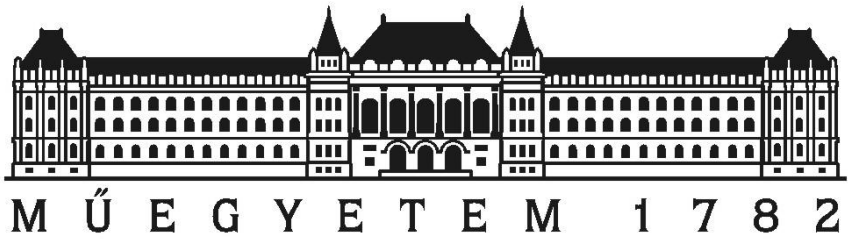
Malware detection through Convolutional Neural Network

Chaitanya Arora
Supervisor: Dr. Levente Buttyan



Laboratory of Cryptography and System Security (CrySys Lab)
Department of Network Systems and Services (BME HIT)
Budapest University of Technology and Economics
Budapest, Hungary

1. Introduction:

Internet of things is described as objects which consist of sensors, software and other technologies which allow it to connect and exchange data with other devices over the internet. The name, “Internet of things” is itself a misnomer as usually IOT devices are connected to a single address instead of being connected to a public network. Taking step back to the history of when it all began- The concept IOT started off in 1982 when a cola wending machine was modified to make sure that it is never out of stock and was able to identify if the cola cans are cold or not. IOT gained more popularity when Bill ken presented that device-to-device communication can be a part of the six webs in Davos.

Moving forward to the present, IOT devices surround us everywhere. Having a wide range of applications in infrastructure, consumer, and industrial sectors. The idea of smart home has been escalated with the progress made in IOT, with devices taking care of things like heating, lighting, automatic door opening and much more. A good example can be the google home and echo dot, which are connected to the internet and are used to connect to many things like air conditioners, handling door locks etc. These are not only IOT devices but also allow other existing appliances in the house to be smart. Besides the technological advancement, it also gives us the opportunity to have long term benefits in terms of energy savings or making the users aware of the usage.

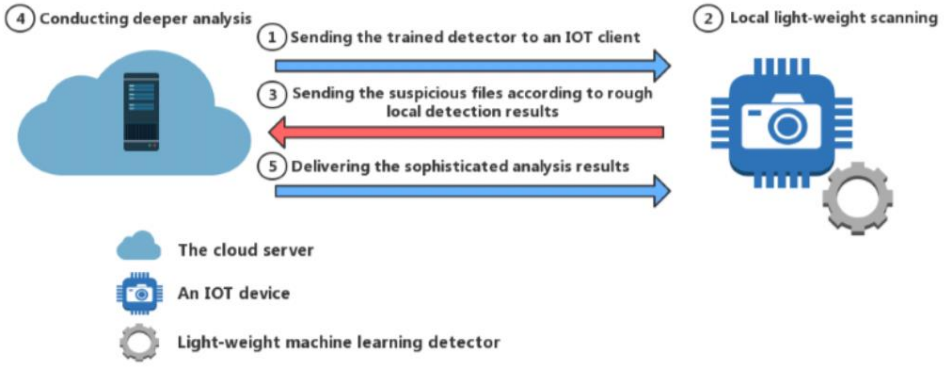
Besides this IOT devices have a wide range of usage in the health and Medicine sector, in monitoring and data collection of various parameters in human body like heart rate, blood pressure etc. and can trigger emergency buttons to get the needful action. Having such devices installed in offices significantly reduces the dependencies on humans also reduces the chances of human error.

Despite having revolutionary advantages and wide ranged applications across domains security of IOT devices always remains a concern. Bringing IOT devices and having high dependence on them in our daily life makes us even more vulnerable in case of any security fallouts. Many of the security vulnerabilities are like the ones that are faced in conventional servers and smartphones. These concerns comprise of MITM (man in the middle attacks), SQL injections, DDOS (Distributed denial of service) etc. Since IOT still being a relatively new field there exists no unified conventions from a security standpoint. Another major concern that we face is the lack of computational power which exists in the IOT devices making it difficult to develop a robust system with firewalls and strong encryption of data. Once in 2016 a DDOS attack running the Mirai malware which took down a DNS sever numerous websites. The Mirai software are found out devices ranging from televisions to printers and had spread the infection to 65000 IOT devices in the first 20 hours and then ranging to 200,000 to 300,000.

