基于卷积神经网络恶意安卓应用行为模式挖掘

张鑫，羌卫中，吴月明，邹德清，金海
（华中科技大学网络空间安全学院，湖北 武汉 430074）

摘　要：现有的安卓恶意应用检测方法所提取的特征冗余且抽象，无法在高级语义上反映恶意应用的行为模式。针对这一问题，提出一种可解释性检测方法，通过社交网络检测算法聚类可疑系统调用组合，将其映射为单通道图像，用卷积神经网络进行分类，并利用卷积层梯度权重类激活映射可视化方法发现最可疑的系统调用组合，从而挖掘理解恶意应用行为。实验结果表明，所提方法在高效检测的基础上，能够正确发现恶意应用的行为模式。

关键词：安卓；快速检测；卷积神经网络；社交网络分析
中图分类号：TP393
文献标识码：A
doi: 10.11959/j.issn.2096−109x.2020073

Mining behavior pattern of mobile malware with convolutional neural network

ZHANG Xin, QIANG Weizhong, WU Yueming, ZOU Deqing, JIN Hai
School of Cyber Science & Engineering, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract: The features extracted by existing malicious Android application detection methods are redundant and too abstract to reflect the behavior patterns of malicious applications in high-level semantics. In order to solve this problem, an interpretable detection method was proposed. Suspicious system call combinations clustering by social network analysis was converted to a single channel image. Convolution neural network was applied to classify Android application. The model trained was used to find the most suspicious system call combinations by convolution layer gradient weight classification activation mapping algorithm, thus mining and understanding malicious application behavior. The experimental results show that the method can correctly discover the behavior patterns of malicious applications on the basis of efficient detection.

Key words: Android, rapid detection, convolutional neural network, social network analysis

---

收稿日期：2020−01−05；**修回日期**：2020−06−20
通信作者：羌卫中，wzqiang@hust.edu.cn
基金项目：国家自然科学基金（61772221）；国家重点研发计划（2017YFB0802205）
Foundation Items: The National Natural Science Foundation of China (61772221), The National Key Research & Development (R&D) Plan of China (2017YFB0802205)

论文引用格式：张鑫, 羌卫中, 吴月明, 等. 基于卷积神经网络恶意安卓应用行为模式挖掘[J]. 网络与信息安全学报, 2020, 6(6): 35-44.
ZHANG X, QIANG W Z, WU Y M, et al. Mining behavior pattern of mobile malware with convolutional neural network[J]. Chinese Journal of Network and Information Security, 2020, 6(6): 35-44.

1 引言

谷歌最新发布的安卓生态安全报告[1]显示，截至 2019 年 10 月，安卓设备中感染潜在有害应用程序（PHA，potentially harmful applications）的比例居高不下，至少 1 470 万台安卓设备面临用户数据泄露风险。新型恶意应用层出不穷，为了抵御恶意应用，谷歌应用商店每天扫描和验证的应用程序多达 500 亿次。

迄今，多种安卓恶意应用检测系统被提出。具有自主学习能力的方法，如基于签名、敏感权限的机器学习检测方法和基于字节码、操作码的深度学习检测方法，单纯提取抽象字节特征，这些方法很容易被基于 Jacobian 的攻击以及混淆技术所躲避[2]，所以这些系统的鲁棒性较差。进而，基于图匹配（控制流图和调用图）的检测系统被提出，但图的相似性度量是 NP 难题，随着移动应用的体积增长，图相似性计算会产生难以接受的性能开销。更重要的是，为了尽可能地提高检测精度，先前的工作往往过量寻求特征，数万维乃至数十万维的冗余特征底层抽象，无法在高级语义上反映恶意应用的行为，所以，无法对研究和发掘理解恶意特征行为模式提供支持。

调用图类似于社交网络，恶意行为相关的敏感 API 之间具有密集的连接，但与良性 API 间连接稀疏。因此，本文首先利用自启发式社交网络检测快速分割调用图，并根据社区敏感度定位敏感社区；然后将社区映射为规则图像，并保留恶意应用原始高级语义；最后设计并训练对应图像尺寸的卷积神经网络（CNN，convolutional neural network）模型，在高效率检测恶意应用的同时，利用卷积层梯度权重类激活映射[3]（Grad-CAM，gradient-weighted class activation mapping）的可视化方法得到 API 调用图像的热力图，从而找出当前应用最可疑的 API 调用组合，为恶意应用特征行为模式领域的研究提供高级素材。

本文的贡献如下。

1) 提出了一种新颖的安卓恶意应用检测系统，利用社交网络检测和统计规律快速分析 API 调用图提取特征，输入卷积神经网络进行快速分类，在保证鲁棒性的同时有效提升了系统检测效率。

2) 保留了恶意应用的原始高级语义，利用已训练模型，通过卷积层梯度权重可视化来挖掘恶意应用最可疑的 API 调用组合，为研究恶意应用特征行为模式提供参考。

3) 使用 13 365 个良性应用和 13 514 个恶意应用对原型系统进行了全面评估。实验结果表明，系统平均可以在 23 s 内以高达 93%的准确率完成应用检测，并找到恶意应用中最可疑的 API 调用组合。

