, which are used for training the CNN model. It obtains 93% accuracy in detecting the 12 categories of Android malwares. In contrast to other image-based methods, DIDroid [9] is based on the images transferred from feature vectors (such as API calls, Permissions, etc.) rather than the images on the binary codes.

Our work is similar to DIDroid, and the benchmark dataset (i.e., CIC2020 [11]) used in DIDroid is also used in this paper. We also reshape the app data with static features into 2D gray images. In contrast to DIDroid, we propose a new data distribution-aware feature learning algorithm that aims to learn stable and discriminate features and to present an LDAM-based loss function for CNN that aims to reinforcement learn minority class samples and hard samples.

2.2 Related Work on Handling the Class Imbalance Problem

As malicious samples are difficult to obtain, malicious samples are fewer than benign samples. The trained model may be biased towards the majority class (i.e., the benign class). This paper mainly utilizes the CNN as the classification algorithm and handles the class imbalance from the aspect of improving the loss function. To handle the class imbalance in CNNs, a variety of loss functions have been proposed. Lin et al. [12] proposed the focal loss. This method is based on modulating the cross entropy loss to reinforce learning the hard samples. Kim et al. [16] proposed CCE (complement cross entropy) loss, which is designed to penalize the model heavily for incorrect predictions. Ye et al. [17] proposed a class-dependent temperature (CDT) loss function that encourages the model to balance its predictions across all classes. Cao et al. [13] proposed LDAM (label distribution aware margin) loss. This loss encourages higher margins for samples from minority classes with the aid of label distribution information to improve the performance of classifying minority classes. Existing works prove that the two factors of label distribution and classification probability are useful for building the loss function for CNN. However, they mainly only consider one of the two factors. This paper pays attention on both of the two factors based on LDAM.

3 Proposed Android Malware Detection Method

This paper utilizes CNN [14, 15] as the malware classification algorithm. This is because that the convolution operation requires less parameters than fully connected networks, which is suitable for the data in high dimension and the sparse data [23, 25]. The features

of apps can be obtained from APIs, activities, intents, permission strings, etc., which may lead to high dimensionality. The values of these features are set according to the occurrence of the corresponding features. This means that the feature vectors of apps are high dimension and sparse. Therefore, the CNN is suitable for the app data.

3.1 The Framework of Our Method

In Android malware detection, the data distribution drifts with the changing of attack behaviors and the updating of apps. In addition, malware samples are difficult to obtain. As a result, the malware class has fewer samples than the benign class. Android malware detection also faces the class imbalance problem. For handling data drift and class imbalance problems, we propose a new method named Malware Detection with data distribution-aware Feature learning and Margin loss (MDFM).

The architecture of MDFM is shown in Fig. 1. There are two stages, including feature learning and model training. In the first stage, DDAFL (data distribution-aware feature learning) is devised for learning features with a stable data distribution. It is inspired by the MFSAN [18], which is designed to learn the invariant features for cross-domain feature alignment. The stable features are helpful for enhancing malware detection performance. In the second stage, the output of conv block1 is the data represented by the learned features. These data will be further flattened and reshaped and then fed into the CNN model in Fig. 1. To handle the class imbalance problem, the loss function named M-LDAM is devised and used for CNN training.

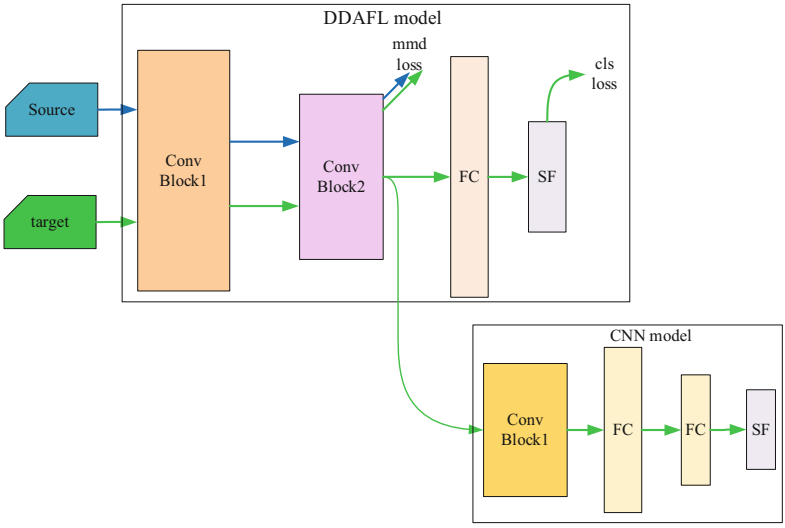


Fig. 1. The neural network architecture of MDFM

3.2 Data Distribution-Aware Feature Learning

The neural network framework of DDAFL is shown as the "DDAFL model" in Fig. 1. Conv Block1 includes three layers of conv nets. Each conv net includes a convolutional layer, batch normalization and pooling. The kernel size is 3*3 in each convolutional layer. The numbers of kernels of the three layers are 32, 64 and 128. The pooling size is 2*2. The activation function of ReLu is used for each convolutional layer. Conv Block2 includes three layers of conv nets, which learn high-level features without changing the feature dimension. Each conv net also includes a convolutional layer and batch normalization. The number of kernels is 256 in each convolutional layer. Three conv nets are followed by a pooling layer with a size of 7*7 to finally decrease the number of features.

