基于 API 分组重构与图像表示的恶意软件检测分类

杨宏宇[1,2], 张宇沛[2], 张　良[3], 成　翔[4,5]

[1] 中国民航大学安全科学与工程学院 天津 中国 300300
[2] 中国民航大学计算机科学与技术学院 天津 中国 300300
[3] 亚利桑那大学信息学院 图森 美国 AZ85721
[4] 扬州大学信息工程学院 扬州 中国 225127
[5] 江苏省知识管理与智能服务工程研究中心 扬州 中国 225127

摘要　针对目前恶意软件检测分类方法在特征提取、检测准确率等方面面临的挑战, 提出一种基于 API 分组重构与图像表示的恶意软件检测分类方法。首先, 对恶意软件调用的 API 类别统一编号, 将 API 指令序列中相同编号的 API 聚合为同一 API 组, 根据恶意软件运行时各类 API 的首次调用顺序对 API 组重排序, 将各 API 组的条目数记录为该类 API 对软件样本的贡献度。经分组重构后, 各 API 组按序组织, 其顺序为软件样本调用各类 API 的顺序。各 API 组内部有序, 其内部各 API 的排列顺序即为软件样本对单个 API 的调用顺序。有序化的 API 分组有助于 API 指令序列信息的图像化表达。基于重组的 API 指令序列提取 API 编号作为全局特征列表、API 贡献度作为局部特征列表、API 顺序索引作为时序特征列表, 对特征列表进行标准化与零填充, 转化为统一尺寸的特征数组。其中, API 编号能清晰地标识 API 类别, API 贡献度可以表征该 API 的调用频繁程度, API 顺序索引可区分各 API 被调用的顺序。然后, 分别用 3 类特征数组填充 RGB 图像的 3 个通道, 生成 3 通道的 API 编号贡献度及顺序索引特征图像(Feature image of API code devotion and sequential index, FimgCDS)。最后, 将 FimgCDS 特征图像输入自主构建的轻量型恶意软件特征图像卷积神经网络(malware feature image convolutional neural network, MficNN)分类器, 实现对恶意软件的检测与分类。实验结果表明, 本文方法在两类数据集上的检测分类准确率分别为 98.66%和 98.35%, 具有较高的恶意软件检测分类性能指标和检测分类速度。

关键词　恶意软件; 分类; API; 特征提取; 图像表示; RGB 图像; 卷积神经网络
中图法分类号　TP309　DOI 号 10.19363/J.cnki.cn10-1380/tn.2024.09.05

Malware Detection and Classification Based on API Block Reconstruction and Image Representation

YANG Hongyu[1,2], ZHANG Yupei[2], ZHANG Liang[3], CHENG Xiang[4,5]

[1] School of Safety Science and Engineering, Civil Aviation University of China, Tianjin 300300, China
[2] School of Computer Science and Technology, Civil Aviation University of China, Tianjin 300300, China
[3] School of Information, University of Arizona, Tucson AZ85721, USA
[4] School of Information Engineering, Yangzhou University, Yangzhou 225127, China
[5] Jiangsu Engineering Research Center for Knowledge Management and Intelligent Service, Yangzhou 225127, China

Abstract　To address the challenges faced by current malware detection and classification methods in terms of feature extraction and detection accuracy, a malware detection and classification method based on API block reconstruction and image representation was proposed. First, the API categories invoked by malware during the malware runtime were numbered uniformly and aggregate the APIs with the same code into the same API block, and the API blocks were reordered according to the invocation order of each API, the number of entries in each API block was recorded as the devotion of such API. After reconstruction, each API block is organized in order, and its order is the order in which each type of API is called by the software sample. The order within each API block is the order in which the software sample calls the individual APIs. The ordered API block sequence helps to represent the API instruction sequence information pictorially. The API codes were extracted as the global feature list, the API devotion as the local feature list, and the API sequential indexes as the temporal feature list, and the feature lists were normalized and zero-padded to transform into feature arrays. The API code clearly identifies the API category, the API devotion characterizes how frequently the API is called, and the API sequential index distinguishes the order in which each API is called. Then, the 3 channels of the RGB image were filled with the 3 types of feature arrays to generate the feature image of API code devotion and sequential index (FimgCDS). Finally, the FimgCDS feature image was fed into a self-built lightweight malware feature image convolu-

通讯作者：杨宏宇, 博士, 教授, Email: yhyxlx@hotmail.com。
本课题得到国家自然科学基金资助项目(No. U1833107)资助。
收稿日期: 2022-08-30; 修改日期: 2022-12-05; 定稿日期: 2024-06-04

tional neural network (MficNN) classifier for malware detection and classification. The experimental results show that the detection and classification accuracies of the method are 98.66% and 98.35% on the two datasets, and the method has high detection and classification performance indicators and speed for malware.

Key words malware; classification; application programming interface; feature extraction; image representation; RGB image; convolutional neural network

1 引言

随着信息技术的发展, 网络与信息系统运行环境日趋复杂, 恶意软件随之不断涌现。恶意软件是一种专门设计的、非法访问计算机、损害或破坏计算机系统的软件, 运行时可对计算机系统中程序或数据的机密性、完整性、可用性等安全特征造成威胁。据权威部门统计, 2022 年 5 月中国境内感染恶意软件的终端数已达 478 万个, 仿冒网站页面数量为 16540 个, 近九成漏洞被用来实施远程攻击, 可窃取用户信息, 将对企业和个人造成巨大的损害[1]。恶意软件的广泛传播已严重威胁到了网络与系统安全, 因此开展恶意软件检测方法研究刻不容缓。

目前的主流恶意软件检测技术包括两类: 静态检测技术和动态检测技术。静态检测在不执行软件的情况下通过对代码的分析, 判断其是否为恶意软件。静态分析方法虽快速安全, 但只能检测已有的恶意软件, 容易受到混淆技术的影响。动态检测在软件运行的状态下检测其是否为恶意软件, 能更好地检测未知恶意软件, 但检测过程对系统资源消耗较大。

使用传统的机器学习方法进行恶意软件检测, 特征选择与提取复杂度较高[2]。近年来, 研究发现同类恶意软件不同样本的特征图像在色彩、纹理与布局上具有相似性, 导致基于图像表示方法的恶意软件检测方法成为研究热点。基于深度学习的恶意代码变体检测方法[3]将恶意软件代码转换为灰度图像, 使用卷积神经网络(convolutional neural networks, CNN)对图像进行识别和分类, 使用蝙蝠算法解决不同恶意软件类别之间的数据不平衡问题。基于可视化技术的同源恶意软件检测方法[4]将恶意软件可执行文件的全局结构信息转换为灰度图像, 提取代码段的操作码语义信息生成操作码图像, 使用 CNN 进行恶意软件的检测。基于可视化和深度神经网络的恶意软件检测方法[5]将恶意软件的代码信息转化为灰度图、RGB 图像和马尔可夫图像, 使用 Gabor 滤波器对图像进行纹理分析。基于图像表示与 CNN 的恶意软件分类方法(image-based malware classification using fine-tuned convolutional neural network architecture, IMCFN)[6]将恶意软件二进制文件转换为 RGB 图像, 使用 CNN 进行恶意软件家族的检测和分类。基于图像表示与深度神经网络的恶意软件检测方法[7]将恶意软件的字节文件转换为 RGB 图像, 基于 SEResNet50、Bi-LSTM 和注意力机制构建深层神经网络检测框架进行恶意软件的检测分类。

在上述研究中, 部分研究[3-4]使用灰度图作为恶意软件的特征图像, 与 RGB 三通道图像相比, 灰度图具有难以识别截面分布信息和信息量较少的局限性[8]。部分研究[5-7]在构建恶意软件特征图像时选取静态特征, 但静态特征的数量较大, 通常需要人工筛选[9], 与包含丰富语义信息、能直观反映程序行为的动态特征相比, 静态特征难以直接反映程序的行为[10]。

针对上述研究存在的不足, 本文提出一种基于 API 分组重构与图像表示的恶意软件检测方法, 本文的贡献如下:

(1)提出一种 API 指令序列分组重构方法。首先将 API 按类别聚合为 API 组(block of API, ABlock), 然后将 ABlock 按各类别 API 第一次被调用的顺序(order of API, AOrder)重排序。分组重构后, 各 ABlock 间有序, 其顺序为软件样本调用各类 API 的顺序。各 ABlock 内部有序, 其内部各 API 的排列顺序即为软件样本对单个 API 的调用顺序。有序化的 ABlock 序列有助于 API 指令序列信息的图像化表达。

(2)提出一种恶意软件图像表示方法。首先, 基于分组重构后的 API 指令序列, 提取 API 编号、API 贡献度、API 顺序索引作为特征。然后, 使用 3 类特征分别填充 RGB 图像的 3 个通道, 生成恶意软件特征图像(feature image of API code devotion and sequential index, FimgCDS)。其中, API 编号能清晰地标识 API 类别, API 贡献度可以表征该 API 的调用频繁程度, API 顺序索引可区分各 API 被调用的顺序。同时, 经分组重构后, 各 ABlock 反映到特征图像上呈现出区域化的颜色与纹理特征, 有助于分类器进行检测分类。

(3)针对恶意软件特征图像的检测与分类, 设计一种轻量型的恶意软件特征图像卷积神经网络(malware feature image convolutional neural network, MficNN)。与经典卷积神经网络相比, MficNN 的网络层数更少, 参数量更低, 有效减少对恶意软件特征图像进行分类的计算资源和时间开销, 可以更及时

高效地检测恶意软件的类别。

2 相关工作

API 指令序列在众多恶意软件特征中最能反映程序的行为, 对恶意软件的检测与分类最为重要[10]。其中, Windows 平台下的 Windows API 是一个广泛的功能集合, 管理着恶意软件与微软程序库之间的交互方式。每类 API 都具有对应功能, 如 CreateFile 用于创建与打开文件, RegGetValue 用于返回注册表某数据项的数值, InternetOpenUrl 用于访问 URL, CreateProcess 用于创建远程 shell。所以, 基于 API 指令序列的恶意软件特征提取, 在恶意软件检测分类中发挥了重要作用。

文献[11]从样本数据集中选出最常用的 300 种 API, 根据功能特征与对系统的危害程度不同, 将 API 分为 17 类, 基于分类后的 API 序列提取语义和结构特征, 构建基于局部注意力机制和滑动窗口方法的检测框架 SLAM 进行恶意软件的检测。文献[12]根据恶意软件的文件名将每个软件的线程调用序列合并为字符串, 对字符串进行矢量化生成线程向量序列。根据线程向量序列的长度截断 API 指令序列, 使用长短期记忆网络(long short-term memory, LSTM)从截断后的 API 指令序列中提取时序特征, 输入 CNN 分类器检测恶意软件的类别。文献[13]采用了基于规则和聚类的方法来评估参数对恶意行为的敏感度, 按照不同敏感度的运行时参数来标记 API。通过连接本地嵌入和已标记的 API 的敏感嵌入对 API 进行编码, 构建基于深度神经网络(deep neural networks, DNN)的二分类器进行恶意软件的检测。

