A Neural Network-Based Approach to Fingerprint Rouge Operating Systems

Zach Mueller*, David Benenati and Stephen Hopkins
Faculty Mentor: Dr. Caroline John
Department of Computer Science, Hal Marcus College of Science and Engineering, University of West Florida
* SURP Student

Introduction

Modern enterprise networks are complex and present countless security challenges. Understanding the nature of the systems that exist within a network environment is a vital step in securing such environments. Hence, operating systems on the network must be identified, tracked, and continuously monitored. In this research, we consider the problem of detecting unauthorized operating systems on an enterprise network, which could exist as a result of the unintentional actions of an authorized user or by the unauthorized actions of internal users or external attackers. A possible scenario involves malware applications that attempt to install virtual machines on a host system within the network as a means of obfuscation and detection avoidance. We utilize a neural network-based classifier which was developed using the pytorch and fastai deep learning libraries. Simulated network traffic was generated through the implementation of a virtual network environment and the generated traffic will be passively collected as it traverses the network boundary.

Operating System (OS) fingerprinting can be used during offensive or defensive operations as shown in the Security Testing Roadmap.

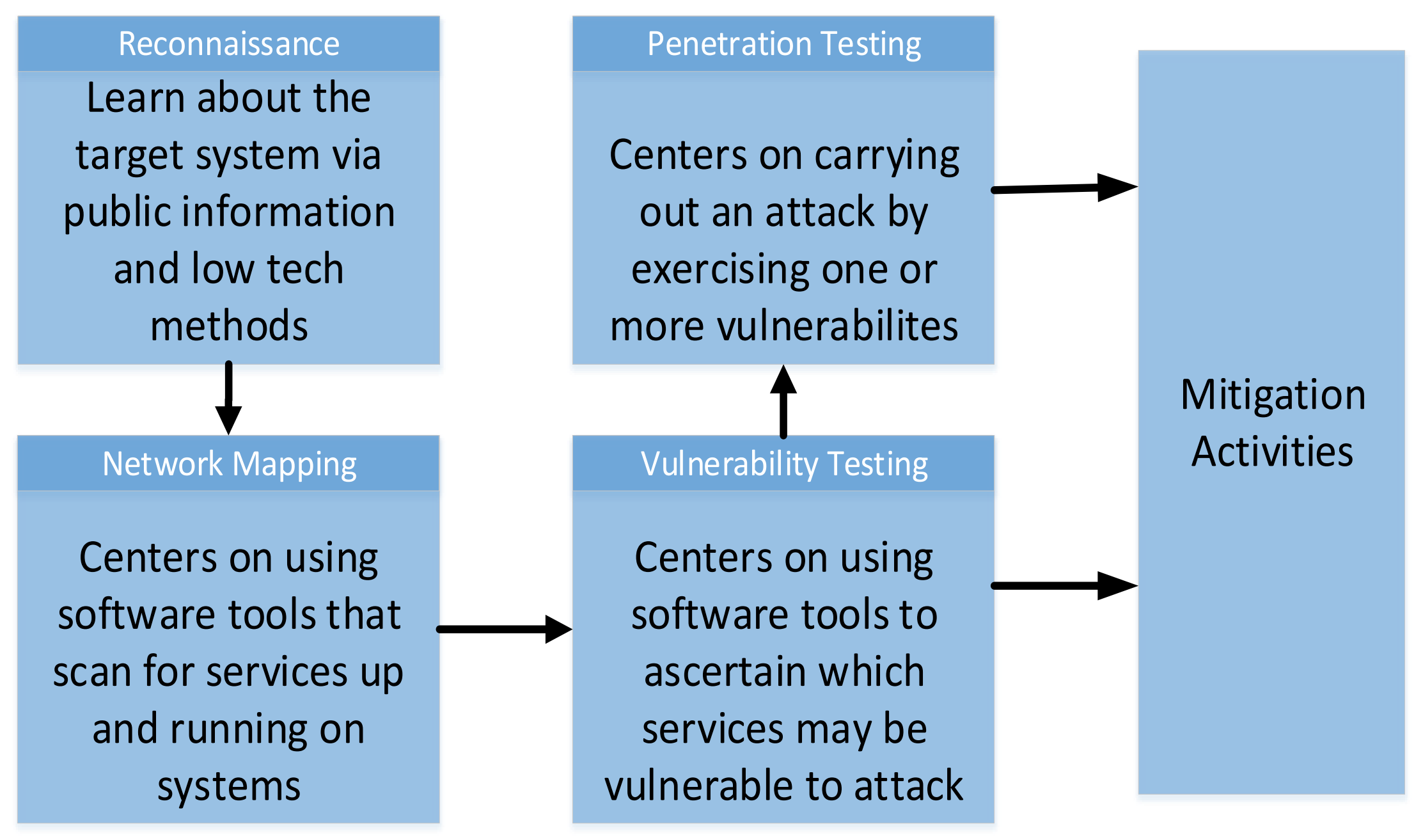
Reconnaissance
Learn about the target system via public information and low tech methods
Penetration Testing
Centers on carrying out an attack by exercising one or more vulnerabilites
Mitigation Activities
Network Mapping
Centers on using software tools that scan for services up and running on systems
Vulnerability Testing
Centers on using software tools to ascertain which services may be vulnerable to attack

