applied sciences

Article

Towards Explainable Quantum Machine Learning for Mobile Malware Detection and Classification †

Francesco Mercaldo [1,2,*], Giovanni Ciaramella [2,*], Giacomo Iadarola [2], Marco Storto [1], Fabio Martinelli [2] and Antonella Santone [1]

[1] Department of Medicine and Health Sciences "Vincenzo Tiberio", University of Molise, 86100 Campobasso, Italy
[2] Institute for Informatics and Telematics, National Research Council of Italy, 56121 Pisa, Italy
* Correspondence: francesco.mercaldo@unimol.it or francesco.mercaldo@iit.cnr (F.M.); g.ciaramella1@studenti.unimol.it (G.C.)

† This paper is an extended version of our paper published in the Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES 2022) entitled Introducing Quantum Computing in Mobile Malware Detection.

Abstract: Through the years, the market for mobile devices has been rapidly increasing, and as a result of this trend, mobile malware has become sophisticated. Researchers are focused on the design and development of malware detection systems to strengthen the security and integrity of sensitive and private information. In this context, deep learning is exploited, also in cybersecurity, showing the ability to build models aimed at detecting whether an application is Trusted or malicious. Recently, with the introduction of quantum computing, we have been witnessing the introduction of quantum algorithms in Machine Learning. In this paper, we provide a comparison between five state-of-the-art Convolutional Neural Network models (i.e., AlexNet, MobileNet, EfficientNet, VGG16, and VGG19), one network developed by the authors (called Standard-CNN), and two quantum models (i.e., a hybrid quantum model and a fully quantum neural network) to classify malware. In addition to the classification, we provide explainability behind the model predictions, by adopting the Gradient-weighted Class Activation Mapping to highlight the areas of the image obtained from the application symptomatic of a certain prediction, to the convolutional and to the quantum models obtaining the best performances in Android malware detection. Real-world experiments were performed on a dataset composed of 8446 Android malicious and legitimate applications, obtaining interesting results.

Keywords: malware; quantum; deep learning; security; Android

check for updates

Citation: Mercaldo, F.; Ciaramella, G.; Iadarola, G.; Storto, M.; Martinelli, F.; Santone, A. Towards Explainable Quantum Machine Learning for Mobile Malware Detection and Classification. *Appl. Sci.* **2022**, *12*, 12025. https://doi.org/10.3390/app122312025

Academic Editors: Gianluca Lax and Antonia Russo

Received: 27 October 2022
Accepted: 17 November 2022
Published: 24 November 2022

1. Introduction

In 2022, Android became the most popular operating system in the world, with over 2.5 billion active users spanning over 190 countries https://www.businessofapps.com/data/android-statistics/ (accessed on 16 November 2022). In the past decade, Google Play (https://play.google.com/ (accessed on 16 November 2022)), the official platform for downloading mobile applications for Android-powered devices, has grown enormously, reaching USD 38.6 billion in revenue only in 2020. In the same year, there were more than 2.9 million apps in the store that had been downloaded 108 billion times [1]. Although it has struggled to overtake Apple in Japan and the U.S. currently, Android is the most-popular platform in most places in the world. In countries such as Brazil, India, Indonesia, Iran, and Turkey, it has an over 85% market share [1]. Thanks to this spread, malicious developers are finding new ways to embed malicious payloads into legitimate applications to exfiltrate private and sensitive data from our mobile devices (and to obtain revenue from the gathered information) [2]. For this reason, the adoption of malicious samples containing Trojans, adware, and ransomware malicious payloads is not a surprise for malware analysts [3]. As demonstrated by Kaspersky, every year, there is a proliferation of malicious payloads devoted to exfiltrating data from Android-powered devices. In particular, the experts found more than 880,000 malicious installation packages only in the second quartile of 2021 (https://securelist.com/it-threat-evolution-q2-2021-mobile-statistics/103636/ (accessed on 16 November 2022)). One of the trends in 2021 has been the introduction of malicious code into third-party advertising modules, which developers of various useful apps often insert to monetize their work. For example, in the spring of 2021, malicious code writers used a malicious advertising SDK to infect APKPure (https://m.apkpure.com/(accessed on 16 November 2022)), a popular alternative Android app store well known to Android users. A similar story happened with the popular FMWhatsApp WhatsApp mod (https://sxprojects.net/fmwhatsapp-fouad-whatsapp/ (accessed on 16 November 2022)): one of the app versions hosted the Triada Trojan within an advertising SDK. This Trojan malware is famous for being very difficult to remove from an infected device. Furthermore, Triada is a malware known for its ability to silently download several additional malicious apps to the victim's device [4]. Malware can also sneak into official app stores. To pass all checks and reach users, malicious code writers employ, for example, loading malicious code into an approved program under the guise of an update [5]. In fact, in 2021, analysts found loaders for various Trojans in applications on Google Play, which included the Joker and Facestealer malware. Joker stealthily takes paid subscriptions for the user, while Facestealer, as the name suggests, specializes in stealing Facebook credentials (https://securelist.com/mobile-malware-evolution-2021/105876/ (accessed on 16 November 2022)). In most cases, to spread their creations via Google Play, malicious code developers add tiny injections of malicious code to an otherwise harmless app that has already been approved by the store [6]. For example, the authors of the Joker Trojan (https://usa.kaspersky.com/blog/mobile-malware-2021/26294/ (accessed on 16 November 2022)) took advantage of the popularity of the Korean TV series Squid Game to hide the malware in an app that offered themed wallpapers. After the spotting of Joker, more than 200 apps on Google Play were identified. In particular, many of them borrowed features from each other. Unsurprisingly, while scanning for such programs, the store moderators let a malicious "update" go unnoticed. Small injections of malicious code are difficult to detect during moderation, which cybercriminals are constantly trying to exploit. Banking Trojans have been on the hunt for several years, not just for bank accounts, but also for online store accounts and other digital services. In 2021, we assisted also in the diffusion of the Gamethief malware, which aimed to steal the login credentials of the mobile version of the PlayerUnknown's Battlegrounds game. This represents the first mobile Trojan specialized in the theft of gaming accounts; just a few years ago, this type of malware was exclusive to desktop computers [7]. Malicious writers have also improved the functionality of the developed malicious payloads: as a matter of fact, the fake calls banking Trojan can drop the call whether the infected user tries to contact their bank, and replace it with a pre-recorded response from a fake bank representative [8]. In this way, the malware lulls the victim into believing that a bank employee answered the call. These mobile threats continue to spread because free, and commercial mobile antimalware is not adequate to detect new malicious payloads whose signatures are not included within the antimalware repository [9].

Due to all of the previously described cases of malware discovered in the technology environment, it is necessary to develop methods aimed at detecting malicious behaviors, with particular regard to the Android environment. To solve those problems, academic and industrial researchers are proposing constantly working in this research direction, mostly proposing techniques aimed at detecting malware. Recently, they started to build antimalware with Machine Learning support. Machine Learning is a very wide field, but the most promising in this context is deep learning, which showed that interesting performances can be obtained with respect to the so-called shallow Machine Learning techniques in many classification tasks [10]: as a matter of fact, they obtained a great interest from researchers involved in malware analysis [11]. Considering that deep-learning-based techniques have been demonstrated to obtain better results in image analysis, security researchers are proposing methods to analyze applications in terms of images [12].

A criticism raised for deep learning techniques is due to the so-called lack of explainability [13,14], aimed to provide a kind of explanation behind a certain classification performed by deep learning models. We consider the explanation in terms of capturing the high-level visual patterns to describe the areas of the input image that have influenced most of the classifier prediction.

In very recent times, we have been witnessing the introduction of quantum computing Machine Learning, devoted to exploiting quantum computing concepts in the Machine Learning field [15], i.e., to perform classification tasks by considering quantum theory [16].

In this paper, we propose several Android malware detectors based on deep learning architectures and quantum computing ones. The main idea is to introduce the concept of quantum computing in malware detection, comparing quantum and classical convolutional models in terms of accuracy for the malware detection task in the Android environment. At the current state-of-the-art, according to the knowledge of the authors, this paper represents the first attempt to introduce quantum computing in image-based malware detection.

This paper represents an extension of the research entitled "Introducing Quantum Computing in Mobile Malware Detection" [17], accepted for publication at the 17th International Workshop on Frontiers in Availability, Reliability, and Security (FARES 2022) to be held in conjunction with the 17th International Conference on Availability, Reliability, and Security (ARES 2022). Concerning the work in [17], we explain the novel contributions introduced in this paper:

- A fully quantum Machine Learning network is presented and considered in the experimental analysis, while in the previous paper, a quantum hybrid network was considered. The latter represents the main contribution of the paper: in fact, this represents the first attempt to apply a fully quantum Machine Learning model to a malware detection task;
- A comparison between two different quantum architectures, i.e., the full and the hybrid quantum one;
- Comparison between state-of-the-art Convolutional Neural Network and quantum architectures;
- We extend the number of experiments presented in [17] by tuning the models with the aim to empirically obtain better detection performances;
- Three state-of-the-art deep learning models are added (i.e., VGG19, MobileNet, and EfficientNet) to perform a more complete comparison:
- To provide explainability behind the model decision, we resort to an algorithm aimed to highlight the areas of the application under analysis (represented as an image) that mostly contributed to a certain prediction (i.e., malware or Trusted). To the best of the knowledge of the authors, this is the second main contribution of the paper. Indeed, this is the first attempt to apply explainability to a quantum Machine Learning model;
- We freely release for research purposes the source code we developed for the fully quantum architecture, to encourage researchers to investigate this area.

The remainder of the paper proceeds as follows: In the next session, preliminary notions about malware detection, quantum computing, quantum Machine Learning, and Gradient-weighted Class Activation Mapping are reported. In Section 3, we present the models we considered; the results are reported in Section 4, while in Section 5, the latter are discussed. Finally, in the last section, the conclusions and future research plans are drawn.

2. Background

This section reports preliminary information about the technique exploited for malware detection in the Android environment (i.e., image-based malware detection) and background notions about quantum Machine Learning. For interested readers, we refer to the literature for further information [18,19].

