Using CNNs as static detectors against malicious Android APKs

Andrea Chezzi[1,*,†], Christian Catalano[2,†], Vita Santa Barletta[2,†], Dario Rollo[1,†] and Danilo Caivano[2,†]

[1]Department of Innovation Engineering, University of Salento, Lecce, Italy
[2]Department of Computer Science, University of Bari Aldo Moro, Bari, Italy

Abstract

Android mobile devices are an ideal target for malware attacks due to their widespread use and open source nature. This study discusses the issues posed by Android malware and introduces a new Convolutional Neural Network (CNN)-based malware detection solution. Using static analysis, the suggested approach converts binary executables into pixel images for classification, separating dangerous from benign software. The study uses four CNN architectures: ResNet50v2, ResNet101v2, VGG16, and InceptionV3, using a dataset of 7,172 malware and 10,442 goodware samples. The ResNet50v2 model outperformed expectations, attaining great accuracy and balance. This study illustrates the potential of CNNs in improving Android malware detection and underscores the importance of novel methods to cybersecurity.

Keywords

CNN, malware, static malware detection, static analysis, security, Android, APK

1. Introduction

The smartphone has become an essential part of our daily lives, and Android stands as the dominant operating system in the smartphone market. Projections indicate that the global population of smartphone users is expected to expand significantly, reaching an impressive 7.7 billion by the year 2028 [1]. This upward trend in smartphone usage highlights the growing importance of these devices in our daily lives. Moreover, when we consider mobile operating systems, Android emerges as the dominant player in the market. As of November 2023, Android held a substantial market share of 70.5% [2], demonstrating its widespread adoption and influence in the mobile technology landscape. Android's open nature allows users to download apps from either the official Google Play Store or alternative third-party app markets. Nevertheless, this very popularity and openness have rendered Android a magnet for malicious intent. In particular, the platform's vast user base and versatile environment have made it an attractive target for cyber attackers [3, 4]. This heightened risk stems from the fact that Android devices can easily fall victim to the insidious infiltration of malicious software, often referred to as malware. Such malware, once introduced into the Android ecosystem, can swiftly disseminate and compromise the integrity of otherwise benign Android devices, posing a substantial threat to the security and privacy of users. As a result, the Android community and security experts must remain vigilant in their efforts to combat and mitigate the impact of these digital threats. Cyber attackers are motivated by a variety of factors, ranging from seeking rewards and self-validation to indulging in entertainment or using their skills as a form of digital weaponry [5]. As these motives drive their actions, it's crucial to consistently evolve and innovate in the realm of detecting Android malware. With the expanding landscape of mobile technology and the increasing sophistication of these cyber

COL-SAI 2025: Workshop on COllaboration and Learning through Symbiotic Artificial Intelligence, in conjunction with the 16th Biannual Conference of the Italian SIGCHI Chapter (CHItaly 2025), October 6-10 2025, Salerno, Italy (2025

*Corresponding author.
†These authors contributed equally.
andrea.chezzi@unisalento.it (A. Chezzi); christian.catalano@uniba.it (C. Catalano); vita.barletta@uniba.it (V. S. Barletta); dario.rollo@unisalento.it (D. Rollo); danilo.caivano@uniba.it (D. Caivano)
0000-0002-0163-6786 (V. S. Barletta); 0000-0001-5719-7447 (D. Caivano)

threats, the development of new, robust methods for identifying and combating such threats becomes an ongoing, critical mission for cybersecurity experts and the tech community at large [6].

