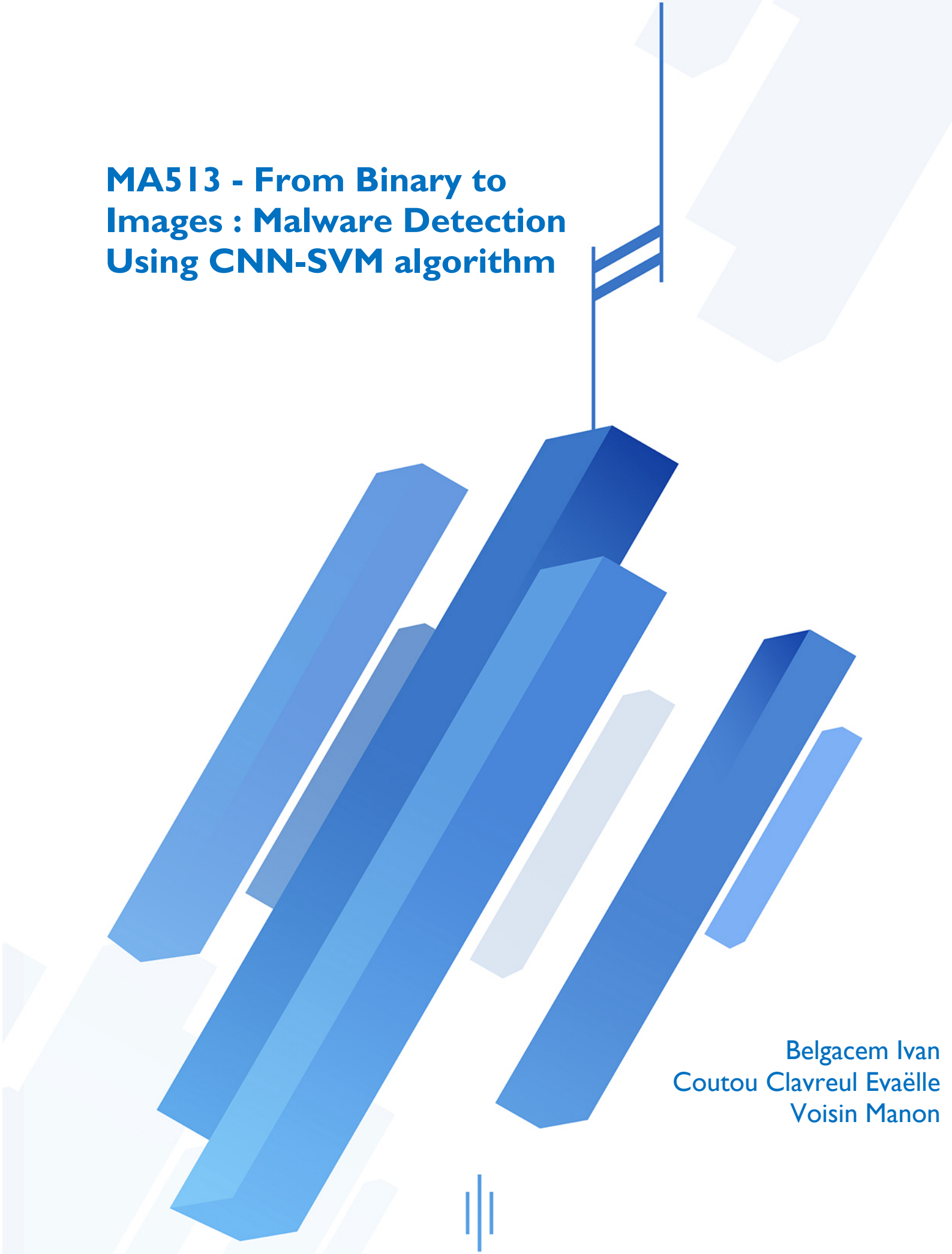
MA513 - From Binary to Images : Malware Detection Using CNN-SVM algorithm



