Zero^2-SMELL: Uma nova abordagem de aprendizado *zero-shot* para detectar vulnerabilidades de dia zero.

Pedro H. Barros[1], Gabriel N. Gomes[2], Leonardo B. Barbosa[1], Heitor S. Ramos[1]

[1]Departamento de Ciência da Computação – UFMG
Belo Horizonte – MG – Brasil

[2]Departamento de Matemática – UFMG
Belo Horizonte – MG – Brasil

{pedro.barros,leob,ramosh}@dcc.ufmg.br, gabrielnfsgomes@ufmg.br

Abstract. *One of the most relevant security problems is inferring whether a program has malicious intent (malware software). Even though Antivirus is one of the most popular approaches for malware detection, new types of malware are released at a fast pace, making most techniques for detecting them quickly obsolete. Thus, regular Antivirus typically fails to detect new malware until their signature is incorporated into their database. Nevertheless, new techniques to identify unknown malware are necessary to protect systems even at day zero of a malware release. Few-shot learning is an approach that consists of using a few examples from each class while training a model. A compelling case of this type of approach is classifying objects that have not yet been used in the training set, namely Zero-shot Learning (ZSL). In the present work, we propose a new Zero-shot learning method to classify malware using visual representation. We propose a new representation space to calculate the similarity between pairs of objects, called S-space. We evaluate our proposal on real-world datasets composed of malware examples. Our proposal achieved* $92.47\,\%$ *of F1-Score and outperforms other methods by a ratio of* $12.11\,\%$ *in a model for zero-day vulnerabilities.*

Resumo. *Um dos problemas de segurança mais relevantes é inferir se um programa tem intenções maliciosas (software malware). Mesmo que o antivírus seja uma das abordagens mais populares para detecção de malware, novos tipos de malware são lançados em um ritmo acelerado, tornando a maioria das técnicas para detectá-los rapidamente obsoletas. Portanto, o antivírus normalmente falha em detectar novos malwares até que sua assinatura seja incorporada ao banco de dados. No entanto, novas técnicas para identificar malwares desconhecidos são necessárias para proteger os sistemas mesmo no dia zero do lançamento de um malware. O aprendizado few-shot é uma abordagem que consiste em usar alguns poucos exemplos de cada classe durante o treinamento de um modelo. Um caso interessante desse tipo de abordagem é a classificação de objetos que ainda não foram usados no conjunto de treinamento, ou seja, aprendizagem Zero-shot (ZSL). No presente trabalho, propomos um novo método de ZSL para classificar malware usando representação visual. Propomos um novo espaço de representação para calcular a similaridade entre pares de objetos, denominado S-espaço. Avaliamos nossa proposta em conjuntos de dados*

do mundo real compostos de exemplos de malware. Nossa proposta atingiu $92.47\,\%$ *na medida Recall@K e supera outros métodos em uma proporção de* $12.11\,\%$ *em um modelo de classificação para vulnerabilidades de dia zero.*

1. Introdução

Com o avanço da tecnologia, muitos processos no nosso mundo foram reformulados, atualizados e digitalizados, fazendo com que os sistemas informatizados se tornassem cada vez mais presentes em nossas vidas. Estes avanços estão migrando interações econômicas e sociais do mundo físico para o espaço cibernético, com várias aplicações sendo desenvolvidas para cidades inteligentes, sistemas educacionais, lojas e bancos eletrônicos, monitoramento de saúde e processos industriais [Pinto and Duarte 2017]. Por isso, questões de segurança estão se tornando fundamentais e ganhando mais relevância.

A detecção de softwares que "deliberadamente cumpre sua intenção de ferir um sistema" (também conhecido como software malicioso ou *malware* [Cozzolino et al. 2012]) é um grande problema de segurança, onde a incidência de um único software malicioso pode causar prejuízos de milhões de dólares [Anderson et al. 2013].

Encontramos diversas abordagens para o tratamento e identificação de *malwares*. Dentre elas, a que se mostra mais comerciável é o software de antivírus. Softwares tradicionais de antivírus usam métodos de assinatura nos problemas de identificação. Quando é encontrado um software suspeito por uma empresa de antivírus, ele é analisado por pesquisadores ou por sistemas dinâmicos de análise. Então, assim que é determinado que é um *malware*, uma assinatura apropriada do arquivo é extraída e adicionada para a base de dados do antivírus [McLaughlin et al. 2017].

Entretanto, diversos softwares se auto encriptam ou aplicam algum método de disfarce, modificando sua assinatura presente no dicionário e, assim, diminuindo a efetividade do antivírus [McLaughlin et al. 2017]. Além disso, muitos softwares maliciosos aparecem todos os dias, e os criadores de *malware* propõem novas abordagens [Nataraj et al. 2011] tornando a tarefa de atualização do banco de dados de *malware* difícil, se não impossível. Portanto, precisamos de novas técnicas para identificar os *malwares* existente e para sermos capazes de identificar um novo *malware* desconhecido – ataque de dia zero.

Portanto, nesse trabalho, propomos uma nova função de similaridade projetada para identificar softwares maliciosos baseados em uma representação visual. Nossa proposta, chamada de Zero^2-SMELL, encontra dois novos espaços de representação dos dados: *espaço de representação latente* e *espaço de similaridade*. No *espaço de representação latente*, temos uma representação obtida através de uma rede neural para redução de dimensionalidade. Nessa representação, usamos uma transformação que mapeia um par de elementos num novo vetor, e então construímos um novo espaço de similaridade, chamado de *Espaço-S*. Para controlar o *overfitting*, é introduzida uma função de regularização agindo em cada um dos espaços de representação dos dados.

Realizamos uma série de experimentos no dataset analisado e mostramos que o Zero^2-SMELL provém ganhos sobre os métodos encontrados na literatura. Em relação ao ZSL, uma métrica usual é o Recall@K, onde Zero^2-SMELL atingiu $92.47\,\%$ para o conjunto de dados analisado. Neste contexto, Zero^2-SMELL supera o outro método por uma proporção de $12.11\,\%$.