以上研究中, 筛选出常用的 API 与截断 API 序列虽然均着眼于占全局信息比重最大的一部分信息, 但仍会造成有关 API 信息的丢失。同时, 人为对 API 进行分类、定义匹配规则过于依赖专家知识, 需要大量的人工筛选, 前期数据处理投入大。本文拟对样本数据集中所有 API 类别进行统一编号, 而非将各类 API 划分到不同的 API 集合, 以解决传统恶意软件检测分类方法在特征提取上过于依赖专家知识, 需要大量的人工筛选的不足。同时, 基于 CNN 构建多分类器以进一步检测恶意软件的类别。

文献[14]认为大多数恶意软件源于被病毒等恶意软件感染的良性软件, 从代码角度看, 大多数恶意代码序列中只有一小部分是恶意的, 即恶意代码具有局部恶意性。研究者根据恶意代码的局部恶意性, 将整个 API 序列切割为长度为 N 的 API 片段, 每个 API 片段均具有一定的恶意性。利用 LSTM 模型对 API 片段进行分类并使用集成学习方法检测恶意软件的类别。虽然局部特征能在一定程度上反映恶意软件的行为, 但恶意软件与系统交互方式复杂, 如创建修改文件、修改注册表、利用网络进行通信、创建或修改进程以执行当前程序之外的代码等, 仅考虑局部特征而忽略全局特征不能捕获软件完整的恶意意图。本文拟从全局、局部和时序等多角度提取恶意软件的特征, 以解决传统恶意软件检测分类方法特征类别单一、仅考虑局部特征或全局特征的不足。

文献[15]将 API 指令信息编码为图结构, 每个 API 为一个节点, 节点之间的边表示各 API 之间存在的连接。利用门控机制和图卷积网络(graph convolution neural networks, GCN)提取 API 图结构的特征对恶意软件进行检测分类。文献[16]构建了一个堆叠卷积神经网络, 用于捕获 API 调用序列的局部语义特征, 设计了基于局部和全局 API 特征的恶意软件联合检测框架(a joint framework based on local and global features for malware detection, LGMal), 将 CNN 和 GCN 相结合进行恶意软件的检测。但 GCN 模型复杂, 特征提取对计算资源的消耗较大。本文拟采用基于图像表示的方法表示恶意软件的特征, 采用自主构建的轻量型卷积神经网络对特征图像进行检测, 以减少系统的计算资源开销。同时, 拟采用多种图像增强方法增强恶意软件特征图像, 以抑制数据不平衡现象。

3 检测方法总体设计

3.1 总体架构

基于 API 分组重构与图像表示的恶意软件检测分类方法的总体架构如图 1 所示。该方法包括 API 分组重构、特征提取、图像生成与增强和图像分类 4 个阶段。本文方法各阶段处理过程设计如下:

(1)API 分组重构

对样本数据集中所有 API 类别编号, 将各软件样本中编号相同的 API 聚合为一个 ABlock, 记录各软件样本运行时对各类别 API 的调用顺序。根据 AOrder 记录的顺序对各 ABlock 进行重排序。最后将软件样本中各 ABlock 的条目数设置为该 API 类别对该软件样本的贡献度。

(2)特征提取

基于重排序的 ABlock, 提取 API 编号列表(list of API code, ListAC)作为全局特征, API 贡献度列表(list of API devotion, ListAD)作为局部特征, API 顺序索引列表(list of API sequential index, ListAS)作为时

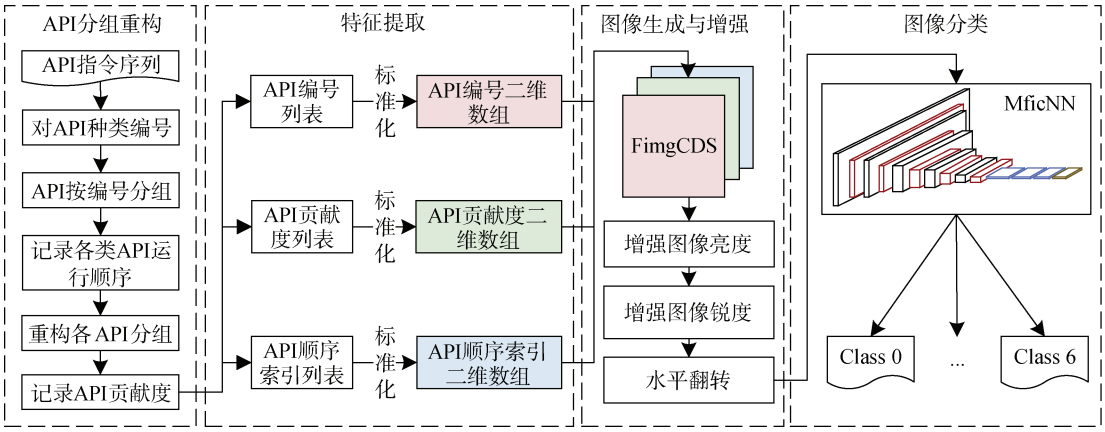


图 1　恶意软件检测分类方法的总体架构

Figure 1　The overall architecture of the malware detection method

序特征。然后对 ListAC、ListAD、ListAS 进行标准化, 并转化为尺寸相同的二维特征数组, 记为 API 编号数组(arrary of API code, ArrayAC)、API 贡献度数组(arrary of API devotion, ArrayAD)和 API 顺序索引数组(array of API sequential index, ArrayAS)。

(3)图像生成与增强

图像生成阶段使用 ArrayAC 填充 R 通道, ArrayAD 填充 G 通道, ArrayAS 填充 B 通道, 生成 FimgCDS 特征图像。

图像增强阶段使用亮度增强和锐度增强技术提高各类别样本的 FimgCDS 特征图像的亮度与锐度, 同时, 使用水平翻转技术对少数类样本的 FimgCDS 特征图像进行水平翻转。图像增强阶段得到经图像增强后的 FimgCDS 特征图像, 样本数量得到扩充, 不同种类软件样本间的数据不平衡现象得到抑制。

(4)图像分类

将 FimgCDS 特征图像输入自主构建的轻量型卷积神经网络 MficNN 分类器, 输出软件样本的检测分类结果。

3.2　方法分析

在特征提取过程中, API 编号作为全局特征, 能够清晰地标识 API 的类别, 可从全局角度反映该恶意软件调用的所有 API。API 贡献度作为局部特征, 能够有效表征某类 API 的调用频繁程度, 反映该恶意软件的主要系统交互活动, 在基于全局 API 调用信息的基础上, 可从局部特征的角度反映恶意软件主要的恶意意图。API 顺序索引作为时序特征, 可以有效区分各 API 被调用的顺序, 可从时序角度描述恶意软件执行各类交互活动的顺序。

API 指令序列经分组重构后, 相同的 API 聚合为一个 ABlock, 每个 ABlock 反映到特征图像上呈现出区域化的颜色与纹理特征, 结构清晰, 有助于 MficNN 分类器对特征图像进行检测与分类。

3 通道 FimgCDS 特征图像包含 3 类恶意软件特征, 能从多角度反映软件的行为。通过使用亮度增强、锐度增强和水平翻转等图像增强技术处理 FimgCDS 特征图像, 在不改变原特征图像的纹理与颜色特征的前提下, 扩充特征图像的数量, 使 MficNN 分类器能从足够数量的图像样本中学习到各类恶意软件特征图像的特征, 以提高检测分类效果。

与传统的非图像分类的检测分类方法相比, 本文方法在对样本数据集中出现的每类 API 进行编号的过程中, 由于考虑到图像结构的特殊性, 将 API 编号映射到取值为(0, 255]的图像像素值, 无需人为对 API 进行分类, 可提取更细粒度的 API 特征。在传统检测分类方法中, 由于其所处理的特征向量的内容过多, 干扰了分类器对长时序特征的提取, 从而降低检测分类效率。而在本文方法的图像分类过程中, 自主构建的 MficNN 分类器中的多个卷积层可关注到图像的全局信息, 从而能够捕获恶意软件的内在特征。

4　API 分组重构

4.1　API 分组重构算法与复杂度分析

统计软件样本调用的 API 种类并编号, 将软件样本中相同编号的 API 聚合为一个 ABlock, 记 ABlock 中的条目数为该 API 对该软件样本的贡献度。最后将该软件样本对各类 API 第一次调用时的顺序记录记为 AOrder, 根据 AOrder 对各 ABlock 进行重排序, 得到重组 API 指令序列。API 指令序列的分组重构过程如算法 1 所示。

算法 1　API 指令序列分组重构算法

输入: 可执行文件 f 对应的 API 指令序列 $A = \{a_1, a_2, \cdots, a_n\}$;

输出: 可执行文件 f 分组重构后的 API 调用序列 A_{new};

```
BEGIN
API 出现顺序表 A_O ← LinkedList()
API 出现次数表 A_D ← Dict()
FOR EACH a ∈ A DO
    id ← A_C[a]
    IF A_O.containsKey(id) THEN
        A_D[id] ← A_D[id] + 1
    ELSE
        A_O.addLast(id)
        A_D[id] ← 1
    END IF
END FOR
令重组后的 API 调用序列 A_new ← {}
FOR i = 0; i < A_O.size; i++ DO
    id ← A_O[i]
    id 出现次数 Time_a ← A_D[id]
    FOR j = 0; j < Time_a; j++ DO
        A_new ← A_new ∪ GetAPI(id)
    END FOR
END FOR
```

对于每个可执行文件 f, 算法 1 对其调用的 API 指令序列进行分组重构, 设该可执行文件 f 调用的平均 API 数量为 n。在进行 API 分组时, 需遍历指令序列中 n 个原始 API, 因此分组过程的时间复杂度为 O(n)。在进行重构时, 尽管存在两层嵌套循环, 但总的计算次数依然为 n 次, 因此重构过程的时间复杂度为 O(n), 故算法的总时间复杂度为 O(n)。算法 1 执行过程中, 主要保存 A_O 和 A_D, 空间复杂度为 O(n)。

4.2 API 分组重构示例

本文使用两类数据集, 数据集 1 为阿里云提供的恶意程序检测大赛数据集[11], 数据集 2 为 Datacon2019-MaliciousCode[17]。以数据集 1 为例, 该数据集共调用 295 类 API, 以[0, 294]范围的整数对数据集 1 中的 API 类别进行编号, 部分 API 类别及编号如表 1 所示。

表 1 数据集 1 中部分 API 类型对应编号

Table 1 Partial API codes of API types

API 类型	API 编号
RegCreateKeyExA	0
GetKeyboardState	1
HttpSendRequestA	2
……	……
LdrGetProcedureAddress	101
……	……
NtDeleteKey	293
WSARecv	294

图 2 所示为数据集 1 中文件编号为 771 号的软件样本的 API 指令序列重组过程。其中, (1)图为文件编号(file_id)为 771 号软件样本的部分信息, 记录该文件的 file_id、标签(label)、API 调用序列(api)、调用该 API 的线程 ID(tid)、该 API 在对应线程中的索引(index)。(2)图为 771 号软件样本的部分 AOrder 信息, 记录该软件样本对各类 API 第一次调用的顺序。(3)图为 771 号软件样本的部分 ABlock 信息, 记录相同编号的 API 按照 AOrder 聚合的情况。(4)图为 771 号软件样本的 API 指令序列经 API 重组后的部分结果。

