What is the Shape of an Executable?

Erick Galinkin
Netskope

$ whoami

Erick Galinkin

Security Research Scientist at Netskope

Applied Mathematics at Johns Hopkins

Father of saasy_boi

Motivation and Overview

There is a lot of information (of varying quality) about Windows PE malware analysis.

There is a lot of information about doing machine learning on Windows PE files.

There is much less information about executable files* for other operating systems.

Is there a way we can use one corpus of knowledge to improve the others?

*Code that is compiled and run on the underlying hardware by interfacing with the operating system.

Motivation and Overview

Inspired by Raff *et al.*[1], neural networks seem to be promising for detection of malicious code.

I wanted to consider the topology of executable code to see if there was some optimization we could gain, especially in terms of transfer learning for cross-platform malware detection.

Li *et al.*[2] provided a way for us to find a filter “shape”.

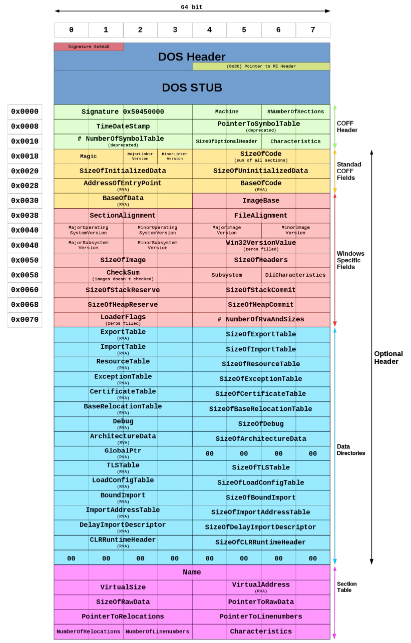
Windows Portable Executable Files

PE files are based on the Common Object File Format (COFF)

Addresses are stored throughout the header along with data about the file.

The executable types of other major operating systems are also based on the Common Object File Format and have similar* properties.

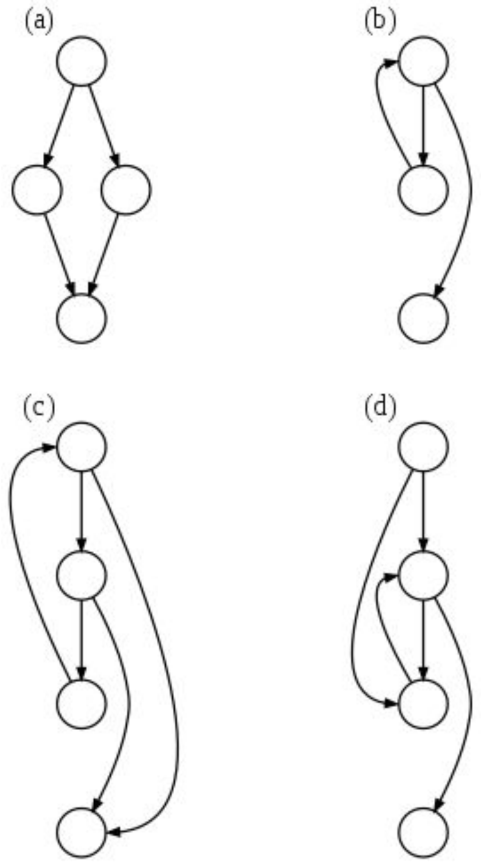
*Caveats cateats



Portable Executable 32 bit Structure from Wikimedia Commons / licensed under CC 4.0

Control-Flow Graph

Thanks to Frances Allen [3], we know that the control flow relationships between sections of an executable are able to be expressed as a directed graph - telling the program where data ought to come from and where it should go.



Some Types of Control Flow Graphs from Wikimedia Commons / licensed under CC 4.0

Why use a neural network?

The transfer learning abilities of neural networks are well-documented and a quick search for “Neural Network Transfer Learning” will yield thousands of pages of varying quality.

The results of Raff *et al.* suggest that a neural network can inherently learn the features of malware vs benignware in a way that does not require hand-engineering features.

Since the 3 major operating systems run on the same architecture (x86), their processor instructions and therefore, the bytes themselves, should be similar, even if the underlying libraries are quite different.