Here, we utilize a source dataset of unlabeled samples for feature learning of the samples in the target dataset. This means that other sources of unlabeled samples could be used for feature learning. The samples from other sources are also in the domain of malware detection but from the dataset collected by other research groups. In addition, our method does not rely on the class label information. This implies that datasets from other research groups (such as unlabeled datasets shared in public) could be used to improve feature learning on target samples.

We assume that the source dataset is denoted by S and the target dataset is denoted by T. Given a batch image $\mathbf{x}_{si}$ from the source sample set $\{\mathbf{X}_{si}\}$ and a batch image x_t from the target sample set $\{\mathbf{X}_t, \mathbf{Y}_t\}$, the two batches are fed into the network of Fig. 1. Assume that F denotes conv Block1 and H denotes conv Block2, H(F($\mathbf{x}_{si}$)) denotes the output H with input of the ith source sample set $\mathbf{x}_{si}$. Similarly, the output of the H network of target sample set $\mathbf{x}_t$ is denoted by H(F($\mathbf{x}_t$)).

DDAFL is devised to learn the stable and discriminative features. This means that we hope that the data distribution of learned features is stable. Therefore, one objective of DDAFL is to minimize the data distribution discrepancy between S and T. The maximum mean discrepancy (mmd) [24] metric is generally applied to evaluate the data distribution discrepancy between S and T. Another objective of DDFAL is to minimize the classification loss for learning discriminative features on T with class label information. Therefore, the loss function of DDFAL is defined as

$$\mathcal{L}_{total} = \mathcal{L}_{cls} + \beta \mathcal{L}_{mmd} \tag{1}$$

where $\mathcal{L}_{mmd}$ denotes the mmd loss, and $\mathcal{L}_{cls}$ denotes the classification loss. Given observations $\mathcal{D}_s := \{x_1^s, x_2^s, ..., x_{ns}^s\}$ and $\mathcal{D}_t := \{x_1^t, x_2^t, ..., x_{nt}^t\}$. They are from the data distributions of P and Q, respectively. The mmd [24] is defined as Eq. (2). Based on the observed samples $\mathcal{D}_s$ and $\mathcal{D}_t$, it is a kernel two-sample test on the null hypothesis P = Q.

$$D_{\mathcal{H}}(P, Q) := \|\mathrm{E}_P[f(x^s)] - \mathrm{E}_Q[f(x^t)]\|_{\mathcal{H}}^2 \tag{2}$$

$\mathrm{E}_P[f(x^s)]$ and $\mathrm{E}_Q[f(x^t)]$ denote expectations with respect to P and Q; f denotes a feature map that maps the original samples to reproducing kernel Hillber space ($\mathcal{H}$) [24]. In reality, the unbiased estimation of $D_{\mathcal{H}}$ is defined as Eq. (3) based on the observations.

The mmd loss is defined as Eq. (4).

$$\widehat{D}_{\mathcal{H}}(P, Q) = \| \frac{1}{n_s} \sum\nolimits_{\mathbf{x}_i \in \mathcal{D}_s} f(x_i) - \frac{1}{n_t} \sum_{\mathbf{x}_j \in \mathcal{D}_t} f(x_j) \|^2_{\mathcal{H}} \tag{3}$$

$$\mathcal{L}_{mmd} = \frac{1}{N} \sum\nolimits_{j=1}^{N} \widehat{D}\big(H\left(F\left(X_{sj}\right)\right), H(F(X_t))\big) \tag{4}$$

In Fig. 1, FC denotes the fully collected layer, and SF denotes softmax. Let G denote the network (i.e., the FC layer) following H, which aims at obtaining the specific target classification loss. We apply the cross-entropy as the classification loss defined as

$$\mathcal{L}_{cls} = J(G(\mathrm{H}(\mathrm{F}(X_t))), Y_t) \tag{5}$$

The learned features are obtained by the output of the *conv Block*2 layer in DDFAL. The samples represented by learned features are used to train an Android malware detection model using CNN, as shown in Fig. 1. These features can also be used with other machine learning algorithms.