2.1. Image-Based Malware Detection

In recent years, a popular method to classify malware consists of converting malicious software into images and then applying deep learning models based on images to perform classification tasks [20,21]. This process starts from the executable file, which is transformed into an array of values by grouping the bits in blocks and casting the bytes to unsigned integers. Then, the array of values is scaled into a 2D matrix and converted to a grayscale (or RGB) image, by casting each value to a pixel [1].

Most of the malware is a variant of previous malware samples, with some differences in the source code to mislead signature-based antimalware: this is the reason why malware variants of the same original malicious sample are grouped into malware families. Assuming that the malware of the same family shares part of the code, also its corresponding images will have patterns in common. Similar to the classical classification of objects within images, a deep learning model trained on a large number of input samples will be able to recognize the pattern that characterizes one malware family rather than another.

Figure 1 reports two samples of two Android malware converted to images, belonging to two different families (i.e., *Mecor* and *Airpush*, respectively). The images may look like random noise, but the information coming from the input executables is preserved.

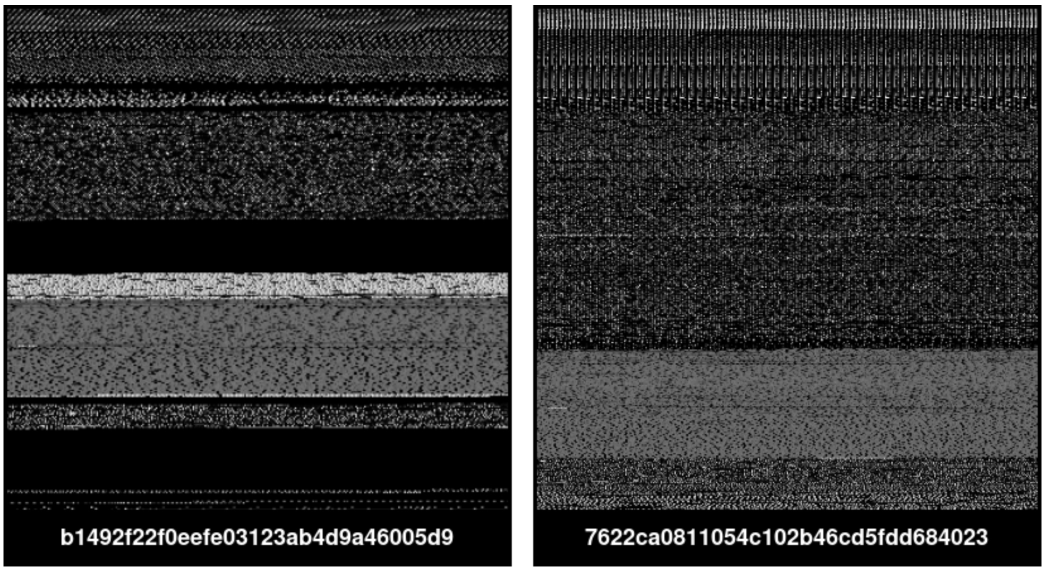
b1492f22f0eefe03123ab4d9a46005d9
7622ca0811054c102b46cd5fdd684023

Figure 1. Two examples of Android malware converted into grayscale images: in detail on the left, a sample belonging to the *Mecor* family, while on the right, a sample obtained from the *Airpush* family.

2.2. Quantum Computing

Quantum computing was born to give answers to unsolvable problems with the use of classic computing, and it takes into account quantum mechanics laws, i.e., the part of physics that studies the smallest particles and how they assume more than one state at the same time [22]. In a nutshell, we can state that quantum mechanics is the basis of quantum computing. Indeed, it refers to the scientific laws that regulate the behavior of molecules, atoms, and subatomic particles and uses the related physical phenomena known as superposition and entanglement for the calculation [23].

Computers normally process information in bits that are zeros and one sequence (i.e., on and off), while quantum computers use Quantum Bits (i.e., qubits), which implement the concept of superposition. Simply speaking, the latter is when a bit can assume a value of zero, one, or even both at the same time. The superposition state represents a combination of all possible configurations. Overlapping groups of qubits can create complex and multidimensional computational spaces. It is in these spaces that complex problems are represented in new ways.

Mathematically, a qubit can be seen as a unit vector described in a two-dimensional complex Hilbert vector space $\mathbb{C}^2$. To represent a complex vectorial space, the use of the notation of Dirac is opportune. It is also possible to obtain the qubit visual representation using the Sphere of Bloch. In that, we can imagine that all possible states are placeable on the surface of a sphere of unit radius, where the two poles represent the two fundamental states, respectively. Starting from Figure 2, it is possible to establish a bi-univocal correspondence between the representation of a generic qubit state:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$$

and its representation on the surface of the sphere in $\mathbb{R}^3$:

$$|\psi\rangle = \cos(\theta/2)\,|0\rangle + e^{\iota\varphi}\sin(\theta/2)\,|1\rangle$$

where θ and φ are real numbers that identify the spherical coordinates of the point. This formula has its correspondence to the real physical world: any physical system with at least two discrete and sufficiently separated energy levels can indeed be used for the qubit representation.

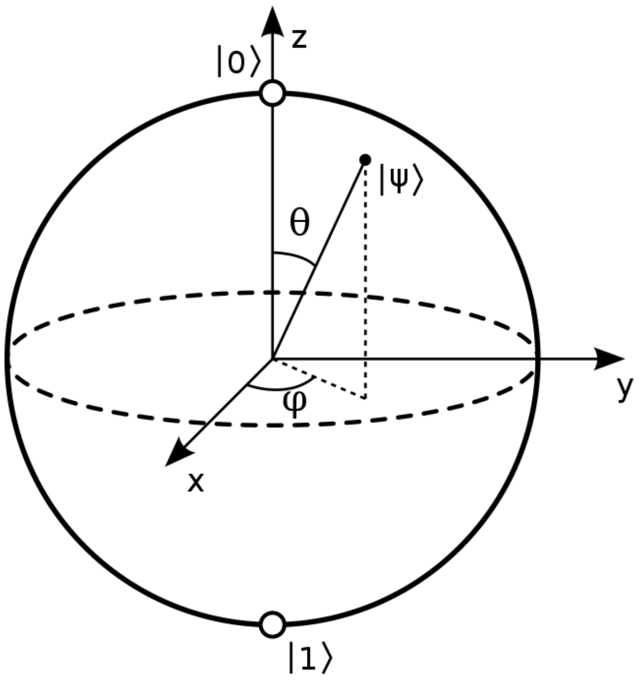

Figure 2. The Bloch Sphere, the geometrical representation of the pure state space of a two-level quantum mechanical system, i.e., a qubit [24].

Another important difference between classical and quantum computing is that, in the traditional field, the system remains constant from measurement to measurement and the outcome never changes. Indeed, in the quantum world, each operation is irreversible, and the result of a measurement is uncertain. A similarity between classical and quantum computing can be found in the usage of a register. A register is a collection of n qubits that can be represented in 2^n different states. The same happens in classic binary encoding. To represent registers and the qubit union, the *tensor product* is represented by the following symbol: $\otimes$.

Due to its unusual processing capacity, quantum computing is aimed at sectors that are traditionally computationally intensive, such as Machine Learning. Machine Learning models themselves can have generalization problems, and given the task of making more and more precise predictions, they become more complex, require more data, and their computations become more expensive. In this case, quantum computing represents an interesting development, as it promises improved performance and better generalization.

2.3. Quantum and Classical Machine Learning

The increasing importance of Machine Learning in recent years has led to many relevant studies that investigate the promise of quantum computers for Machine Learning [25–27]. Quantum Machine Learning (QML) is the field that concerns the integration of quantum algorithms into classical Machine Learning models. Classical Machine Learning algorithms are used to analyze large amounts of data, and quantum computing helps by using qubits and quantum operators to increase the computation and data storage speed of these algorithms. The QML field is still at the forefront of computer science research, and some improvements (the so-called "Quantum Advantage") have not been proven theoretically, yet. With the concept of the "Quantum Advantage", the researchers refer to the advantage of Quantum Models over classical approaches, by leveraging quantum effects. While some attainable advantage of the Quantum Models over generic classical computations has been proven, there is no certainty that the "advantage" may bring complete "supremacy" in the future; the question is still an open debate. The strong quantum speedup is still debatable if a lower bound for the classical algorithms has not been found yet. Indeed, even if the Quantum Advantage was demonstrated for some problems (such as Shor's factoring algorithm [28]), the demonstration represents a quantum speedup over the best-known classical counterpart, but it may be, theoretically, disrupted by improvements in classical calculation techniques and lower bound verifications.

The main difference between a quantum and a classical calculator is the basic unit of calculation. In the classic case, a process is based on the bit, which can assume only two states, generally represented as 0 and 1, corresponding to the state of charge of a transistor. On the other hand, in the quantum scenario, the equivalent of the bit is the quantum bit (or qubit), which follows properties deriving from the postulates of quantum mechanics. Superposition and entanglement are two of the key concepts of quantum theory and contribute to the great computational capacity of quantum computers. The combination of these two principles allows one to create computer systems characterized by a great calculation capacity and speed of execution. While in classical computer science, a system consisting of two bits can store only one of the four possible binary combinations (00, 01, 10, 11), a two-qubit quantum system can store all four combinations.

In [29], the authors conducted research in which they distinguished four ways to combine quantum computing and Machine Learning. There are several approaches to applying quantum principles. Those methodologies are reported in Figure 3 and differ on whether the data are generated by a Quantum (Q) or Classical (C) system and if the information processing device is Quantum (Q) or Classical (C). Briefly, In summary, case *CC* refers to classical data treated conventionally, case *CQ* employs quantum computing to process classical datasets, and case *QQ* uses quantum data processed by a Quantum Computer. In this paper, we focus our attention on quantum data processed by classical algorithms (i.e., *QC*). The quantum Machine Learning algorithms can be a quantum version of conventional ML or a completely new algorithm that addresses a classical problem in a quantum scenario. However, also hybrid classical–quantum methods can be used, where parts of the process, usually computationally expensive ones, are assigned to QC. The aim is to combine quantum and classical algorithms to obtain higher performance and decrease the learning cost. One more classification can be performed on the type of input data, which can be encoded with classical or quantum representation; thus, also the data type generates more hybrid approaches between the classical and quantum worlds.

