Towards an in-depth detection of malware using multi-QCNN

Tony Quertier and Grégoire Barrué

Orange Innovation, Rennes, France

Abstract. Malware detection is an important topic of current cybersecurity, and Machine Learning appears to be one of the main considered solutions even if certain problems to generalize to new malware remain. In the aim of exploring the potential of quantum machine learning on this domain using only a few qubits, we implement a new preprocessing of our dataset using Grayscale method, and we couple it with a model composed of five quantum convolutional networks and a scoring function. We get an increase of around 20% of our results, both on the accuracy of the test and its F1-score.

Keywords: Quantum Machine Learning · Malware detection · Quantum convolutional networks.

Introduction

Malicious software detection has become an important topic in business, as well as an important area of research due to the ever-increasing number of successful attacks using malware. Cybersecurity researchers are recently shifting their attention to Machine Learning (ML) methods to improve malicious files detection [1,2], and they have been incredibly creative in data preprocessing. In fact, this part is essential now that the learning algorithms are already extremely powerful.

In [3], Anderson et al. trained a feature-based malware detection model using a non-optimized LightGBM algorithm, in [4] Nataraj et al. use k-nearest neighbors algorithm on image-based malware.Image-based malware detection is a challenge in both classical and quantum computing, but for different reasons. In classical machine learning, one limitation to obtaining results as good as with standard static features is the number of training data. Because malware images are much more complex and less representative than standard images, convolutional networks need a lot of images to extract information. However, computing resources are not a problem for two-channel images of size 64×64.

In recent years, research into QML has developed very significantly, whether in the area of learning theory [5], generalisation capabilities [6,7], or more practical use-cases as earth observation study by ESA[1]. In particular, quantum convolutional neural networks (QCNNs) have proved to be interesting, because they perform well, and for example do not exhibit Barren Plateaus [8]. They were first

[1] https://eo4society.esa.int/projects/qc4eo-study/

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

introduced in [9], and used for instance in [10] for classical data classification. Their full understanding is still missing, but some works, as [11] start to deeply analyze these models, and we think that it could be a suitable solution for image classification, as is its classical equivalent.

The challenge also comes from the limited number of qubits. On the features, we have restricted the number of qubit to less than 8 [12], but for image-based detection this number is not enough. We still want to give an approach using relatively few qubits, to keep coherence with the current capacities of the quantum computers.It is quite difficult to extract relevant information from a malware image of size 64x64 with just a few qubits, so we turn ourselves to quantum convolutional neural networks, and we split each image of binary file into sub-images, corresponding to specific sections of the binary file.

In this paper, we present an algorithm consisting of five QCNNs trained on the images corresponding to the different sections of a binary file and a scoring function to extract as much information as possible from it.

1 Dataset and preprocessing

1.1 Data

We rely on two different datasets, Bodmas and PEMachineLearning.More details on these two datasets are available in the paper [13].We split the dataset into three sub-datasets: one to train the QCNNs on the binary file's sections, one to train the final scoring function and one to test our architecture. Each sub-dataset is composed of 20k benign and 20k malware and the train/validation ratio is 70/30 for the first two.

The PE format is a file format used by Windows operating systems. A PE file is separated into two parts: the header and the sections. The header describes the file and its contents such as the date the file was created, information about loading the file into memory and the number of sections. Each section is described by a specific header containing its name, size and location in virtual memory. Sections generally contain the executable code (.text) and the variables used with their default values (.data, or in read only .rdata). The relocation section (.reloc) contains relocation information and the resource section (.rsrc) contains resources like icons, menus, and other elements. The resource section is often used by malware to evade detection. For example, scripts can be used to inject payloads directly into this section. When the binary file is executed, the embedded payload is extracted and executed. In this paper we focus in the five sections presented above, because they are present in the majority of the binary files, while others sections could be found in less files and thus give a smaller dataset for the training of a QCNN.