The pseudocode of DDAFL is shown in Algorithm 1. First, it randomly samples m images from S and T, obtaining $\mathbf{x}_s$ and $(\mathbf{x}_t, \mathbf{y}_t)$, respectively. $\mathbf{x}_s$ and $\mathbf{x}_t$ are sequentially fed into *convBlock*1, obtaining $\mathrm{F}(\mathbf{x}_s)$ and $\mathrm{F}(\mathbf{x}_t)$, respectively. The two outputs are further fed into *convBlock*2, obtaining $\mathrm{H}(\mathrm{F}(\mathbf{x}_s))$ and $\mathrm{H}(\mathrm{F}(\mathbf{x}_t))$. Then, it calculates the mmd loss between $\mathrm{H}(\mathrm{F}(\mathbf{x}_s))$ and $\mathrm{H}(\mathrm{F}(\mathbf{x}_t))$ according to Eq. (4). $\mathrm{H}(\mathrm{F}(\mathbf{x}_t))$ is fed into the FC layer. The classification loss is calculated according to Eq. (5). The parameters of neural networks are updated by minimizing the defined loss. The output of Algorithm 1 is the samples from dataset T represented by the learned features. The output is denoted by NT.

Algorithm 1 Data distribution-aware feature learning algorithm.

Input: source datasets $S=\{X_s\}$ and target dataset $T=\{X_t, Y_t\}$, *batch_size*
Output: dataset NT represented by learned features
for i in range (*#inters*)
  $\mathbf{x}_s$=*RS*(S, *batch_size*)
  $(\mathbf{x}_t, \mathbf{y}_t)$ =*RS*(T, *batch_size*)
  mmd_{Loss}=L_{mmd}(H(F($\mathbf{x}_s$)), H(F($\mathbf{x}_t$)))
  cls_{Loss}=L_{cls}(G(H(F($\mathbf{x}_t$))), $\mathbf{y}_t$)
  L_{total}=mmd_{Loss}+$\beta cls_{\mathrm{Loss}}$
  Update the parameters of the DDAFL network by SGD optimization
end for
return NT = H(F(X_t))

3.3 M-LDAM Loss Function

In this paper, the CNN is used as the algorithm for building the malware classification model. We address the class imbalance problem from the aspect of reinforcement learning the hard samples by modifying the loss function of CNN. Our proposed loss function

is based on AM-SoftMax loss [19] and LDAM [13]. Based on the softmax loss [20], the model predicts the sample as class C_1 when

$$|\omega_1||x|cos(\theta_1) > |\omega_2||x|cos(\theta_2) \tag{6}$$

On the class imbalanced dataset, the model may learn less knowledge from the minority class because the minority class has a small number of samples. We can reinforce the learning on the minority class samples by modifying the loss function. In order to reinforce the learning on the samples of a class C_g, the cross entropy loss is modified to reduce the output of C_g. It is defined as

$$\mathcal{L}_{M-LDAM} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\cdot(\hat{y}_g - m_g)}}{e^{s\cdot(\hat{y}_g - m_g)} + \sum_{j=1, j\neq g}^{k} e^{\hat{y}_j}} \tag{7}$$

m_g is set the same for the majority and minority classes [20]. For handling the class imbalance problem, m_g is defined as Eq. (8) in LDAM [13] to make the model biased toward learning the minority samples.

$$m_g = \frac{(1/n_j^{1/4})C}{max_{j\in\{1,...,k\}}(1/n_j^{1/4})} \text{for } j \in \{1, ..., k\} \tag{8}$$

where n_j denotes the sample size of class C_j, k is the number of classes, and C is a hyperparameter to be tuned. However, Eq. (8) is only correlated with the sample size of each class. In the malware detection field, instances are usually sampled from existing apps. The label distribution in the training set may not be the real label distribution in existing apps. In multiclass classification, there is more than one minority class. In addition, some minority class samples may be easily identified if they are far from the boundary. Therefore, the class imbalance problem is also influenced by the difficulty of classification. To address this problem, this paper imports the information of the difficulty level of each class. The new m_g is defined as Eq. (9). Therefore, we obtain a new margin loss with the newly defined m_g, which is named as modulated LDAM (M-LDAM).

$$m_g = \frac{(1-p_j)(1/n_j^{1/4})C}{max_{j\in\{1,...,k\}}((1-p_j)(1/n_j^{1/4}))} \text{for } j \in \{1, ..., k\} \tag{9}$$

p_j is the model's classification probability of a model on class C_j. It denotes the classification confidence of correctly classifying a sample. If its value is small, the confidence is small and it is more difficult to correctly classify. Therefore, $(1 - p_j)$ is introduced to denote the classification difficulty. If $(1 - p_j)$ is high, sample x_j is difficultly identified. It is obtained by the average over the $(1 - p_j)$ obtained on the samples of C_j. $(1 - p_j)$ is used to further modulate the margin. This means that the margin of the class with higher p_j will be further reduced to reinforcement learning the samples that are easily to classify, otherwise to reinforcement learning the samples that are difficulty to classify.