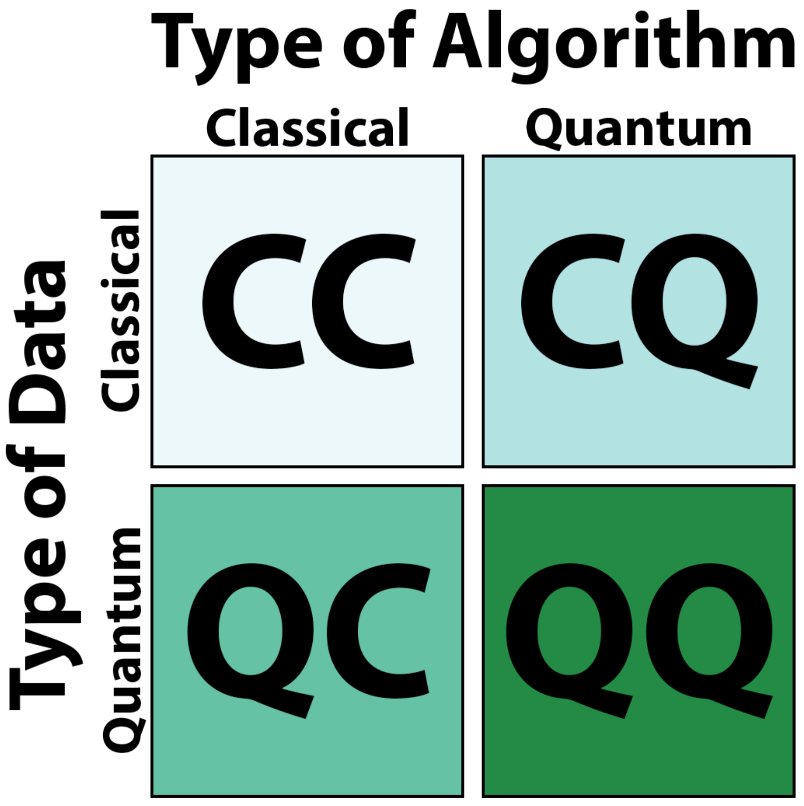

Figure 3. Depending on the type of data and the algorithms used, there are four possible approaches to combine the Machine Learning and Quantum Computing fields.

2.4. Quantum Neural Network

A neural network is inspired by the biological model of information processing in the brain; it can be summarized as a graph consisting of a set of x_m elements linked together by weighted connections with w_{ml} parameters, which represent the equivalent of the synapses of a biological neuronal model.

$$x_l = \sum_{m=1}^{n} w_{ml} * x_m$$

An activation function defines the value of a neuron based on the current value of all other states weighted by the Wml values, and the dynamics of the network unfold as the neurons are continuously updated through the activation function. This kind of model can be seen as a real computational tool, and its programming can be performed by setting the w_{ml} weights and using an activation function that encodes a certain relationship between the input and output. For pattern classification, we usually consider a feed-forward neuronal network, in which neurons are organized into layers (layers) and each layer feeds its information to the next one. A set of initial values is used to feed the input layer, and after subsequent updates on each layer, it is possible to read the final value on the output layer. Feed-forward neural networks often use sigmoid as the activation function:

$$sgm(a,k) = \frac{1}{(1 - e^{-ak})}$$

The network is initialized with a set of inputs, and the initial output is compared with the expected values to adjust the value of the weights to minimize the error of the classification. If an appropriate set of weights is given, these kinds of networks can classify new inputs extremely well. Despite various approaches and ideas in an attempt to adapt neural networks to quantum computation, there is no known concrete proposal describing a quantum classification method with neural networks that is sufficiently performant and functional. Finding this adaptation remains one of the most interesting challenges.

2.5. Grad-CAM

The classical image classification problem is one of the most popular tasks for Deep Learning models. It consists of classifying images that contain items or generic shapes (such as typewritten letters) with the highest accuracy possible. The deep learning model completes the task by leveraging the information of a dataset of input samples. In the training phase, the deep learning models extract and memorizes features and patterns peculiar to a specific output class, thus learning how to distinguish between the different input samples.

One of the most widely used deep learning models for image classification is the Convolutional Neural Network (CNN), which exploits mathematical convolutional operators on the input image to extract features. The input images pass through several layers of convolution, to combine the pixels with the neighboring ones, and subsampling, to reduce the size of the two-dimensional matrix while preserving the most relevant information. Finally, the last part of the CNN is usually composed of dense layers, which are formed by a variable number of *perceptrons*); this last part of the model performs the classification, and it is trainable with the standard backpropagation algorithm. We refer to the literature for further information on CNNs [30,31].

Many complex CNN variants were proposed in the literature; mainly, they differ in the size of the architecture and the number of convolutional layers. For instance, AlexNet introduced the use of ReLU as the activation function instead of the then function and optimization for multiple GPUs. It also addresses overfitting by using data augmentation techniques and the dropout layer. In this paper, we experimented with this architecture and other architectures, by considering also a CNN designed by the authors. We refer to the original papers for further information on their architectures [32,33].

Gradient-weighted Class Activation Mapping (Grad-CAM) is a technique to extract the gradients of the deep learning models' convolutional layers and use them to provide graphical information on the inference step. Briefly, the gradients capture high-level visual patterns and can describe which areas of the input image have influenced most of the model output decision. Furthermore, the convolutional layer preserves spatial information; thus, Grad-CAM uses this data to provide a heat map of the input image. This heat map highlights the input image area that was used by the deep learning model to classify a specific input; it provides a visual "explanation" to a certain decision. The Grad-CAM adopted in this work is an implementation of the one introduced by the paper [34].

3. Method

This section aims to present the proposed method, devoted to directly comparing different deep learning models to discriminate between malicious and legitimate Android applications. Furthermore, those networks are also able to distinguish between malware belonging to different malicious families. To perform our experiments, in addition to the commonly Convolutional Neural Network, we also employed two quantum Machine Learning models, i.e., a hybrid model and a fully quantum one. Figure 4 shows the workflow of the proposed method, composed of four main steps.

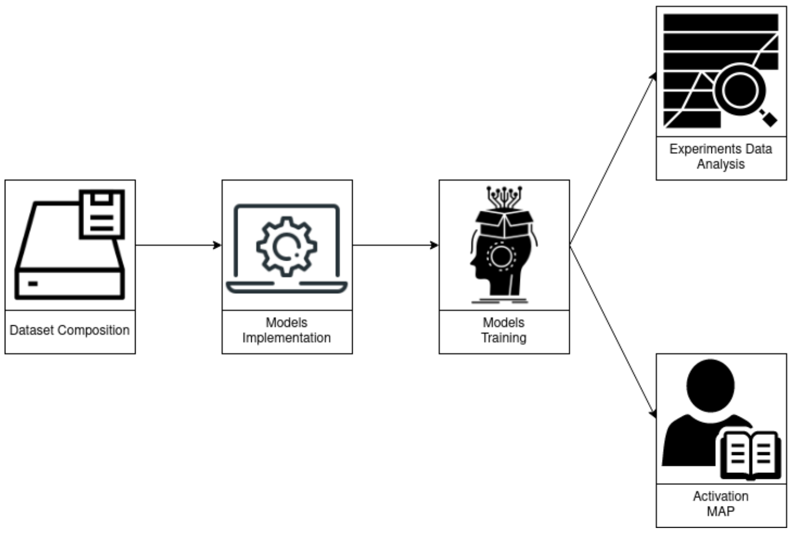

Figure 4. The workflow of the proposed method: starting from a dataset of Android applications, converted to grayscale images, a set of convolutional and quantum models is considered to build classifiers aimed to discriminate between malware and Trusted applications, by considering Grad-CAM to provide explainability in terms of graphical information about the area of the images that influenced the prediction.

3.1. Dataset

The idea behind this step is to obtain a dataset of real-world legitimate and malicious Android applications and convert each application into a grayscale image (as introduced in Section 2.1). We developed a script to automatically unzip an Android application and consider only the .dex file for the image conversion. We confirmed the maliciousness and the trustworthiness of the Android application under analysis by submitting the dataset to the Virustotal (https://www.virustotal.com/(accessed on 16 November 2022)) web service, aimed to check an application under analysis against more than 60 antimalware engines. It is crucial to gather an extended dataset free of bias and duplicated applications to have consistent results.

3.2. Model Research and Implementation

This step is aimed at defining the convolutional and quantum models for the design's comparison. In particular, we considered the following deep learning models: *AlexNet*, a Convolutional Neural Network developed by the authors in [17] (i.e., Standard-CNN), *VGG16*, a Hybrid Quantum Convolutional Neural Network (i.e., CNN with a layer that uses transformations in circuits to simulate a quantum convolution), and a Quantum Neural Network (i.e., a fully quantum model considering two-qubit gates, with the readout qubit always acted upon). In the following, we introduce in detail the considered (convolutional and quantum) deep learning models:

- **Standard-CNN:** From 2012, CNNs conquered a plethora of of the ICISSP, 22 gen–24 gen 2018 2018 tasks and are currently growing at a rapid pace. There are differences in their architecture, but all the CNN models are based on the principle of the convolutional filter. This filter, called the *kernel*, is applied to the pixel matrix that composes the image on the three RGB levels. We report the structure of the CNN proposed by the authors in [17] in Listing 1. The idea is to consider a lighter model if compared to state-of-the-art models (as, for instance, **VGG16** also exploited in the comparison). For the **Standard-CNN**, the following layers (usually considered by typical CNN models) are exploited: *convolutional*, *Maxpooling*, *Dropout*, and *Dense* layers;
- **Visual Geometry Group 16**: This network, also known as VGG-16 [33], was designed and developed in 2014 to solve difficult image classification tasks by exploiting ImageNet, an extended dataset composed of 1000 different output labels belonging to

different domains. The VGG16 network demonstrated that network depth is a crucial factor to increase classification accuracy in deep learning. The VGG16 model is composed of several convolution layers, each one with a 3×3 filter and Maxpooling layers considering a 2×2 filter. The 16 in VGG16 is related to the 16 layers with trainable weights in its architecture. The last model part is composed of 2 fully connected (*Dense*) layers with a softmax activation to perform the classification task. One of the most typical approaches for VGG16's training is devoted to keeping the convolutional part of the model with the weights obtained from training the model on the ImageNet dataset, while the Dense layers part is trained for the specific classification task required;