图 2 中, 标号①～⑥表示 771 号软件样本调用的前六类 API 指令的重组过程。以标号①②为例, ①表示 771 号软件样本调用的第一类 API 指令为_exception_, 首先将_exception_记录在 AOrder 中, 编号为 1, 表明_exception_是该软件样本调用的第一类 API; 然后统计该软件样本对_exception_的总调用次数, 为 1 次, 则_exception_所属的 ABlock 的条目数为 1。②表示该软件样本调用的第二类 API 指令为 LdrLoadDLL, 将 LdrLoadDLL 记录在 AOrder 中, 编号为 2; 统计该软件样本对 LdrLoadDLL 的总调用次数, 为 48 次, 则 LdrLoadDLL 所属的 ABlock 的条目数为 48。待所有 API 均被记录到 ABlock 后, 按照 AOrder 和 ABlock 记录的信息, 对 API 指令序列进行重排序。

5 特征提取

特征提取指从原始数据中学习并挖掘能反映原始数据特征的非冗余派生值[18], 可以促进模型的学习和泛化步骤, 带来更好的可解释性。

在特征提取阶段, 首先从重组后的 API 指令序列中提取 ListAC、ListAD、ListAS 一维特征数组, 并进行标准化与零填充, 转化为统一尺寸的二维特征数组, 记为 ArrayAC、ArrayAD、ArrayAS。

5.1 ArrayAC 生成

基于重组后的 API 指令序列提取 API 编号特征, 记为 ArrayAC, 将其作为全局特征清晰地标识 API 的类别。首先查询各 ABlock 中 API 的对应编号, 将 API 编号记录为一维数组 ListAC。然后对 ListAC 中各编号值进行标准化。最后对经标准化的 ListAC 进行零填充, 转化为尺寸为 128×128 的二维特征数组 ArrayAC, 作为恶意软件的 API 编号特征。标准化的 API 编号为

$$A_{n\_c} = \left(A_c + 1\right) \times \frac{255}{A_n + 1} \tag{1}$$

其中, A_n 为 API 的种类数; A_c 为 API 编号, 取值范围为[0, A_n–1], A_{n_c} 为经标准化的 API 编号, 取值范围为(0, 255)。

771 号软件样本生成 ListAC 的过程如图 3 所示。标号①～⑥表示经 API 指令序列重组后, 771 号软件样

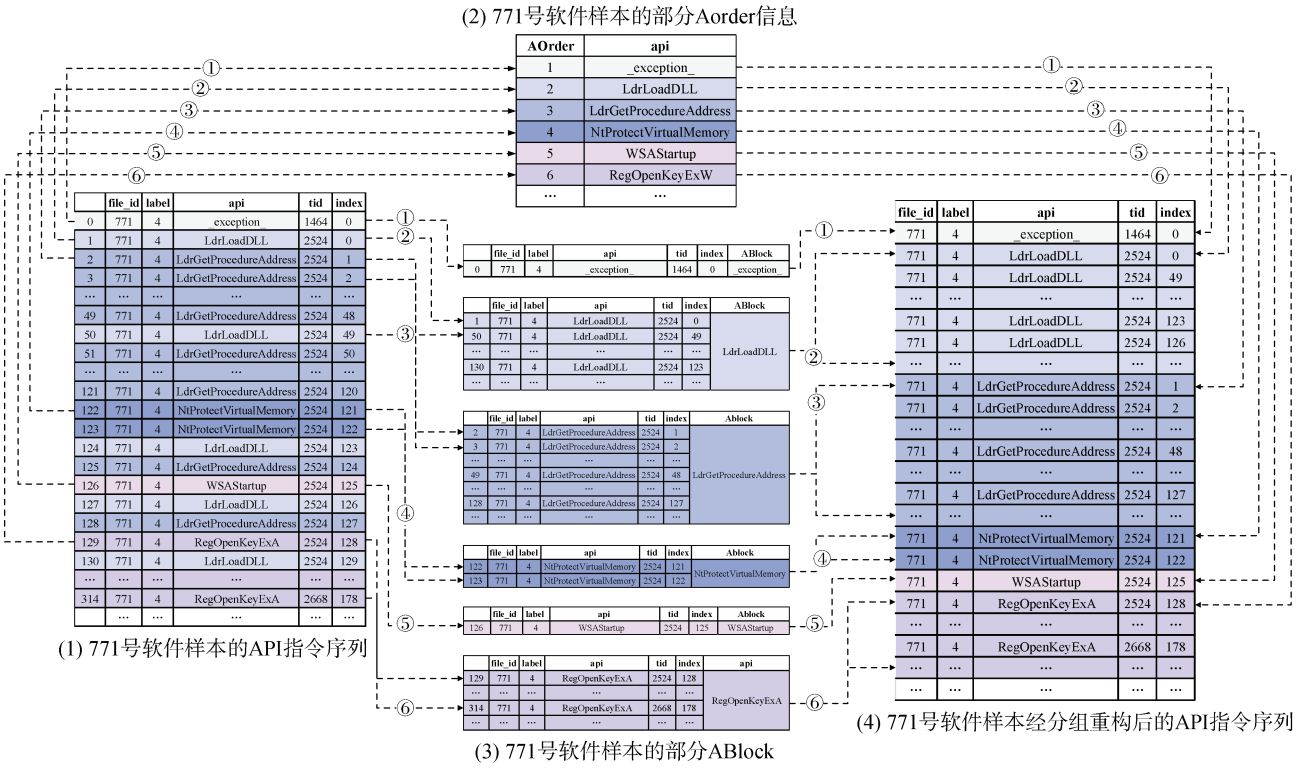


图 2　API 指令序列重组过程示例

Figure 2　An example of API instruction sequence grouping and reconstruction process

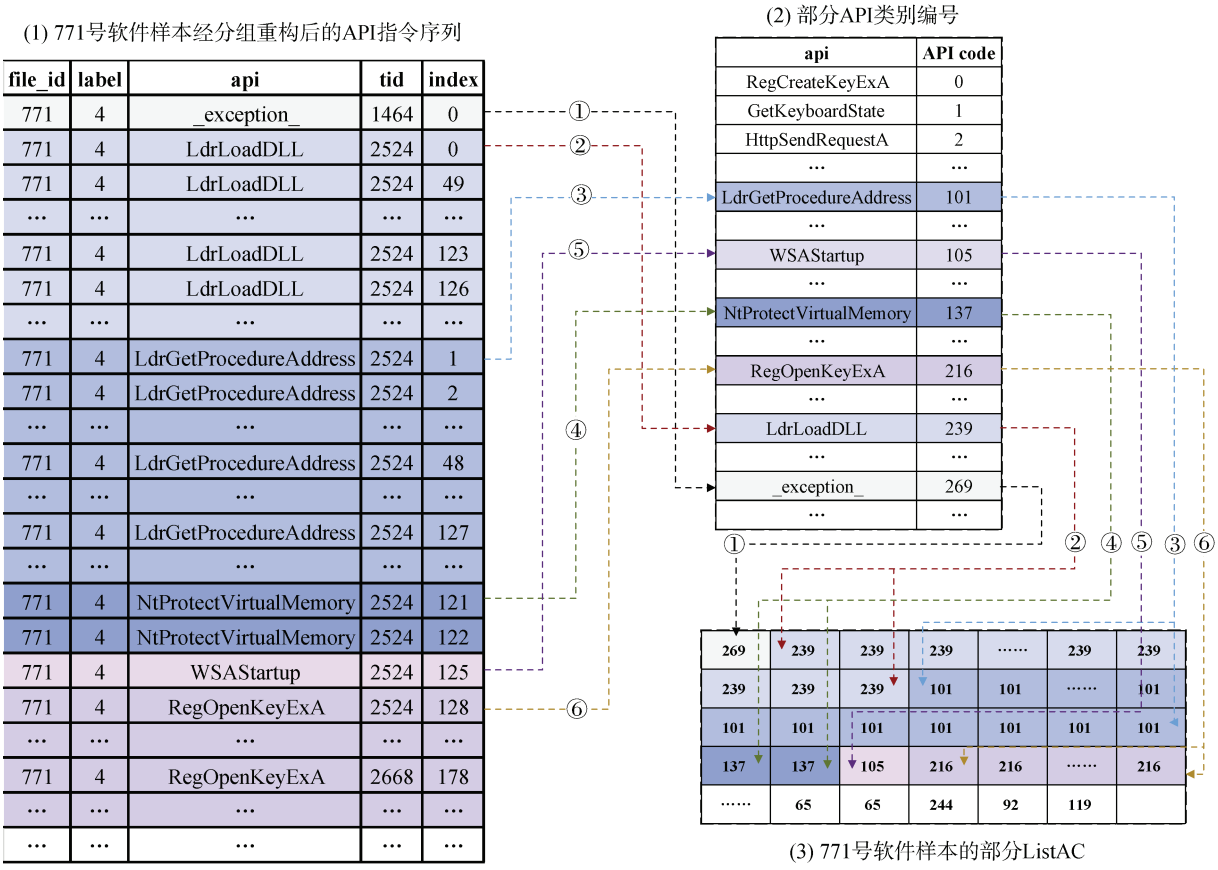


图 3　ListAC 生成过程示例

Figure 3　An example of the ListAC generation process

本对应的前六类 ABlock 生成 ListAC 的过程。图 3 中, ①表示该软件样本调用的第一类 ABlock 为 _exception_, 在 API 编号对照表中查询_exception_的编号为 269, 将 269 记录到 ListAC; _exception_所属 ABlock 条目数为 1, 则①的记录过程出现一次。②表示该软件样本调用的第二类 ABlock 为 LdrLoadDLL, 其编号为 239, 将 239 记录到 ListAC; LdrLoadDLL 所属 ABlock 条目数为 48, ②的记录过程重复 48 次, 直到 LdrLoadDLL 所属 ABlock 中的全部 LdrLoadDLL 对应编号均记录到 ListAC 中。

文件编号为 771 号软件样本由 ListAC 生成 ArrayAC 的过程如图 4 所示。其中, (1)图表示 771 号样本的 ListAC, 为一维数组; (2)图表示使用公式(1)对 ListAC 进行标准化的结果, 为一维数组; (3)图表示对经标准化的 ListAC 进行零填充, 转化为尺寸为 128×128 的二维数组 ArrayAC。