4 Experiments

4.1 Experimental Datasets

In this paper, three benchmark datasets downloaded from public databases are used in our experiments. Two datasets are publicly shared by CIC (Canadian Institute for Cybersecurity) [11], and one dataset is the OmniDroid [21].

4.1.1 CIC Datasets

The two CIC datasets are CIC-InvesAndMal2019 (CIC2019 in short) and CICMalDroid 2020 (CIC2020 in short). In the CIC2019 dataset, the features include permissions and intents, which are from AndroidManifest.xml. In total, there are 8110 features in this dataset. These malware samples fall into five categories. The detail of the class distribution is shown in Table 1. In the CIC2020 dataset, the benign apps were collected from Androzoo. There are 12 malware categories, as shown in Table 2. The main extracted features include activities and permissions. The details of the features can be found in [9]. In total, there are 9503 features in CIC2020 dataset. All features can be obtained by the static analysis method, i.e., extracted from apks before running them.

CIC2019 and CIC2020 datasets have multiple classes. The details of the two datasets are shown in Tables 1 and 2. The label distribution shows that the benign class has more samples than any one of the malware classes. The two datasets face the class imbalance problem.

Table 1. Details of the CIC2019 dataset

Classes	#Samples	Classes	# Samples
Adware	95	Ransomware	93
Benign	1187	SMS	61
PremiumSMS	47	Scareware	111

4.1.2 OmniDroid Dataset

The samples in the OmniDroid dataset [21] were collected from Koodous and AndroZoo. To handle the class imbalance problem, the authors in [21] sampled 11,000 samples for malware and benign classes. The number of samples is balanced between the two classes. These samples are also represented by features obtained by static analysis, including API calls, permissions, activities and so on. The performance of different combinations of features was compared in [21]. The results show that the model with the API and Permission features performs the best. Therefore, the two feature sets are applied in following experiments.

Table 2. Details of the CIC2020 dataset

Classes	#Samples	Classes	# Samples
Adware	47197	Ransomware	6201
Backdoor	1537	Riskware	97348
Banker	886	SMS	3124
Benign	122577	Scareware	1555
Dropper	2301	Spy	3539
FileInfector	668	Trojan	13541
PUA	2050		

4.2 Experimental Results

To evaluate the performance of our proposed method, this section will carry out a variety of experiments as below.

(1) Experiment on comparing different loss functions: To handle the class imbalance problem in the CNN algorithm, multiple loss functions have been presented, including focal loss [12], CCE (complement cross entropy) [16], CDT (class-dependent temperature) [17], and LDAM (label distribution aware of margin loss) [13]. All of them are compared with our loss function M-LDAM.
(2) Experiment on comparing different CNN-based malware detection methods: To evaluate the performance of MDFM, it is compared with previous related methods, including model fine-tuning, TransNorm, and DIDroid. These methods are based on the CNN algorithm. Fine-tuning and TransNorm handle the data drift from different aspects.

In MDFM, the hyper parameters of CNN architecture are introduced in Sect. 3.2. The hyper parameters for training CNN model are set as follows: the optimization function is SGD with the learning rate of 0.01, the number of epochs is 200, and early stopping is used to avoid the overfitting problem. The other parameters in MDFM are set as follows: the batch size is 128; β in the loss function of feature learning is 0.01.

In these experiments, the *accuracy*, *Fscore* and *g-mean* are used as performance evaluation metrics. On each of the three datasets, the split ratio of training and test is 0.5. In the following experiments, each result is obtained by averaging over the ten runs of the corresponding experiment.

4.2.1 Comparison on Loss Functions

Previous works proposed multiple loss functions (CCE, FL, CDT and LDAM) for handling class imbalance problem. This section carries out experiments to compare different loss functions on handling class imbalance. The MDFM with different loss functions on the three datasets are shown in Tables 3 and 4, Fig. 2. CE denotes the cross-entropy loss function that is used as the baseline for comparison.

The results show that M-LDAM obtains 83.6%, 85.6% and 93.3% *accuracy* on the OmniDroid, CIC2019 and CIC2020 datasets, respectively. It obtains 83.6%, 52.5% and 76.8% *g-mean* on the three datasets, respectively. The results show that FL, LADM and M-LDAM perform better than other loss functions in handling class imbalance. This indicates that the existing methods of FL and LDAM perform better than others in the field of malware detection. M-LDAM performs much better than FL and LDAM on the CIC2019 and CIC2020 datasets in terms of *g-mean*. M-LDAM improves the *g-mean* by approximately 0.1%, 7.2% and 0.9% on the OmniDroid, CIC2019 and CIC2020 datasets, respectively, when compared with LDAM. This shows that the improvement of M-LDAM on the multiclass datasets is more significant than that on the binary class dataset. On the CIC2019 dataset, the *Fscores* of Adware and SMS are much worse than those of other classes. M-LDAM improves the *Fscore* of the two classes by approximately 1.6% and 13.8%, respectively. Compared with LDAM, on the CIC2020 dataset, M-LDAM decreases the *Fscores* of some minority classes, but it also improves the *Fscores* of some other classes. On average, it improves the *Fscores* of the minority classes (all classes except Adware and Benign classes) by approximately 1.8% on the CIC2020 dataset.