- **Visual Geometry Group 19:** Also known as VGG19, it was introduced by Oxford University following VGG16 [33]. This network differs from the previous because it exploits 19 layers: sixteen convolutional layers and three fully connected ones;
- **MobileNet:** was introduced in 2017 by Google [35]. This model is based on depthwise separable convolutions, with a single filter applied to each input channel. Different from the standard convolution, MobileNet splits the inputs into two layers, where one layer filters and separates (called depthwise convolutions) and the other combines (called pointwise convolutions). Deeper, the latter layer, allows the creation of a linear combination of the output of the depthwise layer. This strategy significantly reduces the computation and model size. MobileNet can be used in different fields, such as object detection, fine-grained classification, face attributes, and landmark recognition, where the model can obtain good results;
- **EfficientNet:** This was presented in 2019 [36] and is based on the MobileNet-V2 scale-up models in a simple, but effective way using a technique known as the compound coefficient. Using this technique, it is possible to scale each dimension (i.e., depth, width, and resolution) uniformly using a preset set of scaling factors;
- **Hybrid Quantum Convolutional Neural Network**, i.e., Hybrid-QCNN: this model is the first one introducing a quantum computation in deep learning. It consists of quantum and classical neural-network-based function blocks; for this reason, we called it hybrid. In this network, the inner workings (variables, component functions) of the various functions are abstracted into boxes, where the edges represent the flow of classical information through the metanetwork of quantum and classical functions [37]. In a nutshell, it adds to the classic convolutional network a first layer aimed to simulate computations performed by a quantum computer, by performing a quantum convolution. The first layer implements a quantum convolution, aimed to use the transformations in circuits to simulate the quantum computer behavior. Listing 2 shows the implementation of the Hybrid-QCNN, where the first layer represents a convolutional layer built to work on quantum circuits;
- The **Quantum Neural Network**, also known as QNN, was introduced in 2018 by Farhi et al. [38], and it represents a deep learning network exploiting only quantum operation (for this reason, we refer to the QNN also as a fully quantum model). Considering that the classification is based on the expectation of the readout qubit, the authors in [38] proposed the usage of two-qubit gates, with the readout qubit always acted upon. This is similar in some ways to running a small Unitary RNN across the pixels. The proposed QNN for malware detection exploits this approach, where each layer uses n instances of the same gate, with each of the data qubits acting on the readout qubit, by starting with a simple class that will add a layer of these gates to a circuit. Figure 5 shows an example of a circuit layer to better understand how it looks. The quantum circuit consists of a sequence of parameter-dependent unitary transformations, which act on an input quantum state. The input quantum state is an n-bit computational basis state corresponding to a sample string. The idea is to design a circuit made from two-qubit unitaries that can correctly represent the label of any Boolean function of n bits. Listing 3 shows the Python code related to the Quantum Neural Network we implemented. In particular, in Listing 3, two different

layers are exploited: the first one is a Parametrized Quantum Circuit (PQC) layer, typically related to the fully quantum model, and the last one is a Dense layer used to perform the classification (as happens in a classic Convolutional Neural Network). The PQC level is considered for the training of parameterized quantum models. Given a parameterized circuit, this level initializes the parameters. We define a simple quantum circuit on a qubit. This circuit parameterizes an arbitrary rotation on the Bloch Sphere in terms of three angles, i.e., a, b, and c. The source code of the Quantum Neural Network developed by the authors is available, for research purposes, at the following link: https://github.com/vigimella/Quantum-Neural-Network (accessed on 16 November 2022).

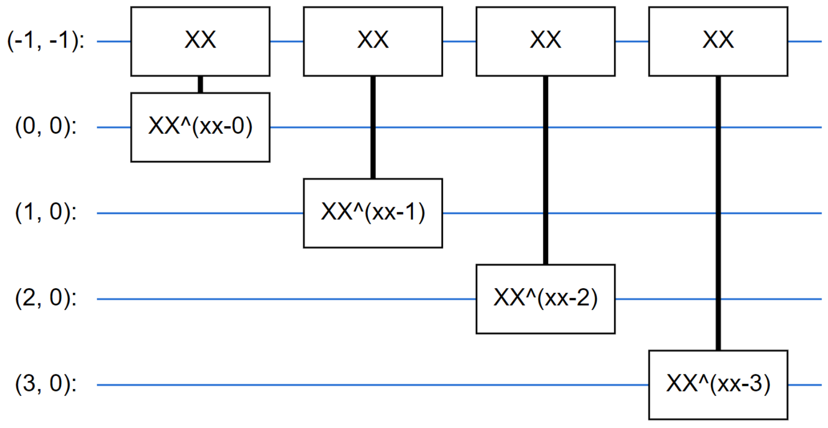

Figure 5. Circuit layer example, where the quantum circuit is composed of a sequence of parameter-dependent unitary transformations aimed to work as an input quantum state. The input quantum state is composed of an n-bit computational basis state that corresponds to a sample string.

Listing 1. The Standard-CNN network designed by the authors is composed of convolutional and Maxpooling layers for the feature extraction part and the Dropout, Flatten, and Dense layers relating to the classification part.

```
model = models.Sequential()
model.add(layers.Conv2D(30, (3, 3), \
          activation='relu',
          input_shape=(self.input_width_height,
                       self.input_width_height,
                       self.channels)))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(15, (3, 3), \
          activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Dropout(0.25))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(50, activation='relu'))
model.add(layers.Dense(self.num_classes, \
          activation='softmax'))
```

Listing 2. The hybrid Convolutional Neural Network, with the QConv layer aimed at performing a quantum convolution, followed by a convolutional layer, a Flatten layer, and two Dense layers for the classification part.

```
NEW_SIZE = 10
[...]
model = models.Sequential()
model.add(QConv(filter_size=2, depth=8, \
  activation='relu', name='qconv1', \
  input_shape=(NEW_SIZE, NEW_SIZE, self.channels)))
model.add(layers.Conv2D(16, (2, 2), \
  activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(self.num_classes, \
  activation='softmax'))
```

Listing 3. The quantum neural network, composed by a PQC layer devoted to the training of parameterized quantum models.

```
[...]

model = models.Sequential()
model.add(PQC(model_circuit, model_readout))
model.add(layers.Dense(self.num_classes, activation='softmax'))
```

Considering that quantum Machine Learning requires huge computational resources if compared to Convolutional Neural Networks, we reduced the image input size for the CNN models to adopt an image size closer to the Hybrid-CNN one, which was 25 $\times$ 25 $\times$ 1. Relating to the QNN, the image size we considered was 4 $\times$ 4 $\times$ 1 due to the more extended huge amount of computational resources required from the fully quantum model. Once having evaluated all the quantum and convolutional models, the obtained results are discussed to provide suggestions and insights.

3.3. Experimental Analysis

To implement the proposed comparison between quantum and convolutional deep learning models, we used Python as a programming language, and we considered two different open-source libraries, *Tensorflow* and *Tensorflow Quantum*, to develop the deep learning models. More details on *Tensorflow Quantum* can be found in [37]. Briefly, Tensorflow Quantum is a framework that offers high-level abstractions for the design and training of both discriminative and generative quantum networks, compatible with existing TensorFlow APIs, along with quantum circuit simulators. In addition, the authors demonstrated that this library can be applied to tackle advances in many fields, i.e., quantum learning tasks including meta-learning, layerwise learning, Hamiltonian learning, sampling thermal states, variational quantum eigensolvers, classification of quantum phase transitions, generative adversarial networks, and reinforcement learning.

3.4. Gradient-Weighted Class Activation Mapping

Gradient-weighted Class Activation Mapping (Grad-CAM) is a technique to provide graphical information on the parts of the input image that have influenced most of the classification output of a CNN. The output of Grad-CAM is a heat map, which can be overlayed on the input image to highlight the relevant part. The heat map is generated using the gradients of the final convolutional layers, which are the one that capture higher-level visual patterns and preserve spatial information on the input image.

The Grad-CAM we adopted [34] is a generalization of the approach proposed in [39]. Grad-CAM does not require any modification to the model architecture and provides a clear way to understand if the model has learned correctly, that is if the model is using the discriminative pattern in the input image to classify that sample. Intuitively, in an image-based malware classification task, the deep learning model should highlight the payload (i.e., the malicious code) of the malware, which is the shared pattern with the other malware of the family. Otherwise, if the model is classifying that sample because of another part of the input image than the payload, the malware could be easily modified by cutting out that highlighted part, preserving the malicious behavior (expressed by the payload), and so generating a new malware variant, which will pass the model check as legitimate.

As a matter of fact, the application of Grad-CAM can be useful to understand in which part of the image under analysis are located the bytes that, from the model point of view, are symptomatic of the malicious payload. It can also be of interest to the security analyst for studying malware families: since samples belonging to the same family share the same payload, Grad-CAM for these samples should highlight the same area of the image with similar color intensities. It can also be useful for identifying malware variants belonging to the same family attackers develop new variants to evade signature-based detection provided by antimalware by applying code obfuscation techniques [40,41]. Therefore, the highlighting of different areas among various samples of the same family can be symptomatic of a new variant of an existing malware family.