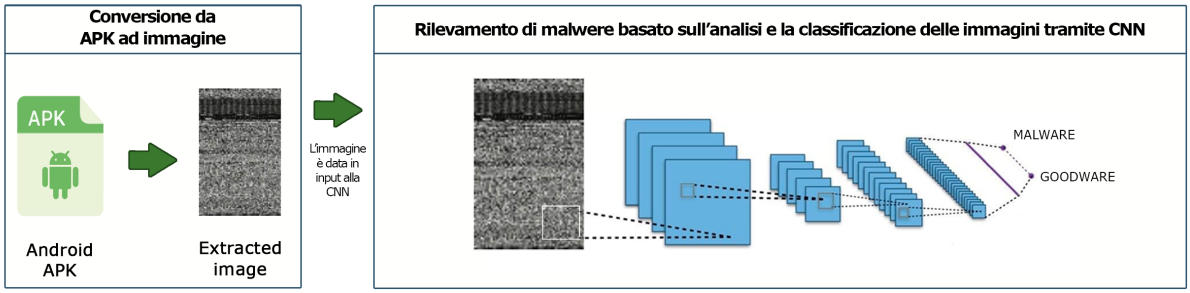

Figure 1: CNN malware detector architecture overview

2. Related Works

Android malware detection has been a focus of numerous studies, reflecting the growing concern over the security of mobile devices. Traditional malware detection methods rely heavily on signature-based techniques, which, while effective, are limited by their inability to detect new or rapidly evolving threats. To address this issue, machine learning (ML) and deep learning (DL) techniques have gained popularity in recent years.

One of the early approaches in malware detection using ML was based on feature extraction from static and dynamic analyses. Arp et al. [7] introduced Drebin, a lightweight Android malware detector using static analysis. They extracted permissions, API calls, and other features from APKs, achieving a high detection rate. Although Drebin was effective, it faced limitations in dealing with new variants of malware that could evade signature-based detection methods.

In recent years, deep learning has emerged as a powerful alternative due to its ability to learn hierarchical representations. Chaymae and Khalid [8] explored the use of Convolutional Neural Networks (CNNs) for Android malware detection by transforming APK files into grayscale images. This method highlighted the capability of CNNs to analyze spatial characteristics within binary data, paving the way for the approach used in this study. Similarly, Hou et al. [9] proposed Deep4MalDroid, a CNN-based model that analyzed images generated from system call sequences, achieving notable success in detecting malware families.

Other studies have employed CNNs in combination with other architectures. For example, Grosse et al. [10] demonstrated a hybrid model incorporating Recurrent Neural Networks (RNNs) to capture sequential information from API calls while using CNNs for spatial feature extraction. This hybrid approach achieved significant accuracy but also required substantial computational resources, making it less practical for real-time deployment.

The effectiveness of CNNs for malware detection is further supported by Zhou et al. [11], who used a CNN-based model trained on binary and opcode sequences, obtaining high accuracy rates. These studies reinforce the idea that CNNs can identify complex patterns within static representations of malware, which are challenging to detect with conventional methods. However, the need for computational efficiency and generalizability remains a central challenge for applying CNNs in real-world settings.

Our study builds upon these findings by using static analysis to convert APK binaries into pixel images. By evaluating multiple CNN architectures, we demonstrate the efficacy of these networks in differentiating malicious from benign applications. This paper extends previous work by focusing on multiple architectures, including ResNet50v2, ResNet101v2, VGG16, and InceptionV3, and by evaluating their relative performances, which provides a comprehensive comparison to aid future researchers in the field.

3. Background

Malware presents a significant challenge within the open-source Android operating system. It's feasible to disassemble an APK file, make alterations to its code, reassemble it, append code triggered by certain events, repackage, sign it, and subsequently release it to the market through reverse engineering tools [12]. This process involves a sequence of steps that can manipulate an application's code, potentially creating security vulnerabilities and allowing for the distribution of malicious software within the Android ecosystem. Malware creators employ a technique known as piggybacking [13] to craft malicious replicas of established software. According to research conducted by Zhou and Jiang [14], approximately 86 percent of Android malware is introduced through piggybacking. While there are indeed other forms of malware, piggybacking stands out as a notably convenient method to implant and execute malicious code on an Android device. This method leverages the familiarity and trust associated with legitimate applications, exploiting them by embedding malicious code within, which is often challenging to detect until it's activated on a user's device [15]. The prevalence of this approach underscores the importance of stringent security measures to protect against such deceptive tactics in the Android environment. The central focus of this paper lies in a comprehensive analysis of the limitations and opportunities inherent in leveraging neural network algorithms to enhance cyber threat detection systems designed for Android platforms. By delving into this subject matter, the paper aims to contribute novel perspectives that can guide the development of more effective practices for constructing models and tools aligned with the evolving landscape of cybersecurity policies and requirements. To ensure clarity in our examination, it becomes pertinent to revisit and reinforce the definitions of the fundamental concepts elucidated in this paper.