1.2 Splitting the image into sub-images

When preprocessing our data, we are using the Grayscale method, that transforms the malware into an image [4], but in a more subtle way that is better

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

adapted to our problem. We first explore the PE file using the LIEF library to identify the start and end of each section. Then we transform the content of each section into an image of size 8×8. To begin with, we have identified 5 sections that we consider to be relevant and we focus on these. We train a QCNN on each section thanks to our first sub-dataset. Then we use these trained QCNNs to assign to each file a score per section when the sections are present. If they are not, we give these sections a score of -1. Initially we had chosen a neutral score of 0.5, but this introduced a bias because this is equivalent to considering that the absence of a section is of no importance, which is not true.

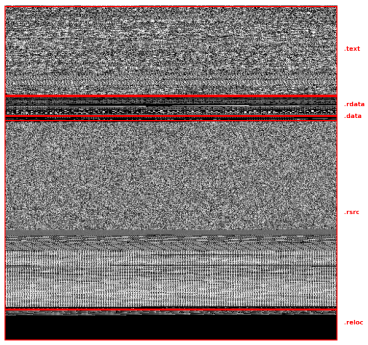
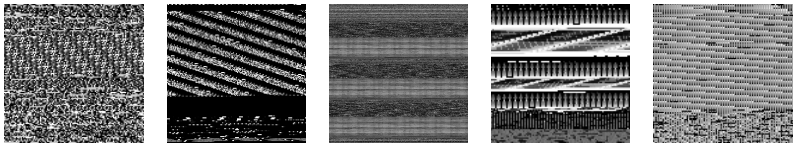


Fig. 1: Sections in a malware



(a) .text (b) .data (c) .rdata (d) .rsrc (e) .reloc

Fig. 2: Images of different binary's sections

2 Framework

2.1 Description of the QCNN and its training on a section image

Here we present the architecture of our algorithm. To train a QCNN for each chosen section, we generate images for each section extracted from the file, then we train the QCNNs on the different sections by setting the number of qubits to 8 and therefore using 3 layers, as shown in Figure 3.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

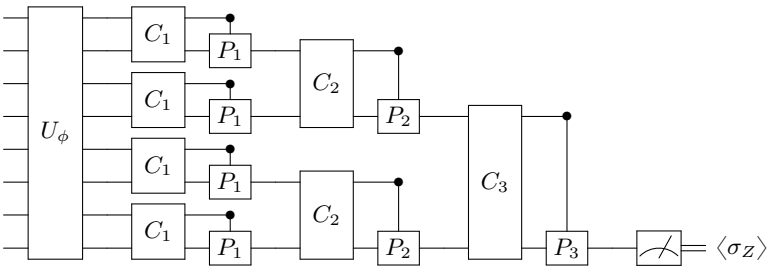
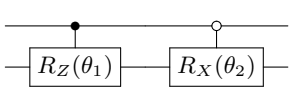
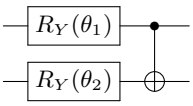


Fig. 3: The architecture of our QCNN, where convolutional and pooling layers alternate up to the measurement by an observable.

The QCNN consists of a layered architecture as shown in [10]. A layer is composed of a convolutional layer (C_i in Figure 3), entangling qubits with parameterized gates, and a pooling layer (P_i in Figure 3), which also entangles qubits but then reduces the number of qubits by tracing out half of them. Figure 4 details the architecture we used for our convolutional and pooling layers. At the end of the quantum circuit, we measure the last qubit in the Z-basis.



(a) Pooling layer



(b) Convolutional layer

Fig. 4: Description of the convolutional and pooling layers used in our algorithm. At the end of the pooling layer we trace out the control qubit in order to reduce the dimension.

Since we use 8 qubits, we use a PCA on the image to reduce 8x8=64 features to 8. This reduction is already less critical than reducing a 64x64=4096 image to 8 features. For the encoding of data, we map all input data $x \in \mathbb{R}^n$ in $[0, \frac{\pi}{2}]^n$, and then we apply the encoding map

$$U_\phi : x \mapsto |\phi(x)\rangle = \bigotimes_{i=1}^{n} (\cos(x_i)|0\rangle + \sin(x_i)|1\rangle) \,. \tag{1}$$

We build five QCNNs with the same architecture, which are trained on images corresponding to a specific section. Scores are available in the Section 3. Each QCNN is trained over five epochs. For the optimization method, we use Simultaneous Perturbation for Stochastic Backpropagation (SPSB) [14].

