Malware Detection with Malware Images using Deep Learning Techniques

Author: Ke He
Supervisor: Dong-Seong Kim

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science with Honours in the
Department of Computer Science and Software Engineering

October 18, 2018
Abstract

Department of Computer Science and Software Engineering

Bachelor of Science with Honours

Malware Detection with Malware Images using Deep Learning Techniques

by Ke He

Driven by economic benefits, the number of malware attacks is increasing significantly on a daily basis. Malware Detection Systems (MDS) is the first line of defence against malicious attacks, thus it is important for malware detection systems to accurately and efficiently detect malware. Current MDS typically utilizes traditional machine learning algorithms that require feature selection and extraction, which are time-consuming and error-prone. Conventional deep learning based approaches use Recurrent Neural Networks (RNN) which is vulnerable to redundant API injection, thus we investigate the effectiveness of Convolutional Neural Networks (CNN) against redundant API injection. We designed a malware detection system that transforms malware files into image representations and classifies the image representation with CNN. The CNN is implemented with spatial pyramid pooling layers (SPP) to deal with varying size input. We evaluate the effectiveness of SPP and image colour space (greyscale/RGB) by measuring the performance of our system on both unaltered data and adversarial data with redundant API injected. Results show that naive SPP implementation is impractical due to memory constraints and greyscale imaging is effective against redundant API injection.
Contents

1 Introduction 1

2 Background 3
   2.1 Malware Detection Systems ........................................ 3
      2.1.1 Signature-based Malware Detection ................................ 3
      2.1.2 Heuristic-based Malware Detection ................................ 3
      2.1.3 Cloud-based Malware Detection ................................... 4
      2.1.4 Limitations and Beyond ........................................... 6
   2.2 Deep Learning ......................................................... 7
      2.2.1 Artificial Neural Networks ........................................ 7
      2.2.2 Convolutional Neural Network ..................................... 9
      2.2.3 Recurrent Neural Network ....................................... 10
   2.3 Deep Learning in Malware Detection ................................. 11
      2.3.1 Recurrent Neural Networks ....................................... 11
      2.3.2 Convolutional Neural Networks ................................... 11

3 Design 13
   3.1 Preprocessing ......................................................... 13
   3.2 Classification ......................................................... 14
   3.3 Evaluation ............................................................ 15

4 Experiments 17
   4.1 Dataset .............................................................. 17
   4.2 Experiment Setup ..................................................... 17
   4.3 Unaltered Samples .................................................... 18
      4.3.1 Time Measurements ............................................... 21
   4.4 Results Comparison .................................................... 21
   4.5 Adversary Samples .................................................... 22

5 Discussion 26
   5.1 Findings .............................................................. 26
      5.1.1 Colour Space ...................................................... 26
      5.1.2 Spatial Pyramid Pooling ......................................... 26
   5.2 Limitations ............................................................ 27
   5.3 Future Work ........................................................... 27

6 Conclusion 29

Bibliography 31

A Raw Experiment Results 36
   A.1 Fixed width ............................................................. 36
   A.2 Fixed Ratio ............................................................. 38
   A.3 Adversary Data .......................................................... 41
A.3.1 RGB with Resnet50 ............................................. 41
A.3.2 Grayscale with Resnet50 ................................. 43
A.3.3 Grayscale with Plain Network ......................... 44
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Signature of a network attack on Windows SMB protocol</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of the difference between NN and 5-NN, using 2D points and 3 classes. Coloured regions are decision boundaries [24].</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>A feature space that cannot be partitioned by any plane is shown on the left. The kernel function $f(x,y) = (x,y,x^2 + y^2)$ turns the 2-dimensional sample space into 3-dimensional space, where a hyperplane can be fitted [29].</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Left: a biological neuron. Right: a computational neuron [36]</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>An example neural network with 3 inputs, 2 hidden layers of 4 neurons and one output layer [36]</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>Unfolding a Recurrent Neural Network. $x_t$ represents the input at time $t$, $o_t$ represents the output at time $t$ and $s_t$ represents the memory at time $t$, calculated based on $x_t$ and $s_{t-1}$. $U, V, W$ are the shared parameters of the network. [5]</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>The structure of malware detection system</td>
<td>13</td>
</tr>
<tr>
<td>3.2</td>
<td>A network with spatial pyramid pooling layer that pools feature maps of any size into fixed-length (16+2+1=19 in this case) arrays [7]</td>
<td>15</td>
</tr>
<tr>
<td>3.3</td>
<td>Left: the Resnet 50 model. Right: the 3 layer plain network</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>RoC Curves of models based on 3 layer plain network</td>
<td>19</td>
</tr>
<tr>
<td>4.2</td>
<td>RoC Curves of different models based on resnet50 network</td>
<td>19</td>
</tr>
<tr>
<td>4.3</td>
<td>RoC Curves of different models based on 3 layer plain network</td>
<td>20</td>
</tr>
<tr>
<td>4.4</td>
<td>RoC Curves of different models based on resnet50 network</td>
<td>20</td>
</tr>
<tr>
<td>4.5</td>
<td>RoC Curves of resnet50 model with RGB input and no SPP on adversary inputs</td>
<td>23</td>
</tr>
<tr>
<td>4.6</td>
<td>RoC Curves of resnet50 model with RGB input and no SPP on adversary inputs</td>
<td>23</td>
</tr>
<tr>
<td>4.7</td>
<td>RoC Curves of plain model with greyscale input and no SPP on adversary inputs</td>
<td>24</td>
</tr>
<tr>
<td>4.8</td>
<td>RoC Curves of resnet model with greyscale input and no SPP on adversary inputs</td>
<td>24</td>
</tr>
</tbody>
</table>
List of Tables

4.1 Summary of experimental parameters ........................................ 17
4.2 Summary of training parameters .............................................. 17
4.3 Results of the experiment ..................................................... 18
A.1 Plain 3 layer network with RGB input and no SPP ....................... 36
A.2 Plain 3 layer network with RGB input and SPP ........................... 36
A.3 Plain 3 layer network with greyscale input and no SPP .................. 36
A.4 Plain 3 layer network with greyscale input and SPP ..................... 37
A.5 Resnet50 network with RGB input and no SPP ........................... 37
A.6 Resnet50 network with RGB input and SPP ............................... 37
A.7 Resnet50 network with Greyscale input and no SPP ...................... 38
A.8 Resnet50 network with Greyscale input and SPP .......................... 38
A.9 Plain 3 layer network with RGB input and no SPP ....................... 38
A.10 Plain 3 layer network with RGB input and SPP ......................... 39
A.11 Plain 3 layer network with greyscale input and no SPP ................ 39
A.12 Plain 3 layer network with greyscale input and SPP ................... 39
A.13 Resnet50 network with RGB input and no SPP .......................... 40
A.14 Resnet50 network with RGB input and SPP ............................. 40
A.15 Resnet50 network with greyscale input and no SPP ..................... 40
A.16 Resnet50 network with greyscale input and SPP ....................... 41
A.17 Input file saturated by 10% with NOPs ................................. 41
A.18 Input file saturated by 20% with NOPs ................................. 41
A.19 Input file saturated by 30% with NOPs ................................. 42
A.20 Input file saturated by 40% with NOPs ................................. 42
A.21 Input file saturated by 50% with NOPs ................................. 42
A.22 Input file saturated by 10% with NOPs ................................. 43
A.23 Input file saturated by 20% with NOPs ................................. 43
A.24 Input file saturated by 30% with NOPs ................................. 43
A.25 Input file saturated by 40% with NOPs ................................. 44
A.26 Input file saturated by 50% with NOPs ................................. 44
A.27 Input file saturated by 10% with NOPs ................................. 44
A.28 Input file saturated by 20% with NOPs ................................. 45
A.29 Input file saturated by 30% with NOPs ................................. 45
A.30 Input file saturated by 40% with NOPs ................................ 45
A.31 Input file saturated by 50% with NOPs ................................ 46
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>malware</td>
<td>malicious software</td>
</tr>
<tr>
<td>AV</td>
<td>Anti-Virus</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>MDS</td>
<td>Malware Detection System</td>
</tr>
<tr>
<td>NN</td>
<td>Nearest Neighbour</td>
</tr>
<tr>
<td>kNN</td>
<td>k Nearest Neighbour</td>
</tr>
<tr>
<td>NBC</td>
<td>Naive Bayes Classifier</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>SPP</td>
<td>Spatial Pyramid Pooling</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>RGB</td>
<td>Red Green Blue</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
</tr>
</tbody>
</table>
1 Introduction

Recent advances in hardware and software technologies have made digital devices extremely powerful. Individuals and co-operations rely heavily on servers and computers to exchange and process information over the internet on a daily basis. It is estimated that there are 3.5 billion internet users as of 2017 and continues to grow rapidly [1]. With a promising future ahead for the digital technologies industry, comes severe consequences. Driven by economic benefits, digital devices are under constant threat of malicious software (malware).

Malware is an umbrella term for any software that intends to violate the confidentiality, integrity and availability of any digital device or system [2]. Attackers typically utilise a wide range of malware, such as viruses, worms and Trojan horses to steal confidential information, hijack computers, crash servers and many other malicious activities. Such malicious activities could potentially cost the global community 500 billion in total [3] and worldwide cyber-security spending is estimated to reach 86.4 billion [4]. In order to protect individuals and large corporations from such a catastrophe, it is important to ensure their digital devices and systems are used in a safe and secure environment.

The first line of defence against malicious attacks is malware detection system. They are distributed by anti-virus (AV) vendors to examine whether a file is malicious or benign. Traditional malware detection systems are based on shallow machine learning algorithms (the term "shallow machine learning" or "traditional machine learning" both refers to machine learning algorithms that are not deep learning, e.g. decision trees, support vector machines and naive Bayes classifier). The performance of traditional machine learning algorithms relies heavily on the quality of features extracted, however, feature selection and extraction are extremely time-consuming, error-prone and requires a high level of knowledge in the field. The next generation of machine learning algorithms, deep learning, has become very popular in recent times due to its ability to automatically extract sophisticated, high-level features that lead to high accuracy [5].

Current deep learning based malware detection systems are mostly based on Recurrent Neural Networks (RNN) with API calls and machine instructions as input and have been shown to have high accuracy. However, RNNs can be vulnerable to adversarial attacks where the attacker mimics the RNN used in the MDS based on the input and output and use an adversarial RNN to add redundant API calls. The file with redundant API calls injected can easily bypass RNN detection [6]. The robustness and effectiveness of using RNN in malware detection are still questionable.

The goal of this project is to investigate the resilience of Convolution Neural Network (CNN) against redundant API injection. We theorise that adding redundant API calls corresponds to a distortion or translation of malware features in the image space, which the CNN can still find. To achieve this task, we first transform the input file into an image and train our CNN to learn the features/textures of the image. A major problem is that conventional CNN takes fixed size inputs, but malware files can come in drastically different sizes, thus we deploy Spatial Pyramid Pooling (SPP) [7] to make our CNN take inputs of arbitrary size.
The rest of this report is organised as follows: Chapter 2 provides background of shallow and deep machine learning algorithms used in malware detection systems, Chapter 3 showcases the design of our proposed malware detection system and the respective rationale behind them, Chapter 4 examines the performance of our proposed malware detection system, Chapter 5 discusses our finding from the experiment as well as any further work and limitations. Finally, we conclude this report in Chapter 6.
2 Background

2.1 Malware Detection Systems

Malware detection systems have made a series of transformations from the early signature-based detection to the modern cloud-based detection. This section gives a brief summary of signature-based and heuristic-based malware detection systems while giving an in-depth analysis of traditional machine learning algorithms used in cloud-based malware detection.

2.1.1 Signature-based Malware Detection

Early anti-malware vendors (e.g., Kingsoft, MacAfee and Symantec) commonly used signatures to identify malware [8]. A signature is a unique sequence of bytes exhibited by known malware samples and extracted by experts. An example of a network attack signature is shown in Figure 2.1. Detection was done by simply checking whether the file contains any known malware signatures. Signature-based detection is able to identify malware with an extremely small error rate and false positive rate of below 0.1% [9] but it is prone to countermeasures such as polymorphism, metamorphism and code obfuscation. The process of signature extraction is very time-consuming and error-prone. On average it takes 54 days between a new malware’s release and its detection [8] and fast-spreading malware such as viruses and worms would have already caused severe damage by the time malicious signatures are found. Due to such limitations, the focus of identifying malware was shifted to be based on heuristics rather than signatures.