Zero^2-SMELL também alcançou um melhor desempenho quando aplicado como um classificador usual (não-ZSL). Nesse caso, treinamos nosso modelo com amostras de todas as classes (esse problema geralmente é altamente desbalanceado). Para tais casos, a medida F1 é uma métrica justa para comparar diferentes técnicas. Zero^2-SMELL atingiu uma pontuação nessa métrica de $97.69\,\%$ para o conjunto de dados analisado. Esses resultados são estatisticamente diferente aos melhores métodos encontrados na literatura e mostram que nossa proposta também é adequada para ser usada em cenários onde os dados rotulados são abundantes.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta os trabalhos relacionados à classificação de *malwares*; a Seção 3 descreve nossa proposta; a Seção 4 descreve a metodologia usada para analisar os dados; a Seção 5 apresenta os resultados principais; e a Seção 6 conclui este trabalho e mostra algumas perspectivas futuras.

2. Trabalhos Relacionados

Para obter eficiência e robustez, técnicas de análise de *malware* estudam o comportamento e estrutura dos executáveis, extraindo *features* que podem descrever suas intenções maliciosas. Tradicionalmente, os métodos são classificados baseados na análise estática e dinâmica dos arquivos do programa e pela abordagem de extração de *features* usada.

A análise dinâmica consiste em um sistema de classificação automatizado baseado numa heurística que examina o comportamento da execução da amostra em suas atividades de rede, lendo, e escrevendo operações em uma área de um ambiente restrito. Apesar de um componente de análise dinâmica provavelmente ser importante para uma solução a longo prazo, tal abordagem ainda é lenta, requerendo alto poder computacional, e necessitando de um ambiente controlado para garantir bons resultados [Wang et al. 2021].

Por outro lado, análise estática detecta um arquivo executável sem executar o código em tempo real. Usando métodos baseados em assinatura, pode-se identificar padrões de bits extraídos através de observações do código binário ou da estrutura interna do arquivo, o que requer que o programa seja descompactado antes de fazer a análise [Saeed et al. 2013].

Entre o conjunto de análise estática de *malware*, abordagens baseadas em imagem são técnicas tradicionais da literatura devido a sua facilidade de uso e da existência de uma infraestrutura para imagens sintéticas. Nestes métodos, arquivos executáveis são convertidos em imagens, tornando possível o uso de métodos tradicionais de processamento de imagem. Tipicamente, a imagem do *malware* é capaz de manter informações do código do programa, textos e recursos de seção, de modo que, resultados promissores tem sido alcançados [Corum et al. 2019].

A análise de *features* de texturas binárias usando descritores locais e globais também se mostram uma alternativa para a classificação de *malware*. [Xiaofang et al. 2014] extraiu *features* locais das imagens usando o descritor SURF (Speeded up robust feature) e executou a correspondência usando LSH (Local sensitive hashing). Tal método atingiu acurácia de 85% na tarefa de classificação.

Aplicando uma abordagem de análise em *features* globais, [Nataraj et al. 2011] propõem o uso de um descritor GIST e do algoritmo K-vizinhos próximos (KNN) para classificação. Seus resultados experimentais indicaram que sua técnica atingiu 96% de

acurácia.

Paralelamente, muitas áreas do conhecimento aplicam aprendizagem profunda em várias tarefas como reconhecimento de fala, classificação de imagens, e redução de dimensionalidade [Sadiq et al. 2021]. Redes neurais (NN) são projetadas para aprender uma representação não-linear dos dados sem interversão humana.

[Singh et al. 2019] usa uma rede neural convolucional (CNN) para classificar arquivos de *malware*. Os autores aplicaram um mapa de cores com base na nova representação visual e obtiveram $96.08\,\%$ de acurácia para seu conjunto de dados. Em [Go et al. 2020], os autores usam uma rede neural baseada em ResNeXt para classificar imagens com base em *malware*.

Em [Agarap and Pepito 2018, Rezende et al. 2018], os autores usam as *features* extraídas das camadas convolucionais de redes neurais e uma máquina de vetores de suporte (SVM) para classificar *malware*. Observe que [Agarap and Pepito 2018] projetou explicitamente um SVM para o problema e treinou a CNN e o SVM simultaneamente, enquanto [Rezende et al. 2018] usar as *features* encontradas pela CNN para alimentar um classificador SVM.

[Vinayakumar et al. 2019] propõe uma arquitetura de rede neural que combina camadas convolucionais e recorrentes (LSTM) para identificar *malware* com base na representação visual. Desse modo, os autores relatam que conseguem extrair *features* espaço-temporais dos arquivos *malwares* para realizar inferência.

[Roseline et al. 2020] utiliza Florestas profundas para inferência de códigos maliciosos. Este método gera um conjunto de florestas aleatórias profundas com uma estrutura em cascata que permite as florestas realizarem o aprendizado de uma nova representação dos dados. Sua capacidade de aprendizado pode ser aprimorada quando as entradas têm alta dimensionalidade.

Abordagens que podem classificar *malwares* desconhecidos (que não estavam no conjunto de treinamento) são promissoras para essa tarefa. [Bozkir et al. 2021] estima uma nova representação dos dados com base em uma transformação linear para redução de dimensão (chamada UMAP) no domínio do problema e avalia sua contribuição para problemas de detecção de *malware* desconhecidos (ZSL).

Os métodos mencionados nesta seção demonstram a viabilidade de se usar representações visuais dos dados de entrada para o aprendizado de classificação de *malware* – Tabela 1. Entretanto, as propostas da literatura tipicamente usam descritores para capturar informações de um único código. Sendo assim, desenvolvemos um novo método que captura informações emparelhadas de pares de códigos utilizando extratores de *features* não-lineares advindos de uma rede neural. Desse modo, propomos uma nova medida de similaridade projetada para o contexto de identificação de *malwares* para cenários de ataque de dia zero.

