XXI Escola Regional de Redes de Computadores / VIII Workshop de Segurança (ERRC/WRSeg). DOI: 10.5753/errc.2024.4676

ARTIGO COMPLETO/FULL PAPER

Otimização de hiperparâmetros da DroidAugmentor para geração de dados sintéticos de malware Android

Hyperparameter Optimization of DroidAugmentor for Synthetic Android Malware Data Generation

Angelo Gaspar Diniz Nogueira • angelodiniznogueira@gmail.com
Universidade Federal do Pampa (UNIPAMPA)
Lucas Ferreira Areias de Oliveira • lukasferreira@hotmail.com
Universidade Federal do Pampa (UNIPAMPA)
Anna Luiza Gomes da Silva • annalgs30@gmail.com
Universidade Federal do Pampa (UNIPAMPA)
Diego Kreutz • diegokreutz@unipampa.edu.br
Universidade Federal do Pampa (UNIPAMPA)
Rodrigo Brandão Mansilha • rodrigomansilha@unipampa.edu.br
Universidade Federal do Pampa (UNIPAMPA)

RESUMO. Neste trabalho, realizamos um estudo abrangente sobre o impacto da otimização de hiperparâmetros, como taxa de *dropout* e número de camadas ocultas, na ferramenta *DroidAugmentor*, com foco no aumento de 10 *datasets* distintos de *malware* Android. Avaliamos os efeitos dessa otimização nas métricas de utilidade, similaridade e no consumo de recursos computacionais, como CPU e memória. Os resultados confirmam que ajustes precisos de hiperparâmetros são essenciais para maximizar a qualidade dos dados gerados, otimizando também a eficiência no uso de recursos computacionais.

ABSTRACT. In this work, we conducted an extensive study on the impact of hyperparameter optimization, such as dropout rate and the number of hidden layers, in the DroidAugmentor tool, focusing on the augmentation of 10 distinct Android malware *datasets*. We evaluated the effects of this optimization on utility and similarity metrics, as well as on the consumption of computational resources like CPU and memory. The results confirm that precise hyperparameter adjustments are essential to maximize data quality and ensure efficiency in the use of computational resources..

PALAVRAS-CHAVE: Malware Android • Geração de dados sintéticos • Otimização de hiperparâmetros • cGAN • Data Augmentation

KEYWORDS: Android Malware • Synthetic data generation • Hyperparameter optimization • cGAN • Data Augmentation

1 Introdução

Ferramentas como a ADBuilder [1] e a AMGenerator [2] são úteis na criação de *datasets* de *malware* Android, permitindo a coleta automatizada, rotulação e extração de características de forma sistemática. Contudo, a rotulação pode ser demorada, levando meses em alguns casos. Por exemplo, a ADBuilder, ao usar o serviço VirusTotal no plano gratuito, leva cerca de 200 dias para rotular 50.000 amostras, evidenciando o desafio dessa tarefa.

Uma alternativa para mitigar essa demora é o *data augmentation*, técnica usada para aumentar a quantidade e diversidade dos dados de treino. Essa técnica aplica transformações nos dados originais, como rotações e adição de ruído, gerando variações sintéticas que mantêm as características essenciais dos dados originais, melhorando a generalização dos modelos [3].

Diversas ferramentas para geração de dados sintéticos em dados tabulares têm sido propostas, como a CTGAN [4], com foco geral em reproduzir com precisão as características dos dados, e a TabFairGAN [5], voltada para cenários com métricas de viés, como dados demográficos, incluindo a pontuação de discriminação. Em contraste, a DroidAugmentor [6], proposta recentemente, é uma ferramenta especializada em dados sintéticos de *malware* Android, baseada em cGANs (*Conditional Generative Adversarial Networks*). A DroidAugmentor permite a geração condicional de amostras rotuladas e avalia tanto a *utilidade* dos dados gerados para treinamento de classificadores quanto a *similaridade* entre os dados sintéticos e reais.