2.1.2 Heuristic-based Malware Detection

Heuristic-based malware detection is a slight improvement of the signature-based detection. Instead of extracting a unique signature which only covers one variant of a malware type [9], a set of heuristics are used. An ideal set of heuristics is composed of rules and patterns that sufficiently describe features belonging to all variants of the same type of malware yet not falsely matched on benign software [10]. An input file is matched against a collection of malicious heuristics to determine its malevolence. This method is able to cover a range of malware samples with a single set of heuristics thus dramatically the reduces the workload of heuristic engineers. However, heuristics do not come for free and requires large amounts of manual labour for extraction, which was made infeasible by the rise of malware creation tool-kits.

Malware creation tool-kits are designed so that even someone with no programming knowledge can develop their own malware. This causes a rapid growth in the number of new malware samples (200,000 new samples per day [11]) making manual extraction of heuristics infeasible. The growing malware sample size has also taken a toll on the client’s disk space, as it is not possible to store all heuristics on client machines. A more scalable architecture and intelligent detection algorithm
are required for malware detection systems to remain effective, thus the cloud-based malware detection system is born.

### 2.1.3 Cloud-based Malware Detection

Current malware detection systems developed by AV vendors typically adopt the client-server model [12][13]. A lightweight client is installed on the host machine, with a white-list to authenticate known benign software and a blacklist to block known malware. All other software is transmitted to the cloud/server, where detection and classification take place. This architecture minimises computations done on the client side, and the cloud/server side is able to rapidly and reliably classify unknown software with machine learning algorithms.

Cloud/server side classification generally requires feature extraction and classification. The feature extractor extracts a number of features from a file and the extracted feature are the inputs of the classifier. We investigate the feature extraction and classification stage in detail.

**Feature Extraction**: Features are characteristics of a file, for example, the permissions required or the serial number. A good selection of features should exhibit a difference between malicious and benign samples. Selecting excessive features often cause overfitting and requires more computational power, thus the set of features should be a concise yet descriptive representation of a file. Based on how the features are selected and extracted, the feature extraction process can be categorised into two main categories: static and dynamic analysis.

Static analysis is performed without the execution of the program, for example analysing the raw source code or opcodes. Common properties to examine in static analysis are permissions [14][15], n-grams [16] and strings [9][17].

Dynamic analysis, on the other hand, is performed with the execution of code and observes the behaviour of the code to extract features. The malware files are often executed in debuggers, emulators and virtual machines [8]. From executing malware files, API calls [18] and system calls [19] can be extracted and used for classification. A novel property such as power consumption of different hardware over execution cycle has been examined and achieved reasonable success [20].
Both static and dynamic analysis has its own advantages and disadvantages. Static analysis is able to perform a comprehensive analysis of malware by exploring all code paths, but static analysis’ focus on source code made it vulnerable to code obfuscation and encryption. Dynamic analysis is resilient to obfuscation by focusing on the file’s behaviour, but unable to examine edge cases that are hidden in the software. Due to the limitations of the static and dynamic analysis, malware detection systems often combine the two in practice to perform hybrid analysis [21], which provides a more comprehensive coverage of features.

**Classification:** Once features from malware file and benign files are extracted, it is up to machine learning algorithms to learn each sample so that it is able to classify unseen files. The features extraction process transforms a file into a point in an n-dimensional feature space, where n is the number of features extracted, and the classification process aims to segment the feature space into multiple sections based on the training data, where each section contains the maximum possible number of points belonging to the same category. Popular classification algorithms that we will examine include Nearest Neighbour, Naive Bayes Classifier and Support Vector Machine.

Nearest Neighbour (NN) is one of the oldest and simplest classification algorithms [22]. The philosophy behind NN is that objects belonging to the same category possess similar characteristics so that they are spatially close to each other in the feature space. Original NN algorithm classifies a new sample to be the same as its closest neighbour, but this approach is prone to overfitting the data. A variant of NN called k-Nearest-Neighbour (kNN) is favoured in practice where the closest k neighbours are found. The classification is based on the k neighbours found through some metric and usually, the category with the highest frequency is chosen. The exact metric used and the number of k depends on the particular problem and is generally determined empirically. Several malware detection models use variants of the kNN algorithm and have shown promising results [17][23].

![Figure 2.2: An example of the difference between NN and 5-NN, using 2D points and 3 classes. Coloured regions are decision boundaries [24].](image)

Naive Bayes Classifier (NBC) [25] calculates the probability of a new file belonging to a particular category based on the features observed. The algorithm is based on Bayes theorem of conditional probability which states that:

\[
p(A|B) = \frac{p(B|A)p(A)}{p(B)}
\]

Thus, given a sample with feature vector \( \mathbf{v} = (v_1, v_2, ..., v_n) \) representing n features, we wish to calculate

\[
p(C_m|\mathbf{v})
\]
Where \( C_m \) is \( m \) possible categories to be classified. Using Bayes’ Theorem, the above equation can be transformed as

\[
p(C_m|v) = \frac{p(C_m)p(v|C_m)}{p(v)}
\]

Note that the denominator is a constant, therefore only the numerator is of interest. The numerator is equivalent to the joint probability model \( p(C_m, v_1, v_2, ..., v_n) \). Using repeated definition of conditional probability and the naive assumption that features \( v_1, v_2, ..., v_n \) are all conditionally independent, the numerator can be written as

\[
p(C_m, v_1, v_2, ..., v_n) = p(v_1|v_2, ..., v_n, C_m)p(v_2, ..., v_n, C_m)
\]

\[
= ... \\
= p(v_1|v_2, ..., v_n, C_m)p(v_2, ..., v_n, C_m)...p(v_n|C_m)p(C_m)
\]

Due to conditional independence,

\[
p(v_i|v_{i+1}, ..., v_n, C_m) = p(v_i|C_m)
\]

Thus,

\[
p(v_1|v_2, ..., v_n, C_m)...p(v_n|C_m)p(C_m) = p(v_1|C_m)p(v_2|C_m)...p(C_m)
\]

\[
= p(C_m) \prod_{i=1}^{n} p(v_i|C_m)
\]

We have assumed independence of features when using NBC, but it has been empirically shown to be optimal even when the independence assumption is violated by a large margin [26]. This observation has made NBC applicable across numerous domains. Various malware detection systems have used NBC for classification and achieved good success [14][27].

Support Vector Machine (SVM) [28] is a type of binary classifier that seeks for the optimal hyperplane (or support vectors) that partitions the feature space in two. The optimal hyperplane is determined by maximising the distance between the hyperplane and the closest sample data point from both categories so that the generalisation error is minimised. An important element of SVM is the kernel function. The kernel function is able to transform a linearly inseparable dataset into higher dimensions so that it can be linearly separated, as shown in Figure 2.3 [29]. The choice of the kernel function is crucial for SVM to correctly classify the dataset as not all datasets can be made separable from the same kernel function. Common nonlinear kernel functions used are radial basis function kernel, fisher kernel and polynomial kernel [8]. Kernel functions do not have to operate on numerical data, kernels that operate on other data-types, for example, string kernels [30] and graph kernels [31] exists. Multiple MDS have used SVM as the classifier and achieved success [32][33].

2.1.4 Limitations and Beyond

Conventional cloud-based malware detection systems are based on traditional machine learning algorithms. Such algorithms’ performance is largely dependant on the features selected and extracted, but it requires a considerable amount of domain knowledge and is extremely time-consuming [5]. A class of machine learning algorithms that do not require manually designed feature extractor while providing high accuracy and classification speed is deep learning. Deep learning has made major
Chapter 2. Background

Figure 2.3: A feature space that cannot be partitioned by any plane is shown on the left. The kernel function $f(x, y) = (x, y, x^2 + y^2)$ turns the 2-dimensional sample space into 3-dimensional space, where a hyperplane can be fitted [29].

break-through across multiple domains such as image recognition, speech recognition and natural language processing [5]. Future cloud-based malware detection systems are highly likely to deploy deep learning algorithms at the cloud end for reliable and responsive detection and classification. We introduce deep learning and examine current deep learning techniques applied in malware detection and other related fields in the next section.

2.2 Deep Learning

The history of deep learning can be traced back to as early as the 1940s. Due to limited computation power and inexperienced training strategies at the time, early deep learning algorithms such as artificial neural networks nearly always resulted in a locally optimal solution and convergence is not guaranteed. It was not until 2006 when Hinton et al. [34] proposed backpropagation, an efficient way of training and tuning the network, causing deep learning algorithms to be in favour [35].

Deep learning’s success is largely due to the absence of manually designed feature extractor. Deep learning algorithms are able to automatically discover features or representations that are needed for classification straight from raw data. In other words, the feature extractor is designed by the algorithm itself. Such feature extractors take less time, are less error-prone and most importantly, are able to extract intricate high dimensional features which humans cannot think of [5].

2.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) or connectionist systems lays the foundation of modern deep learning algorithms and were originally inspired by modelling the biological neural system. The basic building blocks of the biological neural system is a neuron, connected by synapses. Each neuron receives input signals from its dendrites, then the input is processed by the cell nucleus and outputs signals along its axon via synapses. The outputted signal is then received by dendrites of other neurons. The computational model of a neuron is an extremely simplified version of the biological neuron: dendrites and axons are mapped to the input and output of
the computational neuron, synapses are products of the input and weights and the cell nucleus is modelled by an activation function, as shown in Figure 2.4 [36].

Unlike the biological neural system where neurons are clustered and located randomly, neurons in the computational model are organised in layers. The first layer is the input layer and the last layer is the output layer consists of size n, where n is the number of distinct categories for classification. All layers in between are the hidden layers (see Figure 2.5 for architecture). Each neuron is fully connected to all the neurons in the previous and next layer, but not connect to any of the neurons in the same layer. Neurons in each layer receive inputs (e.g. $x_0$) from previous layer, and synapse (e.g. $w_0$) interacts multiplicatively with each input (e.g. $w_0x_0$). The neuron then computes the sum of all synapse interactions, add a bias term, and feeds the sum into an activation function. The output of the activation function is passed forward to the input of neurons in subsequent layers. This process continues until the output layer, where the neuron with the highest output determines the category that the input belongs to. During the training process, the output of each sample is compared with the expected output. The difference between expected and actual values of each neuron is used for backpropagation [37], which slightly adjusts the weights and biases of every neuron in the hidden layer. The core idea behind artificial neural networks is that backpropagation is able to adjust the weights and biases slightly for each sample, and given sufficient samples the weights and biases will converge, giving optimal classification performance [36]. As a side note, The biological model is just an inspiration for neural networks, the computational model is significantly simplified version and current neural network models have deviated significantly from the biological model [38].
Regular neural networks do not scale well to any practical considerations. As an example, a small-sized image are typically $200 \times 200 \times 3$ (200 by 200 pixels with 3 colour channels) and each neuron would need $200 \times 200 \times 3 = 120,000$ weights. Clearly, this is not scalable and a large number of weights are likely to overfit the data [39].

2.2.2 Convolutional Neural Network

Convolutional Neural Networks (CNN) considers each layer to be 3-dimensional volumes rather than a one-dimensional array in ANN. This allows each layer in the network to transform the volume into various sizes and gradually reduce the volume to save computation power. CNN also has mainly two new layers, convolutional layers and pooling layers, that helps the network to classify data with high precision and speed. Convolutional layers in CNN are locally connected as opposed to fully connected, which is both energy efficient and allows features to be found regardless of position. Pooling layers reduce the volume so that only the significant patterns are found and noise is removed. The combination of convolutional layer and pooling layers acts as a feature extractor, the classification is done with fully connected layers similar to ANN with features extracted from convolutional layers and pooling layers as input.

**Convolutional Layer:** The convolutional layer is the core building block of ConvNet. It consists of multiple filters where each filter has a small dimension along width and height but extends fully through the depth of the input volume. Each filter is slid (convolved) across the full width and height of the input volume and computes the dot product of the filter with input. How far filters slide after each operation (stride) can be varied depending on the input size. For each filter, a 2-dimensional activation map corresponding to the responses of the filter at every position is generated. The activation maps are then stacked to produce the output volume [39]. Using multiple small size filters, the convolutional layer is able to take advantages of local groups of pixels that are highly correlated. Also, the property that features are important regardless of its position (spatial invariant) enables neurons to share the same parameters [5]. The parameters in a depth slice (single 2-dimensional slice with the depth value of one) can be shared with all other neurons in the same depth slice. This means instead of storing weights and biases for all neuron in a depth slice, only one set of weights are biases is stored per slice, which drastically reduces the number of parameters [39]. Of course, there are times when spatial invariant assumption does not hold, for example, when recognising centred faces we would expect eye-specific features to be dependant on spatial location. Under such circumstances, it is common to relax parameter sharing.