Figure 1: Security Testing Roadmap [1]

OS fingerprinting involves the capture and examination of network packets. The OS can be fingerprinted based on analysis of packet features.

Attackers and penetration testers use OS fingerprinting to identify and enumerate potential targets.

- Active fingerprinting initiates a scan by sending specially crafted packets to a target and examining response packets.
- Passive fingerprinting examines existing normal network traffic without generating additional traffic.

Effective Network Defense

- Begins with establishing an understanding of the systems that exist on the network.
- Passive OS fingerprinting can be used to achieve this objective without creating excess traffic on the network.

Research Objectives

This study will consider the application of passive OS fingerprinting as a means of collecting packets that traverse the network with goal of identification of individual operating systems on the network along with individual computers.

How and why does fingerprinting work?

- RFC's 791 and 793 specify the required packet structure for IP and TCP protocols.
- Both RFC's leave a number of packet structure decisions up to OS developers.
- Operating systems are identified by examining these differences among TCP/IP packet features [2].
- Figures 3 and 4 display TCP/IP packet structure

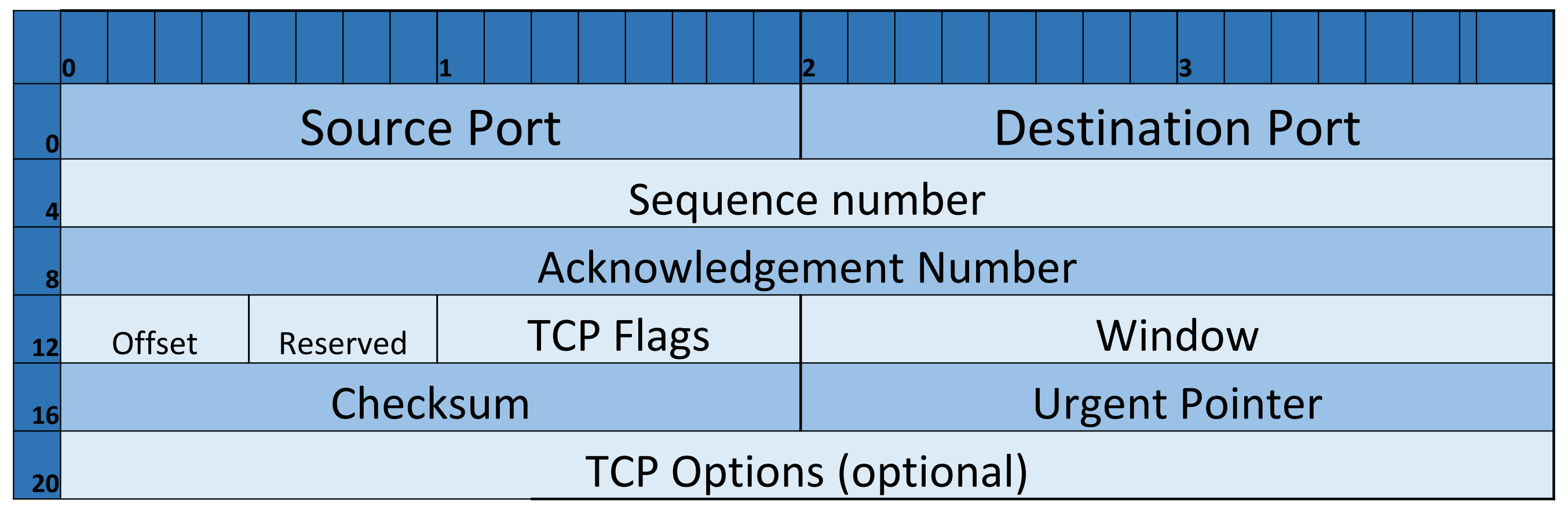

Figure 2: IPv4 Packet Header [1]

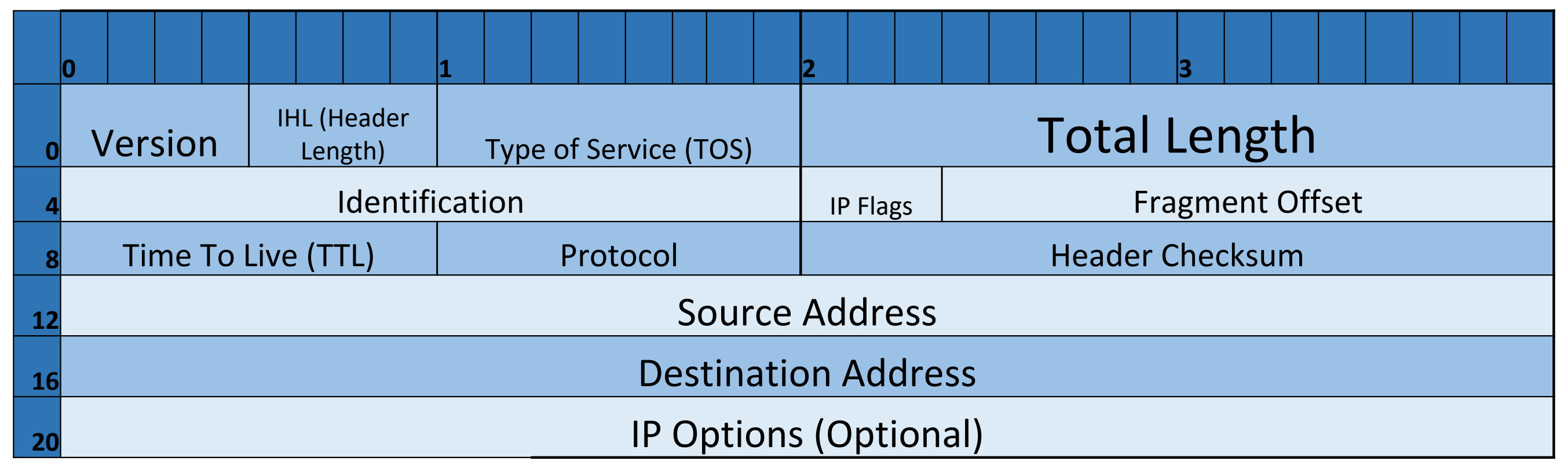

Figure 3: TCP Packet Header [2]

- Common packet features used to identify operating systems include Time To Live (TTL), Window Size, checksum, and total length [5].
- Figure 3 displays many of the TCP packet features that can be examined to identify operating systems.