O desempenho dos algoritmos de aprendizado de máquina está intimamente ligado à escolha dos hiperparâmetros [7], sendo fundamental distinguir entre parâ-

metros e hiperparâmetros. Parâmetros, como pesos em redes neurais e coeficientes em modelos de regressão, são ajustados automaticamente durante o treinamento. Em contraste, hiperparâmetros, como taxa de aprendizado e número de camadas, são definidos antes do treinamento e controlam o processo de aprendizado. Enquanto os parâmetros são otimizados para minimizar uma função de perda, os hiperparâmetros são ajustados para maximizar o desempenho do modelo em dados não vistos.

A escolha de hiperparâmentros é particularmente crítica nas GANs [8], que são extremamente sensíveis a variações nesses valores [9, 10]. Soluções podem demandar significativos recursos computacionais e tempo. Em um exemplo específico (taxa de *dropout* de 0,4 no discriminador, 0,2 no gerador, 4.000 camadas, 2.000 épocas), a execução levou cerca de 4 horas, utilizando 69,5% da CPU e 18% da memória de um sistema.

Este trabalho apresenta um estudo detalhado sobre a otimização de hiperparâmetros de cGANs, aplicados a 10 *datasets* tabulares do domínio de *malware* Android, cada um com propriedades e características distintas, utilizando a ferramenta DroidAugmentor. O estudo investiga o uso de recursos computacionais, como CPU e memória, além de métricas de *similaridade* e *utilidade*. O objetivo é demonstrar a importância da escolha adequada dos hiperparâmetros e avaliar a eficácia da transferência dessas otimizações entre diferentes cenários de dados, destacando a variabilidade e os desafios envolvidos.

2 Trabalhos relacionados

Na Tabela 1, apresentamos alguns dos principais trabalhos sobre a otimização de hiperparâmetros em redes neurais, destacando os modelos empregados, as métricas de avaliação e os *datasets* utilizados. Embora a acurácia seja frequentemente usada como métrica primária para avaliar o desempenho das redes, no contexto de detecção e classificação de *malware* em dispositivos Android, o *recall* é mais relevante, conforme apontado por [11]. Além disso, outros estudos focam na otimização de recursos computacionais por meio da escolha de hiperparâmetros adequados. Diferente de pesquisas anteriores, nosso trabalho integra métricas de *utilidade* e *similaridade*, além de avaliar o consumo de CPU e memória, ampliando o escopo de análise e sua aplicabilidade.

Conforme observado nos trabalhos revisados, a otimização de hiperparâmetros é amplamente utilizada em diversos domínios de dados e modelos de aprendizado de máquina. No entanto, independentemente do domínio ou modelo, a otimização precisa ser realizada para cada *dataset* específico considerado [15]. Existem técnicas voltadas à otimização de múltiplos *datasets* relacionados [18], que devem ser vistas como ferramentas complementares, já que há espaço para otimizações específicas em cada caso. Além dos trabalhos mencionados na Tabela 1, outros estudos propõem diferentes métodos e algoritmos para busca de hiperparâmetros [19–23].

Tabela 1. Trabalhos relacionados.