3. Zero^2-SMELL

Propomos um novo método supervisionado para extrair similaridades entre dois *malwares*. Seja o conjunto de v elementos $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^{v}$, com $\boldsymbol{x}_i \in \mathbb{R}^{m \times m}$, definido em um espaço de representação $m \times m$-dimensional, i.e., representação visual dos *malwares*. Para cada $\boldsymbol{x}_i \in \mathcal{X}$ existe um rótulo associado $y_i \in \mathcal{Y}$, i.e., família de *malware*.

Tabela 1. Sumarização de trabalhos para classificação de *malwares* usando representação visual

Artigo	Abordagem	Método	ZSL
[Nataraj et al. 2011]	GIST + KNN	Extração de Feature	✗
[Xiaofang et al. 2014]	SURF + LHS	Extração de Feature	✗
[Agarap and Pepito 2018]	CNN + SVM	Redes Neurais	✗
[Cui et al. 2018]	Subsampling + CNN	Redes Neurais	✗
[Corum et al. 2019]	Gabor + RF	Extração de Feature	✗
[Singh et al. 2019]	CNN	Redes Neurais	✗
[Rezende et al. 2018]	CNN features + SVM	Redes Neurais	✗
[Vinayakumar et al. 2019]	CNN + LSTM	Redes Neurais	✗
[Go et al. 2020]	ResNeXt	Redes Neurais	✗
[Roseline et al. 2020]	gcForest	Aprendizagem Profunda	✗
[Bozkir et al. 2021]	UMAP	Estimativa de topologia (Manifold)	✓
Nossa proposta	Zero^2-SMELL	Aprendizagem de métrica profunda	✓

O conjunto $\mathcal{X}$ é chamado de *espaço de representação original*. A função de representação $f_\Theta : \mathcal{X} \rightarrow \mathcal{Z}$ obtém uma nova representação para o elemento $\boldsymbol{x}_i$ onde, $\boldsymbol{z}_i = f_\Theta(\boldsymbol{x}_i)$ é associado com o mesmo rótulo que $\boldsymbol{x}_i$, e $\boldsymbol{z}_i \in \mathbb{R}^d$ é um vetor de d-dimensões. Chamamos o conjunto $\mathcal{Z}$ de *espaço de representação latente*.

O autoencoder (AE) é uma rede neural treinada para obter uma cópia da entrada como resultado. O AE é dividido em duas partes: uma função de encoder e uma de decoder. Um encoder pode ser entendido como uma função com um conjunto de pesos Θ que mapeia o *espaço de representação original* $\mathcal{X}$ para o *espaço de representação latente* $\mathcal{Z}$. O decoder é a função inversa $f^{-1}_{\Theta'} : \mathcal{Z} \rightarrow \mathcal{X}$ com um conjunto de pesos Θ'. Neste trabalho, a função de representação f_Θ é um encoder.

3.1. Espaço de similaridade

Definimos a função $f^S : \mathcal{X} \times \mathcal{X} \rightarrow \mathcal{S}$ que mapeia um par de elementos do *espaço de representação original* num novo espaço chamado *Espaço de Similaridade* (ou *Espaço-S*), onde se $\boldsymbol{x}_i$ tem o mesmo rótulo que $\boldsymbol{x}_j$, então ($\boldsymbol{x}_i$, $\boldsymbol{x}_j$) é dito como um par similar. Consideramos então que ($\boldsymbol{x}_i$, $\boldsymbol{x}_j$) é dissimilar se $\boldsymbol{x}_i$ tem um rótulo diferente de $\boldsymbol{x}_j$.

Seguindo [Barros et al. 2020], definimos a função de mapeamento f^S para um par $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ pela seguinte operação de valor absoluto por elemento:

$$\begin{aligned} \boldsymbol{s}_{ij} &= f^S(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &= |f_\Theta(\boldsymbol{x}_i) - f_\Theta(\boldsymbol{x}_j)| \\ &= |\boldsymbol{z}_i - \boldsymbol{z}_j| \\ &= (|z_i^1 - z_j^1|, |z_i^2 - z_j^2|, \ldots, |z_i^d - z_j^d|), \end{aligned} \tag{1}$$

onde $\boldsymbol{s}_{ij}$ tem a mesma dimensão que $\boldsymbol{z}_i$ e $\boldsymbol{z}_j$, z_i^n é a n-ésima *feature* do i-ésimo elemento no *espaço de representação latente* $\mathcal{Z}$.

No *Espaço-S*, temos alguns marcadores para calcular a similaridade entre um par de entrada. Definimos que o conjunto com k elementos $\mathcal{M}^+$ representa o conjunto de marcadores de similaridade. Da mesma forma, marcadores no conjunto $\mathcal{M}^-$ quantificam a dissimilaridade. Então, o conjunto de todos os w marcadores em $\mathcal{M} = \mathcal{M}^+ \cup \mathcal{M}^-$ onde temos que $\mathcal{M}^+ \cap \mathcal{M}^- = \emptyset$.