**Pooling Layer:** The pooling layer’s role is to merge several smaller features into a more general feature [5], for example, several edges detected in the input are merged into a hand. Similar to the convolutional layer, the pooling layer also has a filter, but with a depth value of one and typically computes the maximum value of the filter. The most common filter size is $2 \times 2 \times 1$ with a stride of 2, effectively halving the input along both width and height. The pooling layer not only reduces dimensions of input volume but also prevents overfitting by considering a more general and abstract representation [39]. However, it has been shown by Springenberg et al. [40] that pooling layers can be replaced with convolutional layers with larger stride with no loss in accuracy.

ConvNets are feedforward networks, where neurons are activated precisely once during each classification and have no notion of time and context. Such property
Figure 2.6: Unfolding a Recurrent Neural Network. $x_t$ represents the input at time $t$, $o_t$ represents the output at time $t$ and $s_t$ represents the memory at time $t$, calculated based on $x_t$ and $s_{t-1}$. $U, V, W$ are the shared parameters of the network. [5]

...made ConvNets perform well in image classification tasks where features are spatially invariant, but poorly in natural language processing and speech recognition tasks due to the order and context of the words matters significantly.

2.2.3 Recurrent Neural Network

Recurrent neural networks (RNN) are designed to recognise patterns in sequential data, which made them favourable in natural language processing tasks. The design philosophy of RNN is that the order of inputs contains information as well as the inputs. The overall structure of RNN is similar to CNN, both have an input layer, an output layer and hidden layers. What is different is that RNN’s hidden layer not only considers the input at the current step but also the previous states of the system by recursively activating itself. The hidden layers have a “memory” of what happened previously. The structure of a common RNN is shown in Figure 2.6 [5].

Unfortunately, training the network has proven to be problematic [35]. Memorising long-term behaviour of the system is difficult because the back propagated gradients either grows or shrinks at each time step, causing the gradient to either explode or vanish. As all neural networks rely on gradients to adjust its weights and biases, a gradient of 0 or infinity makes the network untrainable. This phenomenon is largely due to information passes through many multiplicative stages during RNN. Similar to compound interest, an interest rate slightly higher than one can become unimaginably large [41].

Long Short-Term Memory networks (LSTMs) [42] are a widely used variant of RNNs that fixes the exploding/vanishing gradient problem by having a more complicated memory unit. The memory units in LSTM have its own weights and biases that are learning alongside with RNN and are able to decide whether the new input is ignored or impact current memory or forget existing memory (e.g. contexts in a new paragraph or section may be irrelevant to previous paragraph/section). To further reduce the exploding/vanishing rate, the LSTM memory unit adds the input with previous states rather than multiply the two. This naively simple change of calculation has proven to be quite effective.
2.3 Deep Learning in Malware Detection

Deep learning has made major advances across multiple domains such as image recognition, speech recognition, natural language processing and many more [5]. It has even extended into more recreational domains such as automatically generating dank memes [43]. It is not surprising that some of the latest MDS uses deep learning techniques to achieve high accuracy and speed. This section explores current deep learning methods adopted in malware detection system.

2.3.1 Recurrent Neural Networks

Recurrent neural networks (RNN) have been the first choice of most deep learning based malware detection systems. The task of malware detection is analogous to text classification, which RNNs excel at. Designers of RNN-based MDS believe the sequential information contained within machine instructions and API calls extracted from the malware is useful to detect malware, just like text classification [44][45]. A typical RNN based malware detection system encodes each API call/instruction as a one-hot vector of dimension $M$, where $M$ is the number of total API calls/instructions. Given API calls/instructions are labelled from 0 to $M-1$, for an API call/instruction labelled $i$, all numbers in the vector are 0 except for index $i$, where the number is 1. A sequence of embedded API calls is gathered into a matrix and is fed into the RNN for classification. Several variations of RNN have been evaluated [46], showing LSTM network with max pooling and logistic regression as classification outperforms character-level CNN.

A major drawback of RNN is its ability to only learn one language environment or structure. Attackers can exploit this weakness by predicting the language that a particular RNN based malware detection system has learned with a substitute RNN and use another RNN to generate adversarial code to bypass malware detection systems. A simple yet effective way of generating adversarial code is to insert irrelevant and redundant API calls, and most of the generated adversarial code is able to bypass RNN detection [6]. The effectiveness of RNN in malware detection is uncertain.

2.3.2 Convolutional Neural Networks

Convolutional neural network (CNN) extracts hierarchical local features from data samples regardless of location, thus frequently used for image classification. Nevertheless, we theorise CNN is a better candidate for malware detection compared to RNN. In the image space, adding redundant API calls or machine instructions corresponds to a translation or distortion of features, which a CNN can be trained to recognise. As a trivial example, a hand in an image is still a hand regardless of its position or orientation, but a hand in a text segment can be a noun or a verb depending on the context. With RNN-based MDS, attackers can change the context to trick the MDS to think the word "hand" is a noun when it should be a verb by adding "the" in front of it. CNN on the other hand, adding words around the image would distort or translate the feature, which the CNN can easily recognise.

Unfortunately, classifying malware with CNN have not been extensively discussed in literature. Yuan et. al. proposed an MDS that extracts a set of features with hybrid analysis and use CNN to classify the extracted feature vectors [47]. Simply using just a CNN with API calls or machine instructions were rarely done. We suspect it is mainly due to two reasons: 1. Finding a good method of representing a
malware file as an image is difficult. 2. It is difficult to deal with files/images of various sizes.
3 Design

The goal of this project is to examine the effectiveness of CNN in malware detection and its resilience to redundant API injection. With the goal in mind, we designed an MDS consists of three sequential modules: preprocessing, classification and evaluation, as shown in Figure 3.1. We will explain each module in detail below and discuss how they address problems that arise with using CNN for malware detection.

3.1 Preprocessing

The main goal of the preprocessing module is to convert a raw input malware file into a matrix of values which the classification module is able to process. This solves the first problem identified previously. A commonly used method to transform categorical data to numerical data in various deep learning algorithms is one-hot encoding. For text classification, the encoding is commonly applied at character level [48], and for malware detection, encoding is applied at API/instruction level. A drawback of using one-hot encoding is that the transformed data will be large and consist of mostly white spaces, which is not memory efficient. Studies have also shown that character level CNN for malware detection is has a worse performance compared to LSTM networks [46].

A novel, simple and effective way of transforming input files is to compile the malware source code into a binary file and transform the binary file into an image. Each byte ranges from 0 to 255 so it is able to directly represent a pixel in the greyscale image of the malware [49]. It has also been observed that the grey scale images of malware belonging to the same family exhibits similar layout and texture, and grey-scale images have been found to be resilient to section encryption, which is a popular code obfuscation technique [49].

Our preprocessing model chooses to convert malware files to greyscale images as it is simple yet effective. We have also created another conversion method that transforms malware files into RGB images. For RGB images, a group of 3 bytes

![Figure 3.1: The structure of malware detection system](image-url)
represents each pixel in the output image, corresponding to red, green and blue
colour channels respectively. The bytes are processed line by line and the output
pixels are placed sequentially in a row, with the width of the output image fixed to
an arbitrary number of 1920 pixels (it is the width of monitor screen). Any newline
characters are ignored and if the last few bytes cannot fill the entire line, it is padded
with black pixels.

The reason behind transforming into RGB images is that we theorise it will ex-
hibit better performance. CNN considers each input as a volume, for greyscale
image of size $n \times m$, the input volume is $n \times m \times 1$, whereas for RGB images, the
volume would be $n/\sqrt{3} \times m/\sqrt{3} \times 3$. This shortens the distance between each pair
of bytes and allows CNN to find more complex patterns in the malware file.

### 3.2 Classification

The classification module takes the transformed image from the preprocessor and
uses that image to train the CNN. Malware files come in various sizes, however
typical CNN only accept a fixed size input. There are several ways to modify the
input to a fixed size, described below.

- Gilbert [50] proposed a malware detection system based on greyscale image
  representation of malware and down-sampled all malware samples to $32 \times 32$
pixels.
- CNN based image classification methods either crop or warp the original im-
  age to the required size [7].
- Athiwaratkun et al. [46] limited the input to be 1014 characters in the character-
  level CNN and padded shorter ones with zeros.

Modifying the input image is not ideal for malware detection, as it is prone to
information loss before detection stage. Attackers can deliberately put malicious
code in places that are potentially going to be removed before classification, thus
bypassing detection system. For example, if the detection system only takes the first
$32 \times 32$ (1024) pixels as an input, the attacker can simply add the malicious code
after 1024 pixels, bypassing detection.

Manipulating the input file is not ideal, therefore can we design the CNN so
that it accepts input of various sizes? The answer is yes and the method is spatial
pyramid pooling (SPP) [7]. The two main parts of a typical CNN are convolutional
layers and fully connected layers. The convolutional layer operates in a sliding win-
dow manner and the output is dependent on the size of the input. It is only the
fully connected layers that require a fixed size input to output a fixed size output of
the categories. SPP operates in between the last convolutional layer and fully con-
nected layer and divides the features maps from the last convolutional layer into
multiple levels and multiple subsections. Each subsection in each level undergoes
max pooling to extract the most dominant feature and each feature is concatenated
into a fixed-length output array for the fully connected layers. The SPP layer al-
 lows the neural network to take an input image of arbitrary size and outputs a fixed
sized feature array for the fully connected layers. The operation of SPP is shown in
Figure 3.2.

Before training our neural network, we preprocess the images by rescaling all
pixel from 0 to 255 into 0 to 1 and apply sample-wise normalization. These prepro-
cessing steps are commonly done in order to make sure each feature in the image
have equal importance to classify the image, thus increases the accuracy of the network. The classification module uses two different neural networks to train the image, one is a simple 3 convolutional layer network with batch normalisation [51] and drop-out [52] before the final fully connected layer, and the other is a more complex resnet50 [53]. Each neural network has 2 variations, with SPP or without SPP. The SPP layers contain 3 levels, with 1, 2, and 4 divisions along each dimension. With SPP, the input image is not modified for evaluation. For training, each image is resized to (256, 256), (256, 151) and (256, 433) which represents three main ratios of the malware image: square, wide and thin, and corresponds to medium, small and large files. This is because fixed size input allows faster training times and training the network with multiple different sizes allows the network to be more flexible, as suggested by the authors of SPP [7]. Without SPP, the input image is resized to (256, 256) using bilinear interpolation for both training and evaluation stage. The model trains on 90% of the sample data provided. The two network models are shown in Figure 3.3.

The goal of this project is to evaluate the effectiveness of the SPP layer and input colour space against redundant API injections. We are not too concerned with the actual neural network being used. The reason we used two neural networks is to see the interaction of different networks with various configurations and predict the result of using a more complex network.

### 3.3 Evaluation

The evaluation module classifies unseen files as malware or benign file and evaluates the accuracy of our classifier. The output of the CNN is a number between 0 and 1 indicating the likelihood of the files being malicious (1) or benign (0). We introduce
a parameter called threshold so that values above the threshold is considered malicious and otherwise benign. For evaluation, the module uses the remaining 10% of the sample data and generates a confusion matrix. From the confusion matrix, we can calculate accuracy metrics such as precision, recall, false positive rate and f1 score.

Theoretically, the evaluation module with SPP is able to take images of arbitrary size, but in reality, we are constrained by the memory of the computer. There is simply not enough memory to predict a 70 MB file with resnet50. To solve this problem, we have adopted a "divide and conquer" approach. The idea is to divide the input image into smaller, ratio-preserving sub-images and classify each sub-image. If any of the sub-images are classified as malicious, then the entire image will be classified as malicious. The idea behind it is that the sub-groups will exhibit a texture that is similar to a malware texture and the classifier is able to correctly classify the sub-image.

For models that do not have SPP enabled, we simply resize the image to (256, 256) with bilinear interpolation.
4 Experiments

The following experiments are done in order to evaluate the performance of the designed MDS against redundant API injection. The experiments mainly examine the effect of the input colour space, complexity of CNN and spatial pyramid pooling on the performance of our network, with some additional experiments done to prove or disprove our theory. For simplicity, we only consider two distinct levels for each parameter. Table 4.1 shows the configurable parameters and its respective levels.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Level 1</th>
<th>Level 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preprocessing</td>
<td>Greyscale</td>
<td>RGB</td>
</tr>
<tr>
<td>Architecture</td>
<td>3 layer plain CNN</td>
<td>Resnet50</td>
</tr>
<tr>
<td>SPP</td>
<td>No SPP</td>
<td>SPP with bin sizes (1,2,4)</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of experimental parameters

4.1 Dataset

The dataset used for the experiment is kindly provided by Korea University from the Andro-dumpsy study [21]. The dataset consists of 906 malicious binary files from 13 malware families, including smishing and spy applications. The benign files are a variety of popular applications with high rankings downloaded from GooglePlay store. The benign files are checked by VirusTotal [54] to ensure it is indeed benign, and this resulted in 1776 benign files.