Belgacem Ivan
Coutou Clavreul Evaëlle
Voisin Manon

Summary

Abstract

Cybersecurity plays an important role in this increasingly connected world. From big companies to personal computer, each download from the internet is a threat. To protect computers from unwanted malwares, some of the method used is having a database of known malware, so, whenever a file is downloaded, the hash of the malware is compared and can be effectively quarantined if known. But, to work, the malware must be known. Indeed, cybercriminals are creating more and more sophisticated malwares that are undetected. The goal of this work is to use Deep Learning (DL) to detect and classify malwares. It can enable the safe detection of newly created malware. The mathematical model will link the malware **x** and find it's corresponding malware family y

$$f : x \longrightarrow y$$

x being the image malware and y the family of the malware

To train our model, we will be using images_malaware.npz dataset. This dataset consists of malware images corresponding to 25 families. To obtain images of malware, we must convert it to 8 bit vector and to grayscale which will correspond to an image. The algorithm will consist of two parts, the Deep Learning algorithm using Convolutional Layers that will detect patterns in each image, and the classification algorithm Support Vector Machine (SVM). Our model has an accuracy of 90%.

1. Introduction

In the ever-evolving landscape of cybersecurity, combating the proliferation of malware remains a pressing challenge. Traditional methods relying on signature-based detection struggle to keep pace with the exponential growth of new malware variants, leading to undetected threats lurking within systems. Addressing this gap requires innovative approaches that transcend the limitations of conventional detection mechanisms.

This work delves into a paradigm shift in malware analysis and classification, presenting a novel methodology to characterize and combat malware threats. Departing from the traditional reliance on binary signatures or the complexities of static and dynamic code analysis, this research introduces an unconventional approach. By harnessing the transformative power of deep learning (DL) and support vector machine (SVM) techniques, we aim to revolutionize malware detection and classification.

The focal point of this study revolves around visualizing malware binaries as images, leveraging the inherent similarities in their structural patterns. By representing malware executables as binary strings, reshaping them into matrices, and subsequently viewing them as images, we uncover striking visual correlations among malware belonging to the same family. This observation underscores the prevalence of code reuse in generating new malware variants and forms the foundation of our innovative classification methodology.

Using the images_malware.npz dataset, which includes visualized representations of malware binaries, we employ DL-SVM models trained on this data. This amalgamation of DL techniques with the discriminative power of SVM classification, not only enables the identification of known malware, but also, exhibits a remarkable capacity to detect and classify newly released and previously unseen threats. This adaptability arises from the models' ability to generalize from data, offering a robust defense mechanism against evolving and emerging malware.

Throughout this exploration, we will dive into the method of our approach, detailing the process of visualizing malware binaries as images, training DL-SVM models, and assessing their performance in automatically classifying malware families. Furthermore, we critically examine the limitations of this methodology and conclude with insights into the implications and future potential of this intelligent anti-malware system.

2. Methodology proposed for the algorithm

2.1 Libraries used in this work

During this work, we will be using Python programming language. Our Machine learning library will be Scikit – Learn and Tensorflow will be our Deep Learning Library. Alongside these libraries, we will be using Matplotlib to plot effectively graphs of our results, and Numpy to work with image vectors.

2.2 Images_malware.npz Dataset

Images_malware dataset contains 25 families of known malware and their corresponding images. The total size of the dataset is 9,339 images.