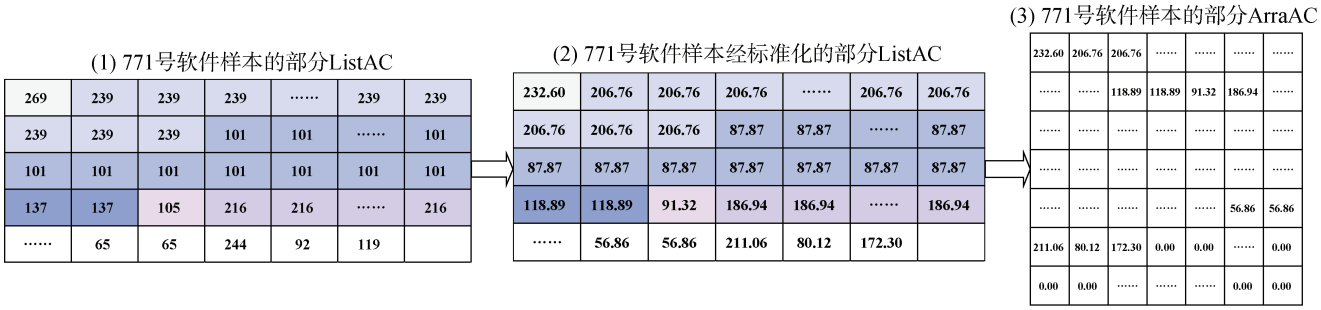


图 4　ArrayAC 生成过程示例

Figure 4　An example of the ArrayAC generation process

5.2　ArrayAD 生成

基于重组后的 API 指令序列提取 API 贡献度特征, 记为 ArrayAD, 作为局部特征反映该恶意软件执行的主要系统交互活动。首先统计各 ABlock 的条目数作为该类 API 对该软件样本的贡献度, 将重组后的 API 指令序列对应的 API 贡献度记录为一维数组 ListAD。然后对 ListAD 中各贡献度值进行标准化; 最后对经标准化的 ListAD 进行零填充, 转化为尺寸为 128×128 的二维特征数组, 记为 ArrayAD, 作为恶意软件的 API 贡献度特征。标准化的 API 贡献度为

$$A_{n\_d} = A_d \times \frac{255}{\max\left(A_d\right)} \tag{2}$$

其中, A_d 为各 API 贡献度, 取值范围为[1, max(A_d)]; max(A_d)为 API 贡献度的最大值; A_{n_d} 为经过标准化的 API 贡献度, 取值范围为(0, 255)。

文件编号为 771 号软件样本生成 ListAD 的过程如图 5 所示。标号①～⑥表示经 API 指令序列重组后, 771 号软件样本对应的前六类 ABlock 生成 ListAD 的过程。

图 5 中, ①表示 771 号软件样本调用的第一类 ABlock 为_exception_, 其对应的 ABlock 条目数为 1, 记_exception_对 771 号的软件样本的 API 贡献度为 1, 表示 771 号软件样本调用_exception_共 1 次, 将 1 记录到 ListAD 中。②表示 771 号软件样本调用的第二类 ABlock 为 LdrLoadDLL, 其对应的 ABlock 条目数为 48, 将 48 记为 LdrLoadDLL 对 771 号的软件样本的 API 贡献度, 并将 48 记录到 ListAD, 同理, ②的记录过程重复 48 次。

文件编号为 771 号软件样本由 ListAD 生成 ArrayAD 的过程如图 6 所示。其中, (1)图表示 771 号样本的 ListAD, 为一维数组; (2)图表示使用公式(2)对 ListAD 进行标准化的结果, 为一维数组; (3)图表示对经标准化的 ListAD 进行零充, 转化为尺寸为 128×128 二维数组 ArrayAD。

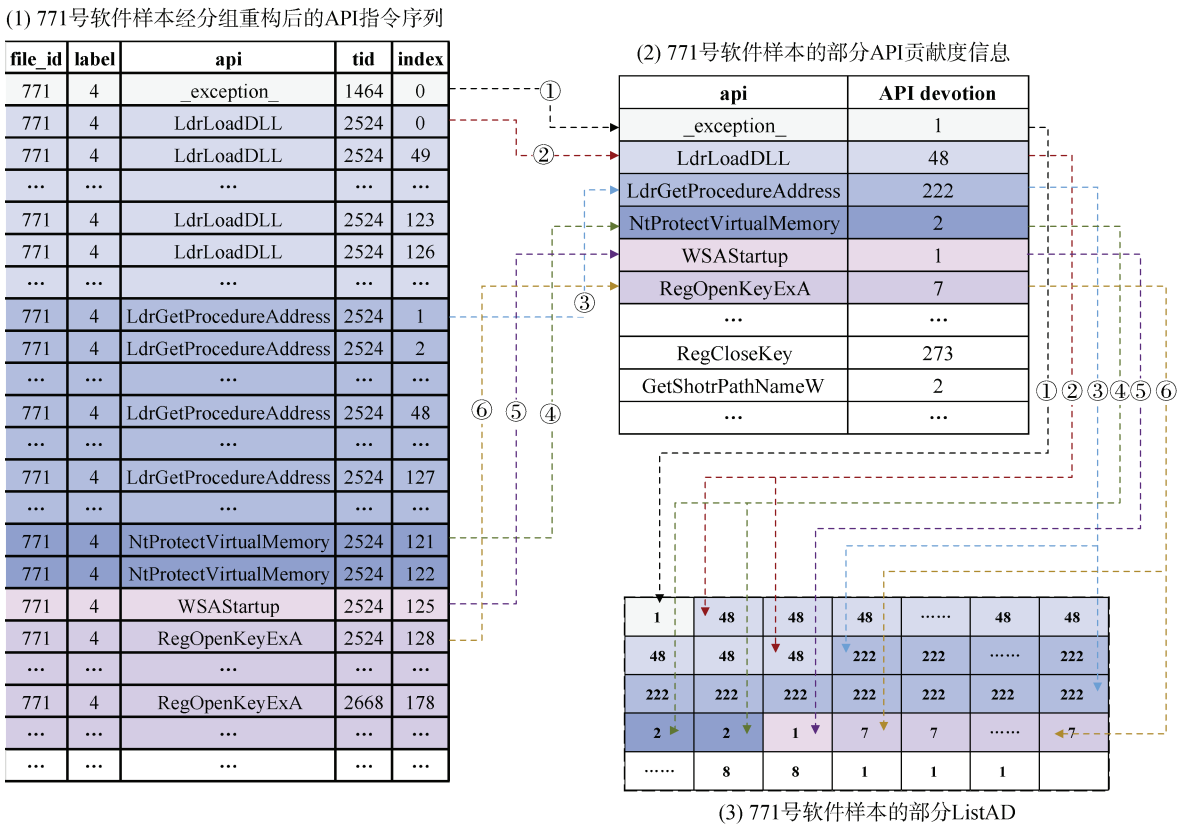
5.3　ArrayAS 生成

基于重组后的 API 指令序列提取 API 顺序索引特征, 记为 ArrayAS, 从时序角度描述恶意软件执行各类交互活动的顺序。首先将重组后的 API 指令序列对应的 API 顺序索引记录为一维数组 ListAS, 并对 ListAS 中各索引值进行标准化。最后对经标准化的 ListAS 进行零填充, 转化为尺寸为 128×128 的二维特征数组, 记为 ArrayAS, 作为恶意软件的 API 顺序索引特征。标准化的 API 顺序索引为

$$A_{n\_s} = (A_s + 1) \times \frac{255}{\max(A_s)} \tag{3}$$

其中, A_s 为 API 顺序索引值, 取值范围为[0, max(A_s)]; max(A_s)为最大 API 顺序索引值; A_{n_s} 为经过标准化的 API 顺序索引值, 取值范围为(0, 255)。

文件编号为 771 号的软件样本生成 ListAS 的过程如图 7 所示。标号①～⑥表示经 API 指令序列重组后, 771 号软件样本对应的前六类 ABlock 生成 ListAS 的过程。



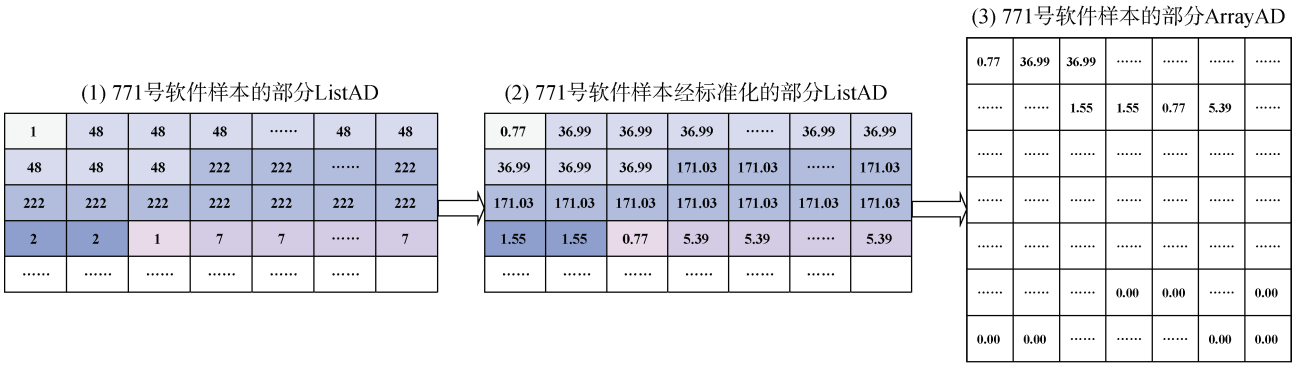
file_id	label	api	tid	index
771	4	_exception_	1464	0
771	4	LdrLoadDLL	2524	0
771	4	LdrLoadDLL	2524	49
…	…	…	…	…
771	4	LdrLoadDLL	2524	123
771	4	LdrLoadDLL	2524	126
…	…	…	…	…
771	4	LdrGetProcedureAddress	2524	1
771	4	LdrGetProcedureAddress	2524	2
…	…	…	…	…
771	4	LdrGetProcedureAddress	2524	48
…	…	…	…	…
771	4	LdrGetProcedureAddress	2524	127
…	…	…	…	…
771	4	NtProtectVirtualMemory	2524	121
771	4	NtProtectVirtualMemory	2524	122
771	4	WSAStartup	2524	125
771	4	RegOpenKeyExA	2524	128
…	…	…	…	…
771	4	RegOpenKeyExA	2668	178
…	…	…	…	…
…	…	…	…	…

api	API devotion
exception	1
LdrLoadDLL	48
LdrGetProcedureAddress	222
NtProtectVirtualMemory	2
WSAStartup	1
RegOpenKeyExA	7
…	…
RegCloseKey	273
GetShotrPathNameW	2
…	…

1	48	48	48	……	48	48
48	48	48	222	222	……	222
222	222	222	222	222	222	222
2	2	1	7	7	……	7
……	8	8	1	1	1	

图 5　ListAD 生成过程示例

Figure 5　An example of the ListAD generation process



1	48	48	48	……	48	48
48	48	48	222	222	……	222
222	222	222	222	222	222	222
2	2	1	7	7	……	7
……	……	……	……	……	……	

0.77	36.99	36.99	36.99	……	36.99	36.99
36.99	36.99	36.99	171.03	171.03	……	171.03
171.03	171.03	171.03	171.03	171.03	171.03	171.03
1.55	1.55	0.77	5.39	5.39	……	5.39
……	……	……	……	……	……	

0.77	36.99	36.99	……	……	……	……
……	……	1.55	1.55	0.77	5.39	……
……	……	……	……	……	……	……
……	……	……	……	……	……	……
……	……	……	……	……	……	……
……	……	……	0.00	0.00	……	0.00
0.00	0.00	……	……	……	0.00	0.00

图 6　ArrayAD 生成过程示例

Figure 6　An example of the ArrayAD generation process

图 7 中, ①表示 771 号软件样本调用的第一类 ABlock 为_exception_, 其对应的 index 为 0, 将 0 记录到 ListAS。②表示 771 号软件样本调用的第二类 ABlock 为 LdrLoadDLL, 其对应的 index 依次为 0, 49, ……, 1716, ……, 将 index 记录到 ListAS。

文件编号为 771 号软件样本由 ListAS 生成 ArrayAS 的过程如图 8 所示。其中, (1)图表示 771 号样本的 ListAS, 为一维数组; (2)图表示使用公式(3)对 ListAS 进行标准化的结果, 为一维数组; (3)图表示对经标准化的 ListAS 进行零填充, 转化为尺寸为 128×128 的二维数组 ArrayAS 的过程。