Table 3. Loss function comparison results on the OmniDroid dataset

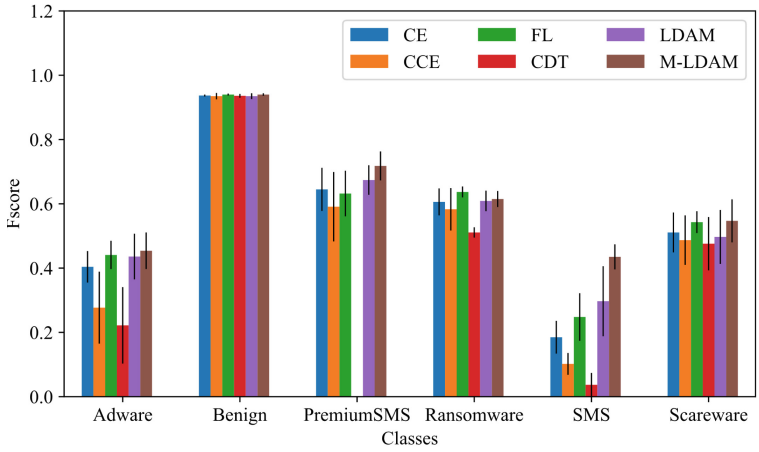
Metrics	Accuracy	Fsorce-M	Fscore-B	G-mean
CE	0.827 ± 0.004	0.823 ± 0.004	0.831 ± 0.005	0.827 ± 0.004
CCE	0.826 ± 0.005	0.823 ± 0.003	0.829 ± 0.007	0.826 ± 0.005
FL	0.831 ± 0.004	0.831 ± 0.003	0.831 ± 0.004	0.831 ± 0.004
CDT	0.827 ± 0.004	0.825 ± 0.003	0.829 ± 0.006	0.827 ± 0.004
LDAM	0.835 ± 0.002	**0.834 ± 0.003**	0.837 ± 0.003	0.835 ± 0.002
M-LDAM	**0.836 ± 0.002**	0.833 ± 0.003	**0.839 ± 0.003**	**0.836 ± 0.002**

Table 4. Loss function comparison results on the CIC2019 and CIC2020 dataset

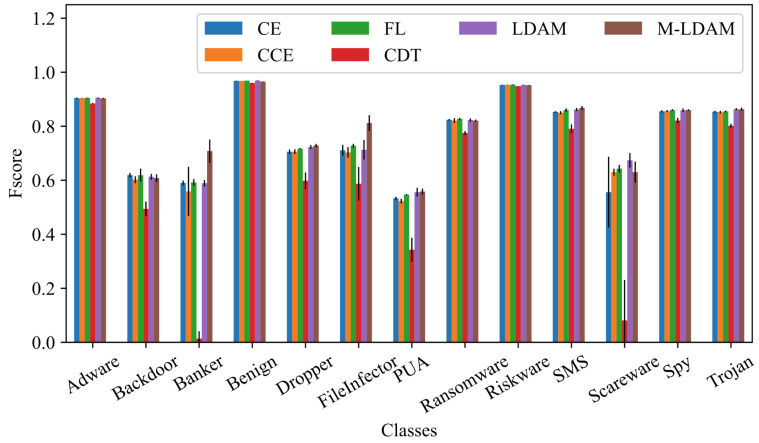
Metrics	CIC2019		CIC2020	
	Accuracy	G-mean	Accuracy	G-mean
CE	0.848 ± 0.007	0.401 ± 0.031	0.933 ± 0.001	0.711 ± 0.025
CCE	0.837 ± 0.014	0.316 ± 0.034	0.933 ± 0.001	0.710 ± 0.018
FL	0.853 ± 0.006	0.449 ± 0.036	**0.935 ± 0.001**	0.735 ± 0.004
CDT	0.819 ± 0.011	0.00 ± 0.00	0.917 ± 0.001	0.193 ± 0.194
LDAM	0.849 ± 0.012	0.453 ± 0.07	0.934 ± 0.001	0.759 ± 0.007
M-LDAM	**0.856 ± 0.008**	**0.525 ± 0.029**	0.933 ± 0.002	**0.768 ± 0.007**

The class label information and the model estimated probability are introduced into the loss function of M-LDAM. There are multiple minority classes on CIC2019 and

CIC2020. The label distribution among those minority classes cannot well reflect the difficulty of classifying those minority classes. The model estimated probability could reflect the difficulty. The difficult class with much fewer samples is double reinforced by M-LDAM to improve the performance of the minority classes.