2 相关工作

很多安卓恶意应用检测的方法被提出。大多数方法基于机器学习寻找尽可能多的特征向量来提升检测准确率。Drebin[4]使用多种静态分析，从应用程序中提取尽可能多的特征，并将它们嵌入联合特征向量中以对安卓应用程序进行分类。但该方法容易通过混淆技术躲避。因此，基于图的方法被提出。DroidSIFT[5]提取加权上下文 API 依赖关系图，以解决恶意应用变种问题。Apposcopy[6]利用静态分析提取应用程序的数据流和控制流属性来识别其恶意软件家族。但是 DroidSIFT[5]和 Apposcopy[6]运行时开销巨大，分别花费 175.8 s 和 275 s 来分析应用程序。MaMaDroid[7]利用从调用图获得的抽象函数调用序列来构建行为模型，并使用其提取频率特征以进行分类，但由于其庞大的功能集和调用图的提取，它在分类检测时需要大量的内存[7]。

少数基于深度学习的方法被提出。文献[8]将二进制应用程序转换为灰度图像，依据图像上纹理布局的区别来区分不同的恶意应用家族。文献[9]进一步提取特征，将字节码映射为三通道的 RGB 图像，然后利用卷积神经网络模型进行分类。

但以上方法所提取特征向量，无论是权限属性、数据流属性、调用序列频率还是字节码图像，都是大量难以理解的底层抽象信息，无法帮助研究人员直观上获取恶意应用的高级特征语义信息。与先前的工作不同，本文保留高级语义信息，直接将敏感 API 调用的组合映射为特征图像，使已训练模型能够提供恶意应用的可疑 API 调用信息，并利用社交网络检测算法来处理调用图，有效避免了混淆攻击并大幅度降低了处理开销，提

升了检测效率。

3　系统模型

3.1　系统总览

本文提出了一个新颖系统，利用卷积神经网络对未知安卓应用进行检测分类，进而依托训练的模型，运用卷积层梯度权重类激活映射可视化算法来理解恶意应用的恶意行为模式。

总体来说，整个系统的操作分为 5 个阶段，如图 1 所示。首先，使用反编译静态分析从应用程序中提取 API 调用图，并在其中标记出敏感 API 调用；然后，选用低开销的社交网络检测算法将调用图划分为不同的社区；之后，在良性和恶意数据集上进行统计计算，得到每个敏感 API 的敏感度，而后按照社区敏感度紧凑排列敏感 API，构建为规则尺寸（如 4×800 ）的图像；接着，用卷积神经网络进行分类训练；最后，利用卷积层梯度权重类激活映射的可视化算法输出样本特征图像对应的热力图，并找出对应的恶意程度最高的一组系统调用及其相互关系，从而挖掘理解恶意应用的特征行为模式。

3.2　提取调用图

APK 文件是由 classes.dex 文件中的 Dalvik 代码和资源文件打包而成的。本文利用工具 Apktool[10]将 Dalvik 代码反汇编为可读的 smail 中间代码，并通过扫描所有.smail 文件中的 invoke 语句来解析 API 之间的调用关系，然后利用 SuSi[11]系统报告所总结的敏感 API 列表对 API 进行标记。且这类系统敏感 API 无法被混淆，而用户自定义的函数名通常被混淆为低熵字符串等。该列表共包含 25 156 个敏感 API，分为两种敏感类型，即 Source 和 Sink。其中，Source 类是来获取敏感信息的，有 17 类别共 17 372 个；Sink 类是来接收敏感信息的，有 19 类别共 7 784 个。API 被解析为格式{<包名: 返回值 方法名(方法参数)>，方法类型，方法权限，方法类别}。

其中，包名和方法名构成一个 API 的实体，唯一标识一个 API。方法类型分为 Source、Sink 和 Normal。方法权限即该敏感 API 申请的安卓权限。方法类别为 API 所属不同敏感类别，按照 API 的功能划分为 26 类。上述格式的每个 API 作为一个节点，API 间的调用关系作为有向边，构成了对应 APK 的 API 调用图。

3.3　社交网络检测

考虑到对大型网络的有效处理，本文选用 Louvain[12]算法。它采用一种无监督的自启发式的划分方法。Louvain[12]算法能够快速处理网络，依赖于简化的模块化计算公式，节点 i 移动到社区而产生的模块化系数的增量计算如下。

$$\Delta Q=\left[\frac{\Sigma_{\text{in}}+k_{i,\text{in}}}{2m}-\left(\frac{\Sigma_{\text{tot}}+k_i}{2m}\right)^2\right]-\left[\frac{\Sigma_{\text{in}}}{2m}-\left(\frac{\Sigma_{\text{tot}}}{2m}\right)^2-\left(\frac{k_i}{2m}\right)^2\right] \tag{1}$$

其中，Σ_{in} 是社区 C 内边的权重之和，Σ_{tot} 是连接到社区 C 的外部节点的边权重之和，k_i 是连接到节点 i 的边的权重之和，$k_{i,\text{in}}$ 是从节点 i 在社区 C 内中所连接的边的权重之和，而 m 是网络中全部

