SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE

Malware Detection with malware images using CNN techinques

AI LAB: COMPUTER VISION AND NLP

Professors:
Daniele Pannone

Students:
Edoardo Allegrini
Samuele Bella

Academic Year 2022/2023

Contents

1 Introduction

According to [1] malware detection is a crucial factor in cybersecurity. Not only organizations but whoever uses a technological appliance should strive to detect and validate malware incidents rapidly to minimize the damage. Our goal is to develop a Malware Detection System (MDS) able to provide a significant contribution to defense against malicious attacks. Instead of traditional MDS that uses feature extractions ML algorithms, we thought that it could be interesting and challenging to develop a Convolutional Neural Network (CNN) trained to classify an executable file as malware or benign just analyzing the corresponding image 2.2. Traditionally, most MDS are based on feature vectors, which contain essential characteristics of malware. The two main categories of MDS are static and dynamic. In static analysis, malicious software is analyzed without executing it. The detection patterns used in static analysis include string signature, byte-sequence n-grams and syntactic library call. Instead, in dynamic analysis, malicious software is analyzed while being executed in a controlled environment (e.g., virtual machine, simulator, emulator, or sandbox). Before executing the malware samples, appropriate monitoring tools like Process Monitor or Capture BAT are installed and activated; thus to what have just been reported dynamic analysis is time consuming and computationally intensive.

2 Development

2.1 Approach

The project includes a training phase in which the CNN learns to detect malware and benign executables by their image representation. The image approach is a focal point of this project, in fact analyzing a potential virus executable is not simple. Converting the .exe in image prevents the time and resource consumption taken by the analysis of its behavior running in a Sandbox, which is one of the phases of traditional malware detection. Our CNN is trained to learn the features/textures of the image and associate them with the corresponding labels.

2.2 Executable conversion to image

To convert an executable to an image we developed a function called "exe_to_png" which takes in input the path of a file α to transform. It starts by instantiating a numpy array of shape [256, 256], then loops through every byte of the file α and converts that byte into a pixel using big-endian approach. At the end of the computation an 256x256 image is returned and saved; if the executable is smaller than the dimensions 256x256 the image is padded with zeros, conversely if the bytes of α exceeds then the exceeding part is discarded. We decided to discard the exceeding bytes because we found out that the loss of information is not comparable to the needed time that needs an eventual CNN trained by very big images of different sizes.

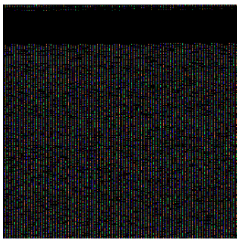


Figure 1: Conversion malware to image

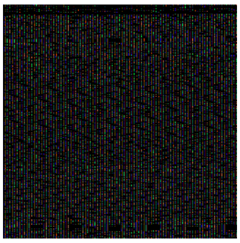


Figure 2: Conversion benign_file to image

2.3 Building the CNN

The CNN has two convolutional layers with a kernel of size (5, 5), every convolutional layer is followed by the non-linear layer Rectified Linear Unit (ReLU) combined with a pooling layer. The details about our CNN's structure are:

```
self.cnn = nn.Sequential(
    nn.Conv2d(3, 50, (5, 5)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),
    nn.Conv2d(50, 60, (5, 5)),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2)),
)
```

At the end of the CNN layers comes into play a multilayer perceptron (MLP) designed as:

```
self.mlp = nn.Sequential(
    nn.Linear(223260, 64),
    nn.ReLU(),
    nn.Linear(64, 2),
    # nn.LogSoftmax(1)
)
```

The last line of MLP is commented because the performace/accuracy tests reported that the CNN performs better without the function `LogSoftmax`, that we initially thought could be useful.