(a)CIC2019



(b)CIC2020

Fig. 2. The *Fscore* obtained by MDFM with different loss functions on the CIC2019 and CIC2020 datasets

4.2.2 Comparison on Malware Detection Methods

Our method is compared with CNN, DIDroid, fine-tuning CNN and TransNorm. To fairly compare all methods, CNN is used as the basic classification algorithm in all malware detection methods. The neural network architecture of the CNN is shown as the CNN model part in Fig. 1. Using our method, the pretrained model on one source of data is

fine-tuned on target source of data. All data are from the field of malware detection. For example, the model pretrained on the CIC2020 can be fine-tuned on the OmniDroid, so as to adapt to the malware detection on OmniDroid data. The reason is that latent feature learned by the hidden layers of CNN can be used by the target domain of malware detection. The DIDroid is performed as in [9]. In the DIDroid method, the extremely randomized trees method is used for feature selection, and the data represented by low dimension features are used for training the CNN model. In TransNorm, the batch norm in CNN is replaced by transferable normalization [22].

Table 5. Comparison results of different models on OmniDroid

Metric	CNN	DIDroid	Fine-tuning	TransNorm	MDFM
Accuracy	0.824 ± 0.002	0.804 ± 0.006	0.731 ± 0.007	0.812 ± 0.004	$\mathbf{0.836 \pm 0.002}$
Fsorce-M	0.822 ± 0.002	0.810 ± 0.006	0.756 ± 0.005	0.807 ± 0.006	$\mathbf{0.833 \pm 0.003}$
Fscore-B	0.827 ± 0.002	0.797 ± 0.009	0.701 ± 0.012	0.816 ± 0.006	$\mathbf{0.839 \pm 0.003}$
G-mean	0.824 ± 0.002	0.814 ± 0.000	0.730 ± 0.000	0.811 ± 0.004	$\mathbf{0.836 \pm 0.002}$

Table 6. Comparison results of different models on CIC2019

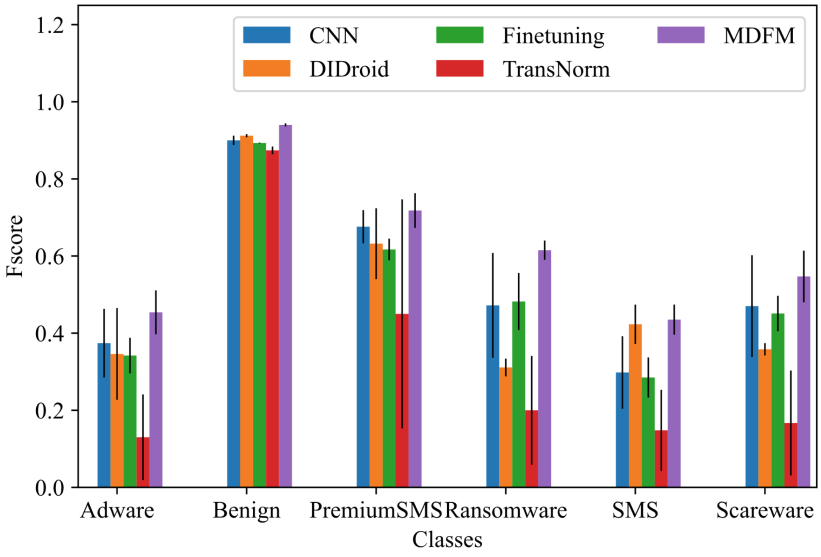
Metric	CNN	DIDroid	Fine-tuning	TransNorm	MDFM
Accuracy	0.821 ± 0.017	0.808 ± 0.009	0.796 ± 0.003	0.780 ± 0.019	$\mathbf{0.856 \pm 0.008}$
G-mean	0.390 ± 0.097	0.376 ± 0.046	0.411 ± 0.051	0.162 ± 0.116	$\mathbf{0.525 \pm 0.029}$

