文献引用格式：梁树彬 , 郑力 , 钟杰 , 等 . 基于卷积神经网络的源代码漏洞检测模型 [J]. 通信技术 ,2022,55(4):493-499.
doi:10.3969/j.issn.1002-0802.2022.04.013

基于卷积神经网络的源代码漏洞检测模型 *

梁树彬[1]，郑　力[2]，钟　杰[2]，胡　勇[1]
（1. 四川大学 网络空间安全学院，四川 成都 610207；2. 中物院成都科学技术发展中心，四川 成都 610299）

摘　要：深度学习被应用于源代码漏洞检测，以解决传统的源代码漏洞检测方法高误报率、高漏报率等问题，但常见的基于循环神经网络的方法面临训练速度慢、可解释性差等问题。针对这些问题，提出了一种结合卷积神经网络和高速神经网络的源代码漏洞检测模型。该模型使用代码切片作为代码表征，通过卷积神经网络提取特征，结合高速神经网络学习代码的高级特征，并使用显著图对模型检测结果进行解释。实验结果表明，与现有方法相比，该模型可大幅降低漏报率，有效提升准确率、F1 分数、马修斯相关系数等指标，还能够提升训练与预测速度。

关键词：源代码漏洞检测；深度学习；卷积神经网络；高速神经网络
中图分类号：TP311　　**文献标识码：**A　　**文章编号：**1002-0802(2022)-04-0493-07

Source Code Vulnerability Detection Model Based on Convolutional Neural Networks

LIANG Shubin[1], ZHENG Li[2], ZHONG Jie[2], HU Yong[1]
(1.School of Cyber Science and Engineering, Sichuan University, Chengdu Sichuan 610207, China;
2. Chengdu Science and Technology Development Center, Chengdu Sichuan 610299, China)

Abstract: Deep learning is applied to source code vulnerability detection to solve the problems of high false positive rate and high false negative rate of traditional source code vulnerability detection methods, but the common recurrent neural network-based methods face problems such as difficulty in capturing long-distance dependencies, slow training speed, and poor interpretability. To address these problems, a source code vulnerability detection model combining convolutional neural network and highway neural network is proposed. The model uses code slices as code representations, extracts features through convolutional neural networks, combines highway neural networks to learn high-level features of the code, and uses saliency maps to interpret the model detection results. Experimental results indicate that the model can significantly reduce the false negative rate, effectively improve the accuracy rate, F1 score, Matthews correlation coefficient and other indicators, and can also enhance the training and detection speed compared with existing methods.

Keywords: source code vulnerability detection; deep learning; convolutional neural network; highway network

0　引　言

信息化的推进使得软件系统功能愈加复杂，而由设计缺陷和实现错误引起的软件漏洞不可避免。频发的软件安全事件和持续增长的披露漏洞数表明软件安全形势仍然十分严峻。源代码漏洞检测是提升软件质量与安全性，以及减少漏洞数和降低漏洞

* **收稿日期：**2021-12-09；**修回日期：**2022-03-10　Received date:2021-12-09; Revised date:2022-03-10

修复成本的重要手段，通常与软件开发生命周期紧密结合。在开发阶段，源代码漏洞检测对源代码进行静态分析，检查编码错误和潜在的漏洞等；在测试阶段，源代码漏洞检测对可执行文件进行动态分析，测试软件实际运行时可能出现的问题。

静态分析在不运行软件的情况下对源代码进行分析来检测代码缺陷。常见的静态分析技术包括词法分析、数据流分析、符号执行、模型检查、定理证明等，但依赖于专家制定规则，存在主观性较强，规则不完善等问题。动态测试执行目标程序并监控其运行状况来挖掘漏洞，由于获取了具体的运行信息，其误报率较低，通常使用模糊测试、动态污点分析等方法，但其存在分析效率低和代码覆盖率低导致的漏报率高、资源消耗大、复杂漏洞难以发现等问题。

上述的漏洞检测方法各有其局限性，一般仅适用于小规模软件，在面对大规模且复杂度较高的软件系统时，通常无法满足安全性分析需求，因此，如何在大规模且复杂的软件系统中快速、准确地检测漏洞成为软件安全领域亟待解决的问题之一。