A malware can be defined as a software which is intent to cause disruption, leak private information in a sever, computer or a network. Currently there are many types of malwares which exist in the market including ransomware, trojan horses, spyware etc. The evolution of computer viruses started even before we had internet. Starting from multitasking viruses in the UNIX system, to going in personal computers by infecting the executable programs of floppy disks. IOT malware as the name suggests is having malware files in IOT devices. There are three main IOT malware codebases, they are namely Kaiten, Qbot and Mirai.

As mentioned, before IOT devices have a considerable low amount of computing power because of which it makes it difficult to conduct a strong security check. But a lightweight approach can be taken up in which can help in classifying a file whether it is malware or not after which we send those files on a remote server to which the hardware IOT device is connected to, wherein the files

are tested upon with strong convolution network and the results are sent back, and accordingly the hardware IOT device responds.



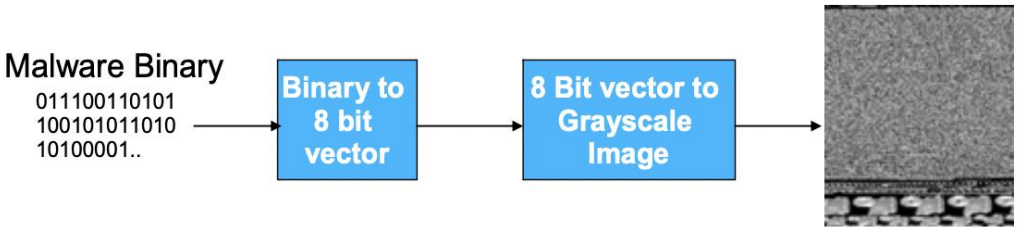
The goal of this paper is to reiterate the approach used in the paper [1] and to follow the similar way used by the authors to classify malware files using simple convolutional neural network but on a different and much smaller sized data set and to compare the results in terms of accuracy, computation power etc. We use two different architecture which are arm and MIPS and the dataset comprises of benign and malware files to test the accuracy.

The application of the project in real life is in the usage of antivirus software's which are used to test and classify malware and other types of viruses to ensure the smooth functioning.

2. The Approach

The approach that we will be following will be to treat malware files and feed them to a convolutional neural network to and use image processing to determine if a file is malware or not.

The paper being referred to would have been only possible because of previous work done by the authors of [4] in which they convert the malware files which usually exist as executable binary file consisting of zeros and ones to greyscale images. This is done by reshaping the binary values into a matrix. The image on the other hand is a big matrix consisting of different values from the range of 0 to 255. In the case of malware files, we read the data as a 8 bit unsigned integers and then organize them into a 2D array which is then visualized as a grey scale image. The process can be better explained with the help of a visual



The steps of implementation of the project are threefold:

1. Converting the malware binary files to images
2. Implementing the convolutional neural network for image processing which is used to prediction
3. Testing the accuracy and analysis of the result in terms of processing power, confusion matrix etc.

About the Dataset: The neural network is implemented on a small dataset of files consisting of 2*2*2000 files each being malware and benign from two architecture namely arm and MIPS.

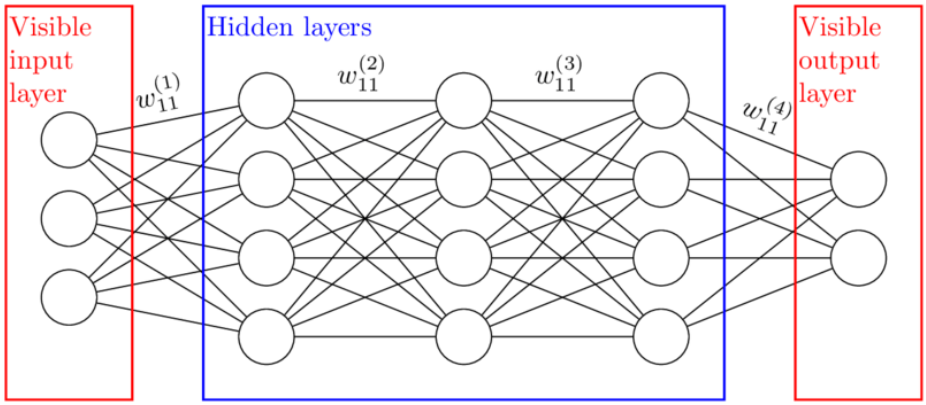
Convolutional Neural Network: In the everyday world words like Artificial Intelligence, Neural Network are becoming buzz words. But they are not the same. Starting with neural networks:

The idea of a neural network is to provide prediction to us, we have some information which we feed to the neural network, and it gives us some output. For the sake of simplicity. Consider Neural network as black box taking a set of inputs and giving some meaningful output. Consider this black box as a function which maps your input to the output.

To answer the question arises of what the neural network is made up of?
A neural network is made up of layers. The first layer of a neural network is always the input, this is what accepts our raw data (the data we want to classify). The number of input neuron depends upon the type of data being used.

A classic example of an image with width and height 28X28, to make a prediction we need all the pixels of that image. So, the number of input neurons that we would need is 784 which is the product of 28 times 28. If the data set is 4 pieces of information, then we need 4 inputs. As for the output layer, the output layer has as many pieces of neuron as the number of amounts of information we want and based on the output we evaluate it into classes. In some instances, it is preferred to have the number of output neurons same as the classes you are predicting and the combination of all these values is 1.

In between these layers which is called the hidden layers. They are called hidden layer because we don't observer it. Every single layer is connected to another layer with a certain number of weights. A densely connected neural network is when every node in the input layer is connected to every node of the output layer. These weights are what the neural network will change to get the output. They are called as the trainable parameters that the neural network will change.



The image above signifies the visual representation of convolutional neural network having the three-layer types (input layer, hidden layer, output layer)

Before deep diving into the working of the neural network, it is a good idea to look at it from a mathematical standpoint.

Data starts at the input layer and is transformed as it passes through subsequent layers. The data at each subsequent neuron is defined as the following.

$Y=(\sum ni=0 wixi)+b$

w stands for the weight of each connection to the neuron
x stands for the value of the connected neuron from the previous value
b stands for the bias at each layer, this is a constant
n is the number of connections
Y is the output of the current neuron
$\sum$ stands for sum

Biases also exists as an input and is another trainable parameter which the neural network uses. The bias allows us to shift the network up or down by a constant value. It is like the y-intercept of a line. One another crucial part is the activation function that we use to the equation above to add complexity and dimensions to our network.

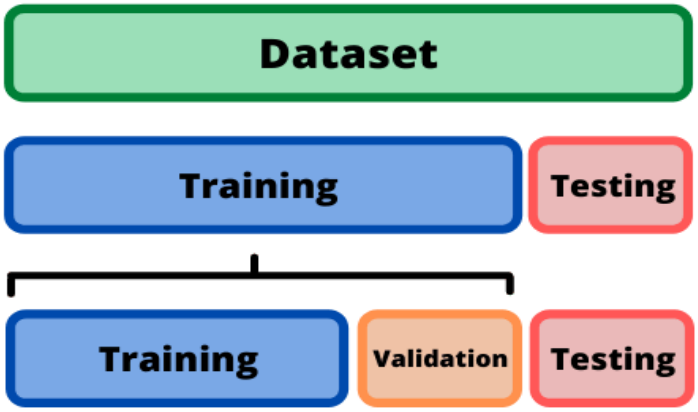
$Y=F((\sum ni=0 wixi)+b)$

Our network will start with predefined activation functions (they may be different at each layer) but random weights and biases. As we train the network by feeding it data it will learn the correct weights and biases and adjust the network accordingly using a technique called **backpropagation**

The working of the project involves implementing an image recognition convolutional neural network. To implement it we start by first creating the data set and preparing the dataset in terms of what the neural network requires, this is one of the most crucial steps because the way we work with the data will affect the whole CNN and to an extension the prediction.

After dealing with the dataset, we create a neural network for us in the similar fashion as discussed previously (the number of layers, input/output neuron varies). To make the neural network do predictions we have to train it first. This is done by first dividing the data set into 3 sections namely training, test and validation. In simpler cases we also have just training and test dataset.