Table 7. Comparison results of different models on CIC2020

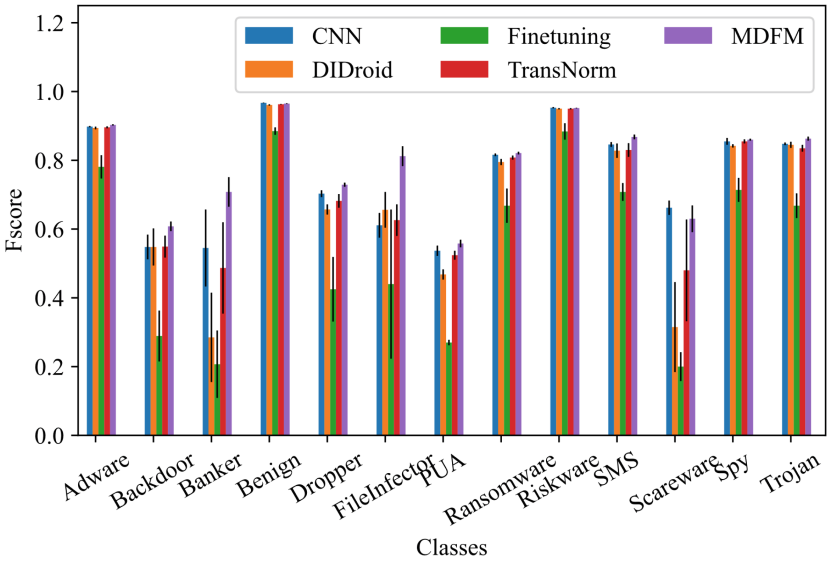
Metric	CNN	DIDroid	Fine-tuning	TransNorm	MDFM
Accuracy	0.930 ± 0.001	0.924 ± 0.004	0.841 ± 0.015	0.926 ± 0.002	$\mathbf{0.933 \pm 0.002}$
G-mean	0.744 ± 0.025	0.615 ± 0.091	0.369 ± 0.047	0.688 ± 0.037	$\mathbf{0.768 \pm 0.007}$

The results obtained on the OmniDroid dataset are shown in Table 5. The results (*accuracy* and *g-mean*) on the CIC2019 and CIC2020 datasets are shown in Tables 6 and 7, respectively, and their *Fscores* are shown in Fig. 3. The results show that MDFM obtains higher *accuracy* and *g-mean* than other methods on the three datasets, especially on the CIC2019 dataset. It improves the *accuracy* and *g-mean* by approximately 1.7% and 5.7%, respectively, when compared with CNN. Figure 3(b) also shows that MDFM obtains higher *recall* for the minority classes of Dropper, FileInfector and PUA. DIDroid also utilizes the feature reduction method before training the CNN model. It applies extremely randomized trees to select features. This paper presents a new data distribution-aware feature learning algorithm for decreasing the feature dimension. The experimental results imply that it outperforms extremely randomized trees.

The results show that the performance of the fine-tuning model is not good. The possible reason is that the data distribution of the target data is much different from that of the source data used for pretraining a model [25]. In such situations, retraining the model (i.e., the CNN model shown in above tables) performs much better. When



(a)CIC2019



(b)CIC2020

Fig. 3. The *Fscore* obtained by different methods on the CIC2019 and CIC2020 datasets

retraining the model, we also utilized the samples from other sources to learn the invariant features that improve the classification performance.

5 Conclusion and Future Work

This paper proposes a new method named MDFM for handling data drift and class imbalance in Android malware detection. To handle the data drift problem, we propose a new data distribution-aware feature learning algorithm named DDAFL. It aims to learn discriminative and stable features. To handle class imbalance, we devised a loss function named M-LDAM for CNN to handle the class imbalance problem. It pays more attention to learn the minority class samples and hard samples. Our method only relies on the assembled data of apps for malware detection. Therefore, it can be used for detecting android malwares before installing apps on the mobile devices. We carry out multiple experiments to compare our method with previous methods. The results are summarized as follows.

(1) The experimental results on comparing loss functions show that M-LDAM can improve the performance of LDAM on the multiclass imbalance problem. This is because it not only considers the label distribution but also considers the classification difficulty. Therefore, it can reinforce the training of the hard samples and minority samples.
(2) The experimental results on comparing MDFM with existing malware detection methods show that MDFM performs the best in detecting malwares. DIDroid is very similar to our work. In contrast to DIDroid, we propose a new feature learning method rather than existing tree-based feature learning. Our feature learning method aims at learning the features with stable distribution and discrimination among classes.

In this paper, we mainly consider the mmd between source data and target data to choose the source data for our feature learning. In the future, we will further analyze how to choose samples from other data sources. Our method mainly used a CNN with three conv blocks, and a CNN with more layers and neurons (such as ResNet50) could be used for DDAFL in the future.

Acknowledgments. We thank the anonymous reviewers for their constructive comments. This work is supported by the Starting Research Fund from the Guangdong University for Foreign Studies [Grant Nos. 2022RC049, 2021RC375, 2022RC100], Guangdong Basic and Applied Basic Research Foundation [Grant No. 2022A1515110980], Guangzhou Science and Technology Project [Grant No. 202201010100], Science Foundation of Ministry of Education of China [Grant No. 23YJAZH106], Guangdong Province Higher Education Teaching Reform Project.