4. Results

In this section are shown the results obtained from the experimental analysis. We executed the experiments using the following hardware and software characteristics: Intel Xeon Gold 6140M CPU at 2.30 GHz, 64 GB RAM, and Ubuntu 22.04.01 LTS distro as the Operating System. Different from the preliminary research work [17], in this paper, we considered more experiments with different settings, where we changed the sample dimension and learning rate regarding the models such as AlexNet, Standard-CNN, and VGG16. In addition, we also performed a comparison with other state-of-the-art methods, e.g., VGG19, EfficientNet, and MobileNet. The image dimension used during the training phase increased from $100 \times 100 \times 1$ to $110 \times 110 \times 1$, while the learning rate went from 0.010 to 0.001, to understand whether different values related to the image dimension and the learning rate can be helpful to obtain better performances in malware detection. The best settings given as the input to each model, such as image size, epochs number, and batch dimension, are reported in Table 1. In Table 2 are shown the experimental analysis results. The dataset used was composed of 8446 Android applications, divided into six different Android malware families: *Airpush* (1170), *Dowgin* (763), *FakeInst* (2129), *Fusob* (1275), *Jisut* (550), and *Mecor* (1499). In addition, we also dedicated a class for the *Trusted* (1060) family, which contains legitimate Android applications. Each application was analyzed using different free and commercial antimalware to avoid possible mistakes in dataset labeling. For this task, we exploited the VirusTotal web service (https://www.virustotal.com/ (accessed on 16 November 2022)), which aggregates more than 60 antimalware and performs an online scan. After the division of all applications into the correct class, we divided them into the training (6759) and test (1687) set, with a percentage of 80–20. In addition, we also included in the training folder another sub-folder called "*VAL*", where 1351 samples were stored. During the training session, the validation technique helped us to understand if the model was learning well or not. Usually, the validation was also used to prevent over-fitting. The last phase of this process was to convert all applications into an image in PNG format. The latter was possible using a Python script developed by the authors.

Table 1. The hyperparameters of the models employed in the experimental analysis to compare convolutional and quantum deep learning networks for Android malware detection.

Model	Image Size	Batch	Epochs	Learning Rate	Training Time (HH:MM:SS)
AlexNet	110 × 1	32	50	0.001	0:47:14
Standard-CNN	110 × 1	32	50	0.001	1:11:54
Standard-CNN	25 × 3	64	20	0.001	0:01:34
VGG16	110 × 3	32	25	0.001	2:56:45
MobileNet	110 × 3	64	25	0.001	0:22:45
VGG19	110 × 3	64	25	0.001	3:35:56
EfficientNet	110 × 3	64	25	0.001	0:36:25
EfficientNet	25 × 3	64	20	0.001	0:07:26
Hybrid-CNN	25 × 1	32	20	0.010	1 day, 5:39:14
QNN	4 × 1	64	10	0.010	0:10:34

As emerges from the settings in Table 1, we used the smallest images for the Hybrid-QCNN and QNN models to perform the experiments. To make a better comparison of the different models, we also executed Standard-CNN and EfficientNet using an image with a 25 × 25 × 1 dimension. Although one of our intents was to compare also the fully quantum model, this did not happen because of the smallest image dimension. The convolutional models and the hybrid model cannot execute experiments using an image dimension such as 4 × 4 × 1. The models were evaluated using several metrics such as the *loss*, *accuracy*, *precision*, *recall*, *F-measure*, and *Area Under the Curve* (AUC). Table 3 shows the experimental results on the test set concerning the output classes (the malware families and the Trusted category). In the case of multi-class classification, where there is no binary choice between positive and negative classes, the analysis targets one class at a time: the true positive is the correct class (the cell cross between the predicted and true label in the confusion matrix representation); the false positive is the sum of target class samples misclassified to another class (the Y-axis sum); the false negative is the sum of target class samples of other classes. Table 3 displays the experimental findings for the output classes on the test set (the malware families and the Trusted category).

Table 2. Experimental results obtained with all the models involved in the evaluation by exploiting the test dataset.

Model	Loss	Accuracy	Precision	Recall	F-Measure	AUC
AlexNet	0.435	0.912	0.920	0.908	0.914	0.983
Standard-CNN (110 × 3)	0.205	0.970	0.972	0.970	0.971	0.993
Standard-CNN (25 × 3)	0.260	0.915	0.927	0.910	0.919	0.993
VGG16	0.182	0.952	0.959	0.949	0.954	0.993
MobileNet	0.124	0.966	0.969	0.963	0.966	0.996
VGG19	0.187	0.951	0.953	0.947	0.950	0.994
EfficientNet (110 × 3)	23.738	0.065	0.065	0.065	0.065	0.455
EfficientNet (25 × 3)	2.343	0.251	0.135	0.022	0.038	0.535
Hybrid-QCNN	1.045	0.905	0.905	0.903	0.904	0.962
QNN	1.584	0.413	0.623	0.103	0.176	0.762

Table 3. Experimental results obtained with the convolutional and the quantum model obtaining the best performances for the family classification task.

Output Classes	Models	Accuracy	Precision	Recall	F-Measure	AUC
Airpush	*Standard-CNN*	0.979	0.906	0.952	0.929	0.968
	Hybrid-QCNN	0.951	0.837	0.811	0.824	0.893
Dowgin	*Standard-CNN*	0.978	0.902	0.848	0.874	0.919
	Hybrid-QCNN	0.945	0.694	0.703	0.699	0.836
FakeInst	*Standard-CNN*	0.997	0.990	1.0	0.995	0.998
	Hybrid-QCNN	0.991	0.985	0.978	0.982	0.987
Fusob	*Standard-CNN*	0.999	0.996	1.0	0.998	0.999
	Hybrid-QCNN	0.998	0.988	1.0	0.994	0.998
Jisut	*Standard-CNN*	0.996	0.956	0.990	0.973	0.993
	Hybrid-QCNN	0.994	0.971	0.936	0.953	0.967
Mecor	*Standard-CNN*	0.999	1.0	0.996	0.998	0.998
	Hybrid-QCNN	0.998	0.996	0.996	0.996	0.997
Trusted	*Standard-CNN*	0.976	0.930	0.882	0.905	0.936
	Hybrid-QCNN	0.930	0.714	0.745	0.729	0.851

In Table 2, we report the best results obtained from several experiments. From it, we can see that the best model is the Standard-CNN, with a precision equal to 0.972 and a recall value of 0.970. In addition, also good performances were obtained using the MobileNet, VGG16, and VGG19 models. The Hybrid-QCNN model exhibited a precision equal to 0.905 and a recall of 0.903, while the fully quantum model, labeled as QNN, obtained lower results, such as a precision value of 0.623 and a recall value of 0.103. Due to the limited computing resources, to perform a better comparison between the fully convolutional and hybrid models, we also executed experiments using Standard-CNN and EfficientNet with a size of $25 \times 25 \times 1$. From Table 2, we can see, albeit by a little, that the Standard-CNN model performed better than the Hybrid model. We can also say that the results obtained from the Hybrid models were far superior than the EfficientNet results. In addition, it is worth noting also the interesting results obtained after the usage of the AlexNet model. Although it usually does not give back a high score for the metrics, in this case, we achieved a precision value equal to 0.920 and a recall value equal to 0.983. We, respectively, report in Figures 6 and 7 the confusion matrix of the Standard-CNN and Hybrid-QCNN models. The images help to perform a more accurate direct comparison of the best deep learning model (in this work) and the quantum model.

In Figure 6, we report the confusion matrix obtained from the *Standard-CNN* model. In that, we can see that the Deep Learning model produced a small number of misclassifications. The higher number of items classified incorrectly was for the Airpush family, with 11 samples classified as Trusted. The confusion matrix obtained after the execution of the Hybrid-QCNN model is shown in Figure 7.

The Hybrid-QCNN model had a significantly larger number of misclassified samples than the Standard-CNN model. Similar to what happened with the Standard-CNN model, the majority of misclassified samples belonged to the Airpush and Dowgwin malware families and were designated as Trusted by the Hybrid-QCNN model. A possible reason for this result is that, by reducing both sets of images, the Airpush and Dowgin malware looks similar to the Trusted applications. As we can see from the confusion matrix shown in Figure 2, the number of Airpush and Dowgwin malware applications labeled as Trusted is, respectively, 24 and 25 samples.

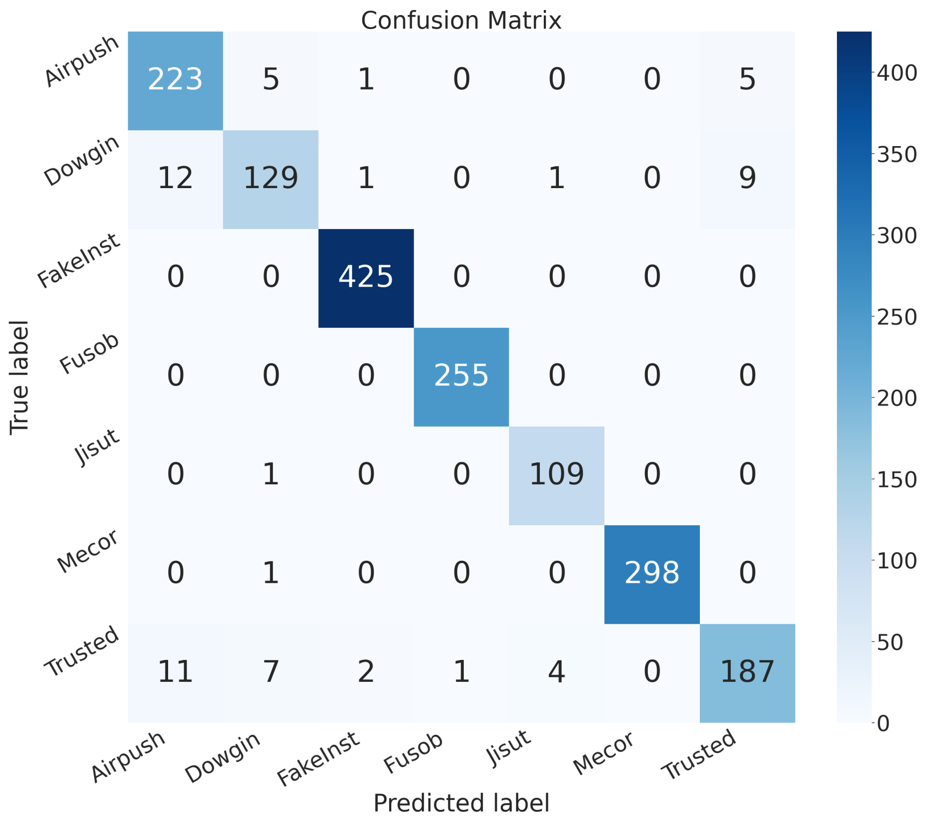

Figure 6. Confusion matrix obtained with the Standard-CNN model for the family classification task.