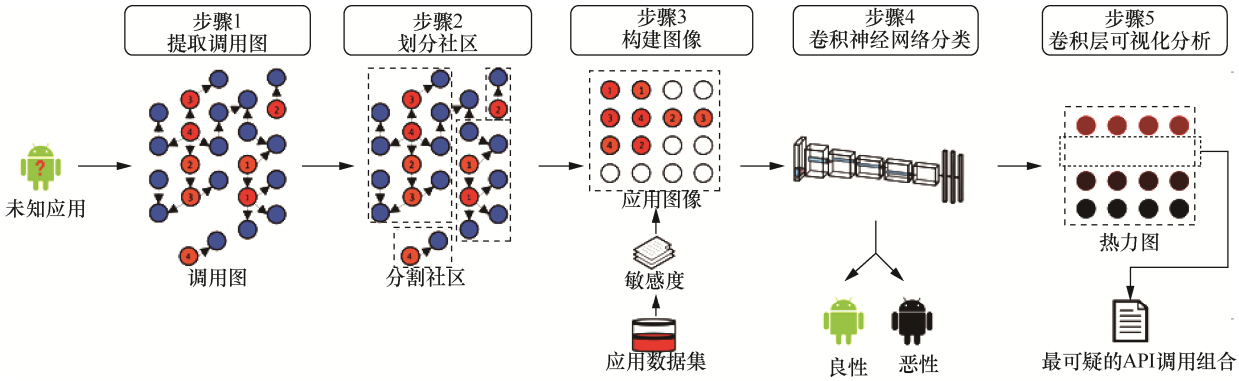

图 1　系统总览

Figure 1　System overview

边的权重之和。

为了尽可能地贴近真实情况，本文从谷歌应用商店中随机抓取了样本，并在 2012—2018 年间每个年份选取良性和恶意样本各 120 个（共计 1 920 个）用于社交网络检测的统计分析。

上述数据集经过社交网络划分后，个别社区中包含的敏感节点数目异常多。排查发现，在异常社区中绝大部分节点连接到一个非敏感 API 节点{<java.lang.RuntimeException: void init(java.lang.String)>,Normal}。它是 java 运行时异常处理 API。大部分自定义的方法中包含异常处理。这种“超级节点”的存在，把原本不存在关系的敏感节点联系在一起，破坏了社区划分的结构。所以，在构建调用图时，需要过滤掉该节点再进行社区检测。

笔者期望 APK 对应的特征图像的每一行排列一个社区，以便于卷积神经网络识别。但目前社区中包含的敏感节点数仍较多且分布离散。本文对每个社区再使用 Louvian[12]算法进行一次划分。如表 1 和图 2 所示，二次社区划分之后的小社区中敏感节点数目集中在 2~5。

表 1　社区中敏感节点分布比例

Table 1　Distribution ratio of sensitive nodes in the community

平均值	最小值	25%	50%	75%	最大值
2.8	1	1	1	3	70

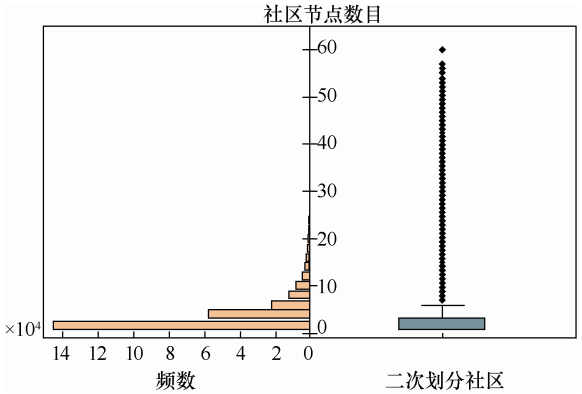

图 2　社区中敏感节点数目分布统计

Figure 2　Distribution statistics of the number of sensitive nodes in the community

根据上述分布结果，可以确定图像的宽度。本文选取图像宽度为 4。这么做是基于这样的考虑，宽度选得过窄，过小的卷积核不能有效地提取模式信息；宽度选得过大，图像中需要填充的空白节点就会增多，从而导致图像的尺寸变大，卷积神经网络模型中参数增多，训练和检测分类时的开销都增大。

3.4　构建规则图像

3.4.1　敏感度计算

本文按照敏感度由大到小将社区中敏感节点排列为规则图像。敏感度是一个统计值，借鉴了词频–逆文本频率[13]（TF-IDF，term frequency-inverse document frequency）的概念。TF-IDF 旨在反映一个术语对语料库中某个文档的重要程度。

衡量一个 API 的敏感度，即该 API 出现所代表的发生恶意行为的可能性大小。基于这样的观察：敏感 API，如 connect()，虽然经常出现在恶意行为模式中，但良性应用也多用它来进行网络连接。所以它的恶意敏感程度并没有很高。参考 TF-IDF，直观上，一个 API 的敏感度应该与其在恶意数据集中出现的频率正相关，而与其在良性数据集中出现的频率负相关[14]。

由此，若敏感 API 定义为 i，那么 i 的敏感度 $S(i)$ 可以定义为

$$S(i)=R_M(i)\cdot\lg\left(\frac{1}{R_B(i)}+1\right) \tag{2}$$

其中，$R_M(i)$ 和 $R_B(i)$ 分别代表 i 在恶意数据集和良性数据集中出现的比例。

$R_M(i)$ 和 $R_B(i)$ 的计算方式如下。

$$R_M(i)=\frac{C_M(i)}{N_M} \tag{3}$$

$$R_B(i)=\frac{C_B(i)+1}{N_B} \tag{4}$$

其中，$C_M(i)$ 和 $C_B(i)$ 分别代表 i 在恶意数据集和良性数据集中出现的比例。N_M 和 N_B 分别代表恶意数据集和良性数据集中包含 APK 的数目。