APK. The Android APK (Android Package) file format serves as a distribution package, encompassing various resources such as layout files and images stored within directories like assets and res, alongside the Android VM bytecode file classes.dex [16]. The DEX (Dalvik Executable) file is a crucial component of Android applications, containing the bytecode that the Android Runtime (ART) or the Dalvik Virtual Machine (DVM) executes. The DEX format is optimized for efficient execution on resource-constrained mobile devices, making it a fundamental part of Android app development [17]. Additionally, DEX files support features like dynamic class loading and memory-efficient storage, contributing to the overall performance and flexibility of Android applications. Understanding the structure and content of DEX files is essential for developers and security researchers alike, as it provides insights into the inner workings of Android apps and aids in tasks such as optimization, analysis, and security assessment.

Malware. Malware, also known as malicious code and malicious software, is a program that is surreptitiously inserted into a system, often with the intention of compromising the confidentiality, integrity, or availability of the victim's data, applications, or operating system. Its purpose may extend to causing annoyance or disruption to the victim. In contemporary computing landscapes, malware has evolved into the most substantial external threat for numerous systems, inflicting widespread damage and posing significant challenges to cybersecurity [18] [19] [20].

Malware detection. A malware detector is a specialized program crafted to identify and flag malicious programs or code within a system [21] [22]. In a broader conceptualization, the role of a malware detector can be encapsulated in a function, as defined below [20]:

$$D(p) = \begin{cases} \text{malicious} & \text{if } p \text{ contains malicious code;} \\ \text{benign} & \text{if } p \text{ does not contain malicious code;} \\ \text{undecidable} & \text{if } D \text{ fails to determine } p. \end{cases}$$

where D signifies the decision function, serving the crucial role of discerning whether a given application or program p falls into the category of benign or malicious.

4. CNN malware detector

The approach for implementing our malware detector is in line with most of the criteria established in the existing literature.

4.1. Architecture overview

The illustrated malware detector is essentially a Convolutional Neural Network algorithm designed to solve a binary image classification problem with two classes: "malware" and "goodware". In this context, "goodware" denotes any benign program or executable that operates without any deliberate malicious intent.

The setup aims to minimize the occurrence of false negatives during the prediction phase by adopting a preventive and more conservative approach. Figure 1 depicts the architecture of the malware detector. The initial phase involves preprocessing, which converts binary executables into pixel images. Once the image samples are prepared, they are inputted into the trained CNN architecture, which provides classification scores used to determine the class of the analyzed sample.

4.2. Main properties

The developed malware detector has the following main properties:

- It employs static analysis: the executable files are examined without being executed. The relevant information used to make decisions is extracted directly from the content of the DEX files.
- Signature detection: the detector relies on the knowledge gained during the machine learning training phase, where it learned specific features and signatures of existing malware from a given dataset.
- Similarities with anomaly detection: this detector learns not only the signatures of the malware samples from a dataset but also the features of harmless goodware. It then makes predictions based on this learning. This behavior is similar to several anomaly detection methods.
- Utilizes images rather than flat byte strings as input.

Although it must be stressed that one cannot rely solely on CNNs for malwere detection, it is worth noting that the present work fully recognizes the novelty and usefulness of CNNs in the context of malware detection having achieved an excellent result.

CNN	Dimension	Classes	Precision	Recall	F1-score	Accuracy AVG
InceptionV3	250x350x3	*Malware*	97,43%	97,99%	97,71%	97,24%
		Goodware	97,04%	96,22%	96,63%	
ResNet50v2	250x300x3	*Malware*	98,04%	98,82%	98,43%	98,11%
		Goodware	98,19%	96,99%	97,59%	
ResNet101v2	250x300x3	*Malware*	98,12%	98,02%	98,07%	97,65%
		Goodware	97,18%	97,31%	97,25%	
VGG16	250x350x3	*Malware*	96,86%	97,19%	97,03%	96,35%
		Goodware	95,84%	95,37%	95,60%	

Table 1
Results of trained CNN models

4.3. Implementation

For this study, we selected four of the most popular CNN architectures: ResNet50v2, ResNet101v2, VGG16 and InceptionV3. They were implemented using the Keras and TensorFlow frameworks for Python 3.10.13 (64-bit).