The distribution of the dataset is as followed:

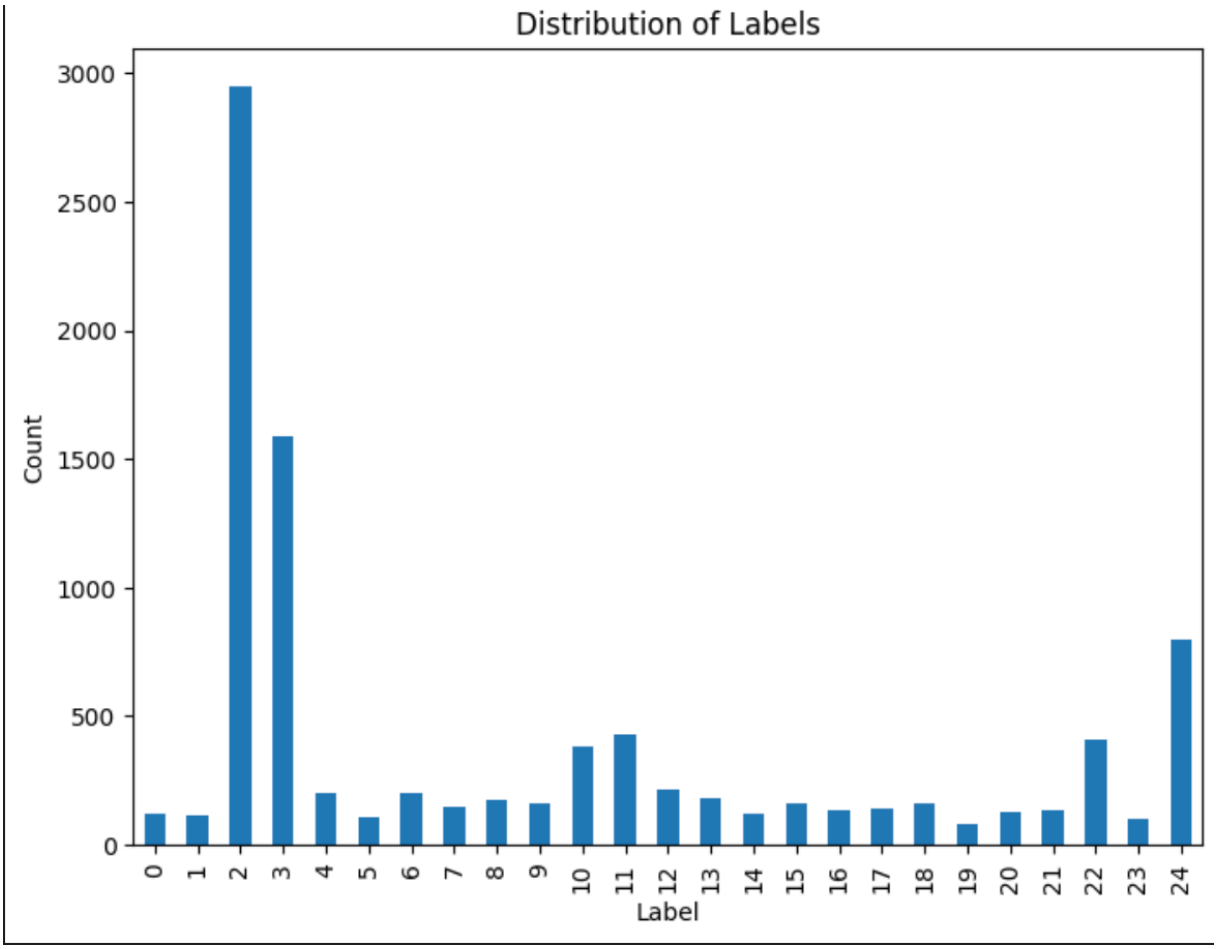


Figure 1 - Distribution of malware families in the dataset

No	Family Name
1	Adialer.C
2	Agent.FYI
3	Allaple.A
4	Allaple.L
5	Alueron.gen!J
6	Autorun.K
7	C2Lop.P
8	C2Lop.gen!G
9	Dialplatform.B
10	Dontovo.A
11	Fakerean
12	Instantaccess
13	Lolyda.AA 1
14	Lolyda.AA 2
15	Lolyda.AA 3
16	Lolyda.AT
17	Malex.gen!J
18	Obfuscator.AD
19	Rbot!gen
20	Skintrim.N
21	Swizzor.gen!E
22	Swizzor.gen!I
23	VB.AT
24	Wintrim.BX
25	Yuner.A

Table showing the family of malware with their associated number.