进而，定义社区敏感度 S_C 为社区 C 所包含的敏感 API 节点的敏感度之和。

$$S_C=\sum_{i\in C}S(i) \tag{5}$$

本文针对目标数据集进行敏感度计算，根据敏感度的定义计算出的敏感度分布如图 3 所示。从图中可以看出，75%敏感 API 的敏感度为零，即大部分敏感 API 并没有在恶意应用数据集中出

现，而能够体现恶意行为的是剩余的 6 415 个敏感 API，其中，5 597 个敏感 API 的敏感度低于 0.2，其余具有较强恶意程度区分度的敏感 API 仅为 818 个，约占总数的 3%。

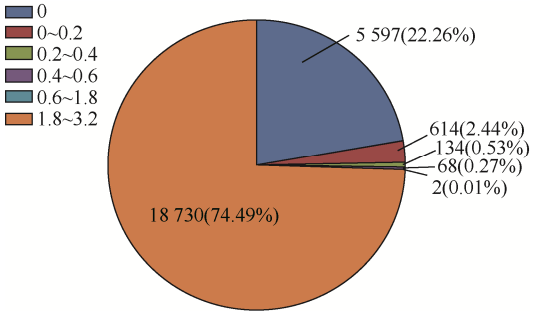

图 3　敏感 API 敏感度分布
Figure 3　Sensitivity distribution of sensitive API

3.4.2　确定图像高度

如图 4 所示，本文按照社区敏感度从大到小递减依次由上到下排列社区，同样地，大社区中划分的小社区也按照敏感度由上到下排序。而一个小社区中的敏感节点则按照敏感度由大到小递减依次从左到右排列到图像中。为了把整个图像像素更紧凑地排列，本文采用如下算法：大社区内的小社区并非严格由上到下排列，如果排列当前小社区后本行剩余的空白像素能够容纳下一个小社区，就在当前行排列下一个小社区；否则，另起一行。同时，为了保存大社区间的界限，大社区间强制换行隔开。这样，相当于将数目较小的小社区（通常是包含 1 到 2 个节点的）填充到大社区空隙中，既有效且紧凑地排列了像素，又保持了不同社区的界限。

如上文所述，图像宽度确定为 4。按照上述排列算法可以得到前述数据集的图像长度分布，如图 5 所示，98.12%APK 的长度在 800 以下。考虑到 API 覆盖率和卷积网络模型运算量之间的平衡，本文将图像长度分为 400、800、1 200 三档做了对比实验。实验指标为准确率和神经网络每秒浮点运算量（FLOPs，floating point operations），实验结果如表 2 所示。由该表可以看出，图像长度从 400 增加到 800 提高了覆盖率，从而带来准确率的上升；800 到 1 200 的准确率基本不变，反而因为空白像素过量填充，导致模型的训练收敛更加困难。由于每次改变图像尺寸需要对卷积神经网络进行调整训练，找到最优长度不太现实，所以在实际操作中本文选择图像长度为 800。少于 800 行用空白像素填充，多于 800 行的像素会被截断丢弃，这样不可避免会漏掉一些使用低敏感度 API 的恶意行为模式。但被丢弃的部分是敏感度最低的节点，其中大部分敏感度为 0，对分类结果的影响很小。最终特征图像尺寸确定为 4×800。