2.4 Training

The batch size of the dataset 3.1 for training the CNN is 64, that phase is split into 3 epochs with a learning rate of 0.001. The function `nn.CrossEntropyLoss()` is used to compute the loss and as optimizer we used the stochastic gradient descent. Executing the test function during the model training we get:

- Epoch: 0 Accuracy: 68.08807134894091
- Epoch: 1 Accuracy: 81.82831661092531
- Epoch: 2 Accuracy: 85.47937569676701

2.5 Testing

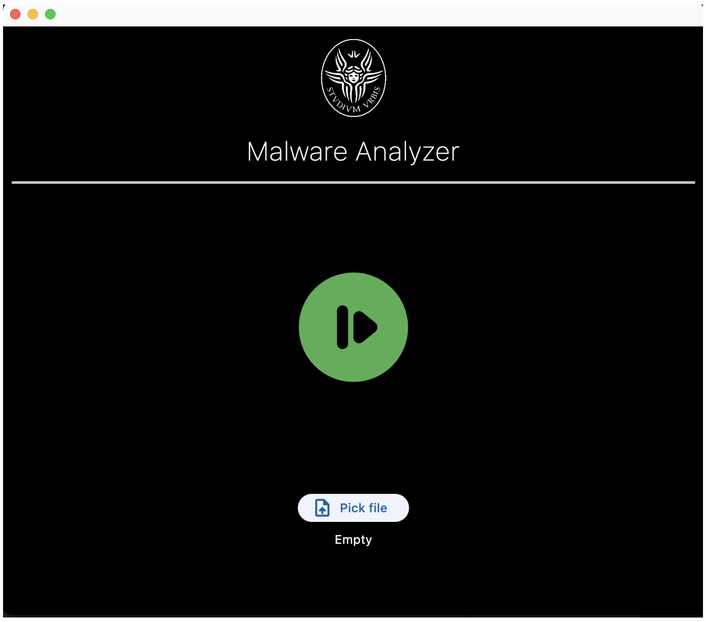
As same as 2.4, in testing phase, the number of epochs is 3 and for each prediction done by the model during testing, if the prediction is correct, the algorithm rises the correct counter for computing the accuracy at the end. Executing the test function on the model pre-trained we get:

- Epoch: 0 Accuracy: 85.47937569676701
- Epoch: 1 Accuracy: 85.47937569676701
- Epoch: 2 Accuracy: 85.47937569676701

2.6 GUI window

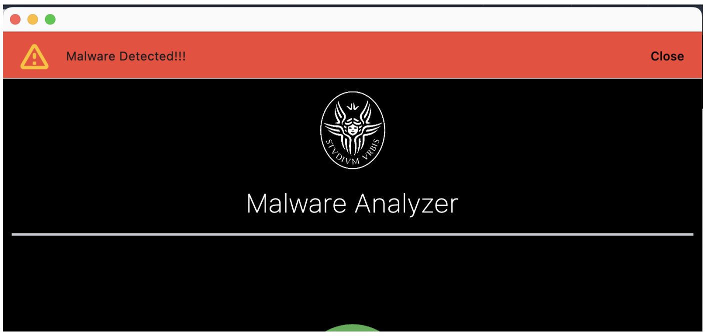
We designed and implemented a GUI for simplifying the interaction between the user and the CNN model. The GUI lets users upload a file and analyze it by clicking the run button 3.

Figure 3: GUI



When the detection is finished and the CNN computed if the file selected is a malware or not the result is shown by a popup 4.

Figure 4: GUI detection result


Malware Detected!!!

3 Conclusions

3.1 Dataset

We created the dataset used for this project by collecting executables of malware and benign files and processing the relative images using 2.2. The dataset is accessible on our kaggle. The dataset is structured as follows:

```
data
 ├─train This directory holds
 │  │     data for training.
 │  ├─malware 7176 files.
 │  └─benign 7176 files.
 └─test This directory holds
    │    data for testing.
    ├─malware 1794 files.
    └─benign 1794 files.
```

3.2 Scalability

In this section, we quantify the scalability of the proposed framework. The malware metadata required for classification can be designed ad hoc for training the model to recognize specific malware classes. We conducted an offline experiment to estimate the classification speed. Our classifier was implemented in Python with pytorch and run on a machine with an M1 CPU and 8GB RAM. It takes an average of $(7,531 \pm 20) \times 10^{-2}$ seconds to evaluate each malware.

3.3 Results

To conclude, we created a Convolutional Neural Network that was trained to classify malware images. Images are generated from 7176 benign software and 7176 malware. Then, by using our CNN, we detect malware by extracting features from the generated images.
From our testing, we are able to assure an accuracy of 86%.

References

[1] Karen Scarfone (Scarfone Cybersecurity) Murugiah Souppaya (NIST). Guide to malware incident prevention and handling for desktops and laptops. Technical Report NIST Special Publication (SP) 800-83, Rev. 1, National Institute of Standards and Technology, Gaithersburg, MD, 2013.