6　图像生成与增强

6.1　图像生成

使用 3 类恶意软件特征数组填充 RGB 图像的 3 个通道, 生成 FimgCDS 特征图像。填充方法为使用 ArrayAC 特征数组的每个数据项填充 FimgCDS 特征图像的 R 通道, 使用 ArrayAD 特征数组的每个数据项填充 FimgCDS 的 G 通道, 使用 ArrayAS 特征数组的每个数据项填充 FimgCDS 的 B 通道。

填充顺序为从第一行第一列的像素值开始, 由左至右使用二维特征数组第一行的数据项填充对应

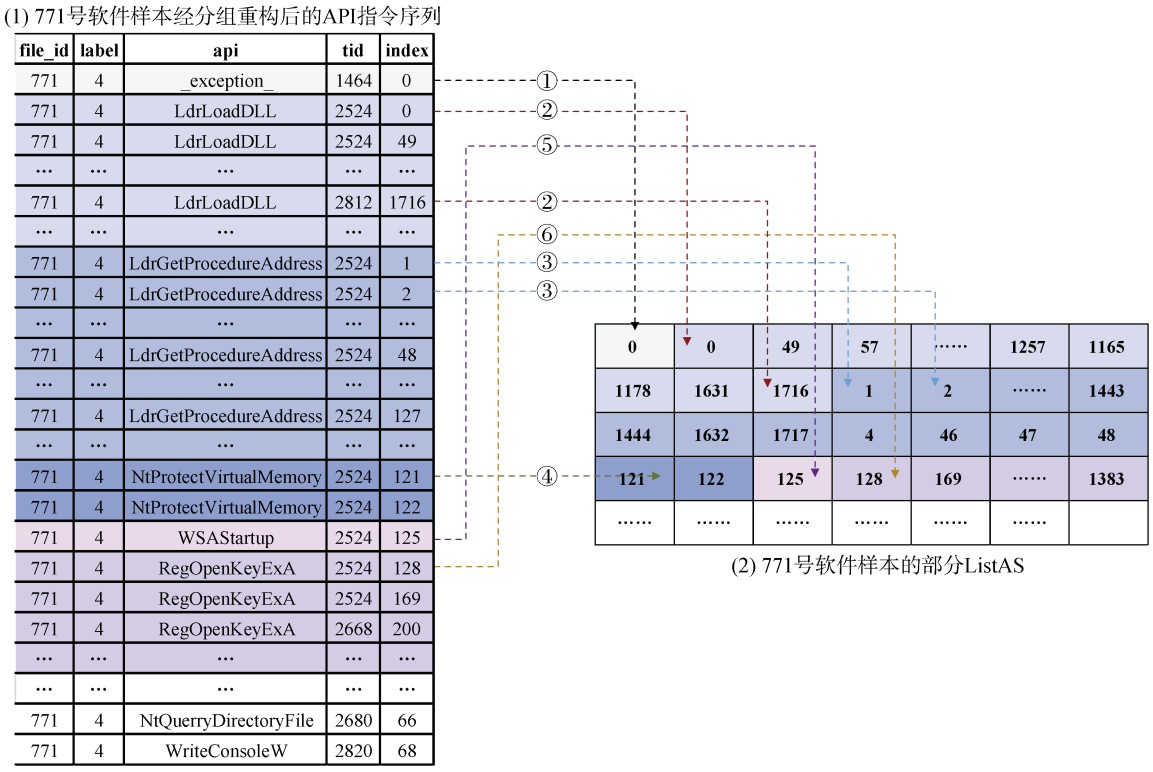


图 7　ListAS 生成过程示例

Figure 7　An example of the ListAS generation process

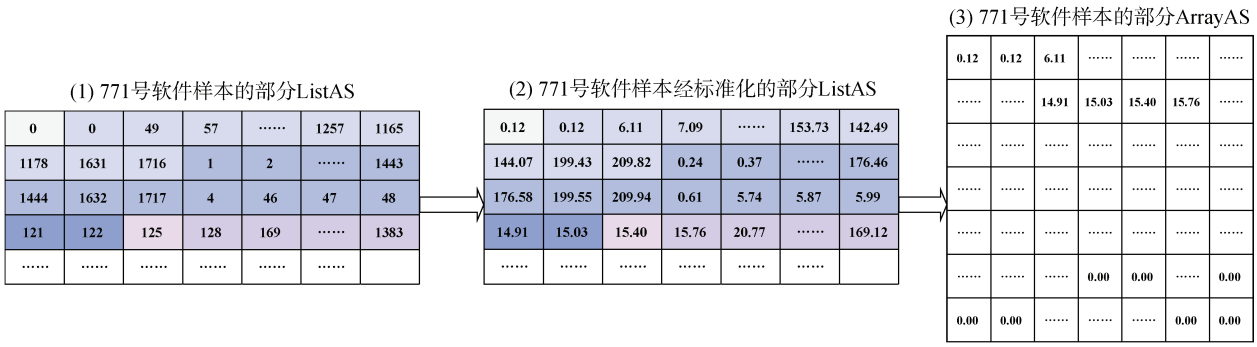


图 8　ArrayAS 生成过程示例

Figure 8　An example of the ArrayAS generation process

通道第一行各列的像素值, 由上至下使用二维特征数组各行的数据项填充对应通道每一行的像素值, 通道尺寸均为 128×128。以数据集 1 中 771 号软件样本为例, 其 FimgCDS 特征图像的三通道填充过程如图 9 所示。

FimgCDS 特征图像 R 通道的单个像素值为恶意软件运行时调用的某类 API 的编号。G 通道对应位置的像素值为该类 API 的贡献度, 在 R 通道的基础上反映对应编号 API 的调用频繁程度。B 通道对应位置的像素值为恶意软件运行时调用该 API 的顺序索引, 在 R 通道的基础上从时序角度反映对应编号 API 的被调用顺序。

FimgCDS 特征图像的各通道均能分别反映恶意软件的一类特征, 即 R 通道反映该恶意软件的全局 API 调用信息, G 通道反映该恶意软件的主要系统交互活动, B 通道反映该恶意软件执行各类交互活动的顺序。特征图像组织结构清晰, 能较为全面地表征恶意软件。以数据集 1 为例, 各软件类别对应的 FimgCDS 特征图像如图 10 所示。

6.2　图像增强

在深度学习中, 为避免过拟合, 通常需要足够的数据来训练模型, 当样本数据集中某类样本数量较少时, 样本比例失衡, 会出现数据不平衡问题。当遇到不平衡数据时, 以检测准确率为学习目标的检

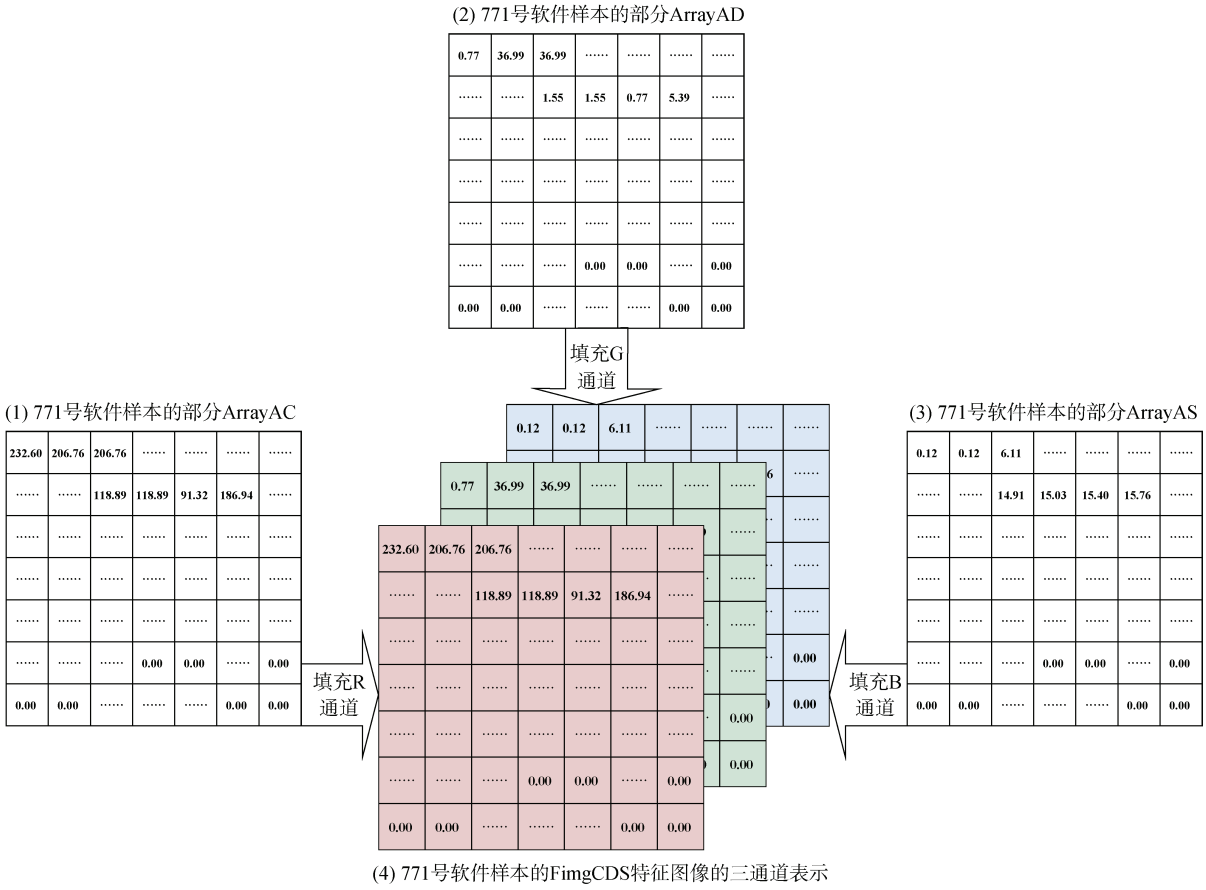


图 9　FimgCDS 特征图像的三通道填充过程示例

Figure 9　Example of 3 channel filling process of FimgCDS feature image

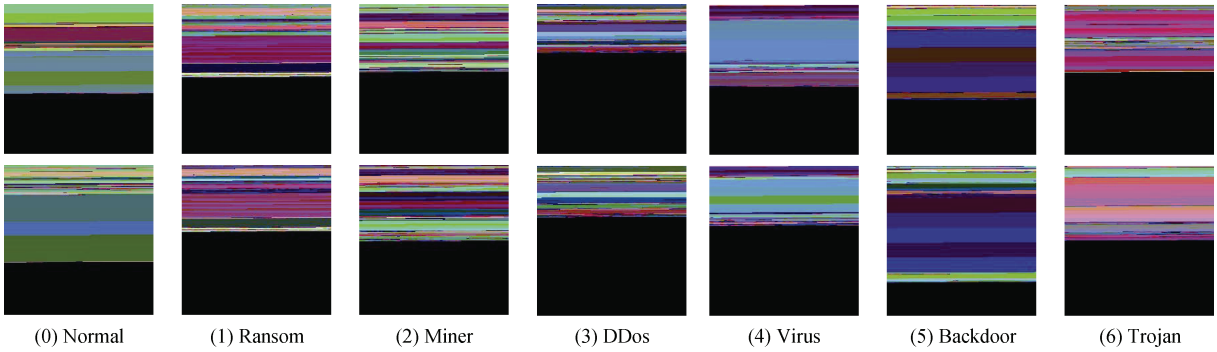


图 10　数据集 1 中各软件类别对应的 FimgCDS 特征图像

Figure 10　FimgCDS feature images for each software category in dataset 1

测算法会过多地关注多数类, 从而使得少数类样本的检测性能下降, 此时需要对少数类样本进行数据增强, 以扩充样本数量, 抑制数据不平衡问题[19]。对于图像数据集, 通常使用旋转、缩放、翻转、随机剪裁、颜色变换等技术进行图像增强。