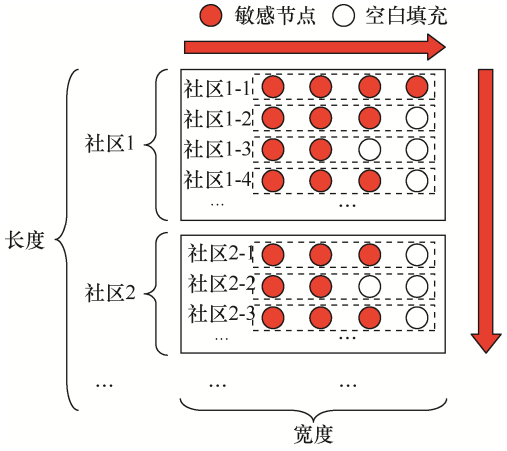

图 4　图像构建方式
Figure 4　Image construction method

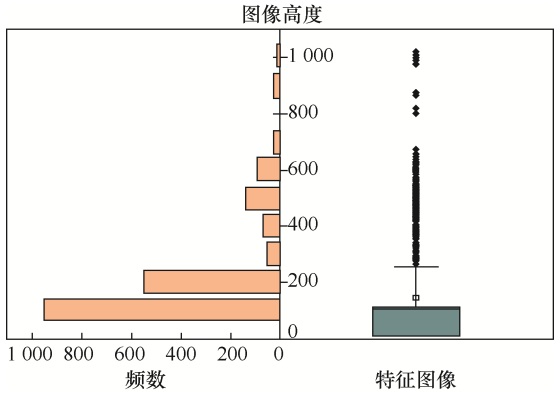

图 5　高度分布统计
Figure 5　Image height distribution statistics

表 2　图像长度效果对比
Table 2　Comparison of image length effects

图像长度	准确率	FLOPs（M）
400	90.31%	417.7
800	92.74%	720.38
1 200	92.73%	1 023.06

最后介绍特征图像中实际排列的像素点。本

文将敏感 API 按类别进行编号并将编号归一化后作为单通道的单通道像素。敏感的 API 总体可以分为两类（Source 和 Sink）。Source 和 Sink 各有 17 372 个和 7 784 个。Source 类 API 从 1 开始递增到 17 372 依次编号，而 Sink 类 API 从–1 开始递减到–7 784 依次编号。相同类别（如网络和位置相关）的 API 会被连续编号。空白像素用 0 表示。为了获得更好的分类效果，需要归一化处理。本文把编号除以类别总数来将像素归一化到–1~1。例如，编号为 20 的 API{<com.android.internal. telephony.sip.SipPhone: java.lang.String getLine1 AlphaTag()>, Source, UNIQUE_ IDENTIFIER,} 属于 Source 类，则其实际对应的像素为 20/17372= 1.15128×10^{-3}。总体来说，特征图像包含了 API 调用组合模式、区域敏感度分布、不同的敏感 API 种类等信息。

3.5 卷积神经网络设计

由于图像的尺寸(1, 4, 800)较小，本文在对比了不同层数的网络 AlexNet[15]、VGG16[16]和 ResNet34[17]效果之后，选用了模型参数稍小的 Alexnet[15]模型。由于输入图像不是规则正方形，所以需要设计相应的 AlexNet[15]网络。第一层卷积层选用了 64 个 4×4 的卷积核。为了减小参数大小，模型选用 kernel_size=3、stride=3、padding=1 的池化层。为了保持矢量形状，卷积层都选用了 5×5 的卷积核。整个网络配置如表 3 所示，为了避免过拟合，保留 Dropout 层，全连接层输出长度为 4 096 的向量，最后调整输出为类别 2。

3.6 卷积层可视化分析

卷积层梯度权重类激活映射[3]可视化是一种可以让已训练的 CNN 模型指出图像中作为分类依据中的区域的方法。该技术的思路是将 CNN 模型最后输出归为类别 C 的特征卷积层，采用反卷积（deconvolution）和导向反向传播（guided-backpropagation）的方法倒推出图像上每个像素点的对最后分类结果 C 权重。这样就可以用热力图的形式直观地解释模型究竟是通过哪些像素点判定图像的类别为 C 的。

定义对应于类别 C 的第 k 个特征图的权重为 α_k^c，计算方式如下。

$$\alpha_k^c = \frac{1}{Z}\sum_i\sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{6}$$

其中，Z 为特征图的大小，y^C 为经过 softmax 层之前类别 C 的分数，A_{ij}^k 代表第 k 个特征图中位置 (i, j)的激活值。

表 3 卷积网络模型

Table 3 Convolutional network model

网络层	参数	输出尺寸
Conv > Relu	kernel_size=4, stride=1	(64, 797, 1)
MaxPool	kernel_size=3, stride=3, padding=1	(64, 266, 1)
Conv > Relu	kernel_size=5, padding=2	(192, 266, 1)
MaxPool	kernel_size=3, stride=3, padding=1	(192, 89, 1)
Conv > Relu	kernel_size=5, padding=2	(384, 89, 1)
Conv > Relu	kernel_size=5, padding=2	(256, 89, 1)
Conv > Relu	kernel_size=5, padding=2	(256, 89, 1)
MaxPool	kernel_size=3, stride=3, padding=1	(256, 30, 1)
AdaptiveAvgPool	output_size=(6, 6)	(256, 6, 6)
Reshape > Dropout	p=0.5	(9,216)
Linear > Relu > Dropout	p=0.5	(4,096)
Linear> Relu	—	(4,096)
Linear	—	(2)

将全部的特征图对类别 C 的权重值进行加权和计算并过滤掉负值后，得到类别 C 的热力值。

$$L_{\text{Grad-CAM}}^c = \text{ReLu}(\sum_k \alpha_k^c A^k) \tag{7}$$

本文将 AlexNet[15]网络的最后一个卷积层（第 11 层）输出的权重反向计算得到梯度权重类激活映射值，并生成样本的热力图。热力图上的每个像素的值，代表样本的每个敏感 API，对神经网络模型，将该样本判别为恶意的影响权重。本文在样本热力图上设置 4×4 的滑动窗口，计算窗口内热力值，向下滑动找到最大热力值的窗口位置，特征图像上对应位置上的敏感 API 调用作为该样本最可疑的敏感 API 调用组合，这样的组合及其互相调用关系一定程度上表征了该样本的恶意行为模式。

表 4　数据集总览
Table 4　Overview of the data set

年份	良性	恶意	总数	平均大小/MB
2012 年	1 875	2 000	3 875	3.73
2013 年	1 896	2 000	3 896	6.56
2014 年	1 826	1 982	3 808	7.15
2015 年	1 811	1 839	3 650	8.35
2016 年	2 015	1 940	3 955	5.12
2017 年	1 884	1 834	3 718	7.92
2018 年	2 058	1 919	3 977	8.15
全部	13 365	13 514	26 879	6.69

4　评估实验

4.1　实验设置

本文通过约 2 000 行 Python 代码实现了原型系统，利用 Apktool[10]进行 APK 文件的反汇编，选用图论与复杂网络建模工具 Networkx 进行调用图的构建，选用 python-igraph 工具包中 Louvain[12]算法包进行社区检测，在深度学习框架 Pytorch 上搭建了神经网络网络，修改了卷积可视化工具 Pytorch-Grad-CAM 来实现卷积层梯度权重可视化。实验软件系统为 Ubuntu 16.04，硬件 CPU 为 Intel Xeon CPU E7-4820，频率 2.0 GHz，内存为 256 GB，显卡为 Tesla P100-PCIE 显存 16 GB。