2012 年，Hindle 等人[1]使用 N 元语言模型对源代码进行建模，表明编程语言与自然语言有相似的可预测的统计学特性，并提出代码自然性假设。受到代码自然性假设的启发，研究人员开始将自然语言处理技术应用于源代码，通过神经网络对代码的词法、语法和语义特征进行提取，进而检测漏洞，解决静态分析误报率高的问题。

本文针对现有基于循环神经网络的方法存在训练速度慢、可解释性差等问题，提出结合卷积神经网络和高速神经网络[2]的源代码漏洞检测模型，主要工作包括两个方面：

（1）使用基于语义的漏洞候选（Semantics-based Vulnerability Candidates，SeVC）[3]作为代码表征，其基于程序依赖图（Program Dependency Graph）对代码进行切片，去除与漏洞无关的冗余信息，保留数据依赖和控制依赖信息；使用卷积神经网络提取代码的局部特征，结合高速神经网络进行特征变换，学习代码的抽象语义特征。实验结果表明，本文提出的模型能大幅降低漏报率，有效提升准确率、F1 分数和马修斯相关系数[4]（Matthews Correlation Coefficient，MCC）等指标。

（2）针对现有方法可解释性较低的问题，使用显著图[5]（Saliency map）对模型检测结果进行解释，实验结果表明，根据显著图得出的重要性分数不能准确反映漏洞的成因及位置，但能反映出对预测结果重要的代码元素，可以辅助人工验证及修复漏洞。

1 相关工作

源代码本质上属于文本信息，于是自然语言处理中使用的技术被应用于源代码漏洞检测，通常利用循环神经网络来提取源代码中的语义特征，从而进行漏洞检测。

Li 等人[6]提出包含数据依赖信息的代码片段（code gadget）作为代码表征，使用双向长短期记忆网络（Bidirectional Long Short-Term Memory，BLSTM）模型检测特定的 code gadget 是否含有漏洞，但 code gadget 只包含数据依赖信息，于是 Li 等人[3]提出基于语义的漏洞候选作为代码表征，包含数据依赖和控制依赖信息，实验结果表明加入数据依赖和控制依赖信息的模型能平均降低误报率 30.4%。Li 等人[7]还认为基于源代码的表征不能充分表现控制流信息和变量之间的关系，一些语义上的信息不能被其捕获，所以提出了基于中间代码的代码表征基于中间表示和语义的漏洞候选（intermediate code- and Semantics-based Vulnerability Candidate，iSeVC），充分捕获代码的语义信息，并实现漏洞的细粒度定位。相比 SeVC，使用 iSeVC 代码表征的模型误报率降低了 1.7%，漏报率降低了 28.8%。Cao 等人[8]先对源代码进行离散傅里叶变换将其转化到频率域，再使用卷积神经网络（Convolutional Neural Network，CNN）和 BLSTM 在频率域捕获局部和全局特征，将其进行逆傅里叶变换后使用注意力机制捕获重要的信息，并利用注意力分数对模型预测结果解释。Zou 等人[9]在 code gadget 的基础上引入控制依赖信息，提出代码注意力（code attention），并使用 BLSTM 模型实现 40 种通用缺陷枚举（Common Weakness Enumeration，CWE）类型的检测，相比 code gadget，漏报率降低 12.27%。上述方法主要使用循环神经网络来构建模型，训练速度较慢，而 Tang 等人[10]使用极限学习机（Extreme Learning Machines）并结合核方法来构建模型，在加速模型训练的同时提升模型精确率。

现有的方法主要存在两个问题：基于循环神经网络的方法的训练与预测速度较慢；大多数方法可解释性较差，对源漏洞检测模型的可解释性研究较少。因此，本文提出结合高速神经网络的卷积神经网络（CNN with Highway，CNNH）模型，提升学习

效率和效果，并使用显著图对模型预测结果进行解释，确定与漏洞相关的代码元素，辅助人工验证与修复漏洞。

2　源代码漏洞检测方法

如图 1 所示，本文所提源代码漏洞检测方法的整体流程为，先对切片后的代码进行符号化，去除标识符信息；再训练神经网络模型；然后，在预测时使用训练好的模型得到检测结果，并使用显著图对模型预测结果进行解释；最后，确定每个代码元素对预测结果的重要性分数。