本文使用亮度增强与锐度增强方法对数据集 1、数据集 2 中各类样本的 FimgCDS 特征图像进行图像增强。同时, 数据集 1 中勒索病毒、后门程序 2 类样本数量较少, 为少数类样本, 样本数据集存在数据不平衡问题, 为解决此问题, 本文使用水平翻转方法处理上述 2 类样本的 FimgCDS 特征图像, 以增加对应类别样本的数量。以数据集 1 为例, 部分 FimgCDS 特征图像经图像增强后效果如图 11 所示。

7　恶意软件特征图像的检测分类

深度学习提供了优于传统机器学习解决方案的性能, 深度学习的模型通过选择适合研究问题的体系结构和优化参数来构建, 其中, CNN 通常用于分类任务[20]。CNN 对图片像素进行学习, 经过大量数

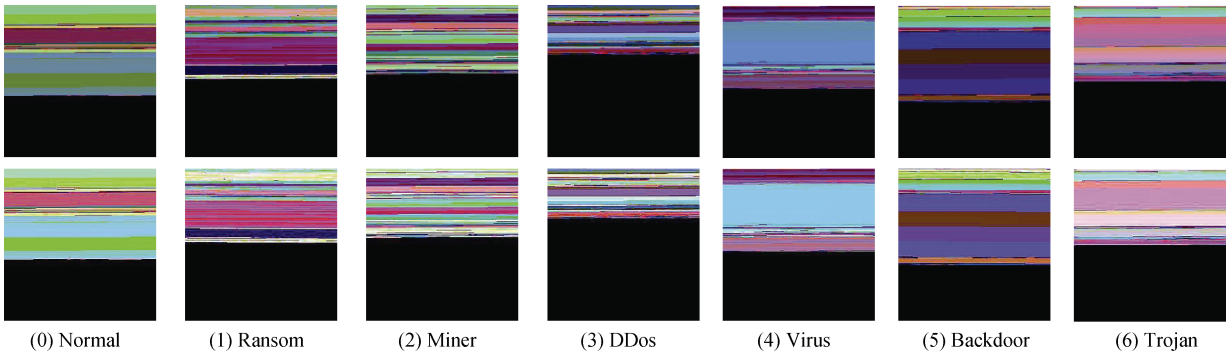


图 11　数据集 1 中各软件类别经过图像增强后对应的 FimgCDS 特征图像

Figure 11　FimgCDS feature images corresponding to each software category after image enhancement in dataset 1

据的训练得到准确稳定的检测结果[21]。卷积层有助于促进抽象和高度非线性模式的学习, 能够捕捉复杂数据的内在结构, 实现特征的自动提取[22]。本文搭建一个轻量型的卷积神经网络 MficNN 分类器, 实现对恶意软件特征图像 FimgCDS 的检测分类, MficNN 分类器的网络结构如图 12 所示。

MficNN 分类器由多个卷积层、最大池化层、全连接层和 Softmax 层组成。卷积层从 FimgCDS 特征图像中提取恶意软件的特征; 最大池化层用于降低由卷积层提取出的特征维度, 减少计算量, 提高分类器的训练速度; 全连接层将卷积层与最大池化层提取的高维特征映射为一维特征向量, 经 Softmax 层处理, 输出该 FimgCDS 特征图像对应的恶意软件样本属于各软件类别的分类概率, 即 FimgCDS 特征图像 y 所属恶意软件类别为 i 的概率 P_i。P_i 为

$$P_i = P(y=i \mid x) = \frac{e^{x^{\top} w_i}}{\sum_{k=1}^{k} e^{x^{\top} w_k}} \tag{4}$$

其中, x 为 Softmax 层的输入, w 为神经元的权重, k 为需要分类的类别数。

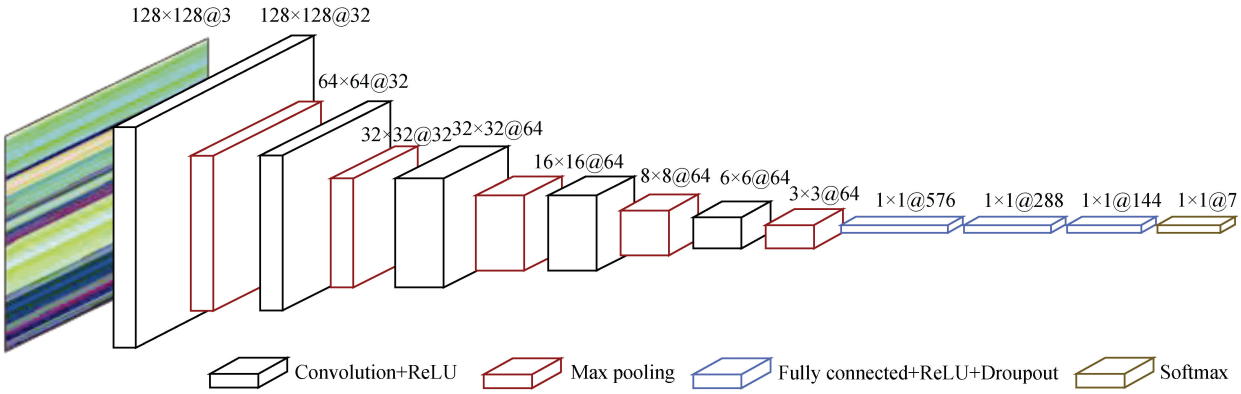


图 12　MficNN 分类器的网络结构示意图

Figure 12　The network structure of the MficNN classifier

将 FimgCDS 特征图像 y 输入 MficNN 分类器, 经过多个卷积层、最大池化层和三层全连接层的处理后, y 将在第三层全连接层被模型处理为长度为 144 的特征向量 $\boldsymbol{x}$。对于该全连接层的输出 x, Softmax 层首先使用公式(4)对 x 进行处理, 处理结果即为 Softmax 层的输出 P_i, 然后将 P_i 的最大值记为 pre,

$$pre = \max\left(P_i\right) \ (i = 0,\ 1,\ 2,\ \ldots, 6) \tag{5}$$

其中, i 表示 FimgCDS 特征图像 y 对应的软件类别。样本数据集包含良性软件和 6 类恶意软件, 即 i 为[0, 6]范围的整数。

记录 pre 对应的 i 值, 图像 y 的检测分类结果即为恶意软件的类别 i。

8　实验与结果分析

8.1　实验数据集及环境

本文研究采用两类恶意软件数据集, 数据集 1 为阿里云提供的恶意程序检测大赛数据集[11]。该样本数据集中的数据是经过沙箱程序模拟运行后的 windows 可执行程序的 API 指令序列, 该样本数据集中软件样本数量为 13787 个, 类别包括良性软件

(Normal)、勒索病毒(Ransom)、挖矿程序(Miner)、DDoS 木马(DDoS)、感染型病毒(Virus)、后门程序(Backdoor)和木马程序(Trojan)。

数据集 2 为 Datacon2019-MaliciousCode[17]。该样本数据集包含 30000 个样本数据, 包括 10000 个黑色样本和 20000 个白色样本。文件类型为 XML 文件, 记录样本软件的执行动作序列, 包括返回值、调用时间、调用的进程号、调用的 API 名等信息。

使用 Python 语言编程实现本文算法, 实验采用 10 倍交叉验证, 硬件计算环境配置为: AMD Ryzen 7 5800H with Radeon Graphics 处理器, NVIDIA GeForce RTX 3060 显卡。

8.2 评价指标

为准确、全面地评价本文恶意软件检测分类方法的性能, 采用准确率(*Accuracy*)、精确率(*Precision*)、召回率(*Recall*)、F1 评分(*F1-score*)4 种指标评价本文方法对恶意软件的检测分类效果, 检测分类性能指标的定义如下:

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

$$Precision=\frac{TP}{TP+FP}\times 100\% \tag{7}$$

$$Recall=\frac{TP}{TP+FN}\times 100\% \tag{8}$$

$$F1-score=\frac{2\times Precision\times Recall}{Precision+Recall} \tag{9}$$

其中, *TP*(True Positive)是将正样本分类为正样本的数量值, *TN*(True Negative)是将负样本分类为负样本的数量值, *FP*(False Positive)是将负样本分类为正样本的数量值, *FN*(False Negative)是将正样本分类为负样本的数量值。

8.3 不同特征组合对检测分类效果的影响与特征提取算法效率评估

不同特征组合对软件样本行为的恶意程度反映不同, 为验证本文方法所选特征组合的有效性, 对不同特征组合下的检测分类效果进行对比实验。在实验中选用 5 组不同的特征组合, 分别为:

1)None 为选择未经重组的原始 API 序列生成的 ArrayAC 作为特征, 填充 R 通道;

2)FimgC 为选择经过重组的 API 指令序列生成的 ArrayAC 作为特征, 填充 R 通道;

3)FimgCD 为在 FimgC 的基础上, 选择经过重组的 API 指令序列生成的 ArrayAD 作为特征, 填充 G 通道;

4)FimgCS 为在 FimgC 的基础上, 选择经过重组的 API 指令序列生成的 ArrayAS 作为特征, 填充 B 通道;

5)FimgCDS 为本文所选特征组合, 即选择经过重组的 API 指令序列生成的 ArrayAC 作为特征, 填充 R 通道、ArrayAD 作为特征, 填充 G 通道、ArrayAS 作为特征, 填充 B 通道。

在相同实验环境和相同参数配置的条件下, 在本文方法中使用上述 5 种特征组合对样本数据集 1 与数据集 2 中的样本进行分类, 得到 5 组恶意软件检测分类性能指标如表 2 所示, 表中粗体表示各类别样本对应检测分类性能指标的最大值。

表 2 不同特征组合的检测分类性能指标

Table 2 Detection and classification performance indicators of different combinations of features

特征组合	数据集 1				数据集 2			
	精确率	召回率	F1 评分	准确率	精确率	召回率	F1 评分	准确率
FimgCDS	**0.9869**	**0.9865**	**0.9867**	**0.9866**	**0.9874**	**0.9879**	**0.9876**	**0.9835**
None	0.7652	0.7647	0.7532	0.7647	0.8753	0.7656	0.8168	0.7710
FimgC	0.8526	0.8325	0.8397	0.8676	0.8778	0.8449	0.8610	0.8181
FimgCS	0.8851	0.8650	0.8744	0.8916	0.8973	0.9060	0.9017	0.8682
FimgCD	0.9136	0.9135	0.9127	0.9135	0.9476	0.9107	0.9288	0.9069