4.2 Experiment Setup

The setup used for conducting the following experiments is a 64 bit Linux mint 18.3 Desktop with a quad-core Intel® Core™ i7-4770 at 3.40GHz, 16GB RAM and GeForce GTX 750 Ti. The program is written in Python 3.5 with Keras and Tensorflow as backend.

The model is trained with 90% of the collected sample, with batch size 16, 150 iterations and 20 epochs respectively. A summary of training parameters is shown in Table 4.2. Once the training is done, all weights are saved for evaluation later.

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epochs</td>
<td>20</td>
</tr>
<tr>
<td>Training Malware Size</td>
<td>815</td>
</tr>
<tr>
<td>Training Benign Size</td>
<td>1598</td>
</tr>
<tr>
<td>Iterations</td>
<td>150</td>
</tr>
</tbody>
</table>

Table 4.2: Summary of training parameters
4.3 Unaltered Samples

Before we evaluate the resilience of the design MDS against adversary result, it is important to evaluate the performance on normal, unaltered data. There is no point having an MDS that is resilient to adversary data but performs poorly with normal data. The precision and recall of all eight classifiers with a threshold value of 0.5 are shown in Table 4.3, confusion matrix and accuracy metrics for all experiments done are listed in Appendix A.

<table>
<thead>
<tr>
<th>Network</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greyscale plain noSPP</td>
<td>0.950</td>
<td>0.620</td>
</tr>
<tr>
<td>Greyscale plain SPP</td>
<td>0.295</td>
<td>0.670</td>
</tr>
<tr>
<td>Greyscale resnet noSPP</td>
<td>0.468</td>
<td>0.791</td>
</tr>
<tr>
<td>Greyscale resnet SPP</td>
<td>0.190</td>
<td>0.044</td>
</tr>
<tr>
<td>RGB plain noSPP</td>
<td>0.666</td>
<td>0.110</td>
</tr>
<tr>
<td>RGB plain SPP</td>
<td>0.156</td>
<td>0.084</td>
</tr>
<tr>
<td>RGB resnet noSPP</td>
<td>0.542</td>
<td>0.703</td>
</tr>
<tr>
<td>RGB resnet SPP</td>
<td>0.0625</td>
<td>0.066</td>
</tr>
</tbody>
</table>

Table 4.3: Results of the experiment

From Table 4.3, we observe that a plain network with greyscale and no SPP have the highest precision but a relatively low recall, while a resnet50 network with greyscale and no SPP have the highest recall but low precision.

We can also see that SPP performs poorly compared to the no SPP counterpart. The reason could be the threshold value of 0.5 is too high for SPP. Since we are using a divide and conquer approach, the entire file is classified as malicious if a sub-image is malicious, with a threshold of 0.5, large images are extremely likely to be classified as malicious even if they are benign. We vary the threshold value from 0.1 to 0.9 in 0.1 increments to examine each model in more detail.

Figure 4.1 and 4.2 shows the RoC curve of each model with various thresholds respectively.

The results show the resnet50 network has better performance when the input image is in RGB colour space, and the plain network performs better with greyscale images. We theorise this is due to RGB images contains more sophisticated patterns due to bytes/pixels being closer together in the input volume, such patterns are too complex for a shallow 3 plain layer network to capture, but the resnet50 network can.

It is also observed that SPP’s performance is worse than the no SPP counterpart regardless of the threshold. During the training period, we have discovered that training images that are too thin (e.g. 19 × 1920) results in the training accuracy to be 0 and loss to be NaN. This could cause SPP’s poor performance. Since we are fixing the width, there will be images that are too thin or wide depending on the size and SPP is unable to learn or classify the input. We conducted a further experiment where the ratio of the images in both training and testing are fixed to 1 : 1 (i.e. all images are square shaped but with different width and height) and check whether SPP shows better performance. The result of the further experiment are shown in Figure 4.3 and Figure 4.4.

From the further experiment, we can conclude that SPP does not work well regardless of threshold or ratio, and having fixed ratio images decreases the accuracy and precision of networks without SPP. The decrease in performance of fixed ratio
Chapter 4. Experiments

**Figure 4.1:** ROC Curves of models based on 3 layer plain network.

**Figure 4.2:** ROC Curves of different models based on resnet50 network.
Figure 4.3: RoC Curves of different models based on 3 layer plain network

Figure 4.4: RoC Curves of different models based on resnet50 network
images is likely due to the varying width causes the textures to change for each sample, thus decreasing the accuracy. To see this, consider a simple binary case where each bit alternates from 0 to 1 \textit{(i.e.101010)}. With width 2, the image will have one vertical line of zeroes and another vertical line of ones. However, with width 3, the image will have a checkered pattern. Note that this effect is different from adding redundant pixels in the image when adding a 1 to our toy example, a vertical line of ones can still be seen at width 2.

We hypothesise the devastating failure of SPP was due to the divide and conquer approach. The problem that arises with the divide and conquers approach is splitting a large malware texture into two sub-images which the classifier is not able to identify and having only a subset of malware features in a sub-image is not sufficient to classify the sub-image as malicious. As an analogy, if we wish to classify whether an image is a face using the divide and conquer method, we might pick up an eye in one image, but only having an eye is not sufficient to classify the sub-image as an eye.

In hindsight, the "divide and conquer" approach is theoretically fraud and destined to fail. The "divide and merge" might be a better solution and is worth exploring. "Divide and merge" divides the image into smaller sub-images, pools dominate features from each sub-image and concatenate it together to form a more concise image. Then, the neural network classifies the merged image. This way, we are able to combine smaller malicious/benign features to form a complete image of a malware. However this approach does not solve the problem of dividing a malware feature into multiple parts, and more experiments need to be done in order to evaluate the significance of splitting malware feature in practice.

\subsection*{4.3.1 Time Measurements}

The plain network takes on average 10 minutes to train one epoch, with 20 epochs it takes 2 hours and 40 minutes, evaluation of each sample takes on average 0.28 seconds. The resnet50 model takes 1 hour and 28 minutes to train one epoch, and with 20 epochs it takes 29 hours 20 minutes to train. The evaluation time for each sample takes 0.41 seconds on average. RGB and greyscale input does not affect the training time to a significant degree, and models with SPP takes 3 times longer to train as we are training the same images at 3 different sizes.

\subsection*{4.4 Results Comparison}

The Andro-Dumpsys [21] MDS, which uses the same dataset as us, extracts a comprehensive set of features and characteristics to classify malware. Some of the main features that Andro-Dumpsys examines are serial numbers, API sequence calls, permissions and memory acquisition. A large majority of the processing is done under 2.8GHz Intel Xeon X5660 with 8GB RAM. The paper reported only 8 benign files are classified as malicious and 16 malware classified as benign, this results in 0.45\% false positive rate and 1.77\% false negative rate. The accuracy of Andro-Dumpsys is significantly higher compared to our model, however, such high accuracy comes at a cost of processing power. Andro-Dumpsys comprehensively analyses many features of the malware, which takes on average 74.18 seconds per megabyte to detect malware. Our model does most of the processing during the training stage and the best model only takes on average 0.41 seconds to evaluate each malware sample since all samples are transformed into a fixed size.
Andro-Dumpsys has high accuracy but it is extremely slow. Our model is much faster than Andro-Dumpsys but with less accuracy. The accuracy of our detection system can be easily improved by fine-tuning the network and use a more advanced network, but this is not the main goal of this project. Nevertheless, we have shown that using convolutional neural networks can drastically improve the time need for detection.

4.5 Adversary Samples

The goal of the experiments is to investigate our MDS’s resilience to redundant API injection. As preliminary results have shown, SPP does not work well in normal classification tasks due to large variations in malware image size and our faulty approach to solve the problem. Thus in the following experiments, we ignore models with SPP and only investigate the performance resizing malware images with bilinear interpolation. We investigate whether bilinear interpolation is able to preserve features after resizing. For example, an eye in a high-quality image is still distinguishable when the image is compressed to a few kilobytes.

The adversary malware samples are generated as close to reality as possible. On an Intel x86 CPU, there are two different types of NOP operations with either 1 byte or multiple bytes ranging from 2 to 9 bytes [55]. To keep things simple we only inject single-byte NOP instruction (0x90) into the binary file. The method for injection NOP is simple, after each byte we add a NOP with a probability of 10%. This means the length of the images will increase by 10% on average. There are other ways of injecting NOPs, such as splitting 10% of the file size into n different redundant chunks and inject such chunks at various places. For this experiment, we only examine the effect of randomly adding NOP after each byte.

The model we wish to examine first is the resnet50 model with RGB input and no SPP. This is because the model has the highest f1 score overall on sample data (See Appendix A for complete metric). We evaluated the performance of the model on adversary samples that are fixed width and fixed ratio, the result is shown in Figure 4.5.

From Figure 4.5, we can see that fixed width inputs are more resilient to adversary inputs compared to fixed ratio ones, but both models are not completely safe from adversary samples. We are curious to see the effect of various levels of NOP saturation on our model’s performance, and a further experiment with files saturated by NOPs from 10% to 50% is conducted, the result is shown in Figure 4.6.

From the results, we can see that as more NOPs are being injected, the performance of our model decreases. This is because of the RGB representation. Adding a single NOP will cause a colour channel shift for all subsequent pixels, which alters the colours of the following pixels that leads to patterns not being recognized by the neural network. For greyscale images, this will not be a problem since there is only 1 colour channel. We also evaluate resnet50 and the plain network’s performance on greyscale adversary samples. The results are shown in Figure 4.7 and Figure 4.8.

From the results of greyscale adversary samples, we can see that adversary curves are very close to each other regardless of the level of saturation. This means bilinear interpolation with greyscale images is effective against redundant API injection. For the plain network, there is a significant drop from no redundant input to redundant input. This is likely due to the plain network is not flexible enough to detect the resized sample with NOP injected. Resnet50, on the other hand, is more complex and
Chapter 4. Experiments

Figure 4.5: RoC Curves of resnet50 model with RGB input and no SPP on adversary inputs

Figure 4.6: RoC Curves of resnet50 model with RGB input and no SPP on adversary inputs
FIGURE 4.7: RoC Curves of plain model with greyscale input and no SPP on adversary inputs

FIGURE 4.8: RoC Curves of resnet model with greyscale input and no SPP on adversary inputs
more robust, which is able to detect the interpolated samples and the initial drop between 0 to 10% adversary sample is a lot smaller than the plain network. However, due to the relatively poor performance of resnet50 on greyscale images, the result shows a high false positive rate.
5 Discussion

5.1 Findings

5.1.1 Colour Space

Resnet model works better with RGB input, while the simple plain network performs better with greyscale input. A possible explanation is that although the raw data is the same, in RGB images the byte values are spatially closer to each other in terms of the input volume. This allows more complex networks such as resnet50 to recognise sophisticated patterns which leads to high accuracy. This implies if we extend our third dimension (colour channel) to more than three, we could get better detection results with complex networks.

Although having more colour channels can lead to high detection accuracy, experimental results show that having more colour channels reduce the ability to detect malware samples with redundant API injected. Our experiment shows RGB input does not deal with redundant API injection as well as greyscale images. This is because injecting single NOP byte shifts the colour channels of subsequent bytes, due to such a shift, the neural network will misclassify the input. This problem can be solved by adding a pixel with value (NOP, 0, 0) everytime a NOP is seen. This prevents colour channel shifts of subsequent images. Alternatively, the preprocessor can remove all NOPs.

5.1.2 Spatial Pyramid Pooling

Due to extremely large variations in input image size, we applied spatial pyramid pooling, and large image sizes made us adopt our own "divide and conquer method" to reduce memory constraints. The "divide and conquer" method splits a large image into multiple ratio-preserving sub-images and classifies the sub-image. If any of the sub-image is classified as malicious, the entire image is classified as malicious. Experimental results show that the combination of SPP and "divide and conquer" does not perform well. In hindsight, the reason is rather obvious: the dividing step may split a malware feature into multiple parts that cannot be extracted as malware feature by the CNN. Even if the malware feature did not get divided, having only a subset of all malware features is not enough to classify the input as malicious. A better strategy may be "divide and merge", where instead of classifying the sub-image, the sub-images are pooled and merged back into a full image, which is then classified by the CNN. Due to time constraints, "divide and merge" has not been tested extensively and further work is needed to evaluate the performance of "divide and merge".

Another possible reason for the failure of SPP is that we have trained the network with images that have been resized significantly, while for evaluation the images are not compressed at all. The classifier has not been trained on sub-images which means accurately classifying the sub-images are extremely difficult.

From a high-level perspective, the reason we used SPP is to have a lossless transformation of the input data, as any form of compression or truncation can
be exploited by an attacker. However, from the experiments, we have found lossless transformation is extremely difficult for CNN-based malware detection because CNN takes the entire image as once and we are limited by the memory of the machine which we run the detection engine on.