2.3 Converting array to image

Our dataset contains array representing images. The array contains two values, the type of the malware and its image. We extract both values as two separates, one for the label (the family) and one for the image. The malware was first binary then converted to 8 bit vector. Our next goal is to convert it to greyscale, since colour is not important, and then normalize it, retaining the normalized 32x32x1(greyscale) form. We then use scikit-learn's MinMaxScaler() function to normalize it and obtain a vector containing values between 0 and 1, as it is easier for the algorithm to work with these values. Finally, we obtain our greyscale malware image.

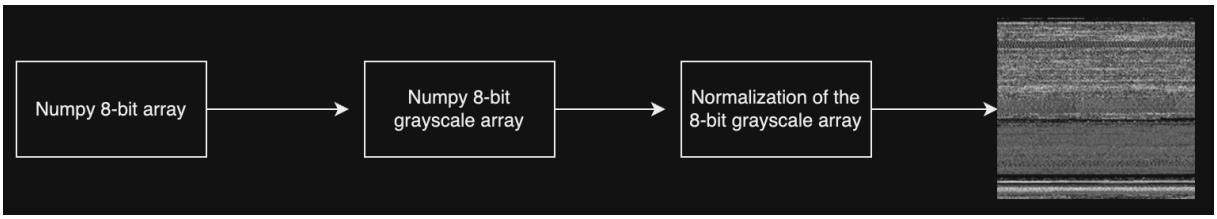


Figure 2 - Converting array to image

The processing of the total arrays takes 4 seconds for our 9,339 values.

2.4 The Deep Learning model to detect image pattern: Convolutional Layers

Convolutional Layers, an integral component of Convolutional Neural Networks (CNNs), stand as a cornerstone in image recognition methodologies. These layers employ filters—compact grids typically 3x3 or larger—imbued with weighted values. Functioning akin to specialized lenses, these filters systematically inspect localized segments of an image, discerning intricate patterns such as edges, textures, or distinctive shapes.

Through a sliding mechanism across the image, these filters generate what is known as a feature map. Conceptually analogous to a cartographic representation highlighting salient geographical features, these feature maps serve to unveil pivotal patterns characteristic of various families of malware, thereby forming a fundamental aspect of our investigative endeavor.

Each convolutional step encompasses an elemental mathematical operation: an element-wise computation, facilitating the identification and emphasis of pertinent features while dampening others. Additionally, the incorporation of an activation function introduces nonlinearities essential for the model to discern increasingly complex patterns within the analyzed images.

The critical significance of Convolutional Layers lies in their exceptional aptitude to discern and isolate specific patterns within images. This unique capability endows our detection framework with a nuanced and clever way to identify and classify malware, marking a significant departure from traditional methodologies.

In summary, the Convolutional Layers within CNNs epitomize a sophisticated toolset adept at discerning intricate visual cues. Their pivotal role in our approach underscores their efficacy in unveiling latent patterns essential for the intelligent detection of malware in image-based contexts.
The network used will be the one used by M. Agarap[2].

It consists of the following layers:

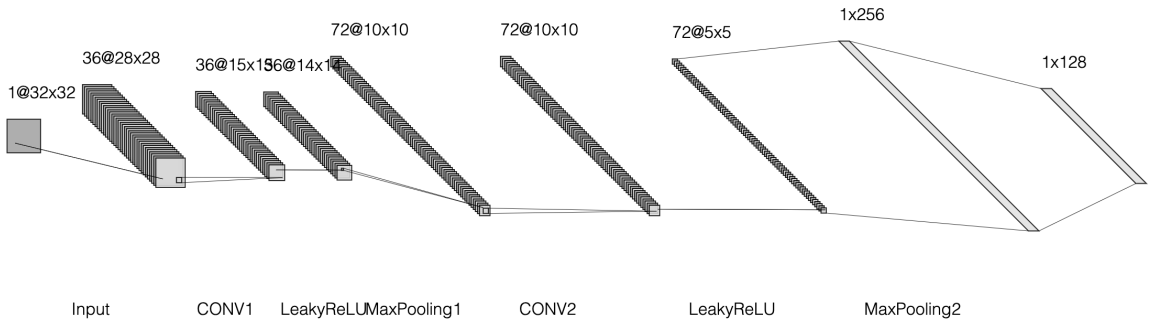


Figure 3 - CNN Layers

Input Layer:

Dimensions: 32×32×1
Convolutional Layer (CONV5): Filter Size: 5 × 5 Number of Filters: 36 Stride: 1
Activation Function: Leaky ReLU
Pooling Layer (POOL): Pool Size: 2 × 2 Stride: 1
Convolutional Layer (CONV5): Filter Size: 5 × 5 Number of Filters: 72 Stride: 1
Activation Function: Leaky ReLU
Pooling Layer (POOL): Pool Size: 2 × 2 Stride: 1
Fully Connected Layer (FC): Number of Neurons: 1024
Activation Function: Leaky ReLU
Dropout Layer: Dropout Probability: 0.85

Output Layer (FC):

Number of Neurons: 25 (representing output classes)

LeakyRelu was used instead of ReLU alone. Indeed, Leaky ReLU's strength lies in preventing neuron death by introducing a small gradient for negative inputs, ensuring continuous learning. Its advantage is that it alleviates the vanishing gradient problem, aiding better training in deeper neural networks compared to traditional ReLU.
Dropout with 0.85 means during training, 85% of the neurons in the previous layer are randomly ignored or "dropped out" in each training step, which helps prevent overfitting by encouraging the network to learn more robust features.
This CNN algorithm will help us find patterns in our malware images, detecting efficiently the specificities of each family of malware and thus being able to transmit the information to our classification algorithm: Support Vector Machine.

2.5 Classification Algorithm: Support Vector Machine

Support Vector Machine (SVM) is a powerful supervised machine learning algorithm used for classification tasks. It works by finding the best possible line or boundary (hyperplane) that separates different classes of data.

SVM aims to maximize the margin, which is the distance between the boundary and the closest data points (called support vectors). By finding this optimal boundary, SVM achieves effective classification by maximizing the separation between different classes.

When the data is not easily separable in its current space, SVM can use a technique called the kernel trick to map the data into a higher-dimensional space, making it easier to find a separating hyperplane. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

SVM is robust, handles high-dimensional data well, and is effective in creating complex decision boundaries, making it widely used in various classification tasks across different domains.

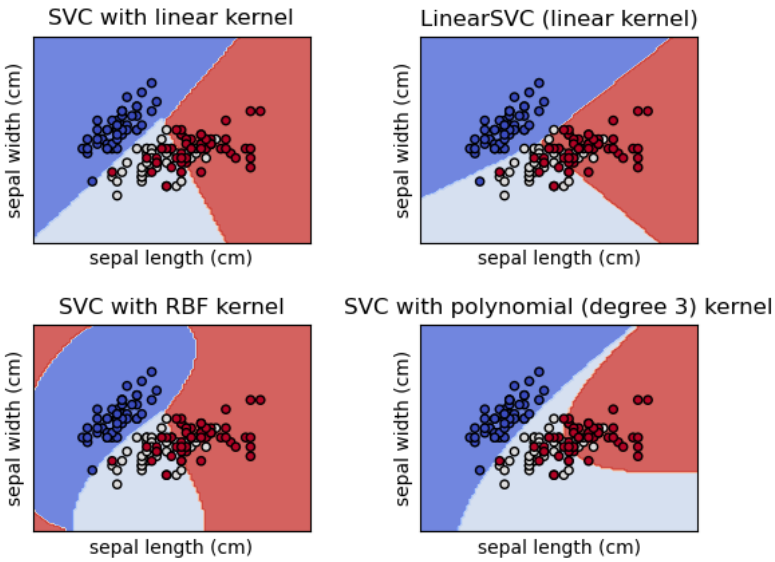


Figure 4 - SVM Classifier classification method

For this classification, we will use specific hyperparameters for the SVM classifier:

Kernel	linear
C	10
Random_state	42

The C parameter is a regularization parameter, it controls the trade-off between maximizing the margin and minimizing the classification error. In other terms, it will affect how much the

SVM will avoid misclassifying each training examples. The higher the C is, the less tolerance we give to our algorithm to misclassify, lowering the margin between each hyperplane done by the algorithm.

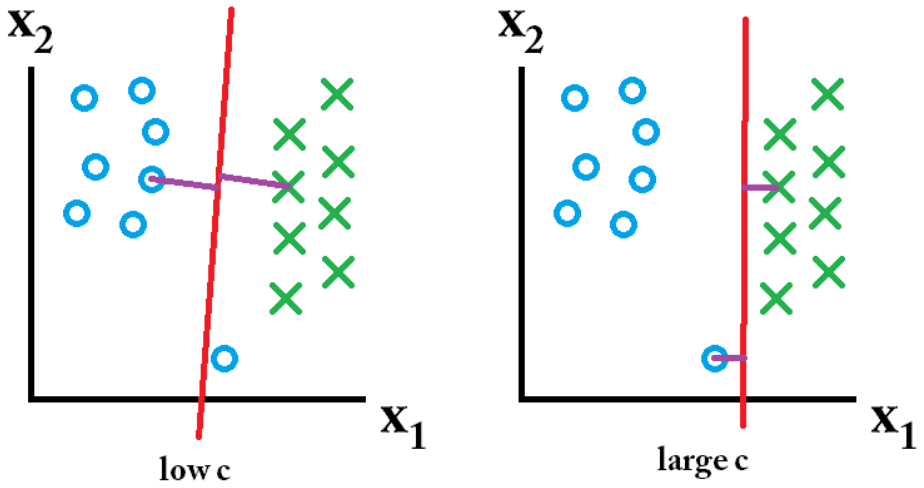


Figure 5 - Impact of C on SVM

2.6 Analysis of the CNN-SVM algorithm

We conducted this study in two parts. The first part was the training part that contains 80% of the data and the testing part that contains 20% of the data. Let's assess the effectiveness of this algorithm.

The deep learning algorithm learning curve shows if the algorithm overfitted or underfitted. Overfits informs us if the convolutional layers where too much trained and cannot effectively classify the training data. On the other part, underfitting means that the algorithm didn't find enough pattern to classify the data, which means more data is needed or more tunning on the algorithm. We want to avoid both.
The experiments were conducted on Collab PRO using a T4 GPU with 50 GB RAM.

Hyperparameters	CNN
Batch Size	256
Hidden layers	2
Epochs	100
Dropout Rate	0.85
Learning Rate	0.0001

The learning curve shows a correct curve. We can see that the validation is slightly below training which means that the convolutional layers slightly struggle on the testing phase which can be because the algorithm trained too much. We find that changing the learning rate didn't improve the learning curve. After much consideration and testing, this difference in training and validation did not impact the classification algorithm.