Ref.	Modelo	Métricas	Dataset	Diferenciais
[12]	GANs	Perda do discriminador e gerador, acurácia do discriminador	1 dataset de malware Android	Enfoque em detecção de malware Android usando GANs para aumentar a quantidade de dados de treinamento.
[13]	CNN	Métricas de classificação binária	1 dataset de tumor cerebral	Uso de CNNs para classificação de imagens médicas, com foco específico em tumores cerebrais.
[14]	DCNN	Métricas de classificação binária, Curva AUC, sensitividade e especificidade	1 dataset de mamografias	Aplicação de DCNN para análise de mamografias, com ênfase na detecção de câncer de mama.
[7]	SVM e Random Forest	Acurácia e AUC	59 datasets gerais de aprendizado de máquina	Comparação entre SVM e Random Forest em uma variedade de datasets, testando a robustez dos algoritmos.
[15]	CNN	Acurácia, MAE e MAPE	3 datasets de falhas de maquinário industrial	CNN aplicada para previsão de falhas em ambiente industrial, com métricas de erro absoluto médio.
[16]	CNN	Acurácia, sensitividade para ruído, entropia cruzada	250 datasets de sensores de canos	Avaliação de ruído nos dados de sensores, destacando a robustez da CNN em dados ruidosos.
[17]	DCNN	Consumo de energia CPU/GPU, tempo de execução	1 benchmark suite	Foco em eficiência energética e tempo de execução, ideal para otimização em dispositivos de baixo consumo.
Este trabalho	cGANs	Métricas de classificação binária, consumo de recursos computacionais	10 datasets de malware Android	Uso de cGANs para aumento de dados específico para segurança cibernética, com foco em consumo computacional otimizado para dispositivos móveis.

3 Metodologia

3.1 Datasets

Na Tabela 2, são apresentadas as especificações dos *datasets* utilizados neste estudo, provenientes do repositório do projeto Malware-Hunter[1]. A tabela detalha o número de características e a quantidade de amostras malignas e benignas, além do total de amostras. Conforme o repositório, os *datasets* incluem até 200 características selecionadas via método Chi-Squared, e cada conjunto de dados possui um total balanceado de até 10.000 amostras por classe.

3.2 Métricas de Desempenho

As métricas de *similaridade* garantem que o *dataset* sintético preserve as propriedades estatísticas do original, sendo avaliadas por meio de similaridade do cosseno, erro quadrático médio e discrepância média para capturar distribuições e correlações multivariadas. As métricas de *utilidade* avaliam o impacto nos algoritmos de aprendizado de máquina usando classificadores como *Random Forest*, *K-Nearest Neighbor* e *Decision Tree*, medindo acurácia, precisão, *recall* e F1-score. Por

[1] Obtidos do repositório https://github.com/Malware-Hunter/datasets/tree/main/preprocessed/binaries/balanced_samples20k_features200

Tabela 2. Datasets considerados neste estudo

Dataset	Características	Malware	Benignos	Total
Adroit	118	3418	3418	6836
Androcrawl	136	10170	10170	20340
Android Permissions	148	9077	9077	18154
DefenseDroid APICalls Closeness	200	5222	5222	10444
DefenseDroid APICalls Katz	200	5222	5222	10444
DefenseDroid APICalls Degree	200	5222	5222	10444
DefenseDroid PRS	200	5975	5975	11950
Drebin215	200	5555	5555	11110
KronoDroid Real Device	200	10000	10000	20000
KronoDroid Emulator	200	10000	10000	20000

fim, consideramos métricas de *consumo* de recursos computacionais, analisando a utilização de CPU e memória para equilibrar a qualidade do *dataset* com a performance computacional.

3.3 Ambiente e Tecnologias

Para a implementação e execução da arquitetura básica da cGAN, utilizamos Python (versão 3.8) com as bibliotecas principais *numpy* 1.21.5, *Keras* 2.9.0, *Tensorflow* 2.9.1, *pandas* 1.4.4 e *scikit-learn* 1.1.1. Os valores das métricas de consumo de recursos computacionais foram monitorados pela plataforma MLflow[2], que capturou o uso de recursos e o tempo de execução dos treinamentos.

Todos os experimentos foram conduzidos em um sistema com Debian GNU 12 (*kernel 6.1.99-1*), com CPU Intel Core i7-9700 3,00GHz (8 núcleos) e 16 GB de RAM, garantindo consistência nas medições.

3.4 Hiperparâmetros

Na Tabela 3, apresentamos os hiperparâmetros considerados nos experimentos e seus respectivos valores. Escolhemos a função de ativação LeakyReLU após uma série de pré-experimentos que mostraram maior estabilidade em comparação com PReLU e ReLU. A escala de incremento de neurônios por camada segue [12], enquanto o número de épocas foi selecionado com base em faixas estabelecidas por [16].