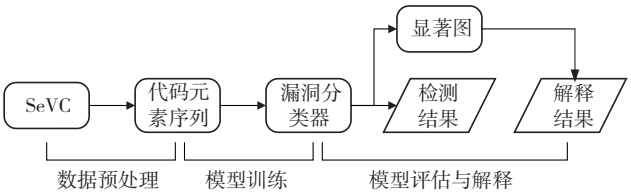

图 1　漏洞检测流程

2.1　数据预处理

本文使用 SeVC 作为代码表征，包含代码的数据依赖和控制依赖信息。数据的预处理主要分为标识符符号化和代码分词两个步骤。

标识符符号化步骤中，用户定义的标识符与漏洞无关，将其转化为符号表示可减小单词表的规模，提升训练效率。此外，数据样本的函数名包含标签信息，需对其进行符号化消除标签信息。根据切片中变量的出现顺序将变量重命名为 variable_0, variable_1 等。根据切片中函数出现的顺序，将函数重命名为 func_0, func_1 等，标识符符号化示例如图 2 所示，其中敏感操作，比如 memcpy、strcat 等不进行符号化。

```
static void goodG2B1 ( )
char * data ;
char dataBuffer [ FILENAME_MAX ] = BASEPATH ;
data = dataBuffer ;
strcat ( data, “file.txt” ) ;
pFile = FOPEN ( data, “wb+” ) ;
If ( pFile != NULL )
Fclose ( pFile ) ;
```

```
static void func_0 ( )
char * variable_0 ;
char variable_1 [ variable_2 ] = variable_3 ;
variable_0 = variable_1 ;
strcat ( variable_0, “file.txt” ) ;
variable_4 = func_1 ( variable_0, “wb+” ) ;
If ( variable_4 != NULL )
fclose ( variable_4 ) ;
```

图 2　标识符符号化示例

代码分词步骤中，将符号化的切片进行分词，将其转化为代码元素序列。

2.2　模型构建

源代码是文本信息，与自然语言文本类似，都具有局部的特征结构，而卷积神经网络具有局部连接的特性，非常适合提取代码局部的特征。本文使用多尺度的卷积神经网络来提取代码的局部特征，并使用高速神经网络将局部特征组合起来形成高层、抽象的代码特征。

本文基于 Kim[11] 的网络模型架构进行改进，采用 4 种卷积核大小的卷积层，并加大卷积核尺寸，使模型能捕获更长距离的依赖信息。在此基础上，添加高速神经网络层形成 CNNH 模型，模型结构如图 3 所示。

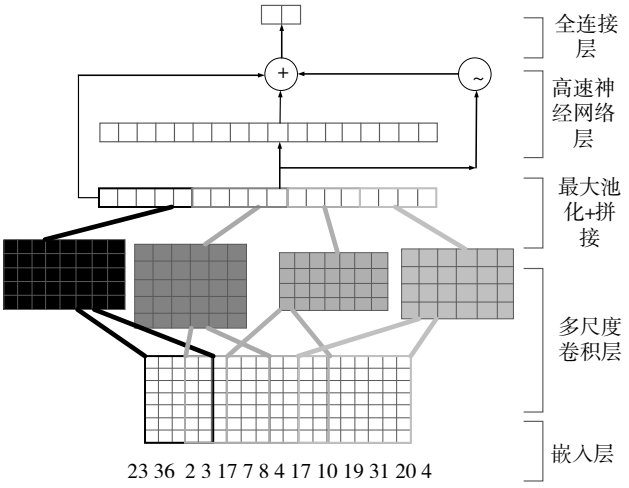

图 3　CNNH 模型结构

如图 3 所示，CNNH 模型前向传播过程为：首先，使用训练数据构建单词表，并将代码元素序列转化为整数索引序列；其次，嵌入层将整数索引映射为低维稠密的嵌入向量，嵌入层并不使用 Word2vec 等预训练词嵌入，而是随机初始化；再次，多尺度卷积层包含 4 种卷积核尺寸的卷积层，每个卷积层后接全局最大池化操作，全局最大池化具有平衡不变性的特点，用于提取最重要的特征，通过池化还可以处理变长序列；最后，拼接层将多尺度卷积层提取的特征进行拼接，得到多尺度的代码特征。

