FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Malware Detection Using Visualization Techniques
Student:	Bc. Ihor Salov
Supervisor:	Mgr. Olha Jurečková
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2023/2024

Instructions

Today, there are many different methods for analyzing and detecting malware. Deep learning and image processing have achieved a good effect on malware classification. A technique introduced in 2008 suggested that malware binaries can be converted into grayscale or RGB (Red, Green, Blue) color images. This representation can be used to identify each type of malware specifically. Different types of malware of the same family have been observed to have identical image structures when converted from binary. Models using deep learning convolutional neural networks (CNN) can embrace images as input simply.

Instructions:
1. Study the existing approaches of malware detection procedures/techniques and review previous research on malware detection using visualization.
2. Implement or use an existing library of current analysis techniques of such visualizations that utilize machine learning and neural networks.
3. Choose an appropriate publicly available dataset and train a neural network able to process the created images and classify them as malicious or benign.
4. Discuss the results and compare them with other malware detection techniques.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 26 January 2023 in Prague.

Master’s thesis

MALWARE DETECTION USING VISUALIZATION TECHNIQUES

Bc. Ihor Salov

Faculty of Information Technology
Department of Information Security
Supervisor: Mgr. Olha Jurečková
January 11, 2024

Contents

List of Figures

List of Tables

List of code listings

Declaration

In Prague on January 11, 2024 ....

Abstract

Recently machine learning (ML) techniques become very popular in many areas, including natural language processing, voice/image recognition, etc. The goal of this master thesis is to create a technique of malware detection by using image visualization. As a method of gaining information from converted malware samples classification using Convolutional Neural Network (CNN) detection was chosen. The malware sample represented as byte array is first converted to gray-scale or RGB image and then fed as input to CNN.

Keywords Malware detection, Malware visualisation, Convolutional Neural Network

Abstrakt

Techniky strojového učení (ML) jsou v poslední době velmi populární v mnoha oblastech, včetně zpracování přirozeného jazyka, rozpoznávání hlasu/obrazu atd. Cílem této diplomové práce je vytvořit techniku detekce malwaru pomocí vizualizace obrazu. Jako způsob získávání informací z převedených vzorků malwaru byla zvolena klasifikace pomocí detekce konvoluční neuronové sítě (CNN). Vzorek malwaru reprezentovaný jako bajtové pole je nejprve převeden na obrázek ve stupních šedi nebo RGB a poté přiveden jako vstup do CNN.

Klíčová slova Detekce malwaru, Vizualizace malwaru, Konvoluční neuronová síť

Abbreviations

ML	Machine Learning
CNN	Convolutional Neural Network
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
CMYK	Cyan Magenta Yellow Black
SHA	Secure Hash Algorithm
FFNN	Feed Forward Neural Network
PE	Portable Executable

Introduction

Virus detection has been a big problem for decades. Humanity has come up with many different ways to detect malware, including signature detection and behavioral analysis. However, virus developers and hackers are coming up with increasingly sophisticated techniques to bypass existing detection methods. But progress does not stand still. Since the invention of neural networks, they have increasingly entered our lives and appear in various areas of our lives. Convolutional neural networks have performed well in image recognition and object detection. This is how the idea arose to combine the problem of malware detection and convolutional neural networks. Convolutional neural networks work well for grid-like structures. For example, an image can be represented as a tensor made up of pixels, where two dimensions are width and height, and the third is color depth. So, if we can turn a malicious file into an image, we can use it as an input to a convolutional neural network. By training it on a sufficient number of samples, it is possible to achieve high accuracy. This is what this thesis will be about.

I apologize, this work was not completed due to personal circumstances. I'll continue to work on it.

Chapter 1

Malware detection

Chapter 2

Malware visualization

This chapter covers the theory about transformation of the PE executable file format into an image, which can be later used for model training and malware detection.