Tabela 3. hiperparâmetros

Parâmetro	Valores
Épocas	1000,2000, 5000
Número de neurônios por camada	64, 256, 512, 1024, 2048, 4096
Taxa de dropout discriminador	0.05, 0.1, 0.2, 0.3, 0.4
Taxa de dropout gerador	0.0025, 0.05, 0.1, 0.15, 0.2
Algoritmo de treinamento	Adam
Função de ativação	LeakyReLU

[2] https://mlflow.org/

Estudos como [24] mostram que taxas de *dropout* entre 0.2 e 0.5 promovem estabilidade nos modelos adversariais. No entanto, taxas mais altas prejudicam o aprendizado, afetando a qualidade das amostras geradas. Selecionamos taxas de *dropout* para a cGAN DroidAugmentor: no gerador, utilizamos valores baixos (0.0025 a 0.2) para preservar características essenciais, e no discriminador, aplicamos taxas moderadas (0.05 a 0.4) para garantir uma regularização equilibrada.

Ademais utilizamos os valores padrões dos hiperparâmetros dos classificadores KNN, RF e DT da biblioteca *Tensorflow*.

No estudo, os algoritmos K-Nearest Neighbors (KNN), Random Forest (RF) e Decision Tree (DT) foram utilizados para avaliar as métricas de utilidade dos dados gerados. A seguir, detalhamos os parâmetros definidos para cada um dos algoritmos:

- K-Nearest Neighbors (KNN): Utilizamos um número de vizinhos (k) igual a 5, com a métrica de distância euclidiana.
- Random Forest (RF): O número de árvores foi configurado para 100, com profundidade máxima das árvores indefinida para capturar variações complexas nos dados, além de divisão de nós baseada na entropia.
- Decision Tree (DT): A profundidade máxima foi deixada indefinida, permitindo que a árvore cresça até a profundidade máxima possível, com o critério de divisão baseado no índice de impureza Gini.

Esses valores foram selecionados para assegurar uma avaliação consistente dos dados sintéticos em relação à acurácia, precisão, recall e F1-score. Além disso, optamos por manter os parâmetros padrão dos algoritmos da biblioteca utilizada, facilitando a replicação e futuras comparações.

4 Resultados

A Figura 1 ilustra o impacto do hiperparâmetro de densidade de neurônios por camada (em escala quadrática) no uso da CPU (cores frias) e tempo de execução (cores quentes). Dois conjuntos de dados são apresentados: (i) a média dos valores obtidos em experimentos onde apenas a densidade da camada foi alterada e (ii) *datasets* com valores extremos de tempo de execução e uso de CPU. O eixo horizontal está em escala quadrática, enquanto os verticais estão em escala linear. A duplicação da densidade da camada resultou em um aumento de CPU entre 4% e 32%, independentemente do *dataset*, e existe uma correlação direta entre o uso

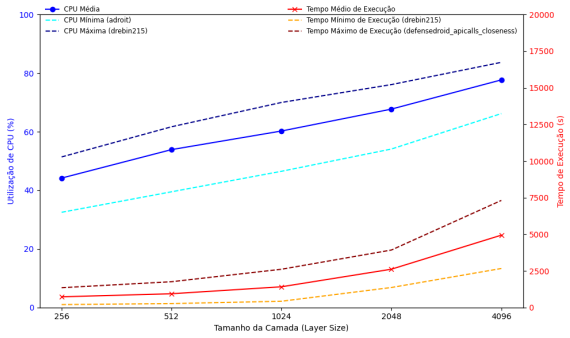
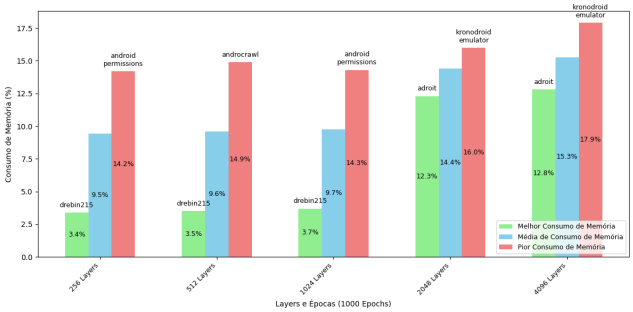