为了评估系统，本文采用 Android 恶意软件检测系统 malscan-android 的数据集，从中选取了 2012 年—2018 年 7 年间的 13 365 个良性应用和 13 514 个恶意应用，如表 4 所示。本文用 4 个指标评估模型分类性能，分别为精确度（P，precision）、召回率（R，recall）、F-分数（F1）和准确率（A，accuracy）。

4.2　实验结果

本文设置了 4 个实验来验证系统的效果，第 1 个实验验证模型有效性，在整体数据集上达到了 93%的精确度和召回率。第 2 个实验验证系统的抗演化能力，即抵御恶意应用变种的鲁棒性。第 3 个实验验证与机器学习检测方法相比，本文系统在性能开销上的优势。第 4 个实验分析验证卷积层权重可视化来挖掘恶意软件行为模式的正确性。

实验 1　有效性

本实验将数据集分为 80%的训练集和 20%的验证集，在验证集上验证模型的有效性。为了模型能够尽快收敛，实验设置的初始学习率为 0.004，批处理大小为 32。

实验结果如表 5 所示，整体来看，模型对约 93%的样本有检测分类能力。结果相较传统基于机器学习的方法效果稍差，主要原因是，本文系统为了保留原始高级语义，提取的特征向量数目 4×800=3 200 与以往基于机器学习的分类方法相比，在数目上相差一到两个数量级。另外，由于本文所用 SuSi[11]总结的敏感列表已经是几年前的工作，错误识别或者遗漏某些应用的行为不可以避免地影响了分类器性能。若将神经网络全连接层输出的特征向量抽出，然后并入权限，特征字符等特征构成更大的特征向量，输入机器学习模型中，可提升分类检测效果。本文的重点是，如何利用有限的高级语义特征向量，保证高效率检测的同时，挖掘出可被研究人员理解的恶意应用高级语义特征。

表 5　有效性
Table 5　Effectiveness

年份	P	R	F1	A
2012 年	94.0	95.6	94.8	94.4
2013 年	93.5	94.2	93.9	93.7
2014 年	92.3	89.2	90.7	90.6
2015 年	91.0	91.9	91.4	91.4
2016 年	91.4	92.6	92.0	93.9
2017 年	93.4	90.4	91.9	91.8
2018 年	95.1	90.9	93.0	93.0
全部	93.0	92.1	92.5	92.7

实验 2　鲁棒性

本实验用 2015 年的数据集进行了训练，并依次在 2016 年、2017 年和 2018 年的数据集进行测试，结果如表 6 所示。实验结果中精确度一直较高，说明模型对学习到的已有的恶意行为模式能正确识别。召回率下降比较明显，说明系统的漏报率较高，主要原因是不同年份的恶意应用的恶意模式不相同。总体上，F1 分数并非逐年下降，如图 6 所示，表明本系统具有相当的鲁棒性，系统的表现依赖于新的未知恶意行为模式在整体数据集中的占比。

表 6　鲁棒性
Table 6　Robustness

指标	2015 年	2016 年	2017 年	2018 年
P	91.0	89.5	79.1	89.2
R	91.9	56.9	37.2	60.0
F1	91.4	69.5	50.6	71.7
A	91.4	75.6	64.2	77.2

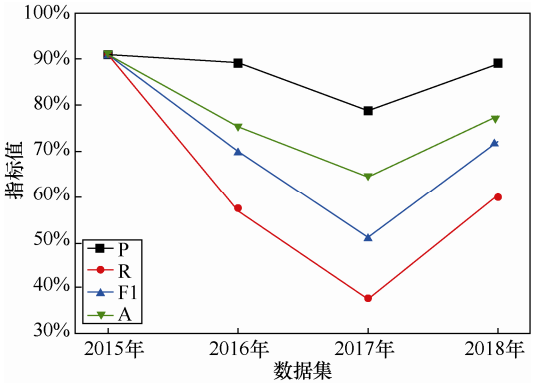

图 6　鲁棒性
Figure 6　Robustness

实验 3　运行性能

本实验在 2018 年的所有恶意样本上统计了方法中各个阶段所需要的平均时间，如表 7 所示。与要提取大量特征的传统机器学习方法 MaMaDroid[7]和 Drebin[4]相比，本文方法在检测效率上表现较好。在平均情况下，反汇编阶段消耗了较多时间，其次是调用图和特征图像构建。如图 7 所示，随着样本的规模变大，调用图和特征图像的构建开销显著增大，主要是社交检测和社区内部多次排序的时间消耗增大。由于最终提取的特征图像的尺寸固定，分类和可视化的时间开销稳定在 2 s 和 1 s 左右。

表 7　运行时间
Table 7　Running time

方法	反汇编/s	调用图/s	特征提取/s	分类/s	可视化/s	总时间/s
MaMaDroid	–	163.2	2.5	0.5	–	166.2
Drebin	–	–	82.4	0.3	–	82.7
本文方法	9.5	4.8	5.0	2.1	1.2	22.7