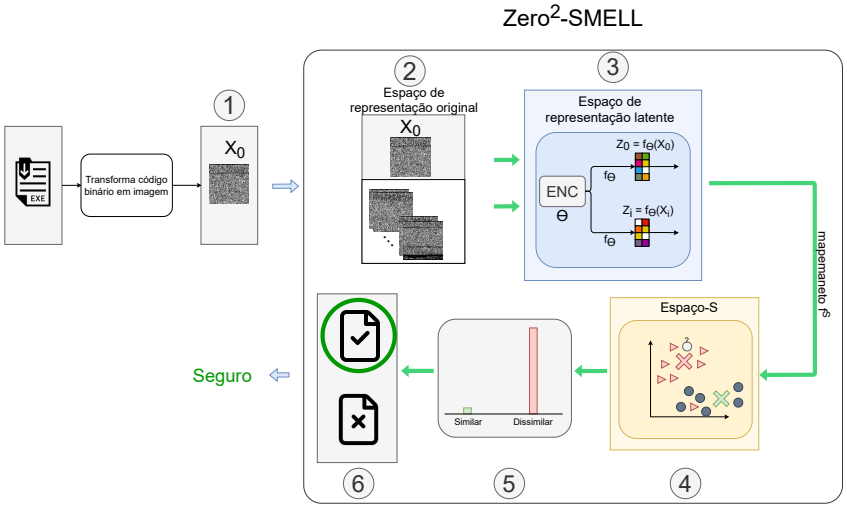

Figura 1. Previsão de seis etapas do Zero^2-SMELL: 1) Transformamos o código binário em uma nova imagem (x_0). 2) Geramos a entrada do encoder emparelhando a nova imagem com algumas imagens do conjunto de dados de treinamento (x_i). 3) Cada par de imagens (x_0, x_i) passa pela função encoder f_θ para ser mapeado no *espaço de representação latente* como (z_0, z_i) = ($f_\theta(x_0)$, $f_\theta(x_i)$). 4) Obtemos então o vetor s_{ij} no *Espaço-S* a partir da nova representação (z_0, z_i) (círculo cinza); 5) No *Espaço-S*, agora calculamos a distância entre o novo vetor (círculo cinza) e os marcadores de dissimilaridade/similaridade (cruz vermelha/cruz verde). 6) Finalmente aplicamos uma função de similaridade baseada em KNN para inferir se o código binário original é seguro ou não (*malware*).

Para o vetor $\boldsymbol{s}_{ij}$, quanto mais próximo $\boldsymbol{s}_{ij}$ é do marcador $\boldsymbol{\mu}^+ \in \mathcal{M}^+$, maior é o valor de similaridade entre o par $(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Analogamente, a distância para $\boldsymbol{\mu}^- \in \mathcal{M}^-$ representa um quantificador de dissimilaridade.

Inspirados por [Barros et al. 2020] usamos a distribuição t de student como um kernel para medir a similaridade entre $\boldsymbol{s}_{ij}$ e um marcador $\boldsymbol{\mu}_m \in \mathcal{M}$, como

$$q_{ij}^m = \frac{(1 + ||\boldsymbol{s}_{ij} - \boldsymbol{\mu}_m||_2^2)^{-1}}{\sum_{\mu_{m'} \in \mathcal{M}} (1 + ||\boldsymbol{s}_{ij} - \boldsymbol{\mu}_{m'}||_2^2)^{-1}}, \tag{2}$$

onde q_{ij}^m é o valor de similaridade/dissimilaridade de $\boldsymbol{s}_{ij}$ em relação aos marcadores $\boldsymbol{\mu}_m$.

Definimos $q_{ij}^+ = \sum_p q_{ij}^p$ para todo $\boldsymbol{\mu}_p \in \mathcal{M}^+$ e $q_{ij}^- = \sum_n q_{ij}^n$ para todo $\boldsymbol{\mu}_n \in \mathcal{M}^-$. Por definição, concluímos que q_{ij}^+ é a probabilidade do par (x_i, x_j) ser similar (rótulos iguais) e q_{ij}^- é a probabilidade do par (x_i, x_j) ser dissimilar (rótulos diferentes). Além disso, não existem marcadores em comum no conjunto $\mathcal{M}^+$ e $\mathcal{M}^-$, i.e., temos $q_{ij}^+ + q_{ij}^- = 1$. Podemos ver um esquemático da nossa proposta na Figura 1.

3.2. Função de custo

Nossa proposta conduz o treino simultâneo do conjunto $\mathcal{M}$ e do *espaço de representação latente* com o parâmetro Θ da função de encoder. Seja a saída do Zero^2-SMELL $\mathcal{Q} = \{\boldsymbol{q}_{ij}\}$ o vetor de 2-dimensões que contém $\boldsymbol{q}_{ij} = (q_{ij}^+, q_{ij}^-)$. Definimos a função de custo

$$J(\{\mathcal{X} \times \mathcal{X}\}) = \sum_{\boldsymbol{x}_i \in \mathcal{X}} \sum_{\boldsymbol{x}_j \in \mathcal{X}} \Big[H_c(\boldsymbol{u}_{ij} || \boldsymbol{q}_{ij}) + R_c(\boldsymbol{x}_i, \boldsymbol{x}_j) \Big], \tag{3}$$

onde H_c é a função de custo de entropia cruzada, $\boldsymbol{u}_{ij} \in \mathcal{U}$ é definido como $\boldsymbol{u}_{ij} = (1, 0)$ se $\boldsymbol{x}_i$ tem o mesmo rótulo que $\boldsymbol{x}_j$ e $\boldsymbol{u}_{ij} = (0, 1)$, caso contrário e R_c é a função de regularização para evitar *overfitting* no processo de treino, como veremos a seguir.

Podemos definir a função de custo contrastive, proposta por [Hadsell et al. 2006], como uma função que minimiza a distância entre pontos similares e impõe uma distância restritiva entre os pontos de classes dissimilares. Então, temos para um par $(\boldsymbol{x}_i, \boldsymbol{x}_j)$ que

$$R_c(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{cases} r_c||f_\Theta(\boldsymbol{x}_i) - f_\Theta(\boldsymbol{x}_j)||_2^2, \text{ se } (\boldsymbol{x}_i, \boldsymbol{x}_j) \text{ é similar} \\ r_c[p - ||f_\Theta(\boldsymbol{x}_i) - f_\Theta(\boldsymbol{x}_j)||_2]_+^2, \text{ caso contrário,} \end{cases} \tag{4}$$

onde r_c é um termo de calibração, e o operador $[.]_+ = max(0, .)$. Nesse trabalho, seguindo [Hadsell et al. 2006], usamos a margem $p = 1$.