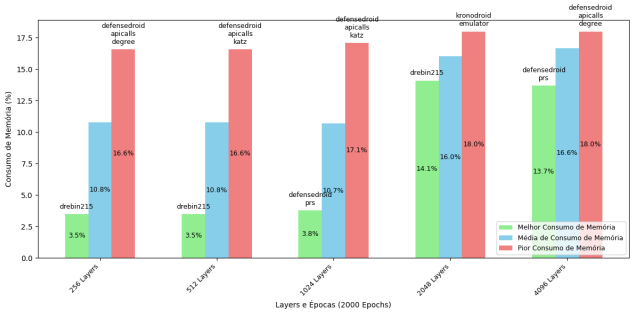
Figura 1. Correlação entre a densidade e utilização de CPU.

da CPU e o tempo de execução, com ambos crescendo proporcionalmente.

A Figura 2 apresenta o impacto da quantidade de camadas (256 - 4096) na utilização de memória em dois cenários com diferentes números de épocas de treinamento (1.000 e 2.000 épocas, nas subfiguras 2(a) e 2(b), respectivamente). Os resultados são agrupados em barras que representam o desempenho médio entre todos os *datasets* (azul), os melhores (verde) e os piores (vermelho).



(a) 1.000 épocas.



(b) 2.000 épocas.

Figura 2. Impacto da densidade no uso de memória.

Como os gráficos mostram, o consumo de memória varia consideravelmente conforme o número de camadas e épocas de treinamento. Por exemplo, para 4096 camadas e 1.000 épocas, o Adroit apresentou o melhor desempenho em termos de uso de memória, enquanto o KronoDroid_emulador foi o pior. Para 2.000 épocas, o DefenseDroid PRS teve o menor consumo de memória, enquanto o DefenseDroid APICalls Degree foi o pior. Esses resultados destacam a influência dos hiperparâmetros no uso de recursos, sendo esta variação dependente do *dataset*.

Para entender o impacto do número de épocas de treinamento no consumo de recursos, realizamos uma campanha de experimentos incrementando o número de épocas em passos de 1.000, mantendo os outros parâmetros constantes. Os resultados dessa campanha para o *dataset* Adroit são apresentados na Figura 3.

Na Figura 3, podemos observar um crescimento linear do tempo de execução, diretamente proporcional ao número de épocas (curva azul). O consumo de memória (curva vermelha) apresenta comportamento semelhante, mas com uma taxa de crescimento menor. Já o uso de CPU (curva verde) se mantém praticamente constante, sem apresentar variações significativas, independentemente do número de épocas. Esses resultados sugerem que, enquanto o tempo de execução e o consumo de memória aumentam com o número de épocas, o uso de CPU se mantém estável.

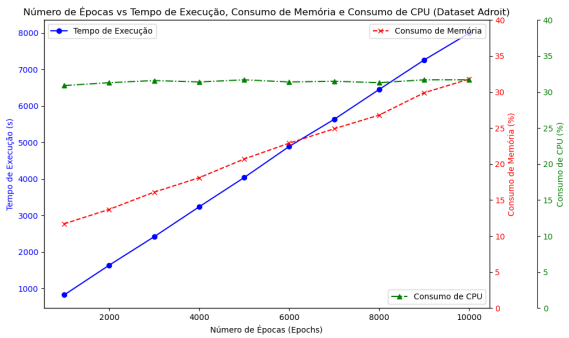


Figura 3. Consumo de recursos por número de épocas

A Figura 4 mostra o consumo de memória e CPU em diferentes *datasets*, organizados em quatro categorias: melhor (MC) e pior caso de consumo de recursos (PC), e melhor (MM) e pior desempenho nas métricas dos classificadores (PM). A análise identifica três padrões: (i) o melhor desempenho dos classificadores se aproxima do melhor caso de consumo de recursos; (ii) ele requer mais memória que o melhor caso de consumo; e (iii) coincide com o pior caso de consumo, indicando que *dropout*, densidade de camadas e número de épocas influenciam o consumo de recursos e o desempenho.