假设序列长度为 n，经过嵌入层后序列可表示为：

$$\boldsymbol{x}_{1:n}=\boldsymbol{x}_1 \oplus \boldsymbol{x}_2 \oplus \cdots \oplus \boldsymbol{x}_n \quad (1)$$

式中：⊕为拼接操作。

假设 $\boldsymbol{x}_{i:i+j}$ 为 $\boldsymbol{x}_i, \boldsymbol{x}_{i+1}, \cdots, \boldsymbol{x}_{i+j}$ 拼接后的张量，对其进行一维卷积操作产生新的特征，比如通过 $\boldsymbol{W}_C$ 过滤器产生特征 c_i：

$$c_i=C(\boldsymbol{x}_{i:i+h-1},\boldsymbol{W}_C,\boldsymbol{b}) \quad (2)$$

式中：C 为非线性变换；$\boldsymbol{W}_C$ 为过滤器参数；$\boldsymbol{b}$ 为偏置项。

将过滤器应用于整个序列产生特征图（feature map）。

$$\boldsymbol{c}=[c_1,c_2,\cdots,c_n] \quad (3)$$

然后在特征图上进行最大池化操作，作为该过滤器提取的特征：

$$\hat{c}_i=\max\{\boldsymbol{c}\} \quad (4)$$

假设使用 k 个过滤器，过滤器大小为 5，得到最大池化后的特征图：

$$\hat{\boldsymbol{c}}_k^5=[\hat{c}_1,\hat{c}_2,\cdots,\hat{c}_k] \quad (5)$$

使用多种过滤器大小来获得不同尺度的代码特征，进行拼接后得到特征，假设使用 4 种大小的过滤器，分别为 5,6,7,8，得到多尺度的代码特征：

$$\boldsymbol{y}=\hat{\boldsymbol{c}}_k^5\oplus\hat{\boldsymbol{c}}_k^6\oplus\hat{\boldsymbol{c}}_k^7\oplus\hat{\boldsymbol{c}}_k^8 \quad (6)$$

随后使用高速神经网络层特征进行组合、变换，高速神经网络基于门机制引入变换门（transformer gate）和进位门（carry gate）对一部分特征进行处理，另一部分特征直接通过，可以让网络高效地进行特征学习，计算方式为：

$$z=H(\boldsymbol{y},\boldsymbol{W}_H,\boldsymbol{b}_H)\cdot T(\boldsymbol{y},\boldsymbol{W}_T,\boldsymbol{b}_T)+\boldsymbol{y}\cdot C(\boldsymbol{y},\boldsymbol{W}_C,\boldsymbol{b}_C) \quad (7)$$

式中：z 为高速神经网络层输出；H 为特征变换全连接层；T 为变换门；C 为进位门；$\boldsymbol{W}_H$ 为特征变换参数；$\boldsymbol{W}_T$ 为变换门参数；$\boldsymbol{W}_C$ 为进位门参数。

为了简单，设置 $C=1-T$，于是得到：

$$\begin{cases} z=\boldsymbol{y},T(\boldsymbol{y},\boldsymbol{W}_T,\boldsymbol{b}_T)=0 \\ z=H(\boldsymbol{y},\boldsymbol{W}_H,\boldsymbol{b}_H),T(\boldsymbol{y},\boldsymbol{W}_T,\boldsymbol{b}_T)=1 \end{cases} \quad (8)$$

高速神经网络根据变换门的输出，输出为 0 的部分直接输出到下一层，输出为 1 的部分进行变换后再输出到下一层。

最后通过全连接层得到每个类别的概率。

2.3 显著图

本文利用显著图（Saliency map）对模型预测结果进行解释，使模型具备一定的可解释性。假定模型及其损失函数是可微分的，对每一个输入嵌入向量可以得到其对应的梯度，使用每个嵌入向量的梯度，可以为每个输入的代码元素计算一个分数，该分数表示模型预测结果对嵌入向量变化的敏感程度，得分越高，表示模型对其越敏感，即重要性越高。本文使用 Allennlp[12] 提供的显著图模块得到代码元素的重要性分数。