2.1 Image formats

Before conversion itself we need to find out, how image data are stored inside of the images. The image height and width are measured in pixels. Each pixel has it's own color. But as we know, all our data are stored in binary representation, so there is no definition of color we used to. In computer each pixel can be represented by single (e.g gray-scale image) or multiple channels (e.g. RGB, RGBA, CMYK). Each channel has it's own value (from 0 to 255 or from 0x00 to 0xFF hex) and by combining them we can get the resulting value. Below I give examples of several formats.

Gray-scale - image format with only one channel, which is presented by one byte. It has color range from 0 (black) to 255 (white).

RGB - image format with three channels: red, green and blue. Each channel has range from 0 to 255, which in total gives us 255 * 255 * 255 = 16581375 values that we can get for each pixel.

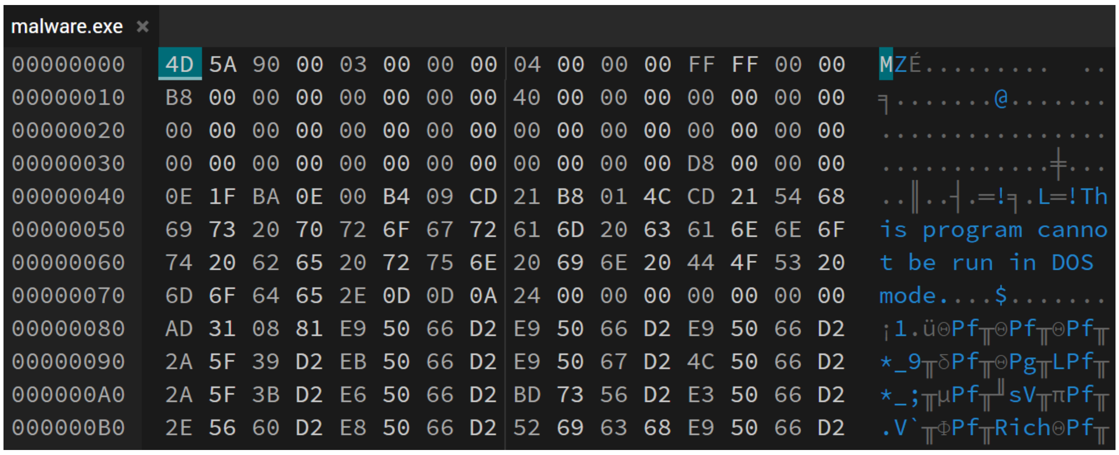
RGBA - similar to RGB, but adds one more channel, which is responsible for transparency. Value 0 corresponds to fully transparent pixel and 255 corresponds to fully opaque pixel.

2.2 Malware-image representation

In this thesis I will work with malware samples in the Portable Executable (PE) format. This is a file format for executables, object code, DLLs and others used in 32-bit and 64-bit versions of Windows operating systems. In general PE file structure is rather complicated, but for our purposes it will be enough to treat it as normal binary file - array of bytes.

This approach will allow us to represent malware in different image formats described in 2.1: if we consider, that each malware byte is one pixel we will get it's gray-scale representation. Similarly if we threat each three bytes as one pixel - we will get RGB representation. This approach was used in [1] and [2]

The following script is used to create such a conversion. It accepts 2 parameters: name of the file to convert and image format/mode. Currently only gray-scale (L) and RGB modes are supported. As an output script produces image representation of the input file of size 512x512 pixels. Then these images will be used to train our neural network model.