5.2 Limitations

One of the biggest limitations of this project is that we did not cross-validate our data. Training the neural network is very time-consuming, especially with a large dataset. Due to time and resource constraints, applying 10-fold cross-validation on our data is infeasible, but doing so would make us statistically more confident in the results we obtained.

The neural networks we used in the classifier are limited to plain 3 layer network and a reasonably complex resnet50 network. Since the main goal is to investigate the effectiveness of the input colour space and spatial pyramid pooling, the model we used was not optimised at all. In practical scenarios, it is worth exploring other CNN architectures, for example, VGG [56], AlexNet [57] and GoogLeNet [58], and optimise these networks.

5.3 Future Work

From our findings we have stated that lossless transformation is impractical, thus it is worth exploring different ways of existing compression/resize methods or design a new compression/resize algorithm tailored for malware files. In our study only used bilinear interpolation as the compression algorithm, and other algorithms such as bi-cubic interpolation, Lanczos interpolation, nearest-neighbour interpolation and area-resampling are worth investigating. Other than interpolations, an RNN based image compression algorithm [59] is able to outperform JPEG at image compression across most bitrates on rate-distortion curve on Kodak dataset and is worth investigating.

It is also worth exploring the effects of different ways of NOP injection on the compression/resizing algorithm. Our study only injects a various number of NOPs sparsely at random positions. Another method is to injection NOPs equivalent of n% of the malware, split it into m chunks and inject the chunks at m randomly generated positions.

From the results we have found having more colour channels increases the detection accuracy. We theorise this is because more colour channels shorten the distance between each pair of pixels, and more patterns can be found. Unfortunately, our experiment only examined 2 distinct level of colour channels, one (greyscale) and three (RGB), which is not enough to prove our theory. An extended study can be conducted with more than three colour channels. This will be useful for image detection as well, as folding the image effectively increases the number of colour channels.

Recent studies done in deep learning based image classification aims to minimise the sample used for training, in order to reduce overfitting and manual labelling. One way for doing so is to distort the original image by flipping, cropping and resizing. This makes one input image equivalent of multiple images. The same idea can use used in CNN-based malware detection, where we preprocess the image by flipping, cropping, resizing and an extra step, which is adding redundant data at random places when training the image. This will make the system more flexible.
Another interesting area to investigate would be transforming malware file to image at the API level. Each API would be encoded as a number between 0 to 255 for greyscale, if there are more than 255 API calls then it is possible to group "safe" API calls (i.e. calls that are extremely unlikely to be malicious) into one. For RGB, each API would be encoded from 0 to $255^3$ which should be sufficient to cover possible all APIs. Using API level transformation shortens the image created, but system updates may change the API calls and require constant maintenance.

Out of curiosity, we have randomly uploaded 5 adversary files we generated with 10% NOP injected to VirusTotal. The adversary files all passed VirusTotal’s scanning engine and is classified as benign. In our study, we have assumed that adding NOPs does not change the behaviour of our sample, thus either VirusTotal’s scanning engine is vulnerable to injecting NOPs or our assumption is incorrect. More research is needed to determine whether the behaviour of a malware file is changed when NOPs are randomly injected. This surprising finding could imply that the attacker has to think carefully about where to add NOPs, which is an easier problem for MDS compared to randomly adding NOPs.
6 Conclusion

Due to RNN-based MDS’s vulnerability to redundant API injection [6], we seek ways of designing an MDS with CNN that are resilient to redundant API injection. Our CNN based MDS transforms the binary malware file into greyscale/RGB images and applies spatial pyramid pooling to allow the neural network to accept variously sized input. As a comparison, bilinear interpolation is used to resize the input image to the same size when SPP is not used. In order to deal with memory constraints, we applied the divide and conquer approach which divides a large image into multiple smaller sub-images and if any of the sub-image is classified as malicious, the entire image is malicious. The best performing model for classifying malware with no redundant API injection is RGB with resnet50 network and no SPP. For malware with redundant API injection, all models have some form of weaknesses which we gave possible explanations in Chapter 5. From the experimental results we find the following:

1. SPP with "divide and conquer" performs poorly and using "divide and merge" may yield better result.

2. Greyscale images are resilient against redundant API injection, and naive RGB conversion is not ideal.

3. Having more colour channels may lead to better detection accuracy.

4. Lossless transformation is impractical and compression/resize algorithms such as bilinear interpolation should be used.

In terms of time, the plain network takes on average 10 minutes to train one epoch, with 20 epochs it takes 2 hours and 40 minutes, evaluation of each sample takes on average 0.28 seconds. The resnet50 model takes 1 hour and 28 minutes to train one epoch, and with 20 epochs it takes 29 hours 20 minutes to train. The evaluation time for each sample is much faster, at 0.41 seconds.

More work in CNN base MDS can be done to increase its resilience against redundant API attacks and detection accuracy on normal input, some of which are:

- Using other types of resize/compression method, for example, bi-cubic interpolation, Lanczos interpolation, nearest-neighbour interpolation and area-resampling

- Explore other ways of redundant API injection, such as injection NOPs equivalent of n% of the malware, split it into m chunks and inject the chunks at m randomly generated positions.

- Conduct more experiments to prove or disprove our theory of more colour channels increases detection accuracy.

- Compare the performance of CNN trained with image modification (e.g. flipping, cropping, resizing and random redundant API injection)
• Research whether randomly adding NOPs alters the malevolence of malware files.

This project has shown that CNN-based MDS have the potential to rapidly classify malware files with high accuracy, and resilient to redundant API injections. To our surprise, Spatial Pyramid Pooling were found to be ineffective against images of drastically various sizes and a complementary method is required. The question still remains as to whether CNNs is more powerful compared to RNN in terms of malware detection accuracy and resilience to adversary inputs. Further research in this topic would be interesting and enlightening to explore.
Bibliography


## A Raw Experiment Results

### A.1 Fixed width

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>67</td>
<td>167</td>
<td>11</td>
<td>24</td>
<td>0.859</td>
<td>0.736</td>
<td>0.736</td>
<td>0.062</td>
<td>0.793</td>
</tr>
<tr>
<td>0.2</td>
<td>64</td>
<td>174</td>
<td>4</td>
<td>27</td>
<td>0.941</td>
<td>0.703</td>
<td>0.703</td>
<td>0.022</td>
<td>0.805</td>
</tr>
<tr>
<td>0.3</td>
<td>62</td>
<td>176</td>
<td>2</td>
<td>29</td>
<td>0.969</td>
<td>0.681</td>
<td>0.681</td>
<td>0.011</td>
<td>0.800</td>
</tr>
<tr>
<td>0.4</td>
<td>61</td>
<td>176</td>
<td>2</td>
<td>30</td>
<td>0.968</td>
<td>0.670</td>
<td>0.670</td>
<td>0.011</td>
<td>0.792</td>
</tr>
<tr>
<td>0.5</td>
<td>59</td>
<td>177</td>
<td>1</td>
<td>32</td>
<td>0.983</td>
<td>0.648</td>
<td>0.648</td>
<td>0.006</td>
<td>0.781</td>
</tr>
<tr>
<td>0.6</td>
<td>55</td>
<td>177</td>
<td>1</td>
<td>36</td>
<td>0.982</td>
<td>0.604</td>
<td>0.604</td>
<td>0.006</td>
<td>0.748</td>
</tr>
<tr>
<td>0.7</td>
<td>52</td>
<td>177</td>
<td>1</td>
<td>39</td>
<td>0.981</td>
<td>0.571</td>
<td>0.571</td>
<td>0.006</td>
<td>0.722</td>
</tr>
<tr>
<td>0.8</td>
<td>45</td>
<td>177</td>
<td>1</td>
<td>46</td>
<td>0.978</td>
<td>0.495</td>
<td>0.495</td>
<td>0.006</td>
<td>0.657</td>
</tr>
<tr>
<td>0.9</td>
<td>30</td>
<td>178</td>
<td>0</td>
<td>61</td>
<td>1.000</td>
<td>0.330</td>
<td>0.330</td>
<td>0.000</td>
<td>0.496</td>
</tr>
</tbody>
</table>

**Table A.1:** Plain 3 layer network with RGB input and no SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>77</td>
<td>159</td>
<td>19</td>
<td>14</td>
<td>0.802</td>
<td>0.846</td>
<td>0.846</td>
<td>0.107</td>
<td>0.824</td>
</tr>
<tr>
<td>0.2</td>
<td>69</td>
<td>165</td>
<td>13</td>
<td>22</td>
<td>0.841</td>
<td>0.758</td>
<td>0.758</td>
<td>0.073</td>
<td>0.798</td>
</tr>
<tr>
<td>0.3</td>
<td>65</td>
<td>170</td>
<td>8</td>
<td>26</td>
<td>0.890</td>
<td>0.714</td>
<td>0.714</td>
<td>0.045</td>
<td>0.793</td>
</tr>
<tr>
<td>0.4</td>
<td>60</td>
<td>173</td>
<td>5</td>
<td>31</td>
<td>0.923</td>
<td>0.659</td>
<td>0.659</td>
<td>0.028</td>
<td>0.769</td>
</tr>
<tr>
<td>0.5</td>
<td>57</td>
<td>175</td>
<td>3</td>
<td>34</td>
<td>0.950</td>
<td>0.626</td>
<td>0.626</td>
<td>0.017</td>
<td>0.755</td>
</tr>
<tr>
<td>0.6</td>
<td>50</td>
<td>175</td>
<td>3</td>
<td>41</td>
<td>0.943</td>
<td>0.549</td>
<td>0.549</td>
<td>0.017</td>
<td>0.694</td>
</tr>
<tr>
<td>0.7</td>
<td>43</td>
<td>177</td>
<td>1</td>
<td>48</td>
<td>0.977</td>
<td>0.473</td>
<td>0.473</td>
<td>0.006</td>
<td>0.637</td>
</tr>
<tr>
<td>0.8</td>
<td>26</td>
<td>178</td>
<td>0</td>
<td>65</td>
<td>1.000</td>
<td>0.286</td>
<td>0.286</td>
<td>0.000</td>
<td>0.444</td>
</tr>
<tr>
<td>0.9</td>
<td>9</td>
<td>178</td>
<td>0</td>
<td>82</td>
<td>1.000</td>
<td>0.099</td>
<td>0.099</td>
<td>0.000</td>
<td>0.180</td>
</tr>
</tbody>
</table>

**Table A.2:** Plain 3 layer network with RGB input and SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>77</td>
<td>159</td>
<td>19</td>
<td>14</td>
<td>0.802</td>
<td>0.846</td>
<td>0.846</td>
<td>0.107</td>
<td>0.824</td>
</tr>
<tr>
<td>0.2</td>
<td>69</td>
<td>165</td>
<td>13</td>
<td>22</td>
<td>0.841</td>
<td>0.758</td>
<td>0.758</td>
<td>0.073</td>
<td>0.798</td>
</tr>
<tr>
<td>0.3</td>
<td>65</td>
<td>170</td>
<td>8</td>
<td>26</td>
<td>0.890</td>
<td>0.714</td>
<td>0.714</td>
<td>0.045</td>
<td>0.793</td>
</tr>
<tr>
<td>0.4</td>
<td>60</td>
<td>173</td>
<td>5</td>
<td>31</td>
<td>0.923</td>
<td>0.659</td>
<td>0.659</td>
<td>0.028</td>
<td>0.769</td>
</tr>
<tr>
<td>0.5</td>
<td>57</td>
<td>175</td>
<td>3</td>
<td>34</td>
<td>0.950</td>
<td>0.626</td>
<td>0.626</td>
<td>0.017</td>
<td>0.755</td>
</tr>
<tr>
<td>0.6</td>
<td>50</td>
<td>175</td>
<td>3</td>
<td>41</td>
<td>0.943</td>
<td>0.549</td>
<td>0.549</td>
<td>0.017</td>
<td>0.694</td>
</tr>
<tr>
<td>0.7</td>
<td>43</td>
<td>177</td>
<td>1</td>
<td>48</td>
<td>0.977</td>
<td>0.473</td>
<td>0.473</td>
<td>0.006</td>
<td>0.637</td>
</tr>
<tr>
<td>0.8</td>
<td>26</td>
<td>178</td>
<td>0</td>
<td>65</td>
<td>1.000</td>
<td>0.286</td>
<td>0.286</td>
<td>0.000</td>
<td>0.444</td>
</tr>
<tr>
<td>0.9</td>
<td>9</td>
<td>178</td>
<td>0</td>
<td>82</td>
<td>1.000</td>
<td>0.099</td>
<td>0.099</td>
<td>0.000</td>
<td>0.180</td>
</tr>
</tbody>
</table>