3 实验与评估

本节对实验使用的实验环境、数据集及评价指标进行介绍，并与其他方法进行对比，验证本文提出方法的有效性。

3.1 实验环境

本文实验使用的计算机配置：GPU 为 GeForce 1080Ti，CPU 为 Intel® Core ™ i7-7700。操作系统为 Linux Mint 20.3，内核为 5.4.0-96-generic，开发语言为 Python（Python 3.8），基于 Allennlp 进行编码，包括数据读取和模型构建，GPU 加速环境为 CUDA 10.3，相关依赖包为 PyTorch 1.10, Allennlp 2.80。

3.2 数据集

采用 Li 等人[3] 从软件保障参考数据集（Software Assurance Reference Dataset，SARD）、美国国家漏洞数据库（National Vulnerability Database，NVD）等提取的 SeVC 数据集对模型进行评估。数据集中每个代码切片有 0 或 1 的标签，1 代码为漏洞样本，0 代码为非漏洞样本。数据集的划分与原文中同样采取 8 ∶ 2 作为训练集和测试集的划分比例，并使用该文献使用的 BLSTM 和 BGRU 模型作为对比。数据集共包含 4 个子数据集，分别为函数调用（API Function Call）、数组使用（Array Usage）、指针使用（Pointer Usage）和算术表达式（Arithmetic Expression）子数据集。本文首先针对每种类别的数据集分别使用单独的模型进行训练与测试，其次将所有数据合并使用模型进行训练与测试。数据集信息如表 1 所示。

表 1 数据集样本数目

数据集类别	样本总数	漏洞样本数	非漏洞样本数
函数调用	64 403	13 603	50 800
算术表达式	22 154	3 475	18 679
数组使用	42 229	10 926	31 303
指针使用	291 841	28 391	263 450
总数据集	420 627	56 395	465 232

3.3 评价指标

本文采用的指标为分类模型中常用的指标，包括误报率、漏报率、精确率、准确率、F1 分数、马修斯相关系数（Matthews Correlation Coefficient，MCC）。判断代码片段是否包含漏洞为二分类问题，将漏洞样本作为正样本，将无漏洞样本作为负样本。误报率表示负样本被预测为正样本的个数占总的负

样本的比例。漏报率为正样本被预测为负样本占总的正样本的比例。准确率为预测正确的样本占总量的比例，数据不均衡情况下不能准确评价模型的性能。精确率为预测为正样本且预测正确的样本数占预测为正样本数的比例。召回率为预测为正且预测正确的样本占实际所有正样本数的比例，召回率与漏报率之和为 1。F1 分数为精确率和召回率的加权平均。MCC[4] 系数表示实际样本与预测样本之间的相关系数，值域为 [–1,1]，当值为 1 时，代表模型完美预测；值为 0 时，代表预测不比随机预测好；值为 –1 时，则代表预测与实际结果完全不一致，适用于评估类别不平衡下的模型性能。

3.4　实验参数

参数设置主要包括模型配置参数与训练配置参数两个部分。

模型配置参数如表 2 所示，其中最重要的参数为卷积核尺寸，包含 4 种卷积核尺寸，4 种卷积核大小的卷积层提取代码的 N 元语法特征。实验表明，与自然语言文本相比，使用大尺寸卷积核在源代码上的表现更好，大尺寸卷积核能捕获较长的依赖关系。