malware.exe

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZÉ.........  ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ╕.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 D8 00 00 00  ............╪...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..║..┤.=!╕.L=!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  AD 31 08 81 E9 50 66 D2 E9 50 66 D2 E9 50 66 D2  ¡1.üΘPf╥ΘPf╥ΘPf╥
00000090  2A 5F 39 D2 EB 50 66 D2 E9 50 67 D2 4C 50 66 D2  *_9╥δPf╥ΘPg╥LPf╥
000000A0  2A 5F 3B D2 E6 50 66 D2 BD 73 56 D2 E3 50 66 D2  *_;╥µPf╥╜sV╥πPf╥
000000B0  2E 56 60 D2 E8 50 66 D2 52 69 63 68 E9 50 66 D2  .V`╥ΦPf╥RichΘPf╥
```

Figure 2.1 Malware byte representation

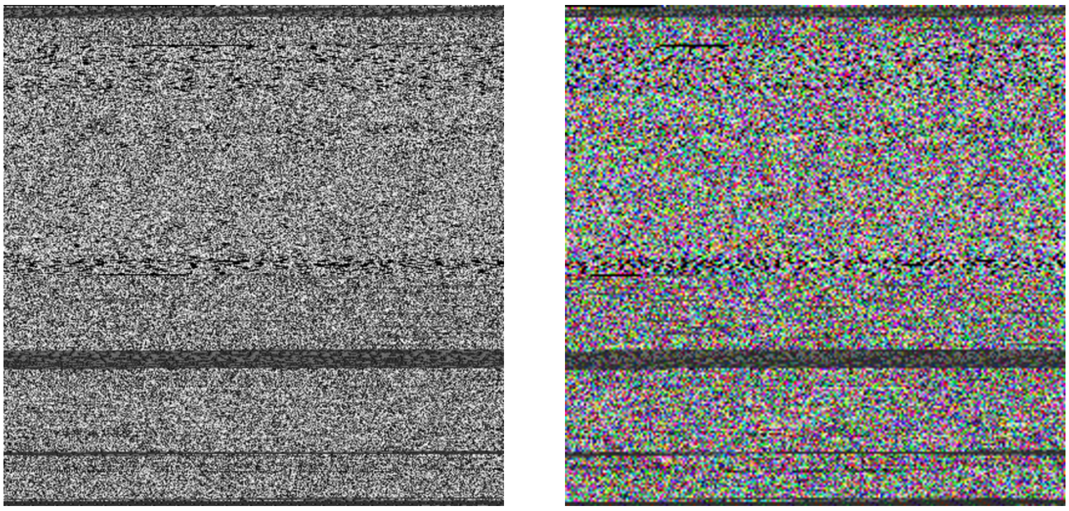

(a) Gray-scale representation of the malware.exe **(b)** RGB representation of the malware.exe

Figure 2.2 Malware image representation

Code listing 2.1 bin2img.py source code

```
import numpy as np
import sys
import math
from PIL import Image

def calculate_width(file_size):
    return int(math.sqrt(file_size))

def calculate_channels(mode):
    if (mode == "L"):
        return 1
    elif (mode == "RGB"):
        return 3
    else:
        raise Exception("Unsupported␣mode")

def convert(data, mode):
    file_size = data.shape[0]
    channels = calculate_channels(mode)
    data = np.pad(data, (channels - file_size % channels, 0))
    img_width = calculate_width(file_size // channels)

    result = []
    for i in range(0, channels):
        ch = data[i::channels]
        ch = np.pad(ch, (img_width - ch.shape[0] % img_width, 0))
        ch = np.reshape(ch, (-1, img_width))
        result.append(ch)
    result = np.stack(result, axis=2)

    if (channels == 1):
        result = result[:, :, 0]
    return Image.fromarray(result, mode)

if len(sys.argv) != 3:
    print(f"Usage:␣python3␣{sys.argv[0]}␣file_name␣RGB/L")
    exit()

filename = sys.argv[1]
mode = sys.argv[2]
with open(filename, "rb") as f:
    data = f.read()