Common TCP/IP Features Used in OS Fingerprinting	
TCP Features	**IP Features**
stream	checksum
options.timestamp.tsval	ttl
window_size_scalefactor	len
port	proto
srcport	opt.ra
analysis.duplicate_ack_frame	dsfield.dscp
analysis.duplicate_ack_num	frag_offset
analysis.out_of_order	
analysis.rto_frame	
flags	

Figure 4: Common Packet Features Used in OS Fingerprinting [3]

Methods

The study made use of the fastai[4] deep learning library in identifying which packet features can be best used to identify operating systems. Along with this, two different model architectures were used. The reason for deep learning as compared to other commonly used algorithms for tabular data is access to embedding layers, which were found to be extremely important.

Data:

3.4 million packets were analyzed, with the train, validation, and test sets consisting of 70, 20, and 10 percent of the data respectively.

Model Architectures:

Architecture 1:

- An embedding layer
- 5 Fully Connected Layers**

Architecture 2:

- An embedding layer
- A Noise Generator
- An encoder which contains an hour-glass pattern of layer sizes from 32 to 300
- Architecture 1
- A total of 10 Fully Connected Layers**

** Fully Connected Layers contain a Linear Layer, a ReLU activation function, and Batch Normalization

Embedding Layers:

The embedding layers are size N-1 by a calculated layer size using the following formula: $1.6 * N^{0.56}$, where N is the cardinality of a categorical variable. These embeddings are based off of Word2Vec[5] and anecdotal findings.

Feature Selection:

Using permutation importance, a selection of features were used that showed to have a relative positive impact on the trained model.

Results

Feature Importance:

TCP Checksum was the most important feature. This is due to its cardinality. Our TCP Checksums had 57,616 giving an embedding matrix of 57,617 by 600 for the model to use and extrapolate from. From these the model was able to discern patterns between both individual machines and overall operating systems.

Results (cont)

Feature Importance

Grouped Operating Systems	
Variable	**Δ Accuracy**
TCP Checksum	0.379748
IP Time to Live	0.128144
Packet Length	0.057037
TCP Window size value	0.054061
TCP Calculated window size	0.041972
TCP Segment Length	0.013613
TCP Window size scaling factor	0.008011
Protocol	0.007163

Individual Machines	
Variable	**Δ Accuracy**
TCP Checksum	0.466541
IP Time to Live	0.145924
Packet Length	0.035088
Protocol	0.018302
TCP Calculated window size	0.012140
TCP Window size scaling factor	0.009189
TCP Window size value	0.008296
TCP Segment length	0.002891

Operating System Identification

Grouped Operating Systems				
	Architecture 1 with all Variables	Architecture 1 with Feature Selection	Architecture 2 with all Variables	Architecture 2 with Feature Selection
Validation	92.73%	**92.78%**	92.11%	**92.29%**
Test	92.75%	**92.77%**	92.14%	**92.31%**

Individual Machines				
	Architecture 1 with all Variables	Architecture 1 with Feature Selection	Architecture 2 with all Variables	Architecture 2 with Feature Selection
Validation	83.29%	**86.92%**	87.02%	**86.98%**
Test	83.22%	**86.99%**	87.01%	**87.02%**

Future Work

Future work involves further analyzing the efficiency based on TCP, UDP, and other categories, as well as discerning if Checksum is important to the other packet types.

References

[1] Information Sciences Institute , "Internet Protocol: DARPA Internet Program Protocol Specification," September 1981. [Online]. Available: https://tools.ietf.org/html/rfc791. [Accessed 8 March 2019].
[2] Information Sciences Institute, "Transmission Control Protocol: DARPA Internet Protocol Specification," September 1981. [Online]. Available: https://tools.ietf.org/html/rfc793. [Accessed 8 March 2019].
[3] A. Aksoy and M. H. Gunes, "Operating System Classification Performance of TCP/IP Protocol Headers," in IEEE 41st Conference on Local Computer Networks Workshops, 2016.
[4] fastai. (2019, April 1). fast.ai: Making neural nets uncool again. Retrieved from fast.ai: https://www.fast.ai/
[5] T. Mikolov et al, "Distributed Representations of Words and Phrases and their Compositionality," 2013