由表 2 可知, 在数据集 1 与数据集 2 下, 各特征组合的检测分类准确率从高到低的排序均为: FimgCDS、FimgCD、FimgCS、FimgC、None。None 条件下生成的特征图像结构与纹理杂乱, FimgC 条件下生成的特征图像结构清晰, 各 ABlock 区域分界明显, 由 FimgC 各检测分类性能指标高于 None 可知, 进行 API 分组与重构能显著提高检测分类效果。由 FimgCD 与 FimgCS 各检测分类性能指标高于 FimgC 可知, API 贡献度和 API 顺序索引的加入能够提高对恶意软件的检测分类效果, 多个维度的特征能从多角度反映恶意软件的行为。FimgCDS 与其他 4 组特征组合相比, 各检测分类性能指标值最高, 表明本文所提特征提取方法及选取的特征组合能全面地反映软件样本的恶意程度, 能提取出更有效的特征,

可准确、有效地检测恶意软件类别。

同时, 为进一步评估本文所提特征提取算法的效率, 在相同实验环境下, 基于各检测分类性能指标值最高的 FimgCDS 特征组合, 记录数据集 1 与数据集 2 中各 API 指令序列经分组重构转化为恶意软件特征图像的耗时, 如表 3 所示。

表 3 特征提取算法的时间消耗对比

Table 3 Comparison of the time consumption of feature extraction algorithm

数据集	最小值(ms)	中位数(ms)	最大值(ms)
数据集 1	0.0150	37.7524	169.7554
数据集 2	15.3728	63.9261	206.1117

由表 3 可知, 在数据集 1 与数据集 2 上, 本文所提特征提取算法的最大时间消耗分别为 169.7554 ms 和 206.1117 ms, 消耗时间的中位数为 37.7524 ms 和 63.9261 ms, 证明本文所提特征提取算法的时间复杂度可控, 具备实际可行性。

8.4 不同图像增强方法组合对检测分类效果的影响

为验证本文所用图像增强方法的有效性, 对图像增强前后的检测分类效果进行对比实验。在实验中使用 7 种图像增强方法处理过的 FimgCDS 特征图像输入分类器进行检测分类, 其中:

1)Base 为不使用图像增强技术对 FimgCDS 特征图像进行处理;

2)With-B 为使用亮度增强技术对 FimgCDS 特征图像进行处理;

3)With-S 为使用锐度增强技术对 FimgCDS 特征图像进行处理;

4)With-R 为使用水平翻转技术处理数据集 1 中的少数类样本与数据集 2 中所有样本的 FimgCDS 特征图像。

在相同实验环境和相同参数配置的条件下, 分别使用上述 4 种图像增强方法的 7 种图像增强方法组合对 FimgCDS 特征图像进行处理, 得到数据集 1、数据集 2 下对经 7 种图像增强方法组合处理的 FimgCDS 特征图像的检测分类性能指标如表 4 所示。

由表 4 可知, 在数据集 1 下, 各图像增强方法组合的检测分类准确率从高到低的排序为: With-BSR、With-SB、With-SR、With-BR、With-S、With-B、Base。在数据集 2 下, 各图像增强方法组合的检测分类准确率从高到低的排序为: With-BSR、With-SR、With-BR、With-BS、With-S、With-B、Base。与 Base 方法相比, With-B 方法与 With-S 方法的各检测分类性能指标更高, 表明使用亮度增强与锐度增强技术可以提高检测分类效果。其中, 亮度增强方法提高了特征图像各区域的亮度, 对于原始图像上较暗的区域, 经亮度增强后该区域的显色更明显。锐度增强方法锐化了特征图像中的色块区域, 使各 ABlock 的边缘区分更明显, 特征图像的结构更清晰。With-BS 方法组合较 With-B 方法与 With-S 方法相比, 各检测分类性能指标更高, 表明同步进行亮度增强与锐度增强可进一步提高检测分类准确率。With-BR 方法组合与 With-SR 方法组合较 With-B 方法与 With-S 方法相比, 各检测分类性能指标进一步提高。其中, 水平翻转方法翻转了特征图像的水平位置, 在未改变图像的颜色与纹理特征的前提下, 有效扩充样本的数量。With-BSR 方法组合的各检测分类性能指标最高, 表明对特征图像进行亮度增强、锐度增强与水平翻转, 可以有效提高检测分类效果。

表 4 不同图像增强方法组合的检测分类性能指标

Table 4 Detection and classification performance indicators of different combinations of image enhancement methods

图像增强方法组合	数据集 1				数据集 2			
	精确率	召回率	F1 评分	准确率	精确率	召回率	F1 评分	准确率
With-BSR	**0.9869**	**0.9865**	**0.9867**	**0.9866**	**0.9874**	**0.9879**	**0.9876**	**0.9835**
Base	0.9145	0.9173	0.9159	0.9195	0.9364	0.9060	0.9210	0.8963
With-B	0.9375	0.9362	0.9368	0.9343	0.9480	0.9385	0.9432	0.9247
With-S	0.9418	0.9360	0.9388	0.9488	0.9446	0.9436	0.9441	0.9255
With-BS	0.9653	0.9612	0.9632	0.9691	0.9616	0.9515	0.9565	0.9424
With-BR	0.9633	0.9642	0.9637	0.9649	0.9688	0.9564	0.9625	0.9504
With-SR	0.9663	0.9581	0.9622	0.9680	0.9709	0.9569	0.9639	0.9522

8.5 不同检测分类方法的性能对比

为验证本文方法对恶意软件检测分类的有效性, 分别采用本文所提方法 FimgCDS, 以及 SLAM[11]方法、CNN-LSTM[12]方法、CruParamer[13]方法、LSTM[14]

方法、SGDNet[15]方法和 LGMal[16]方法等基于 API 指令序列的恶意软件主流检测分类方法, 在相同实验环境和相同参数设置的条件下, 对样本数据集中的样本进行检测分类, 得到 6 种方法对恶意软件的检测分类性能指标如表 5 所示(由于数据集 1 与数据集 2 包含的数据类别有所不同, 基于线程信息的 CNN-LSTM 方法仅在数据集 1 下进行实验, 基于参数信息的 CruParamer 方法仅在数据集 2 下进行实验)。

表 5 不同检测分类方法的检测分类性能指标

Table 5 Detection and classification performance indicators of different combinations of different methods

检测分类方法	数据集 1				数据集 2			
	精确率	召回率	F1 评分	准确率	精确率	召回率	F1 评分	准确率
FimgCDS	0.9869	**0.9865**	**0.9867**	**0.9866**	**0.9874**	**0.9879**	**0.9876**	0.9835
SGDNet	**0.9870**	0.9820	0.9850	0.9730	0.9850	0.9779	0.9814	0.9753
LSTM	0.9703	0.9615	0.9659	0.9734	0.9818	0.9656	0.9736	0.9651
SLAM	0.9863	0.9863	0.9863	0.9723	0.9862	0.9764	0.9812	0.9751
CNN-LSTM	0.9307	0.9265	0.9286	0.9344	—	—	—	—
LGMal	0.8776	0.8808	0.8779	0.8788	0.9256	0.8836	0.9041	0.8750
CruParamer	—	—	—	—	0.9863	0.9841	0.9852	**0.9852**

由表 5 可知, 在数据集 1 下, 6 种检测方法中, FimgCDS 方法的检测分类准确率最高, 为 98.66%, 比 LGMal、CNN-LSTM、SLAM、LSTM 和 SGDNet 分别提高了 10.87%、5.22%、1.43%、1.32%和 1.36%。在数据集 2 下, FimgCDS 方法的检测分类准确率较 CruParamer 方法低 0.17%, 较 LGMal、LSTM、SLAM 和 SGDNet 分别提高了 10.85%、1.84%、0.84%和 0.82%。

与 LGMal 方法和 SGDNet 方法相比, FimgCDS 方法加入了 API 贡献度与 API 顺序索引特征, 多角度的特征有助于更全面地反映软件的恶意行为。与 CNN-LSTM 方法、SLAM 方法和 CruParamer 方法相比, FimgCDS 方法并未忽略非常用的 API 类别, 也未截断 API 序列, 从而保留了恶意软件调用的全局 API 信息, 同时由于没有依赖专家知识对 API 进行分类或定义匹配规则, 减小了人工分类对检测分类结果造成的误差。恶意软件与系统的交互活动复杂, LSTM 方法仅考虑了局部 API 特征, FimgCDS 方法同时考虑局部与全局 API 特征, 有效提高了检测分类性能指标。

上述实验结果表明, FimgCDS 方法在恶意软件检测分类方面具有较好的综合性能。

8.6 不同网络模型的检测分类性能对比

为验证本文所提轻量型卷积神经网络 MficNN 对恶意软件检测的有效性, 在相同实验环境和相同参数配置的条件下, 分别使用本文所提的 MficNN 网络模型与 4 类经典卷积神经网络模型: VGG16[23]、AlexNet[24]、DenseNet[25]、ResNet[26]检测样本数据集生成的 FimgCDS 特征图像, 得到 5 类网络模型对特征图像的检测分类性能评价指标如表 6 所示。

由表 6 可见, 在数据集 1 与数据集 2 下, MficNN 网络模型的检测分类准确率均最高, 分别为 98.66%与 98.35%。比 AlexNet、VGG16、ResNet、DenseNet 等网络模型分别提高了 1.98%、1.40%, 2.74%、3.76%, 35.15%、32.14%和 47.88%、42.03%。同时, MficNN 网络模型的精确率、召回率、F1 评分在上述网络模型中均最高。

表 6 不同网络模型的检测分类性能指标

Table 6 Detection and classification performance indicators of different network models

网络模型	数据集 1				数据集 2			
	精确率	召回率	F1 评分	准确率	精确率	召回率	F1 评分	准确率
MficNN	**0.9869**	**0.9865**	**0.9867**	**0.9866**	**0.9874**	**0.9879**	**0.9876**	**0.9835**
AlexNet	0.9669	0.9668	0.9667	0.9668	0.9834	0.9706	0.9770	0.9695
VGG16	0.9592	0.9592	0.9590	0.9592	0.9784	0.9396	0.9586	0.9459
ResNet	0.5966	0.6351	0.5287	0.6351	0.8473	0.6016	0.7036	0.6621
DenseNet	0.3307	0.5078	0.4005	0.5078	0.7602	0.5016	0.6044	0.5632

为进一步比较各网络模型的性能, 以各项检测分类性能指标均最低的 DenseNet 网络模型为基准, 各网络模型的检测分类性能指标的提升值如图 13 所示, 不同网络模型训练 60 轮次耗时和所占外存空间

对比如图 14 所示。

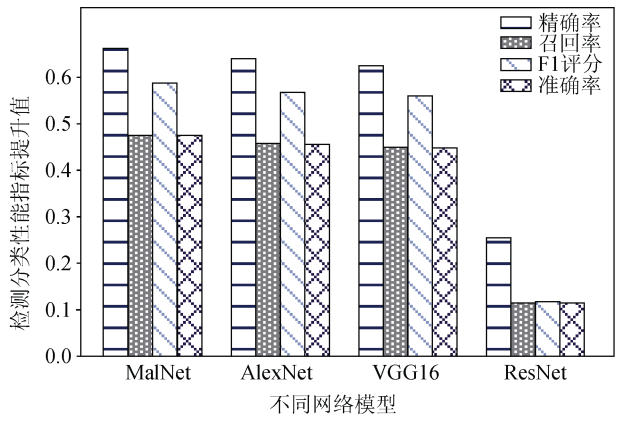


图 13　检测分类性能指标提升值比较

Figure 13　Detection and classification performance indicator values improved by each network model