data = np.frombuffer(data, dtype=np.uint8)
image = convert(data, mode)
image = image.resize((512,512))
image.save(f"{mode}_{filename}.png")
```

Chapter 3

Convolutional Neural Networks

With the development of deep leaning technologies Convolutional Neural Networks (also known as CNN) have become very popular. They have proved their efficiency in processing of grid-like data. That could be anything from time-series 1-D array representing samples taken at regular time intervals to applications where CNN is used for Natural Language Processing. But the most widespread use of this type of neural network is image processing. As any image can be represented as grid of pixels of some dimension, then CNN can be used to perform such operations as: image classification, object detection, image segmentation etc.

In this chapter I will describe what convolution is as well as other building blocks of CNN and mathematics behind of it.

3.1 Convolution

In convolutional neural network the state of each layer is represented by a spatial grid structure. For the first layer its values and shape are determined by the nature of input data. It is called the **input layer**. Parameters, that we want to obtain during training in CNN are organized into 3-dimensional structures also known as **kernels** or **filters**. The filter usually has the same width and height (is square) and its depth is always same as the depth of the layer, to which it is applied. By sequential applying filter to the input layer, we will get more and more layers. These later, newly obtained layers are referred to as **hidden layers**, and their grids are referred to as **feature maps** or **activation maps**. Here ”applying filter” means placing the filter at each possible position in the input (or hidden) layer, so that it stays fully within its borders and then performing a dot product between filter and the matching grid in the input (3.3). This operation is called **convolution**. More formally we can define convolution as follows:

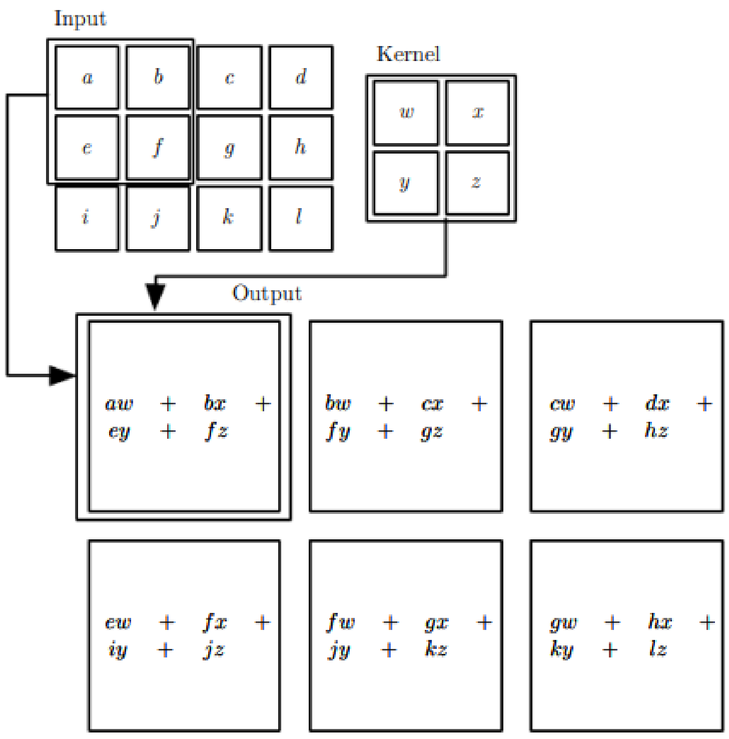


Figure 3.1 An example of a convolution between a 3 × 4 × 1 input and a 2 × 2 × 1 filter