Note que a função R_c funciona somente no *espaço representação latente*, minimizando a distância Euclideana entre pares similares. A função H_c atua no *espaço de representação latente* e no *Espaço-S* simultaneamente, agrupando os elementos $\boldsymbol{s_{ij}}$ próximo dos respectivos marcadores.

Para treinar o Zero^2-SMELL, selecionamos aleatoriamente o minilote com m pares de elementos (metade são similares e a outra metade são dissimilares). Além disso, nossa proposta não tem nenhum critério de seleção de pares específico.

4. Metodologia

4.1. Dataset

Para analisar a performance do Zero^2-SMELL, usamos o dataset Malimg [Nataraj et al. 2011] desenvolvido pelo laboratório de pesquisa visual da Universidade da Califórnia[1]. Esse dataset contém 9339 exemplos de 25 famílias de *malwares* diferentes, obtidas através de experimentos de redes e *malwares* do sistema operacional Windows.

Para gerar o dataset, cada código binário do *malware* é formado por um vetor de números inteiros sem sinal de 8-bit,que é organizado em uma matriz de duas dimensões redimensionada para 64 x 64 e visualizada como uma imagem em tom de cinza. O conjunto de dados malimg possui uma diversidade nos números de exemplos. Especificamente, o numero de amostras das famílias de *malwares* varia de 80 a 2949 – Tabela 2.

4.2. Avaliação do modelo

Nós avaliamos nossa proposta em dois cenários de aprendizagem: ZSL e não-ZSL. Para o experimento envolvendo ZSL, as classes do conjunto de teste não são observadas no conjunto de treinamento. Neste experimento, queremos avaliar a capacidade de Zero^2-SMELL de discriminar malware em suas respectivas famílias. Para o conjunto de dados Malimg, selecionamos 8 famílias de malware para treinamento e, consequentemente, todas as outras famílias de malware para teste.

A fim de avaliar nossa proposta para diferentes cenários, investigamos neste experimento o comportamento do Zero^2-SMELL para o problema de classificação padrão

[1] `https://vision.ece.ucsb.edu/research/signal-processingmalware-analysis`

Tabela 2. Descrição das classes de *malware* e das famílias presentes no dataset Mailing.

Classe	Família	# de amostras
Backdoor	Agent.FYI	116
Backdoor	Rbot!gen	158
Dialer	Adialer.C	125
Dialer	Dialplatform.B	177
Dialer	Instantaccess	431
PWS	Lolyda.AA 1	213
PWS	Lolyda.AA 2	184
PWS	Lolyda.AA 3	123
PWS	Lolyda.AT	159
Rogue	Fakerean	381
TDownloader	Dontovo.A	162
TDownloader	Obfuscator.AD	142
TDownloader	Swizzot.gen!I	132
TDownloader	Swizzot.gen!E	128
TDownloader	Wintrim.BX	97
Trojan	Alueron.gen!J	198
Trojan	C2Lop.gen!g	200
Trojan	C2Lop.P	146
Trojan	Malex.gen!J	136
Trojan	Skintrim.N	80
Worm	Allaple.L	1591
Worm	Allaple.A	2949
Worm	VB.AT	408
Worm	Yuner.A	800
Worm:AutoIT	Autorun.K	106

(não-ZSL), i.e., quando os conjuntos de treinamento e teste têm amostras de todas as famílias de malware.

Para avaliar nossa abordagem, usamos o classificador K-vizinhos próximos (KNN), com 3 vizinhos, de acordo com [Wang et al. 2019]. A performance da classificação KNN pode, por vezes, ser significativamente melhorada através de funções de similaridade supervisionadas. Também, usamos o KNN com distância Euclidiana para comparar nossa proposta, e.g., teste de sanidade.

Usamos validação cruzada com 10-rodadas. Esta validação pode reter amplamente distribuições heterogêneas no conjunto de treino e melhorar a confiança estatística dos resultados. Para o experimento não-ZSL, calculamos a acurácia média (ACC), precisão média (PREC), revocação média (REC), e Medida-F1 média (F1-Score).

Usamos a métrica Recall@K para avaliar o experimento ZSL, como visto em [Wang et al. 2019]. Cada código de teste (consulta) recupera os primeiros K vizinhos mais próximos do conjunto de teste e recebe uma pontuação igual a $1/K$ para cada código da mesma classe recuperada entre os K vizinhos mais próximos e 0 caso contrário. O Recall@K calcula a média dessa pontuação de todas as imagens de teste.

4.3. Inicialização dos parâmetros e arquitetura da rede

Inicializamos a posição dos marcadores com o algoritmo de Lloyd. Para o encoder, usamos as camadas convolucionais da arquitetura VGG 19 como um extrator de *features*, proposto por [Simonyan and Zisserman 2015].

Portanto, após a camada convolucional, desenvolvemos uma rede neural com três camadas de dimensões f-4096-d, onde f é a saída do extrator de *features* convolucional, e d é a representação do *espaço de representação latente*. Para este trabalho, usamos d = 256. Todas as camadas de redes neurais são totalmente conectadas, e usamos como função de ativação ReLU. Além disso, usamos o Gradiente Estocástico Descendente (SGD) com momentum, tendo como taxa de aprendizado igual a 0.01 e o momentum de 0.9.

Devido ao fato do modelo de otimização depender de alguns hiper-parâmetros (r_c, w, k), realizamos uma investigação para determinar qual valor dessas variáveis poderiam maximizar a acurácia do modelo. Usamos a técnica de busca em grade para otimização de hiper-parâmetros, e encontramos $r_c = 10^{-1}$, $k = 3$ marcadores de similaridade, $w - k = 2$ marcadores de dissimilaridade. Estes valores foram utilizados durante o restante do trabalho.