Usually, the idea is to spend considerable amount of time and resources on the training of the model and less time to predict. Therefore, we allocate a larger chunk of our dataset to training and a small part to test.



The way training is conducted is that we provide the neural network the image and the correct answer and we repeat this process over the whole training dataset. The neural network then understands it by feature extraction from the data and by finding similarities. Then after this the testing dataset is used in which we feed the image and the model returns us the answer and depending on this basis we determine how accurately the model is working.

3. Implementation:

As stated previously that the implementation has been divided into three main parts:

1. Converting the malware binary files to grey scale images
2. Implementation of the Convolutional Neural Network (CNN)
3. Evaluation of the results

Converting the images to Greyscale Images:

```
import os
import pandas as pd
from os import listdir
import numpy as np
from PIL import Image
import binascii
```

We used normal data analysis libraries of python like pandas and NumPy. To iterate through the sample there is usage of the “listdir” function from os. The conversion of an image array to an image type object is done with the help of the “pillow” library. The usage of the binascii is done to handle the binary malware files.

```
images_new = []
def convert():
    num_files = listdir("ml-sample-pack-small/benign/arm")
    for file in num_files:
        loc = "ml-sample-pack-small/benign/arm/" + file
        with open(loc,'rb')as f:
            content=f.read()
        hexst=binascii.hexlify(content)
        fh=np.array([int(hexst[i:i+2],16)for i in range(0,len(hexst),2)])
        rn=len(fh)/1024
        fh=np.reshape(fh[:int(rn)*1024],(-1,1024))
    #   print(fh.shape)
        fh=np.uint8(fh)
        im=Image.fromarray(fh)
        im=im.resize((32,32),Image.ANTIALIAS)
        images_new.append(im)
        # im.show()
```

The image above highlights the conversion of the malware binary files to images. The first step involves opening the binary file in the read mode and reading it in the “content” variable. After which the binary data is converted to the hexadecimal with the help of the “hexlify” variable. The data is then converted to an array and the reshaped according to the image and is converted to unsigned 8-bit integer and an image object is generated with the “fromarray” function.

This same process is then repeated over the whole dataset both MIPS and ARM type architecture.

Implementation of CNN model:

```
#utils
import numpy as np
import pandas as pd
from glob import glob
import tensorflow as tf

#visualisation
import seaborn as sns
import matplotlib.pyplot as plt

#Preproc
from keras.preprocessing.image import ImageDataGenerator
from keras.utils.np_utils import to_categorical
from keras import backend as K

#model
from keras import layers
from keras.models import Model
from keras.layers import Conv2D, GlobalMaxPooling2D,MaxPooling2D
from keras.layers import Activation,Dense

#metrics
from sklearn import metrics
from keras.metrics import categorical_accuracy
from sklearn.metrics import classification_report,confusion_matrix,f1_score, precision_score
import time
```

The image above describes the libraries which are used for the implementation of the Neural Network. For the sake of simplicity, we will be going through one architecture.

```
seed = 5
np.random.seed(seed)
#dimensions
img_width, img_height = 64,64

#data dirs
train_data_dir = './img/train/'
test_data_dir = './img/val/'

#Classes/labels
class_types = ['Malware','Benign']
```

After the separation of the dataset into training and validation, we created fixed seeds divided the dataset into two classes namely benign and malware.

```
def model(input_shape):
    input_layer = layers.Input(shape = input_shape)
    x = Conv2D(32,(3,3),activation='relu')(input_layer)
    x = MaxPooling2D(32,(2,2))(x)
    x = Conv2D(72,(3,3),activation='relu')(x)
    x = GlobalMaxPooling2D()(x)
    x = Dense(256,activation='softmax')(x)
    output_layer = Dense(1,activation='sigmoid')(x)
    model = Model(inputs =input_layer,outputs = output_layer,name='CNN')
    return model
```

Creation of the Sequential model.
1st layer: input layer = creates a starting point for the data to enter by giving all of them a uniform shape from where they can traverse through the rest of the model.

2nd Layer: Convolution 2D layer = This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
params: filters, kernel shape, activation function,
The input layer has been passed as the layer before it.