▶ **Definition 3.1** (Convolution). *Let $W^{(p,q)} = [w^{(p,q)}_{i,j,k}]$ be the 3-dimensional tensor $F_q \times F_q \times d_q$, which defines parameters of pth filter of qth layer. Here indices i, j, k denote the positions along the height, width and depth of the filter. Also 3-dimensional tensor $H^{(q)} = [h^{(q)}_{i,j,k}]$ of size $L_q \times B_q \times d_q$ represents the feature maps in the qth layer. If the $q = 1$, H^1 simply represents the input layer. Then, convolutional operations from the qth layer to the $(q+1)$th layer are defined as follows:*

$$h^{(q+1)}_{i,j,p} = \sum_{r=1}^{F_q}\sum_{s=1}^{F_q}\sum_{k=1}^{d_q} w^{p,q}_{rsk} h^{(q)}_{i+r-1,j+s-1,l} \qquad \begin{aligned} \forall i &\in \{1,\dots,L_q - F_q + 1\} \\ \forall j &\in \{1,\dots,B_q - F_q + 1\} \\ \forall p &\in \{1,\dots,d_q + 1\} \end{aligned}$$

The result will be the size of $L_{q+1} \times B_{q+1}$, where $L_{q+1} = (L_q - F_q + 1)$ and $B_{q+1} = (B_q - F_q + 1)$. [3]

3.2 Padding

The operation described in previous section is usually referred as “valid convolution”, because filter is moving within the borders of the layer, without “sticking out”. Experiments show, that it generally does not work well. The number of positions in which filter captures outer pixels is less than such number for pixels that are closer to the center. This way contribution of the pixels around the border will be under-represented in the final result. To avoid this the original size of the layer can be extended by zeros on each side. This operation is called **padding**. The most used types of padding are: valid-padding, half-padding, and full-padding. [3]

Valid-padding - no zeros are added to the input (no padding).

Half-padding - type of padding, where $(F_q - 1)/2$ zeros are padded on all sides of the image. The main property of half-padding is that it preserves the size after convolution.

Full-padding - type of padding, where $(F_q - 1)$ zeros are padded on all sides of the image.

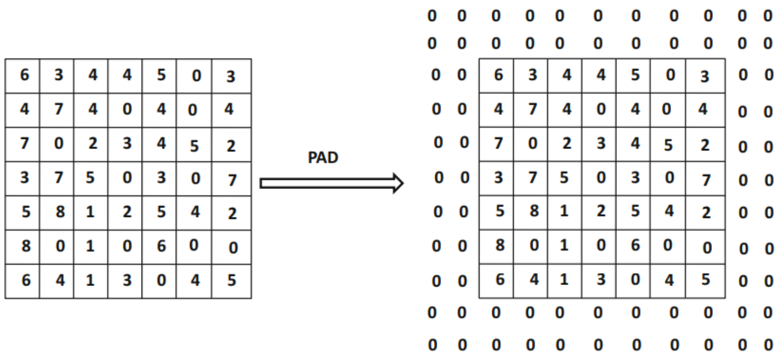


Figure 3.2 Padding example

3.3 Strides

In the previous definition of convolution the filter was always shifted by 1. Nevertheless, it can be taken as another input parameter to convolution. Such parameter is usually referred to as a stride. When a stride of S_q is used, the kernel is shifted by S_q locations during convolution. In such case the size of output layer has height $(L_q - F_q)/S_q + 1$ and a width of $(B_q - F_q)/S_q + 1$. High stride value can help to reduce memory usage and over-fitting, however in most cases stride of 1 or 2 is used. Increasing the strides can quickly expand the receptive field of individual features, while reducing the spatial footprint of the entire layer. [3]