As well as working with a small number of qubits for each QCNN, an interesting advantage is in the interpretation of the results. In addition to the detection score, we are able to see, thanks to the outputs of each QCNN, which section of the binary file is suspicious, and if necessary investigate its content.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

2.2 Training a scoring function

Once the parameters of each QCNN have been set using the first sub-dataset, we need to find a customized scoring function to treat the scores given by these QCNNs to the files of the second sub-dataset. One could think about using a majority vote, but this would include a bias that suggests that every section is equally important. We use the second sub-dataset, and for each binary file, the file is decomposed into images corresponding to the sections present in the binary. The images are then passed through the QCNNs associated with the sections, and each QCNN returns a score. As a reminder, if the section is missing, the score is set to -1 for that section.

As an output of this step, we therefore have a vector composed of the five scores, corresponding to the probability that the file is a malware regarding each section, and we want the algorithm to decide if the file is malicious or not as a final output. We test various functions and train some algorithms to try and find the most optimal function for calculating this score. For the final version of the algorithm, we use the third sub-dataset to test all the steps of our model. We keep a XGBoost model as scoring function, which gave us the best results. The various experiments and results can be found in section 3.

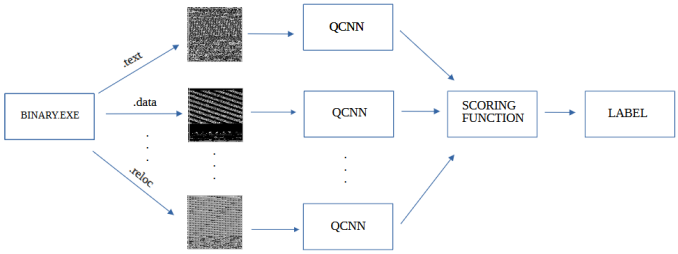


Fig. 5: Architecture of our algorithm. Once everything is trained, we get a model which takes the binary as input and gives its predicted label as output.

3 Experiments and results

In this section we present the results of our algorithm, which contains three steps. First, we train a QCNN model for each five section of the files. When a section is not present in the file, there is no image corresponding to this section, so the different QCNNs are not trained on exactly the same sizes of dataset, but this guarantees the unbiased property of the training.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

Table 1 gathers the training and testing accuracy for each section. We also compute the F1-score, which gives an additional proof of the performances of our results. The results by section are not very good, and actually they are comparable to the model where we train the QCNN on the images of the entire files (see Table 3b). The learning also seems to be better on some sections, which can tell us about their importance for the classification task.

Table 1: QCNN training results for each section

	Train	Test	
Section	Accuracy	Accuracy	F1-score
text	0.67	0.69	0.68
data	0.56	0.55	0.7
rdata	0.66	0.66	0.78
rsrc	0.65	0.63	0.72
reloc	0.69	0.7	0.67

Once the QCNNs are trained, we use them with the second part of the dataset in order to assign to each PE file a vector containing the scores associated to each section. Then we have to train the scoring function on this dataset. We try several scoring functions, which are presented in Table 2. The XGBoost and the Random Forests (RF) models have the best performances, the RF model being better on the train but comparable on the test. We also try a majority vote, which as expected is not relevant in our use-case.

Table 2: Scoring function training results

	Train		Test	
Section	Accuracy	F1-score	Accuracy	F1-score
XGBoost	0.89	0.90	0.84	0.85
LGBM	0.84	0.86	0.82	0.85
RF	0.99	0.99	0.85	0.86
Maj. Vote			0.4	0.27

Finally, once both the QCNNs and the scoring function are trained, we use the third subdataset as a test sample, to evaluate the performances of the whole model. The two compared models in Table 3a, show that the XGBoost model seems to perform slightly better than the RF model.