Training Environment

GPU rig:

Shared among 4 data scientists and myself:

- Intel(R) Core(TM) i9-9960X
- 128 GB RAM
- 4x Nvidia GeForce GTX 1080 Ti
- 1 TB SSD

All data was loaded from a 2TB USB external hard drive (I know.)

Dataset:

- Training/Test/Benchmark - EMBER 2018 [4]
- 500 MacOS samples (250 malicious, 250 benign)
- 500 Linux samples (250 malicious, 250 benign)

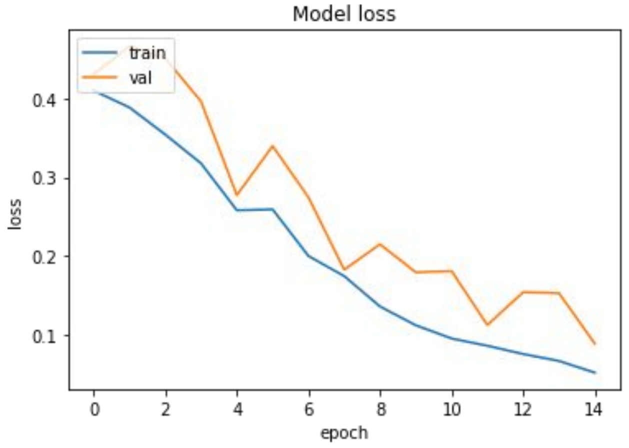
Code stack:

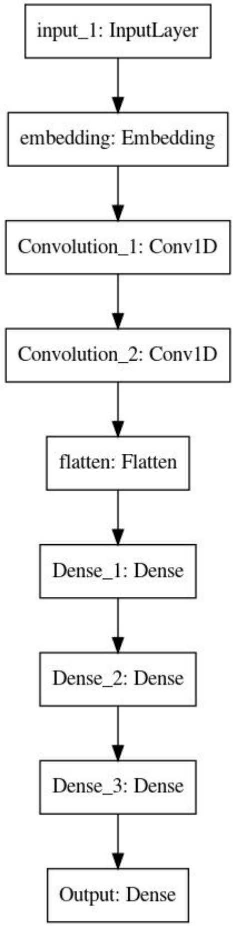
- Python 3.7
- Tensorflow 2.0
- Cuda 10.0

Neural Network Architecture

"Why is your architecture so boring?"

It works for our purposes and takes a really long time to train





Filter Shape Generation

In order to find the filter shapes, 2 things are necessary:

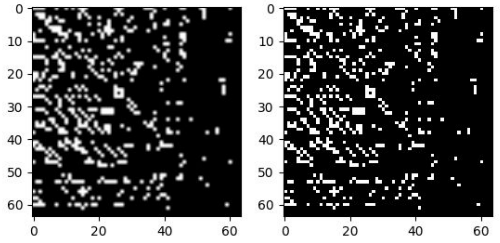
1. A set of filters
2. A covariance matrix from the dataset of interest

Finding the “shape” of the executable was done during training time and added significant overhead (4 hours per epoch) to training time.

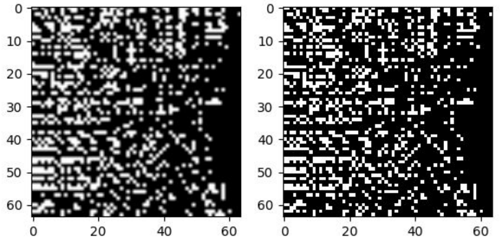
It wasn’t until after 60 additional hours of training that we realized how similar the filters were…