4.4. Dataset and Training

The malware APK samples were downloaded and selected from VirusShare.com, a repository of malware samples [23]. On the other hand, the goodware samples collection were collected by self-produced web scraping tools from en.uptodown.com. Therefore, at this point, the dataset consists entirely of APKs:

- 7172 malware samples
- 10442 goodware samples

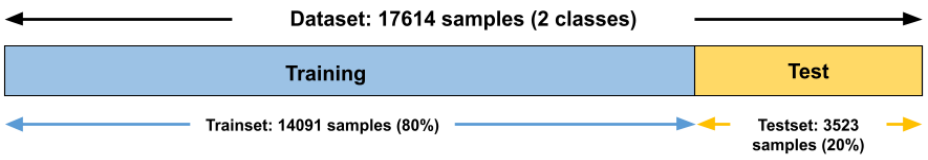

Figure 2: Dataset splitting

To create a dataset suitable for training a CNN learning algorithm, we selected DEX file from APKs to turn them into RGB images through tools written by us: each byte from the DEX file is mapped to a pixel which has each color component labelled from value 0 to value 255. For the training task, it is necessary to divide the dataset into training data and testing data. For this study, the dataset is splitting into two subset: the training set (80%) and the test set (20%). The validation test is derived by TensorFlow directly from the test split. Table 1 shows the results for evaluating the prediction performance of CNN models trained on test set samples. The results clearly show that the ResNet50V2 model is superior in terms of accuracy and balance across criteria. The difference in performance between ResNet50V2 and ResNet101V2 is minor, demonstrating that increasing model depth does not always result in considerable improvements in this particular scenario. Other considerations to consider are training time and computing complexity, and shallower models like VGG16 may nevertheless play an essential role in resource-constrained applications.

5. Future Works

To address the limitations identified in this work, several directions for future work can be considered.

Incorporating dynamic analysis Future research could integrate dynamic analysis to supplement static detection. By analyzing runtime behaviors, such as API calls, system logs, and network traffic, a more comprehensive detection system can be developed. A hybrid model combining static and dynamic analysis may enhance resilience against evasion techniques and improve detection accuracy for new malware variants.

Adversarial Robustness As malware authors continually evolve their techniques to evade detection, future work should focus on improving adversarial robustness. Techniques such as adversarial training, which involves exposing the model to adversarial examples during training, can help enhance the model's ability to detect obfuscated malware. Additionally, incorporating model-agnostic defense strategies may improve generalizability against a wide range of evasion techniques.

Development of Real-Time Detection Systems To improve the practical application of CNNs in malware detection, future work could focus on the development of real-time detection systems that can operate on-device. This would require optimizing the inference process and minimizing latency. Edge computing and federated learning are promising areas for enabling on-device detection, allowing for real-time malware detection without compromising user privacy or device performance.

While this study highlights the potential of CNNs for Android malware detection, addressing these areas in future research can enhance the effectiveness, adaptability, and efficiency of CNN-based solutions. By continuing to innovate and refine these methods, researchers can contribute to building more robust defenses against the ever-evolving landscape of mobile malware.

6. Human-in-the-Loop and Explainability

While the present study emphasizes the efficacy of CNN-based static analysis for Android malware detection, integrating human expertise into the system's workflow remains a crucial direction for practical deployment. Malware detection is rarely a fully automated task; instead, it often requires security analysts, system administrators, or application developers to interpret the results and decide on appropriate mitigation strategies. Designing systems that support human decision-making ensures that automated solutions augment, rather than replace, expert judgment [24].

6.1. Human-in-the-Loop Interaction

The proposed system supports human decision-making by providing, along with its binary classification (malware or goodware), a confidence value associated with each prediction. When this value falls below a predefined threshold, the output is flagged as *uncertain*, signaling the need for human review. This allows analysts to focus on the most ambiguous cases, reducing the risk of misclassification. Moreover, these uncertain samples can be re-labeled by human experts and reintegrated into the training pipeline, enabling the system to continuously improve through human feedback [25]. This interaction creates a feedback loop where AI assists the human operator in prioritizing suspicious samples, while humans contribute to refining the AI's knowledge base.

6.2. Explainability and Transparency

At present, the proposed CNN-based approach does not incorporate explicit explainability features, and therefore operates largely as a black box. While this does not hinder its predictive accuracy, it limits user trust and interpretability. Future work could address this limitation by integrating explainability methods such as saliency maps, Class Activation Maps (CAMs), or feature attribution, which would make the internal decision-making process more transparent [26]. Such methods could help analysts better understand why a particular application is flagged as malicious, strengthening accountability and enabling more informed human–AI collaboration.