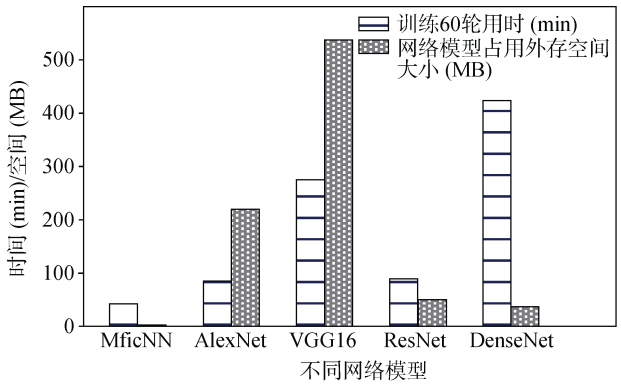


图 14　网络模型所占外存空间与训练耗时

Figure 14　External space occupied by different network models and training time

由图 13 可见, MficNN、AlexNet、VGG16 与 ResNet 网络模型的检测分类性能指标较 DenseNet 网络模型均有明显提升。其中, MficNN、AlexNet、VGG16 三类网络模型的检测分类性能指标提升效果最为明显。由图 14 可知, 在训练耗时方面, DenseNet 网络模型在训练时耗时最多, 为 7.3019 min/轮, VGG16 耗时为 4.5858 min/轮, ResNet 耗时为 1.5625 min/轮, AlexNet 耗时为 1.444 min/轮, MficNN 耗时为 0.6539 min/轮。在占用外存空间大小方面, VGG16 网络占用外存空间最大, 为 527.8639 MB。其次为 AlexNet, 大小为 233.1182 MB。ResNet 与 DenseNet 占用外存空间较小, 分别为 44.7011 MB 和 31.0518 MB。MficNN 模型占用外存空间最小, 为 1.8906 MB。

同时, DenseNet 虽占用外存空间较小, 但在耗时最多的情况下达到最低的准确率, ResNet 虽模型较小、耗时较少, 但所达到的准确率较低, 上述两种网络的综合性能较差, 不适用于检测恶意软件特征图像的工作。VGG16 网络模型的各项检测分类性能指标较高, 增加的网络深度为 VGG16 网络模型带来了良好的检测性能, 但同时也增大了模型大小、降低了检测效率, 使训练耗时较长。AlexNet 网络模型与 VGG16 相比, 网络层数更少, 训练速度更快, 可在更少的训练耗时下达到更高的检测准确率。

MficNN 网络由 5 个卷积层、5 个最大池化层与 4 个线性层组成, 网络层数较少, 训练耗时较少。同时, MficNN 包含多个 ReLU 与 Droupout 算子, ReLU 激活函数对网络层的输出进行线性整流, 有效避免梯度爆炸和梯度消失问题; Droupout 正则化以 0.5 的比例随机忽略或屏蔽部分神经元, 可防止过拟合。实验结果表明, MficNN 网络模型在最短的时间里达到最高的测试准确率, 并且模型存储时占用的外存空间最小, 说明本文提出的轻量型卷积神经网络 MficNN 在恶意软件检测方面具有较好的综合性能, 能够高效、准确地检测恶意软件。

9　总结

为准确、有效地检测恶意软件的类别, 本文选取 API 指令序列这一动态特征, 提出了一种基于重组 API 指令序列图像表示的恶意软件检测分类方法。该方法对不同种类 API 进行编号, 将相同编号的 API 聚合为一个 API 组, 并对各 API 组进行重排序。重排序规则为按照恶意软件运行时对各 API 的调用顺序调整各 API 组的顺序, 同时将各 API 组的条目数记录为该 API 的贡献度。然后基于重排序的 API 组提取 API 编号、API 贡献度和各 API 顺序索引作为特征。最后使用 3 类特征分别填充 RGB 图像的 R 通道、G 通道和 B 通道, 生成 FimgCDS 特征图像, 使用图像增强技术扩充样本数量, 使用自主构建的轻量型卷积神经网络 MficNN 对 FimgCDS 特征图像进行检测分类, 得到恶意软件样本对应的软件类别。

实验结果表明, 与其他方法相比, 本文方法具有较高的恶意软件检测分类准确率、精确率、召回率和 F1 评分。与经典卷积神经网络相比, 本文提出的 MficNN 网络层数更少且参数量更低, 有效减少了恶意软件特征图像检测过程中的计算资源与时间开销。表明本文方法能够有效提取恶意软件的特征, 能够较全面、准确、高效地识别软件的恶意行为。

未来, 将对多个恶意软件数据集进行检测, 并进一步探索更有效的恶意软件特征提取方法和图像增强方法, 提高对少数类样本的检测分类性能指标。同时, 将改进 MficNN 分类器, 在保证模型检测效果的前提下提高网络的健壮性。

参考文献

[1] Miao Y T, Chen C, Pan L, et al. Machine Learning–Based Cyber Attacks Targeting on Controlled Information: A Survey[J]. *ACM Computing Surveys*, 2021, 54(7): 139.

[2] Qiu J Y, Zhang J, Luo W, et al. A Survey of Android Malware Detection with Deep Neural Models[J]. *ACM Computing Surveys*, 2020, 53(6): 126.

[3] Cui Z H, Xue F, Cai X J, et al. Detection of Malicious Code Variants Based on Deep Learning[J]. *IEEE Transactions on Industrial Informatics*, 2018, 14(7): 3187-3196.

[4] Zhu X J, Huang J, Wang B, et al. Malware Homology Determination Using Visualized Images and Feature Fusion[J]. *PeerJ Computer Science*, 2021, 7: e494.

[5] Pinhero A, M L A, Vinod P, et al. Malware Detection Employed by Visualization and Deep Neural Network[J]. *Computers & Security*, 2021, 105: 102247.

[6] Vasan D, Alazab M, Wassan S, et al. IMCFN: Image-Based Malware Classification Using Fine-Tuned Convolutional Neural Network Architecture[J]. *Computer Networks*, 2020, 171: 107138.

[7] Jian Y F, Kuang H B, Ren C L, et al. A Novel Framework for Image-Based Malware Detection with a Deep Neural Network[J]. *Computers & Security*, 2021, 109: 102400.

[8] Xiao M, Guo C, Shen G W, et al. Image-Based Malware Classification Using Section Distribution Information[J]. *Computers & Security*, 2021, 110: 102420.

[9] Han W J, Xue J F, Wang Y, et al. MalInsight: A Systematic Profiling Based Malware Detection Framework[J]. *Journal of Network and Computer Applications*, 2019, 125: 236-250.

[10] Wang J L, Zhang C, Qi X Y, et al. A Survey of Intelligent Malware Detection on Windows Platform[J]. *Journal of Computer Research and Development*, 2021, 58(5): 977-994.
(汪嘉来, 张超, 戚旭衍, 等. Windows 平台恶意软件智能检测综述[J]. *计算机研究与发展*, 2021, 58(5): 977-994.)

[11] Chen J, Guo S Z, Ma X, et al. SLAM: A Malware Detection Method Based on Sliding Local Attention Mechanism[J]. *Security and Communication Networks*, 2020, 2020: 6724513.

[12] Xu A D, Chen L, Kuang X Y, et al. A Hybrid Deep Learning Model for Malicious Behavior Detection[C]. *2020 IEEE 6th Intl Conference on Big Data Security on Cloud*, *IEEE Intl Conference on High Performance and Smart Computing*, *and IEEE Intl Conference on Intelligent Data and Security*, 2020: 55-59.

[13] Chen X H, Hao Z Y, Li L, et al. CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection[J]. *IEEE Transactions on Information Forensics and Security*, 2022, 17: 788-803.

[14] Ma X, Guo S Z, Bai W, et al. An API Semantics-Aware Malware Detection Method Based on Deep Learning[J]. *Security and Communication Networks*, 2019, 2019: 1315047.

[15] Zhang Z K, Li Y D, Dong H R, et al. Spectral-Based Directed Graph Network for Malware Detection[J]. *IEEE Transactions on Network Science and Engineering*, 2021, 8(2): 957-970.

[16] Chai Y H, Qiu J, Su S, et al. LGMal: A Joint Framework Based on Local and Global Features for Malware Detection[C]. *2020 International Wireless Communications and Mobile Computing*, 2020: 463-468.

[17] (2019). Malicious-Code-Dataset. [Online]. Available: https://github.com/kericwy1337.

[18] Yang H Y, Zhang Z X, Xie L X, et al. Network Security Situation Assessment with Network Attack Behavior Classification[J]. *International Journal of Intelligent Systems*, 2022, 37(10): 6909-6927.

[19] Yang H Y, Zeng R Y, Xu G Q, et al. A Network Security Situation Assessment Method Based on Adversarial Deep Learning[J]. *Applied Soft Computing*, 2021, 102: 107096.

[20] Zhang J, Pan L, Han Q L, et al. Deep Learning Based Attack Detection for Cyber-Physical System Cybersecurity: A Survey[J]. *IEEE/CAA Journal of Automatica Sinica*, 2022, 9(3): 377-391.

[21] Yang H Y, Feng Y H. A Pythagorean Fuzzy Petri Net Based Security Assessment Model for Civil Aviation Airport Security Inspection Information System[J]. *International Journal of Intelligent Systems*, 2021, 36(5): 2122-2143.

[22] Lin G J, Wen S, Han Q L, et al. Software Vulnerability Detection Using Deep Neural Networks: A Survey[J]. *Proceedings of the IEEE*, 2020, 108(10): 1825-1848.

[23] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. *3rd International Conference on Learning Representations*, *ICLR 2015 - Conference Track Proceedings*, 2015: 1-14.

[24] Krizhevsky A, Sutskever I, Hinton G E. ImageNet Classification with Deep Convolutional Neural Networks[J]. *Communications of the ACM*, 2017, 60(6): 84-90.

[25] Huang G, Liu Z, Van Der Maaten L, et al. Densely Connected Convolutional Networks[C]. *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017: 2261-2269.

[26] He K M, Zhang X Y, Ren S Q, et al. Deep Residual Learning for Image Recognition[C]. *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016: 770-778.

杨宏宇 CCF 高级会员。于 2003 年在天津大学计算机应用技术专业获得博士学位。现任中国民航大学安全科学与技术学院教授, 博士生导师。研究领域为网络与系统安全。Email: hyyang@cauc.edu.cn

张宇沛 于 2020 年在四川师范大学信息与计算科学专业获得学士学位。现在中国民航大学计算机技术专业攻读硕士学位。研究领域为网络信息安全。Email: ypeizhang@163.com

张良 于 2017 年在天津大学信息与通信工程专业获得博士学位。现为亚利桑那大学研究员。研究领域为强化学习和基于深度学习的信号处理。Email: liangzh@arizona.edu

成翔 于 2021 年在南京航空航天大学计算机科学与技术专业获得博士学位。现任扬州大学信息工程学院实验师。研究领域为网络与系统安全、网络安全态势感知、APT 攻击检测。Email: huozhai9527@126.com