表 2　模型参数

参　数	值
嵌入向量维度	64
过滤器数量	128
卷积核尺寸	5,6,7,8
高速网络层数	2
激活函数	ReLu
Dropout	0.1

训练配置参数包括优化器、损失函数、学习率、权重衰减，参数如表 3 所示。

表 3　训练参数

参　数	值
优化器	Adam
损失函数	交叉熵损失函数
学习率	0.002
权重衰减	0.000 5

3.5　实验结果及分析

3.5.1　评价指标对比

本文与 SySeVR 中的结果进行对比来验证本文提出的 CNNH 模型的有效性，如表 4 为函数调用子数据集的实验结果，表 5 为算术表达式子数据集的实验结果，表 6 为数组使用子数据集的实验结果，表 7 为指针使用子数据集的实验结果，表 8 为整个数据集的实验结果。其中 CNN 代表仅含多尺度卷积神经网络的模型，GRU 和 LSTM 为循环神经网络基线模型。

表 4　函数调用数据集实验结果　　%

模　型	误报率	漏报率	精确率	准确率	F1 分数	MCC 系数
CNNH	3.3	**4.3**	**96.5**	88.9	**92.1**	**90.0**
CNN	2.4	8.5	96.3	**91.3**	91.3	89.0
GRU	3.5	14.5	94.1	86.6	86.1	82.3
LSTM	3.7	13.6	94.2	86.3	86.3	82.6
文献 [3]	**2.1**	17.5	94.7	91.5	86.8	83.6

表 5　算术表达式数据集实验结果　　%

模　型	误报率	漏报率	精确率	准确率	F1 分数	MCC 系数
CNNH	1.9	**6.5**	**97.4**	90.6	**92.0**	**90.5**
CNN	1.8	7.8	97.3	90.8	91.5	89.9
GRU	1.9	11.6	96.6	90.1	89.2	87.2
LSTM	**1.3**	12.0	96.9	**92.6**	90.2	88.5
文献 [3]	1.5	18.3	96.6	87.9	84.7	82.2

表 6　数组使用数据集实验结果　　%

模　型	误报率	漏报率	精确率	准确率	F1 分数	MCC 系数
CNNH	4.8	**4.1**	**95.3**	**87.3**	**91.4**	**88.4**
CNN	4.7	6.7	94.8	**87.3**	90.2	86.7
GRU	4.6	10.6	93.8	87.0	88.2	84.0
LSTM	8.1	9.3	91.6	79.6	84.8	79.3
文献 [3]	**3.8**	17.1	92.7	88.3	85.5	80.7

表 7　指针使用数据集实验结果

模　型	误报率	漏报率	精确率	准确率	F1 分数	MCC 系数
CNNH	**0.1**	**7.6**	**98.4**	**91.4**	**91.8**	**90.9**
CNN	**0.1**	11.8	97.9	90.4	89.3	88.1
GRU	1.6	20.8	96.5	84.3	81.7	79.8
LSTM	1.2	19.6	96.9	87.5	83.8	82.2
文献 [3]	1.3	19.7	96.9	87.3	83.7	82.1

表 8　整体数据集实验结果

模　型	误报率	漏报率	精确率	准确率	F1 分数	MCC 系数
CNNH	**1.2**	**7.4**	**98.0**	**92.1**	**92.3**	**91.2**
CNN	**1.2**	11.7	94.4	92.0	90.1	88.7
GRU	2.2	17.4	95.7	84.8	83.7	81.2
LSTM	1.6	15.6	96.6	89.2	86.7	84.8
文献 [3]	1.7	19.0	96.0	88.0	84.4	82.2

本文使用的数据集存在样本不均衡问题，准确率不能全面评估模型的性能，主要讨论其他指标。在整体数据集中，CNNH 模型的各项指标均优于其他模型，相较文献 [3]，漏报率降低 11.6%，F1 分

数和 MCC 分别提升 8.1% 和 8.8%。通过上述实验可以看出使用多尺度卷积神经网络的整体性能要好于循环神经网络；CNNH 模型相较 CNN 模型漏报率降低 4.3%，精确率提升 3.6%，F1 分数和 MCC 分别提升 2.2% 和 2.5%，可以看出使用高速神经网络层后，模型表现得到进一步提升。

如表 9 所示，卷积循环神经网络的训练速度快于循环神经网络，添加高速神经网络后训练时间又大幅减少，而且高速神经网络基于门机制的残差结构更容易训练，同时使一部分输入不经过变换直接通过网络提升了模型的运行速度。