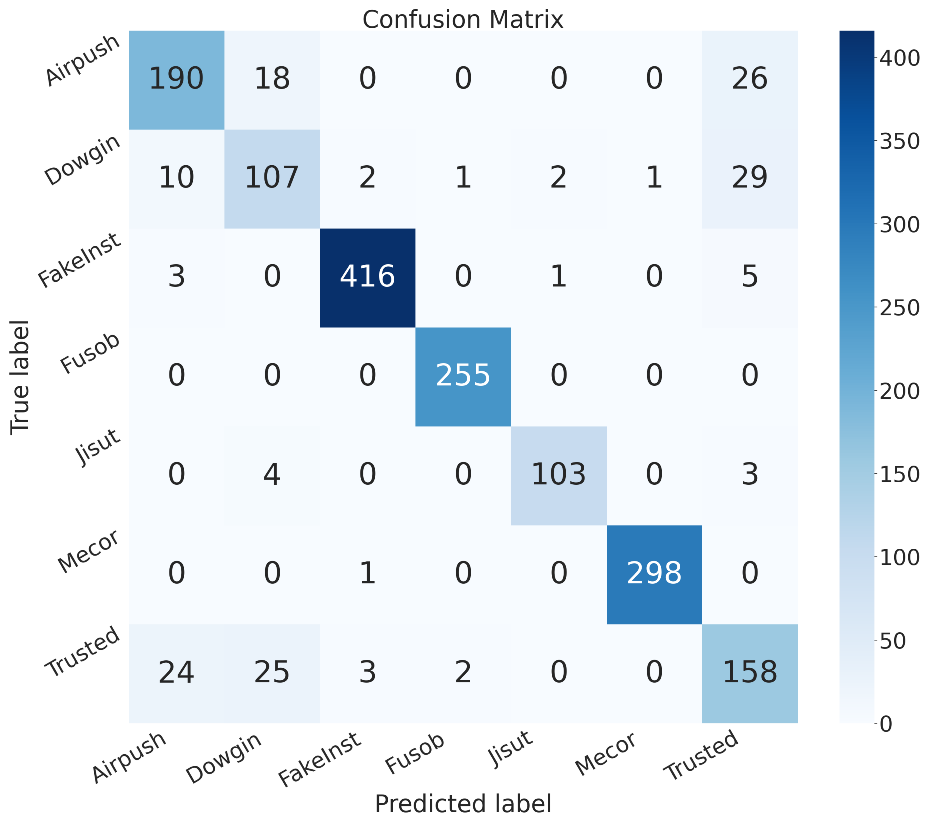

Figure 7. Confusion matrix obtained with the Hybrid-QCNN model for the family classification task.

5. Discussion

Unfortunately, due to hardware limitations, we were not able to experiment with the quantum models (i.e., Hybrid-QCNN and the QNN) using images with dimensions bigger than 25 × 1. Anyway, when we trained the quantum and convolutional models by using the same image dimensions, we can note that the quantum models can obtain better performances. This result is highlighted in Figure 2, in particular by observing the results obtained from the EfficientNet(25 × 3) and Hybrid-QCNN networks, with an accuracy, respectively, equal to 0.251 (for the EfficientNet model) and 0.905 (for the Hybrid-QCNN one). The Standard-CNN model, with images of a size of 25 × 3, obtained an accuracy equal to 0.915, which is slightly higher if compared with the one obtained from the Hybrid-QCNN ones, but the Standard-CNN model was trained with 64 as the dimension of the batch, while the Hybrid-QCNN with a batch size equal to 32. For these reasons, we observed that quantum Machine Learning can be promising in the malware detection task.

After the execution of each experiment, we decided to apply the Grad-CAM algorithm to the best convolutional and quantum models, i.e., the Standard-CNN and Hybrid-QCNN ones. Using this algorithm, it is simple to understand which areas of one image are most influential at the end of the discrimination. Below, we report some examples of a single image that included the PNG image of the file, the activation map, and the image generated by overlaying the initial PNG image with its activation map. The activation map consisted of three unique colors: yellow, green, and blue. The regions colored yellow symbolize the most interesting area, while green designates the sections in the center. Finally, to highlight a section of the image that is unrelated to the model, we used the blue shadow.

In Figure 8, we can see the detection of a sample belonging to the Airpush family with a high precision of 100%. As is possible to see, the most significant area is at the top of the image. In fact, there, the pixels are overlayed with the yellow shadow, while other parts of the image do not appear to be very important to classify the malware. As is described in Section 4, the models can make mistakes in terms of identification. Figure 9 show the wrong classification of malware belonging to the Airpush family recognized as a member of the Downgin family with an accuracy slightly less than 100% (99.9%).

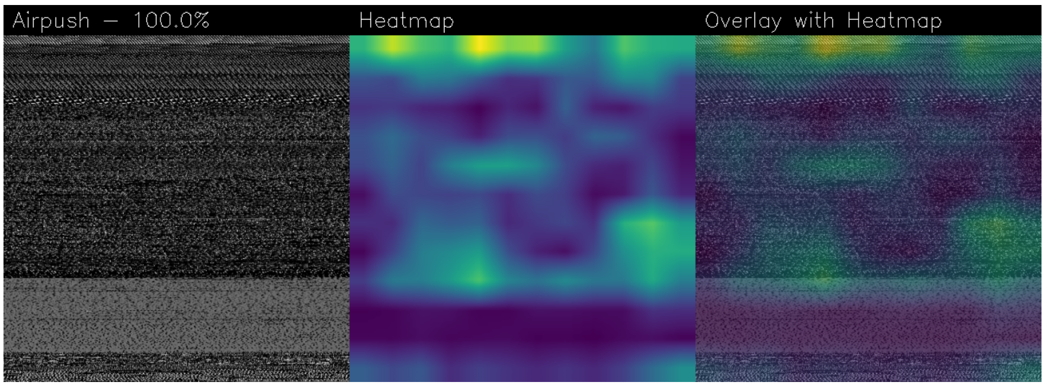

Figure 8. Sample classified as belonging to the Airpush family (4a985c341e1ee08f647395d00640c1f2) with 100%, where we can note that the yellow areas are on the upper side of the image, obtained with the model built using the Standard-CNN network.

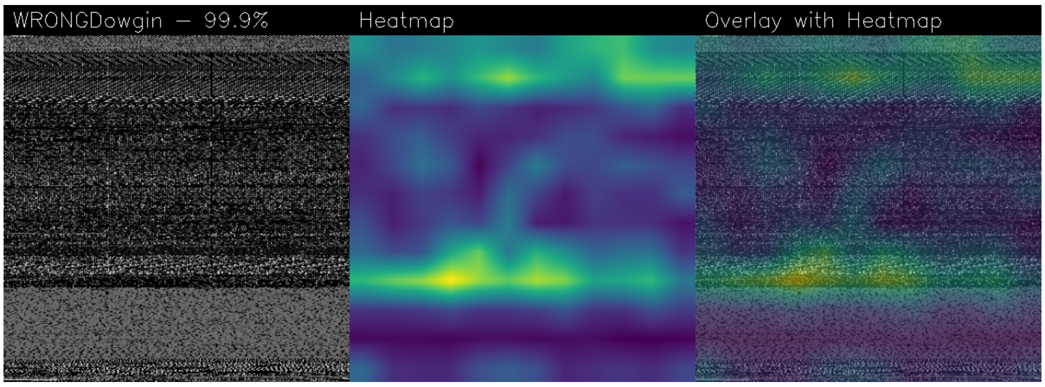

Figure 9. The wrong classification of the sample Airpush (da46241dde7209feb52ff1b37e8f5080) misclassified as belonging to the Dowgi family with 99.9%, where we can note the yellow areas are in the middle part of the image, obtained with the model built using the standard-CNN network.

In Figure 10, we report an example of the classification of a Dowgin malicious sample with a precision value equal to 100%. In that picture, it is possible to understand why in eighteen cases of the images related to this family were classified as Airpush. In fact, as is possible to see in Figures 8 and 10, pixels at the top of the images are highlighted with the yellow shadow. Instead, in Figure 11, the models are classified as Trusted dangerous applications belonging to the Dowgin family.

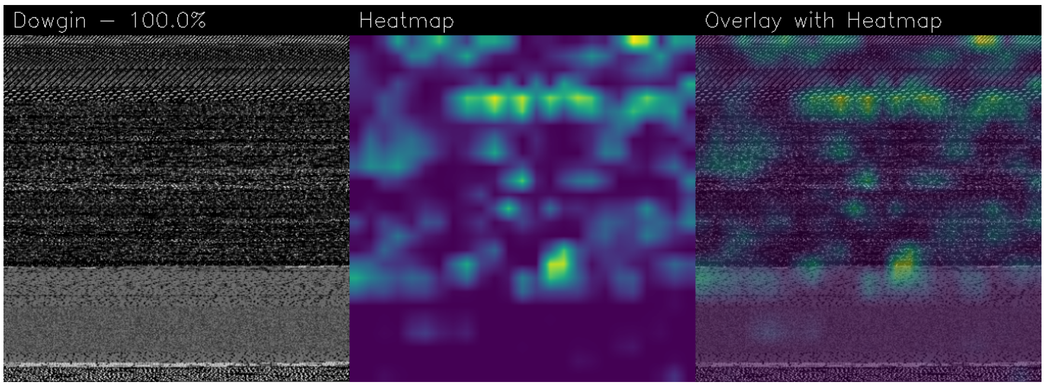

Figure 10. Sample classified as belonging to the Downgin family (0d1366528bf2276fdc686b1f3deb38e5) with 100%, where we can note the yellow areas are really small and in the middle part of the image, obtained with the model built using the Hybrid-QCNN network.