6.3. Usability Considerations

Beyond accuracy, usability remains a key factor in the adoption of AI-driven malware detection. Presenting results through intuitive dashboards that highlight both classification confidence and uncertainty status would empower security analysts to make better decisions, even without deep technical expertise. By combining clarity with confidence thresholds, the system ensures that users can quickly distinguish between highly reliable results and those requiring manual inspection. Human-centered design in this context improves both effectiveness and trust in AI-assisted cybersecurity [27].

6.4. Collaborative Defense and Workshop Relevance

The human-in-the-loop approach also facilitates collaborative defense strategies, where analysts across organizations can share uncertain cases, human-validated samples, and insights derived from AI outputs. This iterative process exemplifies the principles of the *COllaboration and Learning through Symbiotic Artificial Intelligence (COL-SAI)* workshop: fostering symbiotic co-adaptation between humans and AI systems. By allowing AI to filter and prioritize tasks while relying on humans for contextual understanding and validation, the proposed framework supports co-creation and continuous learning. In this way, the system not only strengthens malware detection but also advances the broader goal of

designing AI tools that enhance human decision-making, adapt through human feedback, and enable collaborative problem-solving in cybersecurity.

By explicitly incorporating decision thresholds, human oversight, and potential explainability features into its design, this work moves toward AI systems that are trustworthy, symbiotic, and adaptable—qualities that resonate strongly with the objectives of the COL-SAI workshop.

7. Conclusion

This work revealed the efficiency of Convolutional Neural Network (CNN) models in detecting malware on Android systems. Specifically, the ResNet50V2 model stood out for its accuracy and overall performance balance, demonstrating that increased model depth does not always result in a considerable boost in performance. The adopted process, which entails transforming DEX files into RGB images, has enabled us to fully harness the potential of CNNs for binary classification of malware and benign software.

Possible future advancements in this research include refining the model's architecture by investigating and testing new neural network structures, such as deep neural networks or recurrent neural networks, to see whether they can improve malware detection skills. To improve the model's generalization and resilience, the dataset should be expanded with more samples of malware and goodware from other sources. Another major development area is real-time analysis, which involves creating detection systems that can examine the behavior of programs during execution and combining dynamic analysis approaches with the static analysis approach utilized in this study.

In addition, this work highlights the importance of integrating human-in-the-loop elements. By providing a classification confidence score and flagging uncertain cases for human review, the proposed system fosters a collaborative workflow in which humans and AI systems co-adapt and improve together. Although the current approach does not yet implement explainability features, future efforts in this direction could further enhance transparency, usability, and trustworthiness.

These aspects resonate with the objectives of the *COllaboration and Learning through Symbiotic Artificial Intelligence (COL-SAI)* workshop, which emphasizes symbiotic collaboration between humans and AI for effective decision-making and co-creation. By designing CNN-based malware detection systems that are not only accurate but also supportive of human judgment and adaptable through human feedback, this research contributes to the broader vision of building AI systems that collaborate with, learn from, and evolve alongside their human counterparts in real-world cybersecurity contexts.

Acknowledgment

This work was partially supported by project "SEcurity and RIghts In the CyberSpace - SERICS" (PE00000014 - CUP H73C2200089001) under the National Recovery and Resilience Plan (NRRP) funded by the European Union - NextGenerationEU

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] S. 2023, Number of mobile phone users worldwide from 2016 to 2028 (in billions), 2023. https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ [Accessed: (07/11/2024)].

[2] statcounter, Mobile operating system market share worldwide., 2023. https://gs.statcounter.com/os-market-share/mobile/worldwide/ [Accessed: (07/11/2024)].

[3] M. Angelelli, C. Catalano, D. Hill, H. Koshutanski, C. Pascarelli, J. Rafferty, A reference architecture proposal for secure data management in mobile health, 2022. doi:10.23919/SpliTech55088.2022.9854277.

[4] M. T. Baldassarre, V. S. Barletta, D. Caivano, A. Piccinno, A visual tool for supporting decision-making in privacy oriented software development, in: Proceedings of the 2020 International Conference on Advanced Visual Interfaces, AVI '20, Association for Computing Machinery, New York, NY, USA, 2020. URL: https://doi.org/10.1145/3399715.3399818. doi:10.1145/3399715.3399818.

[5] McAfee, Mcafee mobile threat report q1, 2020., 2020. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf [Accessed: (07/11/2024)].