Esses resultados, combinados com os padrões observados nas Figuras 1 e 2, indicam uma tendência clara: (i) o aumento na densidade de camadas resulta em maior utilização de CPU; e (ii) o incremento no número de épocas leva ao aumento no consumo de memória. Am-

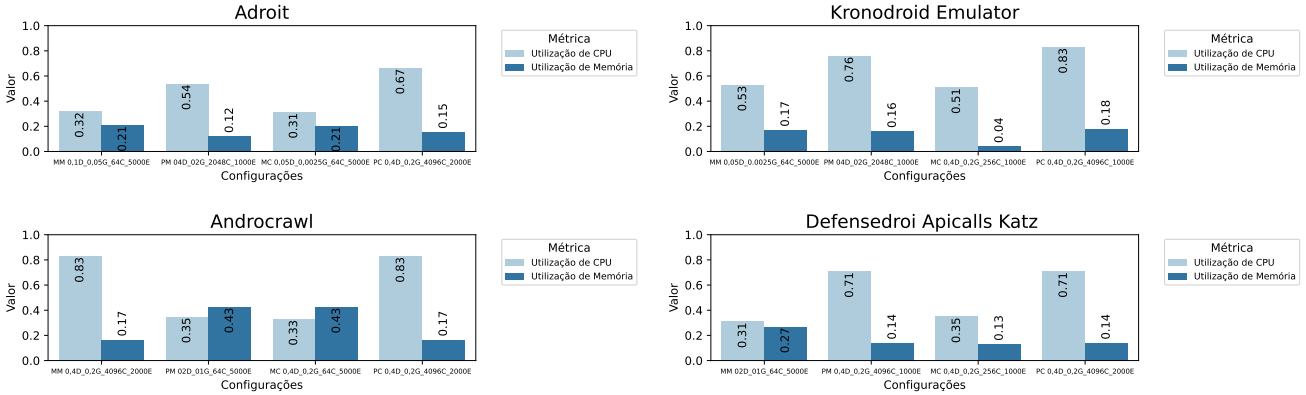


Figura 4. Consumos de recursos computacionais

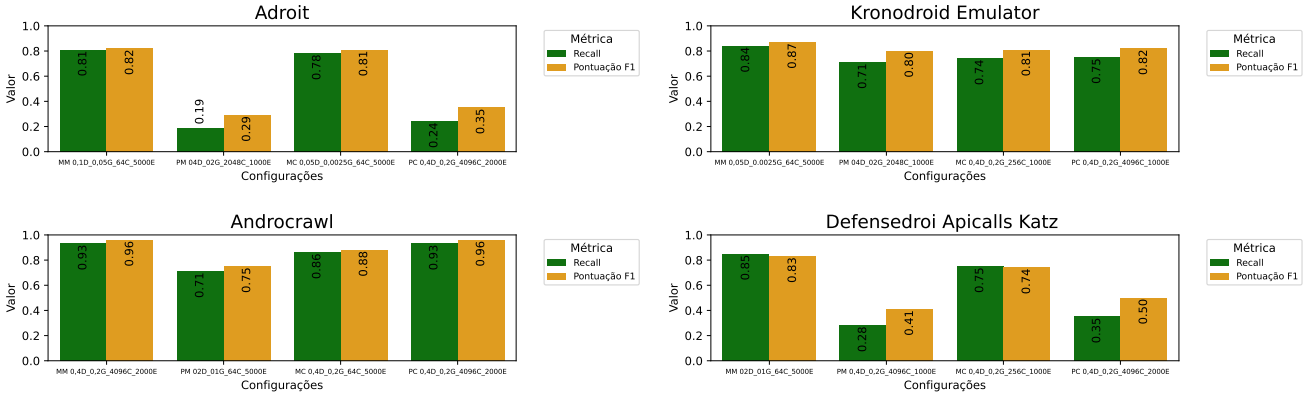


Figura 5. Métricas dos classificadores

bas as variáveis influenciam significativamente o uso de recursos em comparação com outros hiperparâmetros.

Configurações de menor consumo de recursos, embora variem entre os *datasets*, mostram consistentemente uma baixa densidade de neurônios (64 e 256). Em contrapartida, as configurações de maior consumo são homogêneas, caracterizadas por uma alta densidade de neurônios (4096 por camada), taxa de *dropout* de 0,4 no gerador, 0,2 no discriminador e 2000 épocas.

Após entender o uso de recursos de forma isolada, relacionamos esses dados com a *utilidade* dos modelos. A Figura 5 exibe as métricas de *recall* e F1-Score para combinações de hiperparâmetros nos *datasets* Adroit, Kronodroid Emulator, Androcrawal e DefenseDroid API Calls Katz. A análise se concentra em quatro cenários: melhor caso de consumo de recursos, pior caso de consumo de recursos, melhor caso de métricas dos classificadores e pior caso de métricas dos classificadores.

Os cenários de melhor caso de métricas (melhor caso de métricas dos classificadores) tendem a atingir valores próximos de 1. No primeiro cenário, em que o melhor caso de métricas se aproxima do melhor caso de consumo de recursos, as métricas diferem minimamente, com cerca de 3% no *recall* e 1% no F1-Score. No segundo cenário, em que o melhor caso de métricas envolve um uso significativamente maior de memória