References

1. Statcounter: Mobile operating system's market share worldwide (2022). https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed 26 Dec 2022

2. Statista: Number of apps available in leading app stores (2022). https://www.statista.com/statistics/289418/number-of-available-apps-in-the-google-play-store-quarter/. Accessed 26 Dec 2022
3. Crowdstrike: What is mobile malware? (2022) https://www.crowdstrike.com/cybersecurity-101/malware/mobile-malware/. Accessed 26 Dec 2022
4. Guo, Y., Yan, Q.: Android malware adversarial attacks based on feature importance prediction. Int. J. Mach. Learn. Cybern. **2022**, 1–11 (2022)
5. Sharma, T., Rattan, D.: Malicious application detection in android-A systematic literature review. Comput. Sci. Rev. **40**, 100373 (2021)
6. AV-Test: AV-Test Institute report (2022). https://www.av-test.org/en/statistics/malware/. Accessed 26 Dec 2022
7. Ananya, A., Aswathy, A., Amal, T.R., Swathy, P.G., Vinod, P., Mohammad, S.: SysDroid: a dynamic ML-based android malware analyzer using system call traces. Clust. Comput. **23**, 2789–2808 (2022)
8. Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., Xiang, Y.: A survey of android malware detection with deep neural models. ACM Comput. Surv. **53**(6), 126 (2020)
9. Rahali, B., Lashkari, A.H., Kaur, G., Taheri, L., Gaganon, F.: DIDroid: android malware classification and characterization using deep image learning. In: the 10th International Conference on Communication and Network Security (ICCNS 2020), pp. 70–82 (2020)
10. 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. **171**, 107138 (2020)
11. CIC datasets (2020). https://www.unb.ca/cic/datasets/index.html. Accessed 25 May 2022
12. Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: IEEE International Conference on Computer Vision (ICCV), pp. 2999–3007 (2017)
13. Cao, K., Wei, C., Gaidon, A., Arechiga, A., Ma, T.: Learning imbalanced datasets with label-distribution-aware margin loss. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 1567–1578 (2019)
14. Abdel-Hamid, O., Mohamed, A.-R., Jiang, H., Deng, L., Penn, G., Dong, Y.: Convolutional neural networks for speech recognition. IEEE/ACM Trans. Audio Speech Lang. Process. **22**(10), 1533–1545 (2014). https://doi.org/10.1109/TASLP.2014.2339736
15. Lan, S., He, Z., Chen, W., Chen, L.: Hand gesture recognition using convolutional neural networks. In: USNC-URSI Radio Science Meeting (Joint with AP-S Symposium), pp. 147–148 (2018)
16. Kim, Y., Lee, Y., Jeon, M.: Imbalanced image classification with complement cross entropy. Pattern Recogn. Lett. **151**, 33–40 (2021)
17. Ye, H.-J., Chen, H.-Y., Zhan, D.-C., Chao, W.-L.: Identifying and compensating for feature deviation in imbalanced deep learning. arXiv:2001.01385 (2020)
18. Zhu, Y., Zhuang, F., Wang, D.: Aligning domain-specific distribution and classifier for cross-domain classification from multiple sources. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 5989–5996 (2019)
19. Wang, F., Cheng, J., Liu, W., Liu, H.: Additive margin softmax for face verification. IEEE Signal Process. Lett. **25**(7), 926–930 (2018)
20. Liu, W., Wen, Y., Yu, Z., Yang, M.: Large-margin softmax loss for convolutional neural networks, In: Proceedings of the 33rd International Conference on Machine Learning, pp. 1–10 (2016)
21. Martín, A., Lara-Cabrera, R., Camacho, D.: Android malware detection through hybrid features fusion and ensemble classifiers: the AndroPyTool framework and the OmniDroid dataset. Inf. Fusion **52**, 128–142 (2019)

22. Wang, X., Jin, Y., Long, M., Wang, J., Jordan, M.I.: Transferable normalization: toward improving transferability of deep neural networks. In: Advances in Neural Information Processing Systems, pp. 1951–1961 (2019)
23. AjitkumarSingh, S., GautamMeitei, T., Majumder, S.: Short PCG classification based on deep learning. Deep Learn. Tech. Biomed. Health Inf. **2020**, 141–164 (2020)
24. Gretton, A., Borgwardt, K.M., Rasch, M.J., Scholkopf, B., Smola, A.: A kernel two- sample test. JMLR **13**(Mar), 723–773 (2012)
25. Liu, Z., Wang, R., Peng, B., Gan, Q.: A convolutional neural network based Android malware detection method with dynamic fine-tuning. In: 32nd ITNAC, pp. 300–305 (2022)
26. Yadav, P., Menon, N., Ravib, V., Vishvanathan, S., Pham, T.D.: EfficientNet convolutional neural networks-based Android malware detection. Comput. Secur. **115**, 102622 (2022)
27. Ren, Z., Wu, H., Ning, Q., Hussain, I., Chen, B.: End-to-end malware detection for Android IoT devices using deep learning. Ad Hoc Netw. **101**(4), 102098 (2020)
28. Rong, C., Gou, G., Cui, M., Gang, X., Li, Z., Guo, L.: TransNet: unseen malware variants detection using deep transfer learning. In: International Conference on Security and Privacy in Communication Systems, pp. 84–101 (2020)