[6] C. Catalano, A. Chezzi, V. S. Barletta, F. Tommasi, Defeating fido2/ctap2/webauthn using browser in the middle and reflected cross site scripting, Journal of Computer Virology and Hacking Techniques 21 (2025) 11. URL: https://doi.org/10.1007/s11416-025-00556-2. doi:10.1007/s11416-025-00556-2.

[7] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, K. Rieck, Drebin: Effective and explainable detection of android malware in your pocket, in: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS), The Internet Society, 2014. URL: https://www.ndss-symposium.org/ndss2014/ndss-2014-programme/drebin-effective-and-explainable-detection-android-malware-your-pocket/, presented at NDSS Symposium 2014, San Diego, CA, USA, February 23–26, 2014.

[8] C. El Youssofi, K. Chougdali, Android malware detection through cnn ensemble learning on grayscale images, International Journal of Advanced Computer Science and Applications (IJACSA) 16 (2025) —. URL: https://thesai.org/Downloads/Volume16No1/Paper_116-Android_Malware_Detection_Through_CNN_Ensemble_Learning.pdf, accessed: 2025-05-31.

[9] S. Hou, A. Saas, L. Chen, Y. Ye, Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs, in: 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), 2016, pp. 104–111. doi:10.1109/WIW.2016.040.

[10] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial examples for malware detection, in: European Symposium on Research in Computer Security (ESORICS), Springer, 2017.

[11] D. Li, L. Zhao, Q. Cheng, N. Lu, W. Shi, Opcode sequence analysis of android malware by a convolutional neural network, Concurrency and Computation: Practice and Experience 32 (2019) e5308. URL: https://doi.org/10.1002/cpe.5308. doi:10.1002/cpe.5308.

[12] R. Winsniewski, Android–apktool: A tool for reverse engineering android apk files; 2012, 2012.

[13] W. Zhou, Y. Zhou, M. Grace, X. Jiang, S. Zou, Fast, scalable detection of" piggybacked" mobile applications, in: Proceedings of the third ACM conference on Data and application security and privacy, 2013, pp. 185–196.

[14] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: 2012 IEEE symposium on security and privacy, IEEE, 2012, pp. 95–109.

[15] M. T. Baldassarre, V. S. Barletta, D. Caivano, M. Scalera, Privacy oriented software development, Communications in Computer and Information Science 1010 (2019) 18 – 32. doi:10.1007/978-3-030-29238-6_2.

[16] P. Kotzias, J. Caballero, L. Bilge, How did that get in my phone? unwanted app distribution on android devices, in: 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 53–69.

[17] E. B. Karbab, M. Debbabi, D. Mouheb, Fingerprinting android packaging: Generating dnas for malware detection, Digital Investigation 18 (2016) S33–S45.

[18] P. Mell, K. Kent, J. Nusbaum, Guide to malware incident prevention and handling, US Department of Commerce, Technology Administration, National Institute of ..., 2005.

[19] F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-Rimy, T. A. E. Eisa, A. A. H. Elnour, Malware detection issues, challenges, and future directions: A survey, Applied Sciences 12 (2022) 8482.

[20] C. Catalano, A. Chezzi, M. Angelelli, F. Tommasi, Deceiving ai-based malware detection through polymorphic attacks, Computers in Industry 143 (2022) 103751.

[21] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, H. Liu, A review of android malware detection approaches

based on machine learning, IEEE access 8 (2020) 124579–124607.
[22] Ö. A. Aslan, R. Samet, A comprehensive review on malware detection approaches, IEEE access 8 (2020) 6249–6271.
[23] virusShare, virusshare, 2023. https://virusshare.com/research [Accessed: (07/11/2024)].
[24] A. Holzinger, Interactive machine learning for health informatics: when do we need the human-in-the-loop?, Brain Informatics 3 (2016) 119–131.
[25] V. Buhrmester, D. Münch, M. Arens, Analyzing human-in-the-loop learning for interactive neural network training, Frontiers in artificial intelligence 4 (2021) 594234.
[26] A. B. Arrieta, et al., Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai, Information Fusion 58 (2020) 82–115.
[27] A. Abdallah, S. Shan, D. Watson, Human-centered ai for cybersecurity: A survey and research directions, in: Proceedings of the 2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2022, pp. 293–300.