3.4 Pooling

Pooling is another important operation in CNN. Let's have a region of size $P_q \times P_q$. Pooling is taking all the values inside of this region and converts it to the single value. There are two types of conversion: max-pooling and average pooling. [3]

Average pooling - replaces all the values with the average value.

Max pooling - replaces all the values with the max value.

After this operation the region is shifted by some stride. Usually stride 1 or 2 is used. Unlike convolution, pooling is applied separately on each feature map. [3]

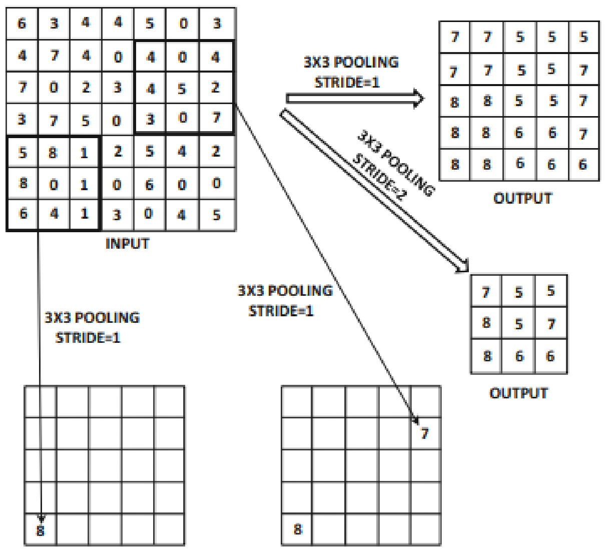


Figure 3.3 Pooling example

3.5 Activation function

3.5.1 Rectified Linear Unit

The most used activation function in CNN is called Rectified Linear Unit function (ReLU). It has the following formula:

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{3.1}$$

Earlier such activation functions as tanh or sigmoid was used, but it appeared that ReLU surpasses them both in speed and accuracy. [4]

Chapter 4

Datasets

This chapter describes existing image malware datasets and one binary dataset, which can be converted to the image representation and used for CNN model training as well.

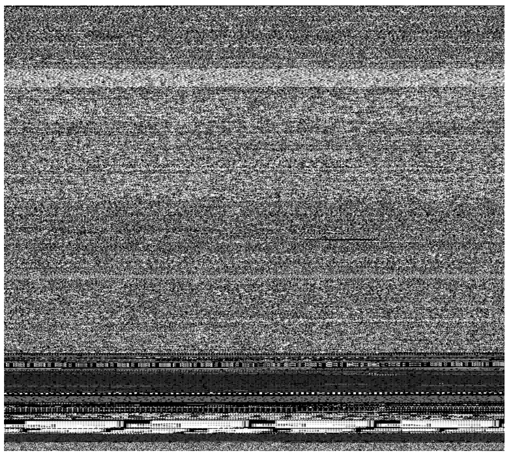
4.1 MalImg

This dataset arose as a result of research carried out by Nataraj at al. [1]. It has 9,458 malware samples splitted into 25 malware families. To convert binary executable into image the binary file is read as vector of 8 bit unsigned integers and then organized into a 2D array. Each integer has value in range [0; 255] and represents a single pixel of gray-scale image, where 0 corresponds to black color, 255 corresponds to white and and other values are intermediate shades of gray. [1]

In their paper authors suggest to determine image width depending on the malware size 4.1 and let the height vary.

File size range	Image width
$<$ 10 kB	32
10 kB - 30 kB	64
30 kB - 60 kB	128
60 kB - 100 kB	256
100 kB - 200 kB	384
200 kB - 500 kB	512
500 kB - 1000 kB	768
$>$ 1000 kB	1024

Table 4.1 Image width for various file sizes


(a) Win32/AutoRun.K

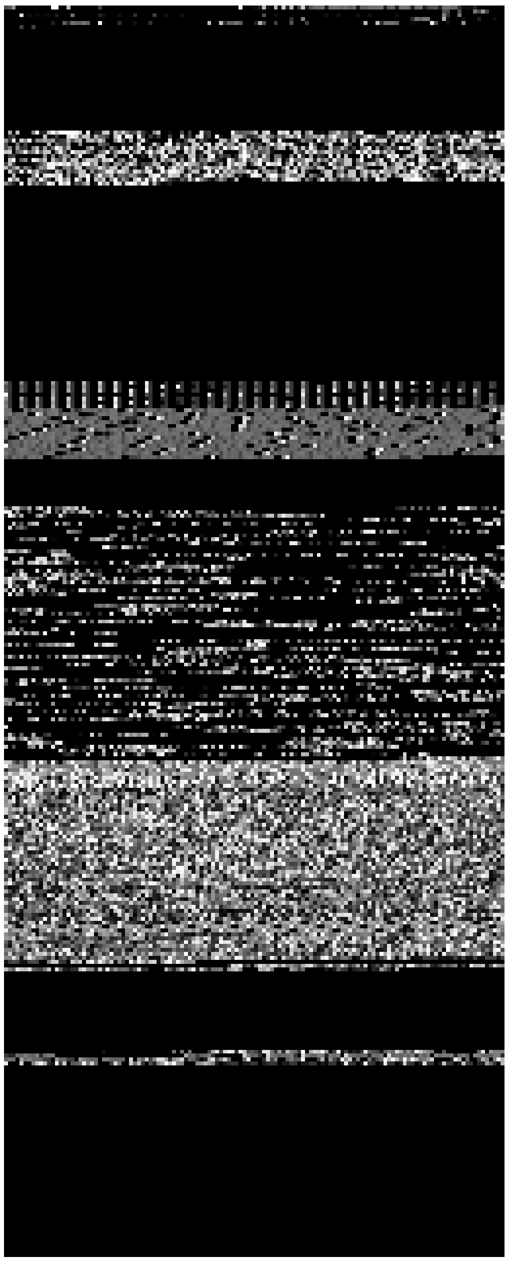
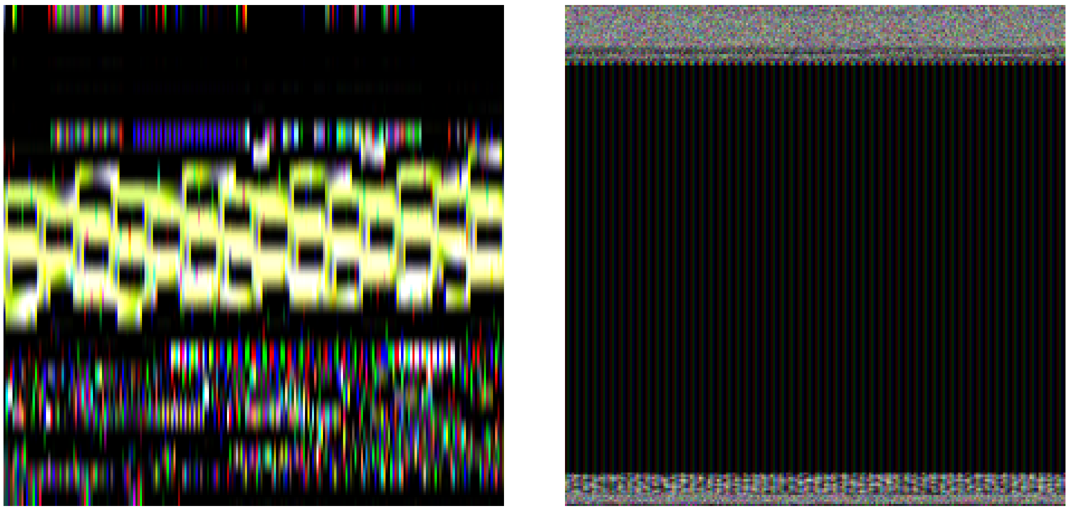

(b) Win32/Dontovo.A

Figure 4.1 MalImg samples representation

4.2 Malevis