实验 4　可视化分析

本实验对 2018 年的全部恶意样本生成了热力图，并输出其特征敏感 API 调用组合。本文挑选重点样本说明可视化效果。以 SHA256 值为 C0D52E768A891B5650CDE6810B6B13B2645FAFC61D7C4A70B64F06A98C70751B 的样本为例，VirusTotal[24]中 21 个引擎报告其为广告恶意软件，该样本包含 365 个敏感 API，特征图像的有效像素行数为 116 行，剩余 684 行填充 0。图 8 左侧为该样本的热力图。根据热力图的颜色可知，神经网络判别该样本为恶意应用的主要依据是特征图像上 1 至 31 行和 63 至 74 行的敏感 API。该热力图 116 行以下都为暗黑色，说明神经网络辨别了填充的无效信息。热力图的头部往往颜色较为明亮，说明神经网络判定的主要依据是头部排列的敏感度较高的 API。该样本中滑动窗口内热力值达到最高的区域为 1 至 4 行。但这并非绝对，有些样本的热力值最高的区域出现在中上部，说明卷积神经网络是依据某些特定的 API 组合而不是单单依赖于敏感度最高的 API 进行检测识别。

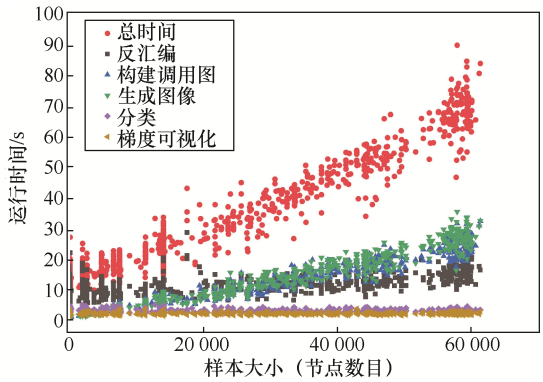

图 7　不同大小的样本运行时间
Figure 7　Running time of samples of different sizes

该样本中滑动窗口找到的最可疑的 API 组合的调用关系如图 8 中右下侧所示，蓝色节点是用户自定义的普通函数，红色节点为 Source 类敏感节点，橘黄色节点为 Sink 类敏感节点。直观上可以定位该部分恶意代码主要位于 com.iflytek.cloud.thirdparty 包中 ai、al、aj、w 等源文件中。这些源文件经过了混淆处理。ai 中定义的 b 函数收集了地区信息，而且 b 函数通过调用 al 中定义的 a 函数间接调用 al 中定义的 b 函数来收集 android.telephony 等敏感信息以及 w 中定义的 a 函数来收集位置敏感信息。从调用图中还能够明确看出恶意应用是通过数据结构 HashMap 来进