**Table A.3:** Plain 3 layer network with greyscale input and no SPP
Appendix A. Raw Experiment Results

### Table A.4: Plain 3 layer network with greyscale input and SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>86</td>
<td>8</td>
<td>169</td>
<td>5</td>
<td>0.337</td>
<td>0.945</td>
<td>0.945</td>
<td>0.955</td>
<td>0.497</td>
</tr>
<tr>
<td>0.2</td>
<td>80</td>
<td>10</td>
<td>167</td>
<td>11</td>
<td>0.324</td>
<td>0.879</td>
<td>0.879</td>
<td>0.944</td>
<td>0.473</td>
</tr>
<tr>
<td>0.3</td>
<td>73</td>
<td>15</td>
<td>162</td>
<td>18</td>
<td>0.311</td>
<td>0.802</td>
<td>0.802</td>
<td>0.915</td>
<td>0.448</td>
</tr>
<tr>
<td>0.4</td>
<td>64</td>
<td>23</td>
<td>154</td>
<td>27</td>
<td>0.294</td>
<td>0.703</td>
<td>0.703</td>
<td>0.870</td>
<td>0.414</td>
</tr>
<tr>
<td>0.5</td>
<td>61</td>
<td>31</td>
<td>146</td>
<td>30</td>
<td>0.295</td>
<td>0.670</td>
<td>0.670</td>
<td>0.825</td>
<td>0.409</td>
</tr>
<tr>
<td>0.6</td>
<td>57</td>
<td>37</td>
<td>140</td>
<td>34</td>
<td>0.289</td>
<td>0.626</td>
<td>0.626</td>
<td>0.791</td>
<td>0.396</td>
</tr>
<tr>
<td>0.7</td>
<td>50</td>
<td>46</td>
<td>131</td>
<td>41</td>
<td>0.276</td>
<td>0.549</td>
<td>0.549</td>
<td>0.740</td>
<td>0.368</td>
</tr>
<tr>
<td>0.8</td>
<td>46</td>
<td>61</td>
<td>116</td>
<td>45</td>
<td>0.284</td>
<td>0.505</td>
<td>0.505</td>
<td>0.655</td>
<td>0.364</td>
</tr>
<tr>
<td>0.9</td>
<td>41</td>
<td>90</td>
<td>87</td>
<td>50</td>
<td>0.320</td>
<td>0.451</td>
<td>0.451</td>
<td>0.492</td>
<td>0.374</td>
</tr>
</tbody>
</table>

### Table A.5: ResNet50 network with RGB input and no SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>8</td>
<td>57</td>
<td>120</td>
<td>83</td>
<td>0.063</td>
<td>0.088</td>
<td>0.088</td>
<td>0.650</td>
<td>0.075</td>
</tr>
<tr>
<td>0.2</td>
<td>8</td>
<td>62</td>
<td>115</td>
<td>83</td>
<td>0.067</td>
<td>0.088</td>
<td>0.088</td>
<td>0.633</td>
<td>0.076</td>
</tr>
<tr>
<td>0.3</td>
<td>8</td>
<td>65</td>
<td>112</td>
<td>83</td>
<td>0.067</td>
<td>0.088</td>
<td>0.088</td>
<td>0.627</td>
<td>0.076</td>
</tr>
<tr>
<td>0.4</td>
<td>8</td>
<td>66</td>
<td>111</td>
<td>83</td>
<td>0.067</td>
<td>0.088</td>
<td>0.088</td>
<td>0.616</td>
<td>0.077</td>
</tr>
<tr>
<td>0.5</td>
<td>8</td>
<td>68</td>
<td>109</td>
<td>83</td>
<td>0.068</td>
<td>0.088</td>
<td>0.088</td>
<td>0.609</td>
<td>0.077</td>
</tr>
<tr>
<td>0.6</td>
<td>7</td>
<td>71</td>
<td>106</td>
<td>84</td>
<td>0.062</td>
<td>0.077</td>
<td>0.077</td>
<td>0.599</td>
<td>0.069</td>
</tr>
<tr>
<td>0.7</td>
<td>7</td>
<td>72</td>
<td>105</td>
<td>84</td>
<td>0.063</td>
<td>0.077</td>
<td>0.077</td>
<td>0.593</td>
<td>0.069</td>
</tr>
<tr>
<td>0.8</td>
<td>7</td>
<td>78</td>
<td>99</td>
<td>84</td>
<td>0.066</td>
<td>0.077</td>
<td>0.077</td>
<td>0.559</td>
<td>0.071</td>
</tr>
<tr>
<td>0.9</td>
<td>7</td>
<td>80</td>
<td>97</td>
<td>84</td>
<td>0.067</td>
<td>0.077</td>
<td>0.077</td>
<td>0.548</td>
<td>0.072</td>
</tr>
</tbody>
</table>

### Table A.6: ResNet50 network with RGB input and SPP
### Appendix A. Raw Experiment Results

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>89</td>
<td>46</td>
<td>132</td>
<td>2</td>
<td>0.403</td>
<td>0.978</td>
<td>0.978</td>
<td>0.742</td>
<td>0.571</td>
</tr>
<tr>
<td>0.2</td>
<td>80</td>
<td>73</td>
<td>105</td>
<td>11</td>
<td>0.432</td>
<td>0.879</td>
<td>0.879</td>
<td>0.590</td>
<td>0.580</td>
</tr>
<tr>
<td>0.3</td>
<td>76</td>
<td>88</td>
<td>90</td>
<td>15</td>
<td>0.458</td>
<td>0.835</td>
<td>0.835</td>
<td>0.506</td>
<td>0.591</td>
</tr>
<tr>
<td>0.4</td>
<td>75</td>
<td>91</td>
<td>87</td>
<td>16</td>
<td>0.463</td>
<td>0.824</td>
<td>0.824</td>
<td>0.489</td>
<td>0.593</td>
</tr>
<tr>
<td>0.5</td>
<td>72</td>
<td>96</td>
<td>82</td>
<td>19</td>
<td>0.468</td>
<td>0.791</td>
<td>0.791</td>
<td>0.461</td>
<td>0.588</td>
</tr>
<tr>
<td>0.6</td>
<td>70</td>
<td>102</td>
<td>76</td>
<td>21</td>
<td>0.479</td>
<td>0.769</td>
<td>0.769</td>
<td>0.427</td>
<td>0.591</td>
</tr>
<tr>
<td>0.7</td>
<td>68</td>
<td>109</td>
<td>69</td>
<td>23</td>
<td>0.496</td>
<td>0.747</td>
<td>0.747</td>
<td>0.388</td>
<td>0.596</td>
</tr>
<tr>
<td>0.8</td>
<td>66</td>
<td>123</td>
<td>55</td>
<td>25</td>
<td>0.545</td>
<td>0.725</td>
<td>0.725</td>
<td>0.309</td>
<td>0.623</td>
</tr>
<tr>
<td>0.9</td>
<td>61</td>
<td>147</td>
<td>31</td>
<td>30</td>
<td>0.663</td>
<td>0.670</td>
<td>0.670</td>
<td>0.174</td>
<td>0.667</td>
</tr>
</tbody>
</table>

**Table A.7: Resnet50 network with Greyscale input and no SPP**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>17</td>
<td>113</td>
<td>64</td>
<td>74</td>
<td>0.210</td>
<td>0.187</td>
<td>0.187</td>
<td>0.362</td>
<td>0.198</td>
</tr>
<tr>
<td>0.2</td>
<td>10</td>
<td>141</td>
<td>36</td>
<td>81</td>
<td>0.217</td>
<td>0.110</td>
<td>0.110</td>
<td>0.203</td>
<td>0.146</td>
</tr>
<tr>
<td>0.3</td>
<td>7</td>
<td>151</td>
<td>26</td>
<td>84</td>
<td>0.212</td>
<td>0.077</td>
<td>0.077</td>
<td>0.147</td>
<td>0.113</td>
</tr>
<tr>
<td>0.4</td>
<td>5</td>
<td>154</td>
<td>23</td>
<td>86</td>
<td>0.179</td>
<td>0.055</td>
<td>0.055</td>
<td>0.130</td>
<td>0.084</td>
</tr>
<tr>
<td>0.5</td>
<td>4</td>
<td>160</td>
<td>17</td>
<td>87</td>
<td>0.190</td>
<td>0.044</td>
<td>0.044</td>
<td>0.096</td>
<td>0.071</td>
</tr>
<tr>
<td>0.6</td>
<td>4</td>
<td>167</td>
<td>10</td>
<td>87</td>
<td>0.286</td>
<td>0.044</td>
<td>0.044</td>
<td>0.056</td>
<td>0.076</td>
</tr>
<tr>
<td>0.7</td>
<td>4</td>
<td>168</td>
<td>9</td>
<td>87</td>
<td>0.308</td>
<td>0.044</td>
<td>0.044</td>
<td>0.051</td>
<td>0.077</td>
</tr>
<tr>
<td>0.8</td>
<td>1</td>
<td>168</td>
<td>9</td>
<td>90</td>
<td>0.100</td>
<td>0.011</td>
<td>0.011</td>
<td>0.051</td>
<td>0.020</td>
</tr>
<tr>
<td>0.9</td>
<td>1</td>
<td>173</td>
<td>4</td>
<td>90</td>
<td>0.200</td>
<td>0.011</td>
<td>0.011</td>
<td>0.023</td>
<td>0.021</td>
</tr>
</tbody>
</table>

**Table A.8: Resnet50 network with Greyscale input and SPP**

### A.2 Fixed Ratio

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>70</td>
<td>172</td>
<td>6</td>
<td>21</td>
<td>0.921</td>
<td>0.769</td>
<td>0.769</td>
<td>0.034</td>
<td>0.838</td>
</tr>
<tr>
<td>0.2</td>
<td>66</td>
<td>175</td>
<td>3</td>
<td>25</td>
<td>0.957</td>
<td>0.725</td>
<td>0.725</td>
<td>0.017</td>
<td>0.825</td>
</tr>
<tr>
<td>0.3</td>
<td>64</td>
<td>177</td>
<td>1</td>
<td>27</td>
<td>0.985</td>
<td>0.703</td>
<td>0.703</td>
<td>0.006</td>
<td>0.821</td>
</tr>
<tr>
<td>0.4</td>
<td>61</td>
<td>177</td>
<td>1</td>
<td>30</td>
<td>0.984</td>
<td>0.670</td>
<td>0.670</td>
<td>0.006</td>
<td>0.797</td>
</tr>
<tr>
<td>0.5</td>
<td>55</td>
<td>177</td>
<td>1</td>
<td>36</td>
<td>0.982</td>
<td>0.604</td>
<td>0.604</td>
<td>0.006</td>
<td>0.748</td>
</tr>
<tr>
<td>0.6</td>
<td>46</td>
<td>177</td>
<td>1</td>
<td>45</td>
<td>0.979</td>
<td>0.505</td>
<td>0.505</td>
<td>0.006</td>
<td>0.667</td>
</tr>
<tr>
<td>0.7</td>
<td>44</td>
<td>178</td>
<td>0</td>
<td>47</td>
<td>1.000</td>
<td>0.484</td>
<td>0.484</td>
<td>0.000</td>
<td>0.652</td>
</tr>
<tr>
<td>0.8</td>
<td>38</td>
<td>178</td>
<td>0</td>
<td>53</td>
<td>1.000</td>
<td>0.418</td>
<td>0.418</td>
<td>0.000</td>
<td>0.589</td>
</tr>
<tr>
<td>0.9</td>
<td>22</td>
<td>178</td>
<td>0</td>
<td>69</td>
<td>1.000</td>
<td>0.242</td>
<td>0.242</td>
<td>0.000</td>
<td>0.389</td>
</tr>
</tbody>
</table>