The Malevis dataset is RGB based ground truth dataset collected by Multimedia Information Lab of Hacettepe University Computer Engineering cooperated with COMODO Inc. In total, the dataset consists of 9100 training and 5126 validation RGB images, divided into 26 classes (25 malware types + 1 that represents ”legitimate” samples). [2]

To convert malware samples into images, authors of the dataset have used bin2png script, which is publicly available on GitHub, then fixed the width to 224 and 300 pixels correspondingly and resized the image by using Lanczos interpolation to obtain square shape. [2]



(a) Win32/AutoRun-PU **(b)** Win32/Adposhel

Figure 4.2 Malevis samples representation

4.3 SOREL-20M

SOREL-20M (Sophos-ReversingLabs 20 Million) - is a malware dataset for ML purposes created by anti-virus company Sophos in 2020. The dataset contains 20 million Windows PE executables, including 10 million disarmed malware samples.

4.3.1 GitHub repository

The GitHub repository of the project is located at https://github.com/sophos/SOREL-20M and is licensed under Apache 2.0 license. It contains supportive scripts and instructions about how to use them, environment requirements, licence information, terms of conditions and answers to the frequently asked questions. Among the presented scripts there are those for training and evaluating ML model ('train.py' and 'evaluate.py'). Also authors provide two baseline models: a Pytorch feed-forward neural network (FFNN) model, and a LightGBM gradient-boosted decision tree model. Their structure can be found in 'nets.py' and 'lightgbm_config.json' respectively [5].

For the purposes of this thesis we are be interested in two scripts that work with SQLite3 "meta.db" database of malware metadata and load the dataset - 'dataset.py' and 'generators.py'. Parts of their code will later be used during the implementation phase. Database structure is described later in the 4.3.3.

4.3.2 AWS S3 bucket

The core files of the dataset are available via AWS S3 bucket: s3://sorel-20m/09-DEC-2020/. It contains such things as: trained FFNN and LightGBM model (checkpoints/), performance results for each model (results/), extracted LightGBM features (ligthGBM-features/), zlib compressed malware binaries (binaries/) and dataset metadata: SQLite db, Ember 2.0 and pefile features (processed-data/). Full bucket structure is shown on the figure below: 4.3.