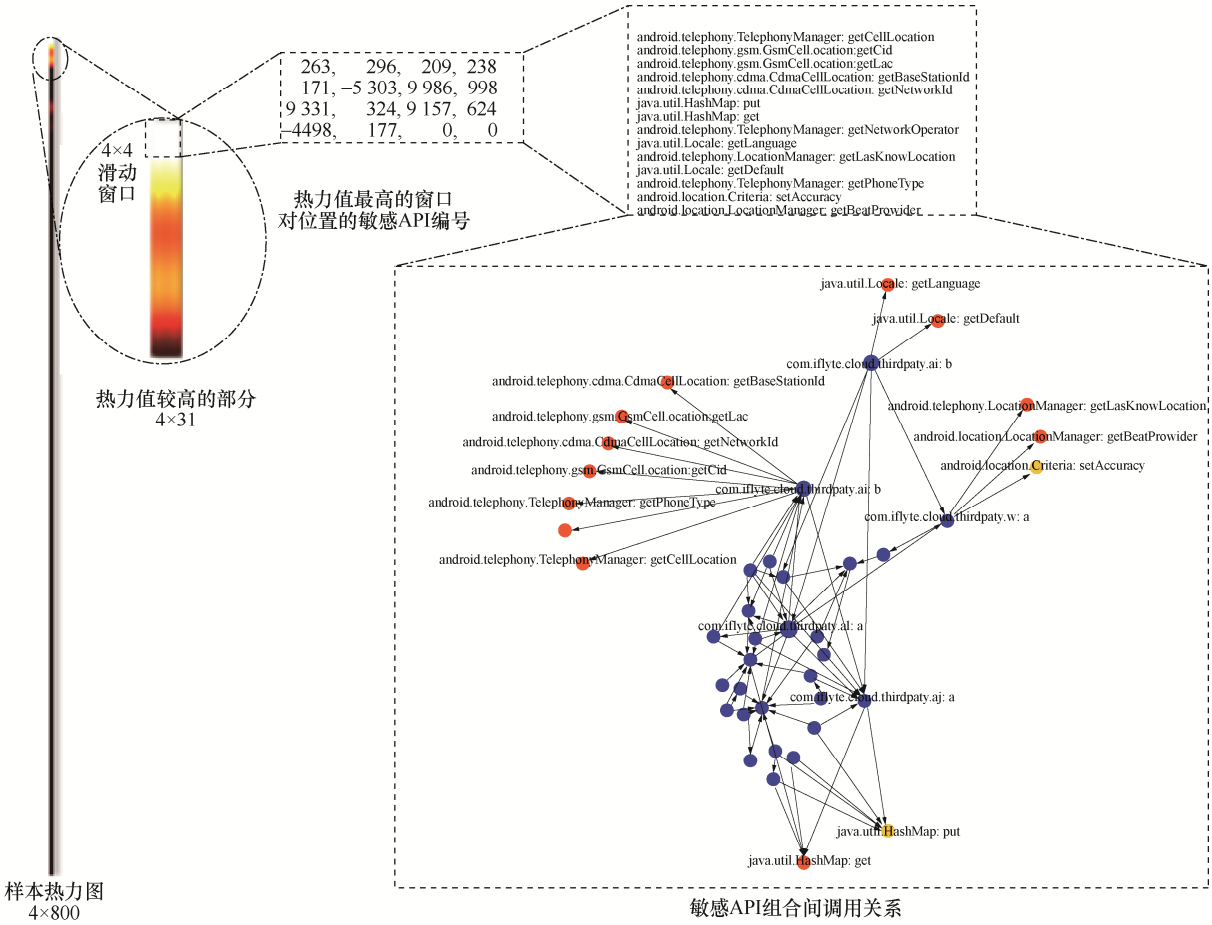

图 8　样本可视化分析

Figure 8　Visual analysis of samples

行敏感信息的收集和传递。总的来说，这 14 个 API 组合的主要功能是获取位置和运营商网络等设备信息。这些信息最终会被加密传输到远端服务器，从而便于黑色产业广告商针对性地进行流量推送，这是一般广告恶意应用的最常见特征行为。

5　结束语

本文提出了一种新颖的安卓恶意应用检测系统，利用社交网络检测和统计规律快速分析 API 调用图提取特征，输入卷积神经网络以高达 93% 的准确率进行快速分类。本文系统保留了恶意应用的原始高级语义，利用已训练模型，通过卷积层梯度权重可视化算法来挖掘恶意应用最可疑的 API 调用组合，为研究恶意应用特征行为模式提供参考。

本文设计的系统虽然可以利用卷积层可视化来解释应用程序的恶意调用模式，但目前粒度还不够细致，恶意信息的挖掘还不够深入准确，如何解决该问题将是下一步的研究工作。

参考文献：

[1] Google android security 2018 report[EB].

[2] RASTOGI V, CHEN Y, JIANG X. Catch me if you can: evaluating android anti-malware against transformation attacks[J]. IEEE Transactions on Information Forensics and Security, 2013, 9(1): 99-108.

[3] SELVARAJU R R, COGSWELL M, DAS A, et al. Grad-cam: visual explanations from deep networks via gradient-based localization[C]//Proceedings of the IEEE International Conference on Computer Vision. 2017: 618-626.

[4] ARP D, SPREITZENBARTH M, HUBNER M, et al. Drebin: effective and explainable detection of android malware in your pocket[C]//NDSS. 2014 23-26.

[5] ZHANG M, DUAN Y, YIN H, et al. Semantics-aware android malware classification using weighted contextual api dependency graphs[C]//Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014: 1105-1116.

[6] FENG Y, ANAND S, DILLIG I, et al. Apposcopy: semantics-based detection of android malware through static analysis[C]// Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014: 576-587.

[7] MARICONTI E, ONWUZURIKE L, ANDRIOTIS P, et al. Mamadroid: detecting android malware by building markov chains of behavioral models[J]. arXiv preprint arXiv:1612.04433, 2016.

[8] NATARAJ L, KARTHIKEYAN S, JACOB G, et al. Malware images: visualization and automatic classification[C]//Proceedings of the 8th International Symposium on Visualization for Cyber Security. 2011: 4.

[9] HSIEN-DE HUANG T T, KAO H Y. R2-D2: color-inspired convolutional neural network (CNN)-based android malware detections[C]//2018 IEEE International Conference on Big Data (Big Data). 2018: 2633-2642.

[10] [EB/OL]. https://ibotpeaches.github.io/Apktool.

[11] RASTHOFER S, ARZT S, BODDEN E. A machine-learning approach for classifying and categorizing android sources and sinks[C]//NDSS. 2014: 1125.

[12] BLONDEL V D, GUILLAUME J L, LAMBIOTTE R, et al. Fast unfolding of communities in large networks[J]. Journal of Statistical Mechanics: Theory and Experiment, 2008(10): 10008.

[13] WU H C, LUK R W P, WONG K F, et al. Interpreting TF-IDF term weights as making relevance decisions[J]. ACM Transactions on Information Systems (TOIS), 2008, 26(3): 13.

[14] FAN M, LIU J, WANG W, et al. Dapasa: detecting android piggybacked apps through sensitive subgraph analysis[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(8): 1772-1785.

[15] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]//Advances in neural information processing systems. 2012: 1097-1105.

[16] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[17] HE K, ZHANG X, REN S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016: 770-778.

[作者简介]

张鑫（1993− ），男，河南汝州人，华中科技大学硕士生，主要研究方向为恶意软件检测。

羌卫中（1977− ），男，江苏南通人，博士，华中科技大学教授、博士生导师，主要研究方向为系统安全及软件安全。

吴月明（1993− ），男，湖北洪湖人，华中科技大学博士生，主要研究方向为恶意软件检测、漏洞检测。

邹德清（1975− ），男，湖南湘潭人，博士，华中科技大学教授、博士生导师，主要研究方向为软件安全。

金海（1966− ），男，上海人，博士，华中科技大学教授、博士生导师，主要研究方向为分布式计算。