In conclusion, we can affirm that, as it did not achieve higher results, using the Hybrid-QCNN makes more mistakes during the classification than the Standard-CNN model. In addition, thanks to the usage of the Grad-CAM algorithm, even inexperienced people can understand where the model has confusion.

Assuming that malware belonging to the same family shares the malicious payload (i.e., the dangerous action), the idea at the bottom of Grad-CAM is to highlight similar areas in the images. Differently, samples belonging to different families should exhibit different areas highlighted by Grad-CAM. The rationale is to find regions of the images obtained from malware symptomatic of certain malicious behaviors: in this way, the security analyst can focus his/her manual effort on a reduced section of the application under analysis. We are aware that images obtained from application source code can be more informative from this point of view; as a matter of fact, in this case, the areas highlighted will be related to code snippets and, for this reason, more understandable from the security analysis.

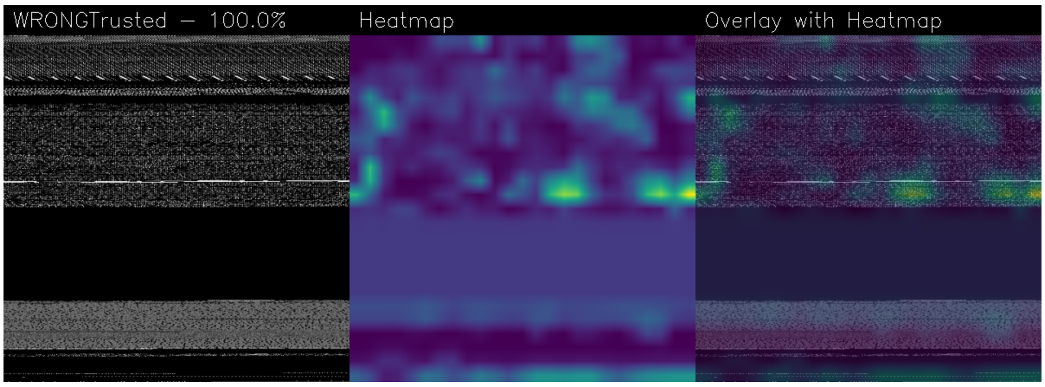

Figure 11. The wrong classification of sample Dowgin (6afbe9d4c41fd46a3c516a9203dc4393) misclassified as Trusted with 100%, where we can identify really small yellow areas are in the middle part of the image, obtained with the model built using the Hybrid-QCNN network.

6. Related Work

In this section, we provide the current state-of-art use of deep learning to detect mobile malware and the adoption of quantum Machine Learning for general tasks.

The use of quantum computing is significant in several fields, such as medicine. In fact, after the global pandemic in 2020, the usage of deep learning helped during the evaluation of the thoracic CT exam. To improve that performance, Amin et al. in [42] used *QML*. The authors concluded that, through the use of *quantum algorithms*, the performances in spotting the infection in its early stages increased.

The authors in [25] implemented Support Vector Machine (SVM) in a quantum computer with the complexity logarithmic in the size of the vectors and the number of training examples. In circumstances when classical sampling algorithms need polynomial time, an exponential speedup is obtained. At the core of this quantum network, the big data algorithm is a non-sparse matrix exponentiation technique for efficiently performing a matrix inversion of the training data inner-product (kernel) matrix. In contrast to them, we used a Hybrid QCNN model with a high level of accuracy, i.e., 0.905.

The researchers in [26] demonstrated that, by using quantum computing, the time required to train a deep restricted Boltzmann machine was reduced and the learning results were better than classical computing. As a result, the optimization of the underlying objective function improved significantly. In addition, quantum computer techniques have been demonstrated to efficiently handle some problems that are intractable on ordinary, classical computers. Furthermore, the quantum method demonstrated may efficiently train entire Boltzmann machines and multilayer, fully connected models for which there are no efficient classical counterparts.

In [43], Seymour and his colleague created a minimalist malware classifier using cross-validation. The accuracy obtained was comparable with the standard Machine Learning algorithms.

The authors in [27] showed that quantum Machine Learning algorithms exponentially increased their speed, opposite the classical models, performing their tasks in logarithmic time both with the number of vectors and their dimension. This study was conducted on supervised and unsupervised quantum Machine Learning algorithms for cluster assignment.

In [44], the researchers proposed a mix of software used to discover the most common hashes and n-grams between benign and malicious software and a quantum search method. The latter will be utilized to find the desired hash value by searching through every permutation of the entangled key and value pairs. This eliminates the need to recompute hashes for a set of n-grams.

The authors in [45] proposed a federated learning approach aimed to detect malware in IoT devices, by modeling the network traffic of several real IoT devices affected by malware. Both supervised and unsupervised federated models (multi-layer perceptron and autoencoder) were exploited by the authors.

The researchers in [46] compared the performance obtained from 26 state-of-the-art pre-trained Convolutional Neural Network models in Android malware detection, by also including the performance obtained by large-scale learning with the SVM and RF classifiers. The model obtaining the best performance was the EfficientNet-B4 one.

The authors in [47] proposed a hybrid multi-level deep learning model, formed by unsupervised and supervised DL models, to encode X86 malware binary files and classify them. It uses similarity matrices to detect relevant similarity patterns between input samples.

Several studies tried to propose methods for malicious malware family detection. More specifically, the authors in [48] presented a solution that is based on the BIRCH clustering algorithm. After extracting static and dynamic analysis features from the malware samples, they constructed the necessary vectors for the clustering algorithm and grouped the samples into families based on this information. Another study that used the call graph clustering was proposed in [49].

In [50], the authors extracted n-gram features from the binary code of the samples. These features were selected using the Sequential Floating Forward Selection (SFFS) method with three classifiers, and the accuracy of the method reached 96.64%. However, their methodology was different from the one presented in this paper, and they did not provide results specifically referring to ransomware.

In [51], the authors made use of the PrefixSpan algorithm to create groups of malware families, based on sequence patterns. However, these patterns are related only to network traffic packets and not to the overall behavior of the malware. In [6] also, malware samples were assigned to a specific family by exploiting model checking. In [52], ARIGUMA was proposed, a malware analysis system for family classification. The authors considered the number of functions being called in a function and the number of functions that call a specific function. Furthermore, the local variables, the arguments, and the instructions were taken into account. Even though this method can also detect obfuscated APIs, the classification accuracy was only 61.6%. In [53], the authors presented a malware family classification system that was based on the instruction sequence.

The researchers in [54] adopted supervised classification algorithms for ransomware family identification. Moreover, they considered the binary trees generated by these algorithms to infer the phylogenetic relationships.

As emerged from the current state-of-the-art analysis, this paper represents the first attempt to introduce quantum Machine Learning in the malware detection research field. Moreover, this is the first paper devoted to introducing an explanation behind the decision made by a quantum Machine Learning classifier.

7. Conclusions and Future Works

In the last few years, we have been witnessing an increase in mobile malware, with particular regard to malicious payloads targeting the Android platform, the most diffused one. Considering the inefficiency of the currently adopted signature-based approach in the detection of zero-day malicious behaviors, the research community, from both the academic and industrial sides, is focused on proposing new techniques to mitigate malware, mainly exploiting deep learning. In this last context, deep learning recently has introduced quantum computing to train models using the integration of quantum algorithms within Machine Learning algorithms. For these reasons, we proposed to apply quantum Machine Learning models for Android malware detection. We considered a comparison between five state-of-the-art deep learning models, i.e., AlexNet, MobileNet, EfficientNet, VGG16, VGG19, and a deep learning network developed by the authors, which we called Standard-CNN, and two quantum models, the first one being a hybrid quantum model and the second one a fully quantum model: all these (convolutional and quantum) models were trained to perform the malware detection in the Android environment. The input for the models was a set of images obtained from the conversion of Android applications into grayscale images. In the experimental analysis, 7386 applications belonging to six different malicious families and 1060 legitimate applications were considered. As a result of the experiments, we obtained that the architecture obtaining the best performance was the Standard-CNN model, with an accuracy equal to 0.970. Relating to the quantum models, the best model in terms of accuracy was the Hybrid-QCNN model, with an accuracy of 0.905. In future work, more complex quantum models (in terms of PQC layers) will be considered. Furthermore, considering that the proposed method is not operating as system-dependent (as a matter of fact, the images are generated directly from the application bytes), we will evaluate whether the proposed approach is working in ransomware detection. Additionally, to provide more explainability, different images will be proposed, for instance an interesting improvement of the proposed method is represented by the adoption of an image obtained from the application code; in this case, Grad-CAM will be able to highlight areas that are immediately traceable to code snippets and, presumably, the malicious payload.