Filter shapes

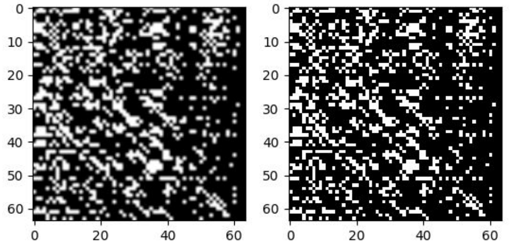
Linux



Mac



Windows

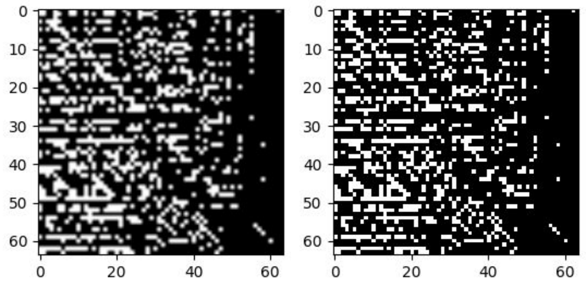


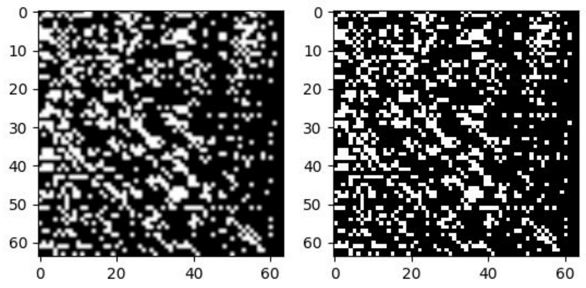
Similarity between Mac and Windows	.9152
Similarity between Linux and Mac	.7333
Similarity between Linux and Windows	.7394

An interesting finding!

For the Ember dataset, no matter whether the binary mask (read: shape) was generated at the beginning of training, at every epoch, or only at the last epoch, the shape was extremely similar! (Cosine similarity of .9484)

This suggests that only 1 sufficiently good mask needs to be generated, and that relatively few weights need to be trained, speeding up training time.



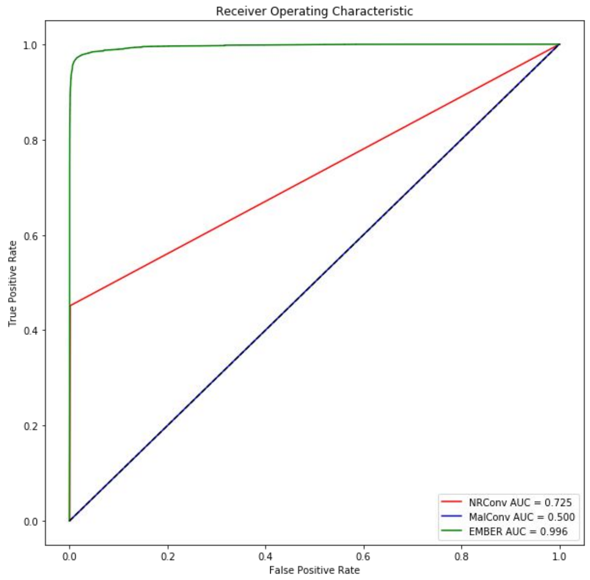


Testing Results

The Non-rectangular convolutional model was tested against 2 benchmarks, both graciously provided by Endgame - the EMBER and MalConv models.

Accuracy on EMBER 2018 test set:

NRConv:	.72509
EMBER Benchmark:	.97802
MalConv:	.50022



Testing for Transfer Learning

<u>Detection accuracy</u>
Linux (no training): .522
Mac (no training): .714

Linux (output only): .6
Mac (output only): .774

Conclusions and Future Work

The training overhead was extremely cumbersome and though it did improve on the results of Raff *et al.*, it fell well below the EMBER benchmark.

Doing more work to explore how we can research executables topologically seems promising, as there is a lot of overlap between the various operating systems.

Expanding the testing sets would greatly help in evaluating the ability of models like this to use transfer learning.

The architecture used was extremely simple and improvements could certainly be made on that front.

References

[1] Raff, Edward, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. “Malware Detection by Eating a Whole EXE.”, October 25, 2017. http://arxiv.org/abs/1710.09435.
[2] Li, Xingyi, Fuxin Li, Xiaoli Fern, and Raviv Raich. “FILTER SHAPING FOR CONVOLUTIONAL NEURAL NETWORKS,” 2017, 14.
[3] Allen, Frances E. “Code Flow Analysis.” SIGPLAN Notices, July 1970.
[4] Anderson, H. S., and P. Roth. “EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models.” ArXiv E-Prints, April 2018.

Thanks for listening to my Talk!

Twitter: @erickgalinkin
Blog: galinked.in

netskope