**Table A.9: Plain 3 layer network with RGB input and no SPP**
### Appendix A. Raw Experiment Results

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>78</td>
<td>10</td>
<td>167</td>
<td>13</td>
<td>0.318</td>
<td>0.857</td>
<td>0.857</td>
<td>0.944</td>
<td>0.464</td>
</tr>
<tr>
<td>0.2</td>
<td>61</td>
<td>26</td>
<td>151</td>
<td>30</td>
<td>0.288</td>
<td>0.670</td>
<td>0.670</td>
<td>0.853</td>
<td>0.403</td>
</tr>
<tr>
<td>0.3</td>
<td>54</td>
<td>39</td>
<td>138</td>
<td>37</td>
<td>0.281</td>
<td>0.593</td>
<td>0.593</td>
<td>0.780</td>
<td>0.382</td>
</tr>
<tr>
<td>0.4</td>
<td>43</td>
<td>58</td>
<td>119</td>
<td>48</td>
<td>0.265</td>
<td>0.473</td>
<td>0.473</td>
<td>0.672</td>
<td>0.340</td>
</tr>
<tr>
<td>0.5</td>
<td>35</td>
<td>77</td>
<td>100</td>
<td>56</td>
<td>0.259</td>
<td>0.385</td>
<td>0.385</td>
<td>0.565</td>
<td>0.310</td>
</tr>
<tr>
<td>0.6</td>
<td>25</td>
<td>93</td>
<td>84</td>
<td>66</td>
<td>0.229</td>
<td>0.275</td>
<td>0.275</td>
<td>0.475</td>
<td>0.250</td>
</tr>
<tr>
<td>0.7</td>
<td>19</td>
<td>122</td>
<td>55</td>
<td>72</td>
<td>0.257</td>
<td>0.209</td>
<td>0.209</td>
<td>0.311</td>
<td>0.230</td>
</tr>
<tr>
<td>0.8</td>
<td>11</td>
<td>144</td>
<td>33</td>
<td>80</td>
<td>0.250</td>
<td>0.121</td>
<td>0.121</td>
<td>0.186</td>
<td>0.163</td>
</tr>
<tr>
<td>0.9</td>
<td>8</td>
<td>164</td>
<td>13</td>
<td>83</td>
<td>0.381</td>
<td>0.088</td>
<td>0.088</td>
<td>0.073</td>
<td>0.143</td>
</tr>
</tbody>
</table>

**Table A.10:** Plain 3 layer network with RGB input and SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>65</td>
<td>175</td>
<td>3</td>
<td>26</td>
<td>0.956</td>
<td>0.714</td>
<td>0.714</td>
<td>0.017</td>
<td>0.818</td>
</tr>
<tr>
<td>0.2</td>
<td>56</td>
<td>176</td>
<td>2</td>
<td>35</td>
<td>0.966</td>
<td>0.615</td>
<td>0.615</td>
<td>0.011</td>
<td>0.752</td>
</tr>
<tr>
<td>0.3</td>
<td>48</td>
<td>176</td>
<td>2</td>
<td>43</td>
<td>0.960</td>
<td>0.527</td>
<td>0.527</td>
<td>0.011</td>
<td>0.681</td>
</tr>
<tr>
<td>0.4</td>
<td>39</td>
<td>177</td>
<td>1</td>
<td>52</td>
<td>0.975</td>
<td>0.429</td>
<td>0.429</td>
<td>0.006</td>
<td>0.595</td>
</tr>
<tr>
<td>0.5</td>
<td>30</td>
<td>177</td>
<td>1</td>
<td>61</td>
<td>0.968</td>
<td>0.330</td>
<td>0.330</td>
<td>0.006</td>
<td>0.492</td>
</tr>
<tr>
<td>0.6</td>
<td>21</td>
<td>177</td>
<td>1</td>
<td>70</td>
<td>0.955</td>
<td>0.231</td>
<td>0.231</td>
<td>0.006</td>
<td>0.372</td>
</tr>
<tr>
<td>0.7</td>
<td>15</td>
<td>178</td>
<td>0</td>
<td>76</td>
<td>1.000</td>
<td>0.165</td>
<td>0.165</td>
<td>0.000</td>
<td>0.283</td>
</tr>
<tr>
<td>0.8</td>
<td>9</td>
<td>178</td>
<td>0</td>
<td>82</td>
<td>1.000</td>
<td>0.099</td>
<td>0.099</td>
<td>0.000</td>
<td>0.180</td>
</tr>
<tr>
<td>0.9</td>
<td>7</td>
<td>178</td>
<td>0</td>
<td>84</td>
<td>1.000</td>
<td>0.077</td>
<td>0.077</td>
<td>0.000</td>
<td>0.143</td>
</tr>
</tbody>
</table>

**Table A.11:** Plain 3 layer network with greyscale input and no SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>79</td>
<td>7</td>
<td>170</td>
<td>12</td>
<td>0.317</td>
<td>0.868</td>
<td>0.868</td>
<td>0.960</td>
<td>0.465</td>
</tr>
<tr>
<td>0.2</td>
<td>73</td>
<td>11</td>
<td>166</td>
<td>18</td>
<td>0.305</td>
<td>0.802</td>
<td>0.802</td>
<td>0.938</td>
<td>0.442</td>
</tr>
<tr>
<td>0.3</td>
<td>64</td>
<td>13</td>
<td>164</td>
<td>27</td>
<td>0.281</td>
<td>0.703</td>
<td>0.703</td>
<td>0.927</td>
<td>0.401</td>
</tr>
<tr>
<td>0.4</td>
<td>61</td>
<td>17</td>
<td>160</td>
<td>30</td>
<td>0.276</td>
<td>0.670</td>
<td>0.670</td>
<td>0.904</td>
<td>0.391</td>
</tr>
<tr>
<td>0.5</td>
<td>53</td>
<td>21</td>
<td>156</td>
<td>38</td>
<td>0.254</td>
<td>0.582</td>
<td>0.582</td>
<td>0.881</td>
<td>0.353</td>
</tr>
<tr>
<td>0.6</td>
<td>42</td>
<td>30</td>
<td>147</td>
<td>49</td>
<td>0.222</td>
<td>0.462</td>
<td>0.462</td>
<td>0.831</td>
<td>0.300</td>
</tr>
<tr>
<td>0.7</td>
<td>34</td>
<td>41</td>
<td>136</td>
<td>57</td>
<td>0.200</td>
<td>0.374</td>
<td>0.374</td>
<td>0.768</td>
<td>0.261</td>
</tr>
<tr>
<td>0.8</td>
<td>26</td>
<td>46</td>
<td>131</td>
<td>65</td>
<td>0.166</td>
<td>0.286</td>
<td>0.286</td>
<td>0.740</td>
<td>0.210</td>
</tr>
<tr>
<td>0.9</td>
<td>16</td>
<td>66</td>
<td>111</td>
<td>75</td>
<td>0.126</td>
<td>0.176</td>
<td>0.176</td>
<td>0.627</td>
<td>0.147</td>
</tr>
</tbody>
</table>

**Table A.12:** Plain 3 layer network with greyscale input and SPP
### Appendix A. Raw Experiment Results

#### Table A.13: Resnet50 network with RGB input and no SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>80</td>
<td>133</td>
<td>45</td>
<td>11</td>
<td>0.640</td>
<td>0.879</td>
<td>0.879</td>
<td>0.253</td>
<td>0.741</td>
</tr>
<tr>
<td>0.2</td>
<td>74</td>
<td>149</td>
<td>29</td>
<td>17</td>
<td>0.718</td>
<td>0.813</td>
<td>0.813</td>
<td>0.163</td>
<td>0.763</td>
</tr>
<tr>
<td>0.3</td>
<td>74</td>
<td>155</td>
<td>23</td>
<td>17</td>
<td>0.763</td>
<td>0.813</td>
<td>0.813</td>
<td>0.129</td>
<td>0.787</td>
</tr>
<tr>
<td>0.4</td>
<td>72</td>
<td>157</td>
<td>21</td>
<td>19</td>
<td>0.774</td>
<td>0.791</td>
<td>0.791</td>
<td>0.118</td>
<td>0.783</td>
</tr>
<tr>
<td>0.5</td>
<td>72</td>
<td>159</td>
<td>19</td>
<td>19</td>
<td>0.791</td>
<td>0.791</td>
<td>0.791</td>
<td>0.107</td>
<td>0.791</td>
</tr>
<tr>
<td>0.6</td>
<td>70</td>
<td>162</td>
<td>16</td>
<td>21</td>
<td>0.814</td>
<td>0.769</td>
<td>0.769</td>
<td>0.090</td>
<td>0.791</td>
</tr>
<tr>
<td>0.7</td>
<td>69</td>
<td>164</td>
<td>14</td>
<td>22</td>
<td>0.831</td>
<td>0.758</td>
<td>0.758</td>
<td>0.079</td>
<td>0.793</td>
</tr>
<tr>
<td>0.8</td>
<td>68</td>
<td>166</td>
<td>12</td>
<td>23</td>
<td>0.850</td>
<td>0.747</td>
<td>0.747</td>
<td>0.067</td>
<td>0.795</td>
</tr>
<tr>
<td>0.9</td>
<td>60</td>
<td>173</td>
<td>5</td>
<td>31</td>
<td>0.923</td>
<td>0.659</td>
<td>0.659</td>
<td>0.028</td>
<td>0.769</td>
</tr>
</tbody>
</table>

#### Table A.14: Resnet50 network with RGB input and SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>86</td>
<td>3</td>
<td>174</td>
<td>5</td>
<td>0.331</td>
<td>0.945</td>
<td>0.945</td>
<td>0.490</td>
<td>0.400</td>
</tr>
<tr>
<td>0.2</td>
<td>85</td>
<td>5</td>
<td>172</td>
<td>6</td>
<td>0.331</td>
<td>0.934</td>
<td>0.934</td>
<td>0.489</td>
<td>0.489</td>
</tr>
<tr>
<td>0.3</td>
<td>83</td>
<td>7</td>
<td>170</td>
<td>8</td>
<td>0.328</td>
<td>0.912</td>
<td>0.912</td>
<td>0.483</td>
<td>0.483</td>
</tr>
<tr>
<td>0.4</td>
<td>80</td>
<td>9</td>
<td>168</td>
<td>11</td>
<td>0.323</td>
<td>0.879</td>
<td>0.879</td>
<td>0.472</td>
<td>0.472</td>
</tr>
<tr>
<td>0.5</td>
<td>77</td>
<td>9</td>
<td>168</td>
<td>14</td>
<td>0.314</td>
<td>0.846</td>
<td>0.846</td>
<td>0.458</td>
<td>0.458</td>
</tr>
<tr>
<td>0.6</td>
<td>75</td>
<td>10</td>
<td>167</td>
<td>16</td>
<td>0.310</td>
<td>0.824</td>
<td>0.824</td>
<td>0.450</td>
<td>0.450</td>
</tr>
<tr>
<td>0.7</td>
<td>68</td>
<td>11</td>
<td>166</td>
<td>23</td>
<td>0.291</td>
<td>0.747</td>
<td>0.747</td>
<td>0.418</td>
<td>0.418</td>
</tr>
<tr>
<td>0.8</td>
<td>64</td>
<td>13</td>
<td>164</td>
<td>27</td>
<td>0.281</td>
<td>0.703</td>
<td>0.703</td>
<td>0.401</td>
<td>0.401</td>
</tr>
<tr>
<td>0.9</td>
<td>55</td>
<td>17</td>
<td>160</td>
<td>36</td>
<td>0.256</td>
<td>0.604</td>
<td>0.604</td>
<td>0.359</td>
<td>0.359</td>
</tr>
</tbody>
</table>

#### Table A.15: Resnet50 network with greyscale input and no SPP

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>88</td>
<td>81</td>
<td>97</td>
<td>3</td>
<td>0.476</td>
<td>0.967</td>
<td>0.967</td>
<td>0.545</td>
<td>0.638</td>
</tr>
<tr>
<td>0.2</td>
<td>83</td>
<td>103</td>
<td>75</td>
<td>8</td>
<td>0.525</td>
<td>0.912</td>
<td>0.912</td>
<td>0.421</td>
<td>0.667</td>
</tr>
<tr>
<td>0.3</td>
<td>82</td>
<td>111</td>
<td>67</td>
<td>9</td>
<td>0.550</td>
<td>0.901</td>
<td>0.901</td>
<td>0.376</td>
<td>0.683</td>
</tr>
<tr>
<td>0.4</td>
<td>81</td>
<td>125</td>
<td>53</td>
<td>10</td>
<td>0.604</td>
<td>0.890</td>
<td>0.890</td>
<td>0.298</td>
<td>0.720</td>
</tr>
<tr>
<td>0.5</td>
<td>80</td>
<td>134</td>
<td>44</td>
<td>11</td>
<td>0.645</td>
<td>0.879</td>
<td>0.879</td>
<td>0.247</td>
<td>0.744</td>
</tr>
<tr>
<td>0.6</td>
<td>75</td>
<td>135</td>
<td>43</td>
<td>16</td>
<td>0.636</td>
<td>0.824</td>
<td>0.824</td>
<td>0.242</td>
<td>0.718</td>
</tr>
<tr>
<td>0.7</td>
<td>73</td>
<td>142</td>
<td>36</td>
<td>18</td>
<td>0.670</td>
<td>0.802</td>
<td>0.802</td>
<td>0.202</td>
<td>0.730</td>
</tr>
<tr>
<td>0.8</td>
<td>67</td>
<td>149</td>
<td>29</td>
<td>24</td>
<td>0.698</td>
<td>0.736</td>
<td>0.736</td>
<td>0.163</td>
<td>0.717</td>
</tr>
<tr>
<td>0.9</td>
<td>59</td>
<td>160</td>
<td>18</td>
<td>32</td>
<td>0.766</td>
<td>0.648</td>
<td>0.648</td>
<td>0.101</td>
<td>0.702</td>
</tr>
</tbody>
</table>
### A.3 Adversary Data