表 9 训练一个 epoch 所需要的时间

模 型	训练时间 /（s/epoch）
CNNH	178.7
CNN	247.8
GRU	408.8
LSTM	438.5

3.5.2 显著图结果

图 4、图 5、图 6、图 7 分别为在函数调用数据样本、算术表达式数据样本、数组使用数据样本和指针使用样本的显著图结果示例。图中颜色越深代表代码元素的分数越高。

```
void func_0 ( )
int variable_0 ;
int * variable_1 = & variable_0 ;
variable_0 = - 1 ;
variable_0 = - 1 ;
int variable_0 = * variable_1 ;
char variable_2 [ 100 ] ;
char variable_3 [ 100 ] = " " ;
memset ( variable_2 , \' A \' , 100 - 1 ) ;
variable_2 [ 100 - 1 ] = \' \\ 0 \' ;
if ( variable_0 < 100 )
memcpy ( variable_3 , variable_2 , variable_0 ) ;
variable_3 [ variable_0 ] = \' \\ 0 \' ;
func_1 ( variable_3 ) ;
void func_2 ( const char * variable_4 )
if ( variable_4 != NULL )
```

图 4 函数调用样本示例

如图 4 所示，此函数调用样本问题为 if 判断没有考虑 variable_0 为负数的情况，于是 memcpy 时发生内存访问错误，可以看出 variable_0 与 memcpy 具有较高的分数。

如图 5 所示，此算术表达式样本问题为 variable_0 进行平方时可能会产生整数溢出问题，可以看出 variable_0 和 variable_4 有较高分数。

如图 6 所示，此数组使用样本问题为 strcpy 没有限制复制的长度，可能会产生非法的内存访问，可以看出 variable_0 有较高的分数。

```
void func_0 ( )
int variable_0 ;
variable_0 = 0 ;
if ( variable_1 )
variable_0 = variable_2 ;
if ( variable_1 )
int variable_3 = variable_0 * variable_0 ;
func_1 ( variable_3 ) ;
void func_2 ( int variable_4 )
printf ( " % d \\ n " , variable_4 ) ;
```

图 5 算术表达式样本示例

```
void func_0 ( )
char variable_0 [ 100 * 2 ] ;
memset ( variable_0 , \' C \' , 100 * 2 - 1 ) ;
variable_0 [ 100 * 2 - 1 ] = \' \\ 0 \' ;
strcpy ( variable_0 , variable_1 ) ;
func_1 ( variable_0 ) ;
void func_2 ( const char * variable_2 )
if ( variable_2 != NULL )
printf ( " % s \\ n " , variable_2 ) ;
```

图 6 数组使用样本示例

```
void func_0 ( )
variable_0 * variable_1 ;
if ( variable_2 == 5 )
variable_1 = ( int64_t * ) malloc ( 50 * sizeof ( variable_0 ) ) ;
memmove ( variable_1 , variable_3 , 100 * sizeof ( variable_0 ) ) ;
func_1 ( variable_1 [ 0 ] ) ;
void func_2 ( int64_t variable_4 )
printf ( " % lld \\ n " , variable_4 ) ;
```

图 7 指针使用样本示例

如图 7 所示，此指针使用样本问题为 variable_1 长度为 50，而 memmove 时长度为 100，可以看出 variable_1 有较高的分数。

通过以上样本可以看出，使用显著图得到每个代码元素的重要性分数不能准确反映漏洞成因与位置，但仍能得出对预测结果重要的一些代码元素。显著图使模型具备一定的可解释性，有助于审计人员人工审查与验证。

3.5.3 结果分析

从实验结果来看，卷积神经网络模型整体性能优于循环神经网络模型，误报率、漏报率、准确率、F1 分数、MCC 系数等各种指标均有较大提升的同时，运行速度也有较大提升。CNNH 模型相对于 CNN 模型性能有小幅度提升，同时运行速度也有提高，这是由于高速神经网络利用门机制对一部分输入进行处理，而另一部分输入直接通过，从而提高了运行速度。