5. Resultados e discussão

5.1. Resultados ZSL

Como uma abordagem de aprendizado de métrica profunda, nossa proposta estima semelhanças entre um par de objetos. Desta forma, podemos detectar até mesmo malware que não apareceram no conjunto de treinamento (ZSL). Portanto, investigamos como o Zero^2-SMELL se comporta quando comparado a outras propostas de detecção de malware. Observe que Zero^2-SMELL e UMAP estimam um novo espaço de representação para classificar malware desconhecido. Assim, além do nosso trabalho, apenas UMAP [Bozkir et al. 2021] também é capaz de realizar ZSL sem outras adaptações.

Inicialmente, para o experimento ZSL usando o conjunto de dados Malimg, nós selecionamos aleatoriamente uma família de cada classe do treinamento totalizando 8 classes[2] e 1504 imagens. Todas as outras famílias foram usadas para teste, resultando em 17 famílias de malware e 7.835 imagens. A Figura 2 exibe os resultados do Zero^2-SMELL e UMAP para a configuração do experimento ZSL.

Conforme descrito na seção 4.2, usamos o Recall@K para avaliar o desempenho das propostas ZSL para diferentes valores de K. Assim, adotamos $K = \{1, 2, 4, 8, 10, 20, 40, 80\}$. É importante notar que como *Skintrim.N* possui apenas 80 amostras, portanto, não poderíamos adotar $K > 80$.

A Figura 2 mostra que Zero^2-SMELL tem o melhor desempenho para todos os valores de K analisados. Além disso, à medida que K aumenta, a degradação do desempenho do Zero^2-SMELL é menor do que a observada no UMAP. Para $K = 1$, nossa proposta atinge um Recall@K de $92.47\,\%$, enquanto UMAP tem $82.72\,\%$, resultando em uma diferença de $9.75\,\%$. No entanto, para K maiores, ou seja, $K = \{2, 4, 8, 10, 20, 40, 80\}$, temos uma diferença entre Zero^2-SMELL e UMAP de $7.81\,\%$, $7.83\,\%$, $10.76\,\%$, $12.01\,\%$, $15.97\,\%$, e $13.36\,\%$, respectivamente.

5.2. Resultados não-ZSL

Comparamos a nossa proposta com 12 técnicas tradicionais encontradas na literatura e um teste de sanidade com o KNN usual para distância Euclidiana usando as métricas

[2] *Rbot!gen*, *Dialplatform.B*, *Lolyda.AT*, *Fakerean*, *Swizzot.gen!E*, *C2Lop.P*, *Allaple.A*, e *Autorun.K*

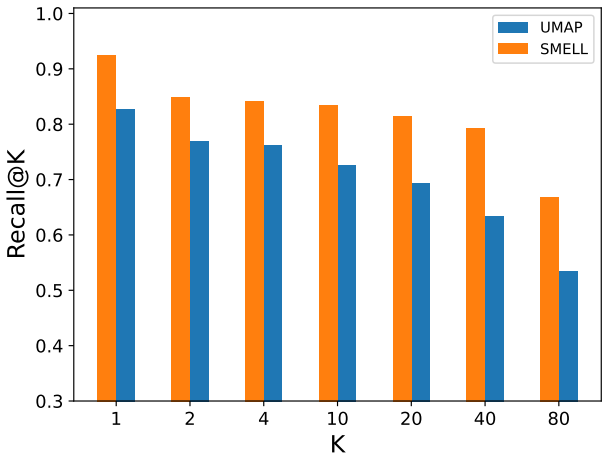

Figura 2. *Espaço latente de feature* encontrado pelo Zero^2-SMELL para o conjunto de dados do Malimg.

descritas na Seção 4.2. Os resultados foram apresentados na Tabela 3. Para o teste de sanidade, percebemos uma diferença significante entre nossa proposta e o KNN padrão para a métrica F1-Score igual a 55.70%.

[Vinayakumar et al. 2019] atingiu a melhor média de acurácia quando comparado com outras técnicas, resultando em 98.59%. O segundo e terceiro melhor resultado foram obtidos por Zero^2-SMELL e [Roseline et al. 2020] com 97.76% e 97.21% respectivamente, representando 0.83% e 1.36% de diferença para a melhor proposta. Entretanto, quando analisamos as outras métricas, esse comportamento muda. Provavelmente isto ocorre porque o dataset é desbalanceado, com a soma de duas classes majoritárias igual a 4540 (representando 48.61% do dataset). Portanto, a acurácia por si só não é suficiente para avaliar este problema de classificação, uma vez que pode ser fortemente enviesado pelas classes com mais amostras. É comum usar outras métricas para uma avaliação mais

Tabela 3. Tabela mostrando os resultados para o dataset analisado. Os melhores valores são representados em negrito e * representa os valores sem uma diferença estatística significante.

Proposta	ACC	PREC	REC	F1-Score
SURF + SVM	31.58% (0.02%)	1.26% (0.01%)	3.99% (0.01%)	1.92% (0.01%)
KNN (Euclidiano)	39.86% (0.07%)	64.62% (0.09%)	39.85% (0.07%)	41.99% (0.09%)
GLCM + SVM [Cui et al. 2018]	62.96% (0.81%)	56.04% (1.51%)	62.96% (0.81%)	53.64% (1.09%)
BAT [Cui et al. 2018]	69.15% (43.39%)	66.45% (43.43%)	67.80% (41.78%)	66.42% (42.35%)
[Corum et al. 2019]	80.89% (0.79%)	78.74% (0.72%)	80.89% (0.79%)	78.49% (0.80%)
HOG + SVM	89.09% (0.84%)	88.28% (0.93%)	89.09% (0.84%)	88.23% (0.91%)
[Rezende et al. 2018]	92.20% (1.35%)	92.50% (2.57%)	91.40% (3.57%)	92.41% (3.20%)
[Agarap and Pepito 2018]	94.12% (5.55%)	88.86% (8.69%)	88.82% (8.62%)	88.44% (9.16%)
VGG 19 [Singh et al. 2019]	95.15% (0.58%)	89.84% (1.25%)	90.23% (1.15%)	89.81% (1.23%)
[Singh et al. 2019]	95.57% (4.39%)	95.89% (3.58%)	95.56% (4.39%)	95.21% (4.91%)
UMAP [Bozkir et al. 2021]	96.62% (0.33%)	95.26% (0.62%)	94.29% (0.81%)	94.54% (0.74%)
[Roseline et al. 2020]	97.21% (0.38%)	97.23% (0.38%)	97.21% (0.37%)	97.14% (0.38%)
[Vinayakumar et al. 2019]	***98.59% (0.26%)**	96.57% (0.76%)	96.33% (0.74%)	96.33% (0.74%)
Zero^2-SMELL	97.76% (0.49%)	***97.84% (0.52%)**	***97.76% (0.49%)**	***97.69% (0.51%)**