Author Contributions: Methodology, F.M.; Software, G.C., G.I. and M.S.; Validation, F.M., G.C., G.I. and M.S.; Formal analysis, F.M. and A.S.; Writing—original draft, F.M. and F.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by the EU DUCA,EU CyberSecPro, and EU E-CORRIDOR projects and PNRR SERICS_SPOKE1_DISE, RdS 2022-2024 cybersecurity.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mercaldo, F.; Santone, A. Deep learning for image-based mobile malware detection. *J. Comput. Virol. Hacking Tech.* **2020**, *16*, 157–171. [CrossRef]
2. Casolare, R.; Lacava, G.; Martinelli, F.; Mercaldo, F.; Russodivito, M.; Santone, A. 2Faces: A new model of malware based on dynamic compiling and reflection. *J. Comput. Virol. Hacking Tech.* **2022**, *18*, 215–230. [CrossRef]
3. Iadarola, G.; Martinelli, F.; Mercaldo, F.; Santone, A. Formal methods for android banking malware analysis and detection. In Proceedings of the 2019 IEEE Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), Granada, Spain, 22–25 October 2019; pp. 331–336.
4. Kumar, K.A.; Raman, A.; Gupta, C.; Pillai, R. The Recent Trends in Malware Evolution, Detection and Analysis for Android Devices. *J. Eng. Sci. Technol. Rev.* **2020**, *13*, 240–248 . [CrossRef]
5. Cimitile, A.; Martinelli, F.; Mercaldo, F. Machine Learning Meets iOS Malware: Identifying Malicious Applications on Apple Environment. In Proceedings of the ICISSP , Porto, Portugal, 19–21 February 2017; pp. 487–492.
6. Cimino, M.G.; De Francesco, N.; Mercaldo, F.; Santone, A.; Vaglini, G. Model checking for malicious family detection and phylogenetic analysis in mobile environment. *Comput. Secur.* **2020**, *90*, 101691. [CrossRef]
7. Elsersy, W.F.; Feizollah, A.; Anuar, N.B. The rise of obfuscated Android malware and impacts on detection methods. *Peerj Comput. Sci.* **2022**, *8*, e907. [CrossRef]
8. Dave, D.D.; Rathod, D. Systematic Review on Various Techniques of Android Malware Detection. In Proceedings of the International Conference on Computing Science, Communication and Security, Mehsana, India, 6–7 February 2022; pp. 82–99.
9. Ferrante, A.; Medvet, E.; Mercaldo, F.; Milosevic, J.; Visaggio, C.A. Spotting the malicious moment: Characterizing malware behavior using dynamic features. In Proceedings of the IEEE 2016 11th International Conference on Availability, Reliability and Security (ARES), Salzburg, Austria, 31 August–2 September 2016; pp. 372–381.
10. Casolare, R.; Ciaramella, G.; Iadarola, G.; Martinelli, F.; Mercaldo, F.; Santone, A.; Tommasone, M. On the Resilience of Shallow Machine Learning Classification in Image-based Malware Detection. *Procedia Comput. Sci.* **2022**, *207*, 145–157. [CrossRef]
11. Yuxin, D.; Siyi, Z. Malware detection based on deep learning algorithm. *Neural Comput. Appl.* **2019**, *31*, 461–472. [CrossRef]
12. Buduma, N.; Buduma, N.; Papa, J. *Fundamentals of Deep Learning*; O'Reilly Media, Inc.: Sevastopol, CA, USA, 2022.
13. Giannotti, F. Explainable Machine Learning for trustworthy AI. In *Artificial Intelligence Research and Development*; IOS Press: Amsterdam, The Netherlands, 2022; p. 3.
14. Pedreschi, D.; Giannotti, F.; Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F. Meaningful explanations of black box AI decision systems. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 9780–9784.
15. Schuld, M.; Sinayskiy, I.; Petruccione, F. An introduction to quantum Machine Learning. *Contemp. Phys.* **2015**, *56*, 172–185. [CrossRef]
16. Martín-Guerrero, J.D.; Lamata, L. Quantum Machine Learning: A tutorial. *Neurocomputing* **2022**, *470*, 457–461. [CrossRef]

17. Ciaramella, G.; Iadarola, G.; Mercaldo, F.; Storto, M.; Santone, A.; Martinelli, F. Introducing Quantum Computing in Mobile Malware Detection. In Proceedings of the 17th International Conference on Availability, Reliability and Security, Vienna, Austria, 23–26 August 2022; pp. 1–8.
18. Gandotra, E.; Bansal, D.; Sofat, S. Malware analysis and classification: A survey. *J. Inf. Secur.* **2014**, *2014*, 44440. [CrossRef]
19. Massoli, F.V.; Vadicamo, L.; Amato, G.; Falchi, F. A Leap among Entanglement and Neural Networks: A Quantum Survey. *arXiv* **2021**, arXiv:2107.03313.
20. Vasan, D.; Alazab, M.; Wassan, S.; Safaei, B.; Zheng, Q. Image-Based malware classification using ensemble of CNN architectures (IMCEC). *Comput. Secur.* **2020**, *92*, 101748. [CrossRef]
21. Iadarola, G.; Martinelli, F.; Mercaldo, F.; Santone, A. Towards an interpretable deep learning model for mobile malware detection and family identification. *Comput. Secur.* **2021**, *105*, 102198. [CrossRef]
22. Hirvensalo, M. *Quantum Computing*; Springer Science & Business Media: Berlin, Germany, 2003.
23. Gill, S.S.; Kumar, A.; Singh, H.; Singh, M.; Kaur, K.; Usman, M.; Buyya, R. Quantum computing: A taxonomy, systematic review and future directions. *Softw. Pract. Exp.* **2022**, *52*, 66–114. [CrossRef]
24. Boyer, M.; Liss, R.; Mor, T. Geometry of entanglement in the Bloch Sphere. *Phys. Rev. A* **2017**, *95*, 032308. [CrossRef]
25. Rebentrost, P.; Mohseni, M.; Lloyd, S. Quantum support vector machine for big data classification. *Phys. Rev. Lett.* **2014**, *113*, 130503. [CrossRef] [PubMed]
26. Wiebe, N.; Kapoor, A.; Svore, K.M. Quantum deep learning. *arXiv* **2014**, arXiv:1412.3489.
27. Lloyd, S.; Mohseni, M.; Rebentrost, P. Quantum algorithms for supervised and unsupervised Machine Learning. *arXiv* **2013**, arXiv:1307.0411.
28. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332. [CrossRef]
29. Aïmeur, E.; Brassard, G.; Gambs, S. Machine Learning in a quantum world. In Proceedings of the Conference of the Canadian Society for Computational Studies of Intelligence, Québec City, QC, Canada, 7–9 June 2006; pp. 431–442.
30. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
31. Khan, S.; Rahmani, H.; Shah, S.A.A.; Bennamoun, M. A guide to Convolutional Neural Networks for computer vision. *Synth. Lect. Comput. Vis.* **2018**, *8*, 1–207.
32. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
33. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
34. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 618–626.
35. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient Convolutional Neural Networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
36. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International conference on Machine Learning. PMLR, Vancouver, BC, Canada, 13 December 2019; pp. 6105–6114.
37. Broughton, M.; Verdon, G.; McCourt, T.; Martinez, A.J.; Yoo, J.H.; Isakov, S.V.; Massey, P.; Halavati, R.; Niu, M.Y.; Zlokapa, A.; et al. Tensorflow quantum: A software framework for quantum machine learning. *arXiv* **2020**, arXiv:2003.02989.
38. Farhi, E.; Neven, H. Classification with quantum neural networks on near term processors. *arXiv* **2018**, arXiv:1802.06002.
39. Zhou, B.; Khosla, A.; Lapedriza, A.; Oliva, A.; Torralba, A. Learning deep features for discriminative localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2921–2929.
40. Bacci, A.; Bartoli, A.; Martinelli, F.; Medvet, E.; Mercaldo, F. Detection of obfuscation techniques in android applications. In Proceedings of the 13th International Conference on Availability, Reliability and Security, Hamburg, Germany, 27–30 August 2018; pp. 1–9.
41. Bacci, A.; Bartoli, A.; Martinelli, F.; Medvet, E.; Mercaldo, F.; Visaggio, C.A. Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis. In Proceedings of the ICISSP, Funchal, Portugal, 22–24 January 2018; pp. 379–385.
42. Amin, J.; Sharif, M.; Gul, N.; Kadry, S.; Chakraborty, C. Quantum Machine Learning architecture for COVID-19 classification based on synthetic data generation using conditional adversarial neural network. *Cogn. Comput.* **2022**, *14*, 1677–1688. [CrossRef] [PubMed]
43. Seymour, J.J. *Quantum Classification of Malware*; University of Maryland, Baltimore County: Baltimore, MD, USA, 2014.
44. Allgood, N.R. A Quantum Algorithm to Locate Unknown Hashes for Known n-Grams Within a Large Malware Corpus. Ph.D. Thesis, University of Maryland, Baltimore County, Baltimore, MD, USA, 2020.
45. Rey, V.; Sánchez, P.M.S.; Celdrán, A.H.; Bovet, G. Federated learning for malware detection in iot devices. *Comput. Netw.* **2022**, *204*, 108693. [CrossRef]
46. Yadav, P.; Menon, N.; Ravi, V.; Vishvanathan, S.; Pham, T.D. EfficientNet Convolutional Neural Networks-based Android malware detection. *Comput. Secur.* **2022**, *115*, 102622. [CrossRef]
47. Venkatraman, S.; Alazab, M.; Vinayakumar, R. A hybrid deep learning image-based analysis for effective malware detection. *J. Inf. Secur. Appl.* **2019**, *47*, 377–389. [CrossRef]

48. Pitolli, G.; Aniello, L.; Laurenza, G.; Querzoni, L.; Baldoni, R. Malware family identification with BIRCH clustering. In Proceedings of the 2017 International Carnahan Conference on Security Technology (ICCST), Madrid, Spain, 23–26 October 2017; pp. 1–6. [CrossRef]
49. Kinable, J.; Kostakis, O. Malware classification based on call graph clustering. *J. Comput. Virol.* **2011**, *7*, 233–245. [CrossRef]
50. Liangboonprakong, C.; Sornil, O. Classification of malware families based on N-grams sequential pattern features. In Proceedings of the 2013 IEEE 8th Conference on Industrial Electronics and Applications (ICIEA), Melbourne, VIC, Australia, 19–21 June 2013; pp. 777–782. [CrossRef]
51. Boukhtouta, A.; Lakhdari, N.E.; Debbabi, M. Inferring Malware Family through Application Protocol Sequences Signature. In Proceedings of the 2014 6th International Conference on New Technologies, Mobility and Security (NTMS), Dubai, United Arab Emirates, 30 March–2 April 2014; pp. 1–5. [CrossRef]
52. Zhong, Y.; Yamaki, H.; Yamaguchi, Y.; Takakura, H. Ariguma code analyzer: Efficient variant detection by identifying common instruction sequences in malware families. In Proceedings of the 2013 IEEE 37th Annual Computer Software and Applications Conference, Kyoto, Japan, 22–26 July 2013; pp. 11–20.
53. Huang, K.; Ye, Y.; Jiang, Q. ISMCS: An intelligent instruction sequence based malware categorization system. In Proceedings of the 2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication, Hong Kong, China, 20–22 August 2009; pp. 509–512. [CrossRef]
54. Martinelli, F.; Mercaldo, F.; Michailidou, C.; Saracino, A. Phylogenetic Analysis for Ransomware Detection and Classification into Families. In Proceedings of the SECRYPT, Porto, Portugal, 26–28 July 2018; pp. 732–737.