```
s3://sorel-20m/09-DEC-2020/
├── Terms and Conditions of Use.pdf
├── baselines/
│   ├── checkpoints/
│   │   ├── FFNN/
│   │   └── lightGBM/
│   └── results/
│       ├── ffnn_results.json
│       ├── lgbm_results.json
│       ├── FFNN/
│       │   └── seed0-seed4
│       └── lightGBM/
│           └── seed0-seed4
├── binaries/
├── lightGBM-features/
│   ├── test-features.npz
│   ├── traing-features.npz
│   └── validation-features.npz
└── processed-data/
    ├── meta.db
    ├── ember_features/
    └── pe_metadata/
```

Figure 4.3 Sorel-20M S3 structure

The most important for this work is binaries/ folder, which contains around 9,919,251 disarmed malware samples ∼ 8TB of data. These samples were collected by Sophos-ReversingLabs from January 1, 2017 to April 10, 2019.

4.3.3 Metadata structure

The SQLite database schema for the meta.db file within the 'processed-data' sub-directory is as follows:

```
CREATE TABLE meta (sha256 text primary key,
  is_malware SMALLINT,
  rl_fs_t DOUBLE,
  rl_ls_const_positives INTEGER,
  adware INTEGER,
  flooder INTEGER,
  ransomware INTEGER,
  dropper INTEGER,
  spyware INTEGER,
  packed INTEGER,
  crypto_miner INTEGER,
  file_infector INTEGER,
  installer INTEGER,
  worm INTEGER,
  downloader INTEGER );
```

- sha256 – the sha256 of the unmodified file (note that all provided files are "disarmed")
- is_malware – a value of 0 indicates benign-ware, 1 indicates malware

- rl_fs_t – the first time (in Unix epoch time) a given sample (unique per sha256) was seen in the ReversingLabs feed
- rl_ls_const_positives – the total number of 'positive' (i.e. malware) results from all detectors at the most recent time that the samples was seen (assuming that more recent scans will be higher quality due to signature updated etc
- adware, flooder, ransomware, dropper, spyware, packed, crypto_miner, file_infector, installer, worm, downloader – the number of tokens appearing in detection names that related to the specified tag; a value >0 indicates a positive result, larger values may indicate higher certainty in the tag

4.4 Benign samples

To train a neural network to distinguish between malware and non-malware samples, the benign dataset is needed. As a source of benign files we will Windows 10 binaries.

Chapter 5

Implementation

Bibliography

1. NATARAJ, Lakshmanan; KARTHIKEYAN, Shanmugavadivel; JACOB, Grégoire; MANJUNATH, B. Malware Images: Visualization and Automatic Classification. 2011. Available from DOI: `10.1145/2016904.2016908`.
2. BOZKIR, Ahmet; CANKAYA, Ahmet; AYDOS, Murat. Utilization and Comparision of Convolutional Neural Networks in Malware Recognition. In: 2019. Available from DOI: `10.1109/SIU.2019.8806511`.
3. AGGARWAL, Charu C. *Neural Networks and Deep Learning - A Textbook*. Springer, 2018. ISBN 978-3-319-94462-3. Available from DOI: `10.1007/978-3-319-94463-0`.
4. VENKATESAN R., Li B. *Convolutional Neural Networks in Visual Computing: A Concise Guide (1st ed.)* CRC Press, 2017. ISBN 978-1-315-15428-2. Available from DOI: `10.4324/9781315154282`.
5. HARANG, Richard; RUDD, Ethan M. *SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection*. 2020. Available from arXiv: `2012.07634 [cs.CR]`.