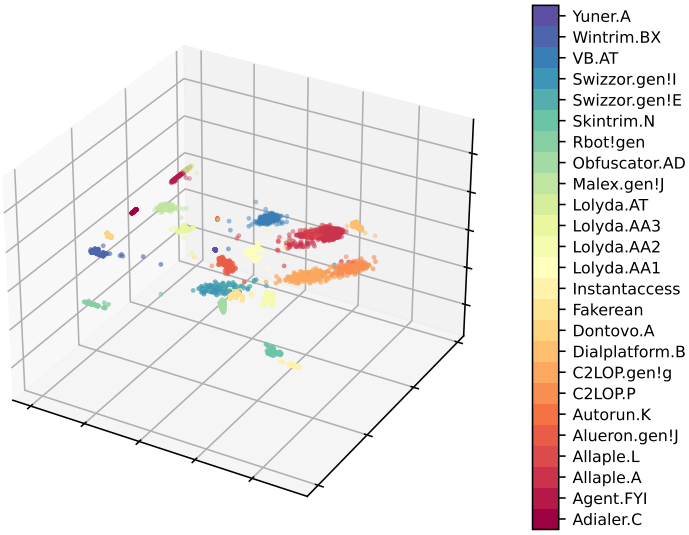

Figura 3. *Espaço latente de feature* encontrado pelo Zero^2-SMELL para o conjunto de dados do Malimg.

abrangente.

Analisando a precisão, observamos que o Zero^2-SMELL atinge 97.84%. O segundo melhor resultado obteve 97.23% com a técnica de [Roseline et al. 2020], totalizando então 0.61% de diferença. Essa diferença se torna maior quando comparada com [Vinayakumar et al. 2019] (terceiro melhor resultado), igual a 1.27%.

Observando a revocação, vemos que a segunda e terceira melhor performance no dataset analisado é obtido pelas propostas [Roseline et al. 2020] e [Vinayakumar et al. 2019], com 97.21% e 96.33% respectivamente. Quando comparados com o Zero^2-SMELL, que obteve a melhor performance com 97.76%, notamos uma diferença de 0.36% e 1.43%.

Também calculamos o teste t de student com 95 % do nível de significância para comparação estatística entre os dois melhores resultados para cada métrica. Para ACC, o p-valor para o teste entre [Vinayakumar et al. 2019] e Zero^2-SMELL é igual a 0.0001. Para PREC, REC e F1-Score, o p-valor entre [Roseline et al. 2020] e Zero^2-SMELL é igual a 0.008, 0.011 e 0.013 respectivamente. Para todas as métricas analisadas, nós rejeitamos a hipótese nula (p-valor$<$0.05) , i.e., nós encontramos diferença estatística significante entre o resultado obtido pelo Zero^2-SMELL e as outras abordagens.

A Figura 3 mostra o *espaço de representação latente* encontrado pelo Zero^2-SMELL para o conjunto de dados do Malimg. Nós projetamos o *espaço de representação latente* em 3-dimensões para melhor visualização (utilizando o PCA). Nesta Figura, nós percebemos que o Zero^2-SMELL agrupa as famílias de malwares (representadas por cores similares). Além disso, observamos que os grupos possuem semântica; por exemplo, embora nossa proposta agrupasse o *malware Allape.L* e *Allape.A* em grupos diferentes,

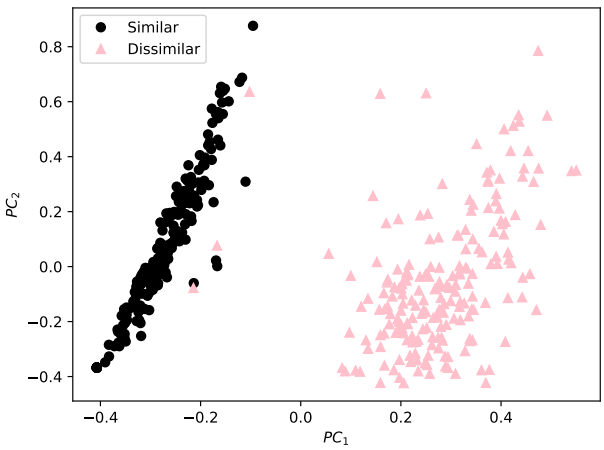

Figura 4. Espaço de similaridade para nossa proposta. Aleatoriamente selecionamos 400 pares dos dados (200 similares e 200 dissimilares).

todos eles acabaram em regiões próximas – Figura 3. Esse comportamento era esperado uma vez que essas duas familias de *malwares* pertencem à mesma classe, ou seja, Worm.

Nós vemos um comportamento similar em *C2Lop.gen!J* e *C2Lop.P* (classe Trojan); e *Swizzor.gen!I* e *Swizzor.gen!E* (classe TDownloader). Esse resultados corroboram com a hipótese que o *Espaço-S* realça a separabilidade entre os grupos de *malwares*.