do que o melhor caso de consumo de recursos, a diferença se amplia para entre 9% e 10% no *recall* e entre 6% e 8% no F1-Score. No terceiro cenário, onde o melhor caso de métricas coincide com o pior caso de consumo de recursos, a diferença é de 7% a 10% no *recall* e de 8% a 9% no F1-Score.

5 Considerações Finais

Este trabalho demonstrou o impacto dos hiperparâmetros na qualidade dos dados gerados e na eficiência do uso de CPU e memória. Observou-se que a densidade de camadas influencia o uso de CPU, enquanto o número de épocas afeta o consumo de memória, destacando a importância de ajustes precisos para maximizar o desempenho do DroidAugmentor.

Planejamos explorar novas combinações de algoritmos e funções de ativação, expandir as aplicações do DroidAugmentor e realizar comparações com ferramentas similares, além de otimizar os hiperparâmetros dos classificadores.

Declarações complementares

Financiamento

Esta pesquisa foi parcialmente financiada pela FAPERGS, por meio dos termos de outorga 24/2551-0001368-7 e 24/2551-0000726-1, e pela Rede Nacional de Pesquisa (RNP), no âmbito do GT Malware DataLab do Programa Hackers do Bem, além de contar com o apoio da CAPES – Código de Financiamento 001.

Referências

1 Barcellos, L. V. ADBuilder: uma ferramenta de construção de datasets para detecção de malwares Android. Universidade Federal do Pampa, 2023.

2 Rocha, V. *et al.* Amgenerator e amexplorer: Geraçao de metadados e construçao de datasets android. *In*: SBC. ANAIS Estendidos do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. 2023. P. 41–48.

3 Chlap, P. *et al.* A review of medical image data augmentation techniques for deep learning applications. *Journal of Medical Imaging and Radiation Oncology*, Wiley Online Library, v. 65, n. 5, p. 545–563, 2021.