通过与其他方法的对比，本文提出的模型具有以下优势：

（1）多尺度卷积神经网络特征提取能有效提升漏洞检测的各类指标，相较于循环神经网络有较大提升，且运行速度大幅提升；

（2）高速神经网络对特征进行变换，有利于模型学习抽象代码特征，进一步提高模型的表现和运行速度；

（3）利用显著图，使模型具备一定的可解释性，可辅助审计人员确认漏洞。

4　结　语

本文提出了一种基于卷积神经网络的源代码漏洞检测模型。该模型使用切片后的代码作为代码表征提高训练及学习效率；使用多尺度的卷积神经网络提取代码特征，相比循环神经网络不仅漏报率大幅降低，训练和预测速度也得到提升；使用高速神经网络不仅能进一步提升模型性能，其门机制还进一步提升模型训练和预测速度。实验结果表明，本文提出的基于 SeVC 代码表征和卷积神经网络及高速神经网络的源代码漏洞检测模型，相比 SySeVR 可大幅降低漏报率，有效提升准确率、F1 分数、MCC 系数等指标。

本文提出的方法仍存在改进的空间：第一，本方法使用 SeVC 作为代码表征，SeVC 为平整的代码序列，虽然包含代码的数据依赖和控制信息，但 SeVC 并不包含代码的结构信息，未来尝试加入代码的结构信息，比如代码的图表征；第二，本方法对漏洞检测作为二分类问题处理，只能预测是否有漏洞，并不能判断漏洞类别，未来尝试解决漏洞的多分类问题并对其进行解释，实现更细粒度的源代码漏洞检测。

参考文献：

[1] HINDLE A,BARR E T,SU Z D,et al.On the naturalness of software[C]//2012 34th International Conference on Software Engineering,2012:837–847.

[2] SRIVASTAVA R K,GREFF K,SCHMIDHUBER J.Highway networks[C]//ICML 2015 Deep Learning workshop,2015:1–6.

[3] LI Z,ZOU D Q,XU S H,et al.SySeVR:A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing,1525(99):1.

[4] MATTHEWS B W.Comparison of the predicted and observed secondary structure of T4 phage lysozyme[J]. Biochimica et Biophysica Acta (BBA) – Protein Structure,1975,405(2):442–451.

[5] ITTI L,KOCH C,NIEBUR E.A model of saliency–based visual attention for rapid scene analysis[J].IEEE Transactions on Pattern Analysis and Machine Intelligence, 1998,20(11):1254–1259.

[6] LI Z,ZOU D Q,XU S H,et al.VulDeePecker: A deep learning–based system for vulnerability detection[C]//2018 Network and Distributed System Security Symposium, 2018:1–15.

[7] LI Z,ZOU D Q,XU S H,et al.VulDeeLocator:A deep learning–based fine–grained vulnerability detector[J]. IEEE Transactions on Dependable and Secure Computing,6142(99):1.

[8] CAO D F,HUANG J,ZHANG X Y,et al.FTCLNet: convolutional LSTM with Fourier transform for vulnerability detection[C]//2020 IEEE 19th International Conference on Trust,Security and Privacy in Computing and Communications,2021:539–546.

[9] ZOU D Q,WANG S J,XU S H,et al. μVulDeePecker:A deep learning–based system for multiclass vulnerability detection[J].IEEE Transactions on Dependable and Secure Computing,2021,18(5):2224–2236.

[10] TANG G G,YANG L,REN S Y,et al.An automatic source code vulnerability detection approach based on KELM[J]. Security and Communication Networks,2021:5566423.

[11] KIM Y.Convolutional neural networks for sentence classification[C]//2014 Conference on Empirical Methods in Natural Language Processing,2014:1746–1751.

[12] GARDNER M,GRUS J,NEUMANN M,et al.AllenNLP: A deep semantic natural language processing platform[C]// Workshop for NLP Open Source Software,2018:1–6.

作者简介：

梁树彬（1996—），男，硕士研究生，主要研究方向为装备软件安全；

郑　力（1971—），男，硕士，高级工程师，主要研究方向为网电安全及系统设计；

钟　杰（1964—），男，硕士，研究员级高级工程师，主要研究方向为通信、目标识别及网电安全；

胡　勇（1973—），男，博士，研究员，主要研究方向为内容安全、物联网安全、大数据分析与安全。