For comparison, we trained a QCNN on the second sub-dataset and then tested it on the third sub-dataset with the same parameters. Table 3b, shows that training a model with a PCA on the entire image of file is not a suitable solution.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

Table 3: Final tests to compare multi-QCNN and single QCNN.

	Accuracy	F1-score
RF	0.82	0.80
XGBoost	0.83	0.83

(a) Testing multi-QCNN with RF and XGBoost on the third dataset.

	Accuracy	F1-score
Train	0.68	0.70
Test	0.60	0.66

(b) Results of QCNN on a complete binary image.

4 Conclusion and future work

The power of our proposed algorithm is that we can explore many different directions to enhance its performances. First we can increase the number of sections taken into account, in order to understand their impact on the final score. This could be done very efficiently as we do not have to re-train the QCNNs on the already studied sections, but just train some new QCNNs on the additional sections and take them into account in the input of the scoring function. We can also investigate what scoring function would be the most relevant for our problem. For example, the use of random forests allows to identify which sections are the most likely to help the classification task. Besides, once this information gathered, we could implement a weighted average scoring function to give a percentage of maliciousness of the files, hence giving a more nuanced response to the problem. Finally, we could improve the QCNNs themselves, in particular those trained on the more important sections.

In conclusion, we proposed in this work a multi-QCNN model in order to classify malware and benign files. We identified the different sections of these files, and transformed each section into an image thanks to Grayscale method. Then, we selected five specific sections, and trained a QCNN on each section. We restricted the size of the QCNNs to 8 qubits, to remain consistent with our approach aiming to think how to make better use of the information contained in the data rather than increase the capacities of our models. Once the training done, we gathered the five QCNNs outputs into vectors, that we studied using a scoring function in order to identify the type of each file. Our results show the efficiency of this model compared to a QCNN classifying images of entire files, namely it drastically increased the accuracy and F1 score for our detection task. Besides, the structure of this model makes it easy to modify, adapt, and allows to get more information about the data, increasing our understanding about malware PE files.

References

1. Edward Raff and Charles Nicholas. A Survey of Machine Learning Methods and Challenges for Windows Malware Classification. 2020.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29

2. Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers and Security*, 81:123–147, 2019.
3. Hyrum Anderson and Phil Roth. Ember: An open dataset for training static pe malware machine learning models. 04 2018.
4. L Nataraj, S Karthikeyan, G Jacob, and B S Manjunath. Malware images: Visualization and automatic classification. In *ACM International Conference Proceeding Series*, 2011.
5. M. Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J. Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2(9):567–576, sep 2022.
6. Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, jun 2021.
7. Martin Larocca, Nathan Ju, Diego García-Martín, Patrick J. Coles, and M. Cerezo. Theory of overparametrization in quantum neural networks, 2021.
8. Arthur Pesah, M. Cerezo, Samson Wang, Tyler Volkoff, Andrew T. Sornborger, and Patrick J. Coles. Absence of barren plateaus in quantum convolutional neural networks. *Physical Review X*, 11(4), oct 2021.
9. Iris Cong, Soonwon Choi, and Mikhail D Lukin. Quantum convolutional neural networks. *Nature Physics*, 15(12):1273–1278, 2019.
10. Tak Hur, Leeseok Kim, and Daniel K. Park. Quantum convolutional neural network for classical data classification. *Quantum Machine Intelligence*, 4(1), feb 2022.
11. Chukwudubem Umeano, Annie E. Paine, Vincent E. Elfving, and Oleksandr Kyrienko. What can we learn from quantum convolutional neural networks?, 2023.
12. Grégoire Barrué and Tony Quertier. Quantum machine learning for malware classification. *arXiv preprint arXiv:2305.09674*, 2023.
13. Benjamin Marais, Tony Quertier, and Stéphane Morucci. Ai-based malware and ransomware detection models. In *Conference on Artificial Intelligence for Defense*, 2022.
14. Thomas Hoffmann and Douglas Brown. Gradient estimation with constant scaling for hybrid quantum machine learning, 2022.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63778-0_29