Observamos também na Figura 4, o espaço de similaridade formado pelo Zero^2-SMELL. Selecionamos aleatoriamente 400 pares de exemplos (200 similares e 200 dissimilares). Como definido na seção 3.1, note que um par de objetos representam um único ponto no *Espaço-S*. Círculos pretos e triângulos rosas representam pares similares e dissimilares, respectivamente. Percebemos que o *Espaço-S* forma duas regiões disjuntas bem definidas. Esse comportamento indica que o modelo obteve uma boa estimativa de similaridade.

6. Conclusão e trabalhos futuros

Neste trabalho, propomos uma nova função de similaridade para identificar *malwares* usando representação visual. Nossa proposta, chamada de Zero^2-SMELL, encontra duas representações (*espaço de representação latente* e *Espaço-S*) que quantifica a similaridade entre dois códigos. Nossa proposta mostra-se uma técnica promissora, quando comparada com outras técnicas encontradas na literatura.

Conduzimos uma variedade de experimentos com o conjunto de dados analisado e mostramos que o Zero^2-SMELL supera de forma consistente outras abordagens. Zero^2-SMELL pode detectar malwares que não foram utilizados no treinamento do modelo (ZSL) com um Recall@K de $92.47\,\%$. Além disso, mostramos a robustez do Zero^2-SMELL, alcançando resultados estatisticamente diferentes às melhores abordagens quando comparados às técnicas tradicionais de detecção de malware conhecido, ou seja, não-ZSL.

Em trabalhos futuros, construiremos o *Espaço-S* usando uma Rede Generativa Adversarial bem como estenderemos nossa análise para mais datasets. Acreditamos que

com um modelo generativo, seremos capazes de melhorar a similaridade extraída por nossa proposta. Além disso, pretendemos investigar o cenário de aplicações no contexto de aprendizagem federada. Acreditamos que, devido ao fato do Zero^2-SMELL possuí um bom desempenho em tarefas de ZSL, ele se mostra uma proposta promissora em problemas não identicamente distribuídos, tipicamente encontrados em aplicações de aprendizagem federada.

Agradecimentos

Este trabalho foi parcialmente financiado pela FAPEMIG; grant #2020/05121-4, São Paulo Research Foundation (FAPESP); CNPq e CAPES.

Referências

Agarap, A. F. and Pepito, F. J. H. (2018). Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (SVM) for malware classification. *CoRR*, abs/1801.00318.

Anderson, R., Barton, C., Böhme, R., Clayton, R., Van Eeten, M. J., Levi, M., Moore, T., and Savage, S. (2013). Measuring the cost of cybercrime. In *The economics of information security and privacy*, pages 265–300. Springer.

Barros, P. H., Queiroz, F., Figueredo, F., dos Santos, J. A., and Ramos, H. S. (2020). A new similarity space tailored for supervised deep metric learning. *arXiv preprint arXiv:2011.08325*.

Bozkir, A. S., Tahillioglu, E., Aydos, M., and Kara, I. (2021). Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. *Computers & Security*, 103:102166.

Corum, A., Jenkins, D., and Zheng, J. (2019). Robust pdf malware detection with image visualization and processing techniques. In *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*, pages 108–114.

Cozzolino, M., Martins, G., Souto, E., and Deus, F. (2012). Detecção de variações de malware metamórfico por meio de normalização de código e identificação de subfluxos. *Anais do XII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 30–43.

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G., and Chen, J. (2018). Detection of malicious code variants based on deep learning. *IEEE Transactions on Industrial Informatics*, 14(7):3187–3196.

Go, J. H., Jan, T., Mohanty, M., Patel, O. P., Puthal, D., and Prasad, M. (2020). Visualization approach for malware classification with resnext. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., and Joon Ahn, G. (2017). Deep android

malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, page 301–308, New York, NY, USA. Association for Computing Machinery.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. S. (2011). Malware images: Visualization and automatic classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*, VizSec '11, New York, NY, USA. Association for Computing Machinery.

Pinto, D. R. and Duarte, J. C. (2017). Static analysis on disassembled files: A deep learning approach to malware classification. In *Annals of the XVII Brazilian Symposium in Information Security and Computational Systems. Sociedade Brasileira de Computação, Brasília, DF*, volume 804.

Rezende, E., Ruppert, G., Carvalho, T., Theophilo, A., Ramos, F., and Geus, P. d. (2018). Malicious software classification using VGG16 deep neural network's bottleneck features. In Latifi, S., editor, *Information Technology - New Generations*, pages 51–59, Cham. Springer International Publishing.

Roseline, S. A., Geetha, S., Kadry, S., and Nam, Y. (2020). Intelligent vision-based malware detection and classification using deep random forest paradigm. *IEEE Access*, 8:206303–206324.

Sadiq, M. T., Yu, X., and Yuan, Z. (2021). Exploiting dimensionality reduction and neural network techniques for the development of expert brain–computer interfaces. *Expert Systems with Applications*, 164:114031.

Saeed, I. A., Selamat, A., and Abuagoub, A. M. (2013). A survey on malware and malware detection systems. *International Journal of Computer Applications*, 67(16).

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations (ICLR)*.

Singh, A., Handa, A., Kumar, N., and Shukla, S. K. (2019). Malware classification using image representation. In Dolev, S., Hendler, D., Lodha, S., and Yung, M., editors, *Cyber Security Cryptography and Machine Learning*, pages 75–92, Cham. Springer International Publishing.

Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., and Venkatraman, S. (2019). Robust intelligent malware detection using deep learning. *IEEE Access*, 7:46717–46738.

Wang, P., Tang, Z., and Wang, J. (2021). A novel few-shot malware classification approach for unknown family recognition with multi-prototype modeling. *Computers & Security*, 106:102273.

Wang, X., Hua, Y., Kodirov, E., Hu, G., Garnier, R., and Robertson, N. M. (2019). Ranked list loss for deep metric learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xiaofang, B., Li, C., Weihua, H., and Qu, W. (2014). Malware variant detection using similarity search over content fingerprint. In *The 26th Chinese Control and Decision Conference (2014 CCDC)*, pages 5334–5339. IEEE.