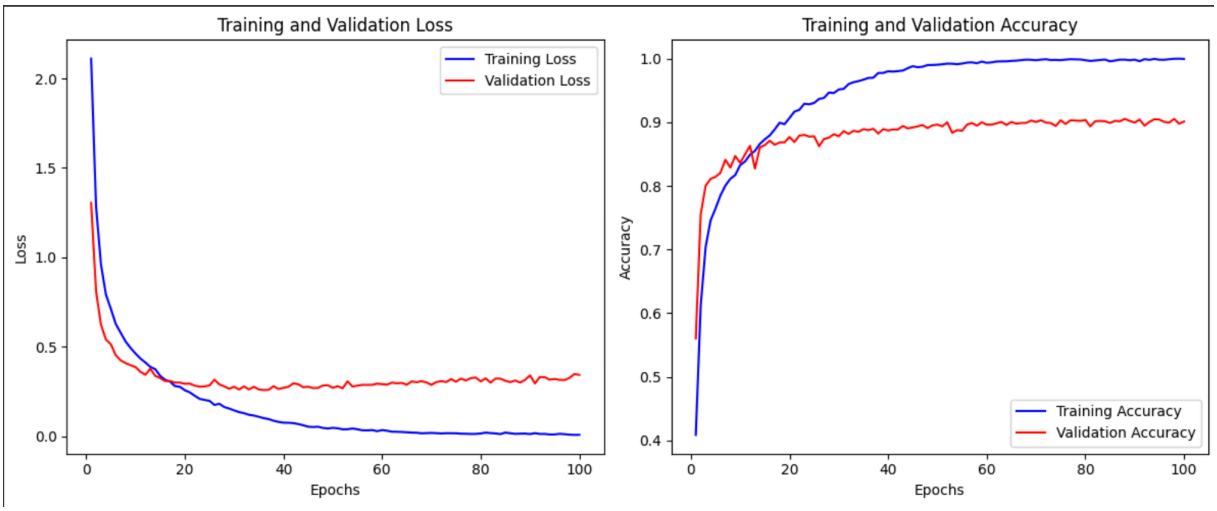


Figure 6 - Learning Curve

With 20% of the data being allocated to validation, we could have used 30% for validation and would have gotten a different learning curve. The result is still good as it means that I can detect malware images 90% of the time. We should aim to get more data to have an even better algorithm.

The SVM algorithm is as followed.

Hyperparameters	SVM
C	10
Kernel	Linear

The reports show:

Classification Report:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	36
1	1.00	1.00	1.00	19
2	0.88	0.86	0.87	592
3	0.76	0.84	0.80	310
4	1.00	1.00	1.00	45
5	1.00	1.00	1.00	18
6	0.63	0.76	0.69	34
7	0.76	0.48	0.59	27
8	1.00	1.00	1.00	49
9	1.00	1.00	1.00	34
10	1.00	1.00	1.00	69
11	1.00	1.00	1.00	89
12	1.00	0.97	0.99	37
13	1.00	1.00	1.00	41
14	1.00	1.00	1.00	23
15	1.00	1.00	1.00	27
16	0.95	0.67	0.78	27

17	1.00	1.00	1.00	25
18	0.91	1.00	0.96	32
19	1.00	1.00	1.00	13
20	0.36	0.47	0.41	19
21	0.57	0.28	0.37	29
22	1.00	0.99	0.99	84
23	1.00	0.95	0.97	19
24	1.00	1.00	1.00	170

Accuracy		0.90	1868	
Macro avg	0.91	0.89	0.90	1868
Weighted avg	0.90	0.90	0.89	1868

The test accuracy of the algorithm is 90%, which is an excellent result.

The confusion matrix gives us the information of where the algorithm has failed. It helps us to understand which malware families are more difficult to classify.