3rd layer: MaxPooling2D layer= Down samples the input along its spatial dimensions(number of pixels in this case) or(height and width) by taking the maximum value over an input window (of size defined by pool size) for each channel of the input The window is shifted by strides along each dimension.
params: filters, pool size.

The last layer has been passed as a parameter to get the input shape.

Global MaxPooling2D = The ordering of the dimensions in the inputs. channels last corresponds to inputs with shape (batch, height, width, channels) while channels first corresponds to inputs with shape (batch, channels, height, width) .

Dense layer = The dense layer's neuron in a model receives output from every neuron of its preceding layer, where neurons of the dense layer perform matrix-vector multiplication. params: units, activation function

Output Layer = Again a Dense layer with only 1 unit and sigmoid activation to normalize the output

```
Model: "CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 64, None, 1)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 62, None, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, None, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, None, 72)      20808
_________________________________________________________________
global_max_pooling2d (Global (None, 72)                0
_________________________________________________________________
dense (Dense)                (None, 256)               18688
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 40,073
Trainable params: 40,073
Non-trainable params: 0
_________________________________________________________________
```

The above image signifies the output we get after which we compile the model (model. Summary ()). While compiling the model we use loss function "binary crossentrop" because we are classifying between 2 classes, benign and malware.

```
train_generator = train_datagen.flow_from_directory(
                                train_data_dir,
                                color_mode='grayscale',
                                target_size = (64,64),
                                batch_size = 10,
                                class_mode = 'binary')


test_generator = test_datagen.flow_from_directory(
                    test_data_dir,
                    color_mode='grayscale',
                    target_size=(img_width, img_width),
                batch_size=10,
                class_mode='binary')
```

```
Found 3200 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
```

The last step is the training of the model, we implement 2 data generators here which create the training and test data sets. The generators take in the following parameters:

params: path_to_directory,color_mode,target_size, batch_size, class_mode.

The directory passed as parameter has 2 child directories which contain the 2 classes of image data (benign and malware). All the data images are greyscale images. The target size is 64x64 pixels. We create batches of 10 images which will be processed at once.

4. Evaluation:

Systems /Metrics	Approach Followed
Accuracy	94%
Classifier	CNN
Number of Layers	2
Number of Nodes	104
Fully Connected Layer	256
Preprocess	Re-organizing Binary
Input Dimension	64X64

Arm Architecture:

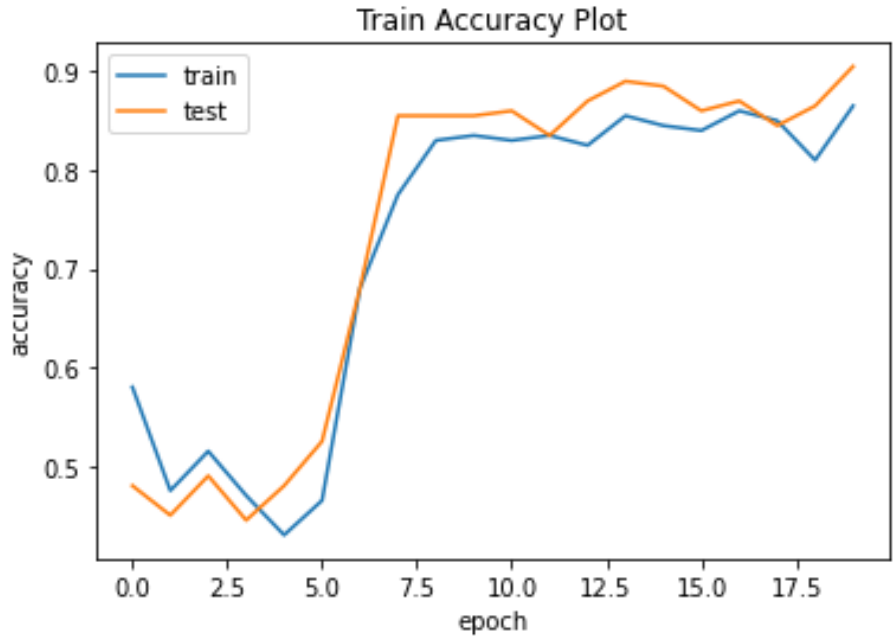
Accuracy	Training Time	Validation Accuracy
84.11%	100.230 sec	90.5%

Confusion Matrix:

Total Malware files: 400
Total Benign files: 400

Malware (400)	**True Positive** (174)	**False Positive** (226)
Benign (400)	**False Negative** (153)	**True Negative** (247)

Graphical Visualization:



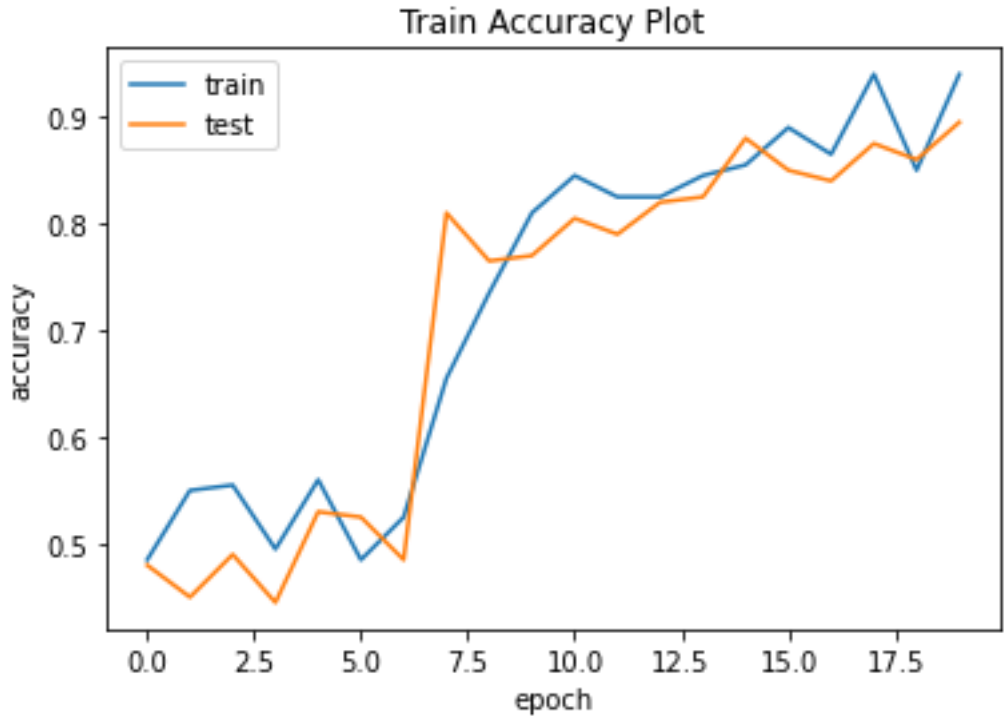
MIPS Architecture:

Accuracy	Training Time	Validation Accuracy
92.23%	90.427 sec	89.50%

Confusion Matrix:

Malware (400)	**True Positive** (163)	**False Positive** (237)
Benign (400)	**False Negative** (166)	**True Negative** (234)

Graphical Visualization:



5. Conclusion

By this project we lay ground for the lightweight approach which can be implemented as a preliminary classifier for the malware files, and can be used inside the IOT hardware. It provides a moderately high accuracy way with minimal computation power to classify harmful files. The most important part of the solution is confusion matrix- here the focus on the true malware and true benign values and time complexity - where we analyse the training time and testing time for a comparative analysis for other models.
The future work in this sector can focus in increasing the accuracy this can be achieved by:

- Creating distinctions in different malware families.
- Applying new approaches like feature extraction.
- Bringing in the use of parameters like the image texture as new features.

6. References

[1] J. Su, D. V. Vasconcellos, S. Prasad, D. Sgandurra, Y. Feng and K. Sakurai, "Lightweight Classification of IoT Malware Based on Image Recognition," 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), 2018, pp. 664-669, doi: 10.1109/COMPSAC.2018.10315.

[2] https://en.wikipedia.org/wiki/Internet_of_things

[3]https://en.wikipedia.org/wiki/Malware#:~:text=Malware%20(a%20portmanteau%20for%20malicious,unknowingly%20interferes%20with%20the%20user's

[4] Nataraj, Lakshmanan & Karthikeyan, Shanmugavadivel & Jacob, Grégoire & Manjunath, B.. (2011). Malware Images: Visualization and Automatic Classification. 10.1145/2016904.2016908.