4 Xu, L. *et al.* Modeling Tabular Data Using Conditional GAN. *Advances in NIPS*, v. 32, 2019.

5 Rajabi, A.; Garibay, O. O. TabfairGAN: : Fair Tabular Data Generation with Generative Adversarial Networks. *ML and Knowledge Extraction*, MDPI, v. 4, n. 2, p. 488, 2022.

6 Casola, K. *et al.* Droidaugmentor: uma ferramenta de treinamento e avaliação de cgans para geração de dados sintéticos. *In*: SBC. ANAIS Estendidos do XXIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. 2023. P. 57–64.

7 Weerts, H. J.; Mueller, A. C.; Vanschoren, J. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*, 2020.

8 Goodfellow, I. *et al.* Generative adversarial nets. *Advances in neural information processing systems*, v. 27, 2014.

9 Kurach, K. *et al.* A large-scale study on regularization and normalization in GANs. *In*: PMLR. INTERNATIONAL conference on machine learning. 2019. P. 3581–3590.

10 Sabiri, B.; El Asri, B.; Rhanoui, M. Effect of Convulsion Layers and Hyper-parameters on the Behavior of Adversarial Neural Networks. *In*: SPRINGER. INTERNATIONAL Conference on Enterprise Information Systems. 2022. P. 222–245.

11 Tam, G.; Hunter, A. Machine learning to identify Android malware. *In*: IEEE. 2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). 2018. P. 1–5.

12 Alarsan, F. I.; Younes, M. Best selection of generative adversarial networks hyper-parameters using genetic algorithm. *SN Computer Science*, Springer, v. 2, n. 4, p. 283, 2021.

13 Minarno, A. E. *et al.* Convolutional neural network with hyperparameter tuning for brain tumor classification. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control*, 2021.

14 Saranyaraj, D.; Manikandan, M.; Maheswari, S. A deep convolutional neural network for the early detection of breast carcinoma with respect to hyper-parameter tuning. *Multimedia Tools and Applications*, Springer, v. 79, n. 15-16, p. 11013–11038, 2020.

15 Van Den Hoogen, J. *et al.* Hyperparameter analysis of wide-kernel cnn architectures in industrial fault detection: an exploratory study. *International Journal of Data Science and Analytics*, Springer, p. 1–22, 2023.

16 Antunes, A. *et al.* Hyperparameter optimization of a convolutional neural network model for pipe burst location in water distribution networks. *Journal of Imaging*, MDPI, v. 9, n. 3, p. 68, 2023.

17 Li, D. *et al.* Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs. *In*: IEEE. 2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom). 2016. P. 477–484.

18 Hutter, F.; Lücke, J.; Schmidt-Thieme, L. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, Springer, v. 29, p. 329–337, 2015.

19 Putatunda, S.; Rama, K. A modified bayesian optimization based hyper-parameter tuning approach for extreme gradient boosting. *In*: IEEE. 2019 Fifteenth International Conference on Information Processing (ICINPRO). 2019. P. 1–6.

20 Kiran, M.; Ozyildirim, M. Hyperparameter tuning for deep reinforcement learning applications. *arXiv preprint arXiv:2201.11182*, 2022.

21 Dhake, H.; Kashyap, Y.; Kosmopoulos, P. Algorithms for hyperparameter tuning of lstms for time series forecasting. *Remote Sensing*, MDPI, v. 15, n. 8, p. 2076, 2023.

22 Mantovani, R. G. *et al.* A meta-learning recommender system for hyperparameter tuning: Predicting when tuning improves SVM classifiers. *Information Sciences*, Elsevier, v. 501, p. 193–221, 2019.

23 Karamchandani, A. *et al.* A methodological framework for optimizing the energy consumption of deep neural networks: a case study of a cyber threat detector. *Neural Computing and Applications*, Springer, v. 36, n. 17, p. 10297–10338, 2024.

24 Seybold, C. *et al.* Dropout-GAN: Learning from a Dynamic Ensemble of Discriminators. *arXiv preprint arXiv:1807.11346*, 2018.