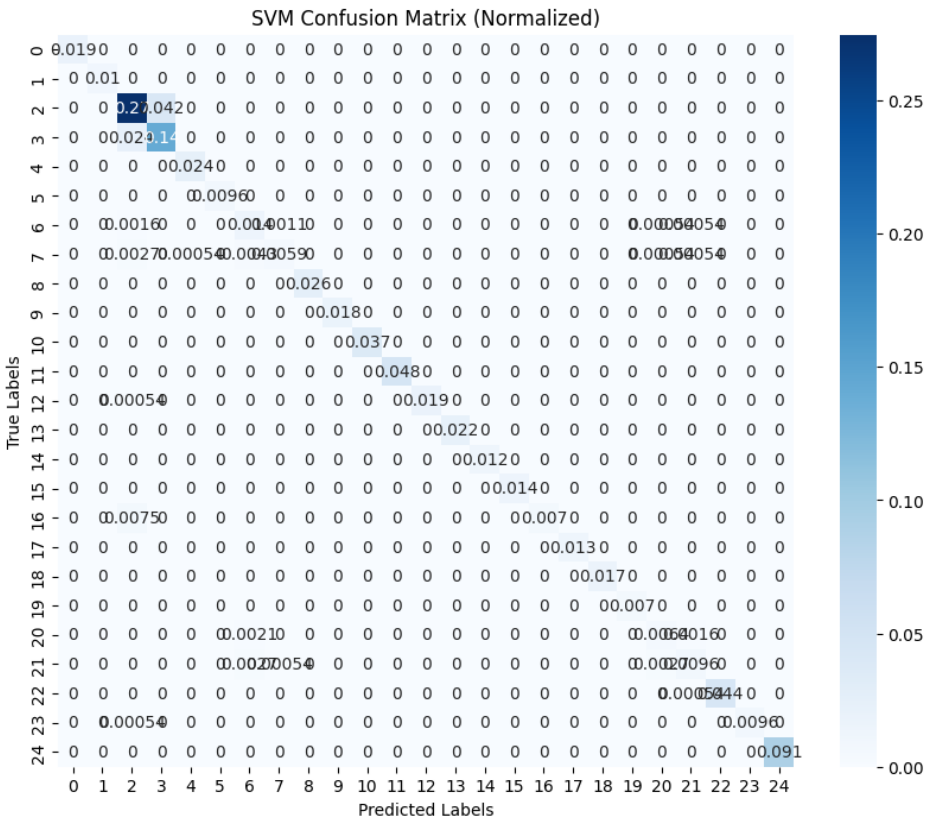


Figure 7 - Confusion Matrix

We see that family 21 and 22 but also 1,2 and 6 caused some misclassification. This can be due to low data on these families, but also because they're harder to classify due to their hard pattern.

3. Conclusion

In conclusion, our innovative approach combining a convolutional layer with a support vector machine classifier presents a promising method for detecting malware without sole reliance on databases. The conversion of malware binaries into grayscale images, followed by deep learning analysis, yielded a commendable 90% classification accuracy, demonstrating a low false-detection rate. However, challenges in handling sparse data and complex models led to occasional false detections, an issue we've actively addressed using a confusion matrix.

While our algorithm demonstrated efficiency by predicting on 15 images within 3 seconds, the scalability to millions of files remains a concern. It's evident that further research is vital to optimize algorithms for wider-scale applications, especially for integration into anti-malware systems like Malwarebytes.

It's important to note that our study was focused on the CNN-SVM algorithm. Future investigations could explore alternative algorithms or hybrid models to enhance malware classification capabilities. This pioneering work opens doors for more robust and efficient malware detection methods, promising advancements in the cybersecurity landscape.

References

[1]2017. Deep MNIST for Experts. (Nov 2017). https://www.tensorflow.org/get_started/mnist/pros

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, San- jay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Leven- berg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015).

[3] Abien Fred Agarap. 2017. A Neural Network Architecture Combining Gated Recurrent Unit (GRU) and Support Vector Machine (SVM) for Intrusion Detection in Network Traffic Data. arXiv preprint arXiv:1709.03082 (2017).

[4]Vinayakumar Ravi. Binary image to malware. Binary image to malware

[5] Dipendra Pant,Rabindra Bista. 2021. Image-based Malware Classification using Deep Convolutional Neural Network and Transfer Learning.