#### A.3.1 RGB with Resnet50

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>74</td>
<td>129</td>
<td>48</td>
<td>17</td>
<td>0.607</td>
<td>0.813</td>
<td>0.813</td>
<td>0.271</td>
<td>0.695</td>
</tr>
<tr>
<td>0.2</td>
<td>66</td>
<td>152</td>
<td>25</td>
<td>25</td>
<td>0.725</td>
<td>0.725</td>
<td>0.725</td>
<td>0.141</td>
<td>0.725</td>
</tr>
<tr>
<td>0.3</td>
<td>62</td>
<td>159</td>
<td>18</td>
<td>29</td>
<td>0.775</td>
<td>0.681</td>
<td>0.681</td>
<td>0.102</td>
<td>0.725</td>
</tr>
<tr>
<td>0.4</td>
<td>53</td>
<td>168</td>
<td>9</td>
<td>38</td>
<td>0.855</td>
<td>0.582</td>
<td>0.582</td>
<td>0.051</td>
<td>0.693</td>
</tr>
<tr>
<td>0.5</td>
<td>47</td>
<td>171</td>
<td>6</td>
<td>44</td>
<td>0.887</td>
<td>0.516</td>
<td>0.516</td>
<td>0.034</td>
<td>0.653</td>
</tr>
<tr>
<td>0.6</td>
<td>46</td>
<td>173</td>
<td>4</td>
<td>45</td>
<td>0.920</td>
<td>0.505</td>
<td>0.505</td>
<td>0.023</td>
<td>0.652</td>
</tr>
<tr>
<td>0.7</td>
<td>37</td>
<td>174</td>
<td>3</td>
<td>54</td>
<td>0.925</td>
<td>0.407</td>
<td>0.407</td>
<td>0.017</td>
<td>0.565</td>
</tr>
<tr>
<td>0.8</td>
<td>32</td>
<td>175</td>
<td>2</td>
<td>59</td>
<td>0.941</td>
<td>0.352</td>
<td>0.352</td>
<td>0.011</td>
<td>0.512</td>
</tr>
<tr>
<td>0.9</td>
<td>20</td>
<td>176</td>
<td>1</td>
<td>71</td>
<td>0.952</td>
<td>0.220</td>
<td>0.220</td>
<td>0.006</td>
<td>0.357</td>
</tr>
</tbody>
</table>

**Table A.17: Input file saturated by 10% with NOPs**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>65</td>
<td>128</td>
<td>49</td>
<td>26</td>
<td>0.570</td>
<td>0.714</td>
<td>0.714</td>
<td>0.277</td>
<td>0.634</td>
</tr>
<tr>
<td>0.2</td>
<td>52</td>
<td>142</td>
<td>35</td>
<td>39</td>
<td>0.598</td>
<td>0.571</td>
<td>0.571</td>
<td>0.198</td>
<td>0.584</td>
</tr>
<tr>
<td>0.3</td>
<td>49</td>
<td>157</td>
<td>20</td>
<td>42</td>
<td>0.710</td>
<td>0.538</td>
<td>0.538</td>
<td>0.113</td>
<td>0.613</td>
</tr>
<tr>
<td>0.4</td>
<td>44</td>
<td>163</td>
<td>14</td>
<td>47</td>
<td>0.759</td>
<td>0.484</td>
<td>0.484</td>
<td>0.079</td>
<td>0.591</td>
</tr>
<tr>
<td>0.5</td>
<td>39</td>
<td>170</td>
<td>7</td>
<td>52</td>
<td>0.848</td>
<td>0.429</td>
<td>0.429</td>
<td>0.040</td>
<td>0.569</td>
</tr>
<tr>
<td>0.6</td>
<td>36</td>
<td>173</td>
<td>4</td>
<td>55</td>
<td>0.900</td>
<td>0.396</td>
<td>0.396</td>
<td>0.023</td>
<td>0.550</td>
</tr>
<tr>
<td>0.7</td>
<td>33</td>
<td>174</td>
<td>3</td>
<td>58</td>
<td>0.917</td>
<td>0.363</td>
<td>0.363</td>
<td>0.017</td>
<td>0.520</td>
</tr>
<tr>
<td>0.8</td>
<td>29</td>
<td>177</td>
<td>0</td>
<td>62</td>
<td>1.000</td>
<td>0.319</td>
<td>0.319</td>
<td>0.000</td>
<td>0.483</td>
</tr>
<tr>
<td>0.9</td>
<td>21</td>
<td>177</td>
<td>0</td>
<td>70</td>
<td>1.000</td>
<td>0.231</td>
<td>0.231</td>
<td>0.000</td>
<td>0.375</td>
</tr>
</tbody>
</table>

**Table A.18: Input file saturated by 20% with NOPs**
<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>72</td>
<td>133</td>
<td>44</td>
<td>19</td>
<td>0.621</td>
<td>0.791</td>
<td>0.791</td>
<td>0.249</td>
<td>0.696</td>
</tr>
<tr>
<td>0.2</td>
<td>60</td>
<td>153</td>
<td>31</td>
<td>44</td>
<td>0.594</td>
<td>0.659</td>
<td>0.659</td>
<td>0.232</td>
<td>0.625</td>
</tr>
<tr>
<td>0.3</td>
<td>53</td>
<td>160</td>
<td>17</td>
<td>38</td>
<td>0.714</td>
<td>0.582</td>
<td>0.582</td>
<td>0.136</td>
<td>0.686</td>
</tr>
<tr>
<td>0.4</td>
<td>47</td>
<td>164</td>
<td>13</td>
<td>44</td>
<td>0.783</td>
<td>0.516</td>
<td>0.516</td>
<td>0.073</td>
<td>0.623</td>
</tr>
<tr>
<td>0.5</td>
<td>39</td>
<td>166</td>
<td>11</td>
<td>52</td>
<td>0.780</td>
<td>0.429</td>
<td>0.429</td>
<td>0.062</td>
<td>0.553</td>
</tr>
<tr>
<td>0.6</td>
<td>32</td>
<td>171</td>
<td>6</td>
<td>59</td>
<td>0.842</td>
<td>0.352</td>
<td>0.352</td>
<td>0.034</td>
<td>0.496</td>
</tr>
<tr>
<td>0.7</td>
<td>29</td>
<td>172</td>
<td>5</td>
<td>62</td>
<td>0.853</td>
<td>0.319</td>
<td>0.319</td>
<td>0.028</td>
<td>0.464</td>
</tr>
<tr>
<td>0.8</td>
<td>23</td>
<td>172</td>
<td>5</td>
<td>68</td>
<td>0.821</td>
<td>0.253</td>
<td>0.253</td>
<td>0.028</td>
<td>0.387</td>
</tr>
<tr>
<td>0.9</td>
<td>13</td>
<td>176</td>
<td>1</td>
<td>78</td>
<td>0.929</td>
<td>0.143</td>
<td>0.143</td>
<td>0.006</td>
<td>0.248</td>
</tr>
</tbody>
</table>

**Table A.19: Input file saturated by 30% with NOPs**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>61</td>
<td>136</td>
<td>41</td>
<td>31</td>
<td>0.594</td>
<td>0.659</td>
<td>0.659</td>
<td>0.232</td>
<td>0.625</td>
</tr>
<tr>
<td>0.2</td>
<td>49</td>
<td>157</td>
<td>20</td>
<td>42</td>
<td>0.710</td>
<td>0.538</td>
<td>0.538</td>
<td>0.113</td>
<td>0.613</td>
</tr>
<tr>
<td>0.3</td>
<td>43</td>
<td>164</td>
<td>13</td>
<td>48</td>
<td>0.768</td>
<td>0.473</td>
<td>0.473</td>
<td>0.073</td>
<td>0.585</td>
</tr>
<tr>
<td>0.4</td>
<td>40</td>
<td>171</td>
<td>6</td>
<td>51</td>
<td>0.870</td>
<td>0.440</td>
<td>0.440</td>
<td>0.034</td>
<td>0.584</td>
</tr>
<tr>
<td>0.5</td>
<td>34</td>
<td>171</td>
<td>6</td>
<td>57</td>
<td>0.850</td>
<td>0.374</td>
<td>0.374</td>
<td>0.034</td>
<td>0.519</td>
</tr>
<tr>
<td>0.6</td>
<td>31</td>
<td>173</td>
<td>4</td>
<td>60</td>
<td>0.886</td>
<td>0.341</td>
<td>0.341</td>
<td>0.023</td>
<td>0.492</td>
</tr>
<tr>
<td>0.7</td>
<td>25</td>
<td>174</td>
<td>3</td>
<td>66</td>
<td>0.893</td>
<td>0.275</td>
<td>0.275</td>
<td>0.017</td>
<td>0.420</td>
</tr>
<tr>
<td>0.8</td>
<td>22</td>
<td>176</td>
<td>1</td>
<td>69</td>
<td>0.957</td>
<td>0.242</td>
<td>0.242</td>
<td>0.006</td>
<td>0.386</td>
</tr>
<tr>
<td>0.9</td>
<td>8</td>
<td>176</td>
<td>1</td>
<td>83</td>
<td>0.889</td>
<td>0.088</td>
<td>0.088</td>
<td>0.006</td>
<td>0.160</td>
</tr>
</tbody>
</table>

**Table A.20: Input file saturated by 40% with NOPs**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>61</td>
<td>118</td>
<td>59</td>
<td>30</td>
<td>0.508</td>
<td>0.670</td>
<td>0.670</td>
<td>0.333</td>
<td>0.578</td>
</tr>
<tr>
<td>0.2</td>
<td>51</td>
<td>142</td>
<td>35</td>
<td>40</td>
<td>0.593</td>
<td>0.560</td>
<td>0.560</td>
<td>0.198</td>
<td>0.576</td>
</tr>
<tr>
<td>0.3</td>
<td>44</td>
<td>160</td>
<td>17</td>
<td>47</td>
<td>0.721</td>
<td>0.484</td>
<td>0.484</td>
<td>0.096</td>
<td>0.579</td>
</tr>
<tr>
<td>0.4</td>
<td>34</td>
<td>164</td>
<td>13</td>
<td>57</td>
<td>0.723</td>
<td>0.374</td>
<td>0.374</td>
<td>0.073</td>
<td>0.493</td>
</tr>
<tr>
<td>0.5</td>
<td>28</td>
<td>171</td>
<td>6</td>
<td>63</td>
<td>0.824</td>
<td>0.308</td>
<td>0.308</td>
<td>0.034</td>
<td>0.448</td>
</tr>
<tr>
<td>0.6</td>
<td>23</td>
<td>172</td>
<td>5</td>
<td>68</td>
<td>0.821</td>
<td>0.253</td>
<td>0.253</td>
<td>0.028</td>
<td>0.387</td>
</tr>
<tr>
<td>0.7</td>
<td>18</td>
<td>175</td>
<td>2</td>
<td>73</td>
<td>0.900</td>
<td>0.198</td>
<td>0.198</td>
<td>0.011</td>
<td>0.324</td>
</tr>
<tr>
<td>0.8</td>
<td>11</td>
<td>176</td>
<td>1</td>
<td>80</td>
<td>0.917</td>
<td>0.121</td>
<td>0.121</td>
<td>0.006</td>
<td>0.214</td>
</tr>
<tr>
<td>0.9</td>
<td>4</td>
<td>176</td>
<td>1</td>
<td>87</td>
<td>0.800</td>
<td>0.044</td>
<td>0.044</td>
<td>0.006</td>
<td>0.083</td>
</tr>
</tbody>
</table>

**Table A.21: Input file saturated by 50% with NOPs**
### A.3.2 Grayscale with Resnet50

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>88</td>
<td>40</td>
<td>137</td>
<td>3</td>
<td>0.391</td>
<td>0.967</td>
<td>0.967</td>
<td>0.774</td>
<td>0.557</td>
</tr>
<tr>
<td>0.2</td>
<td>70</td>
<td>64</td>
<td>113</td>
<td>21</td>
<td>0.383</td>
<td>0.769</td>
<td>0.769</td>
<td>0.638</td>
<td>0.511</td>
</tr>
<tr>
<td>0.3</td>
<td>69</td>
<td>75</td>
<td>102</td>
<td>22</td>
<td>0.404</td>
<td>0.758</td>
<td>0.758</td>
<td>0.576</td>
<td>0.527</td>
</tr>
<tr>
<td>0.4</td>
<td>67</td>
<td>78</td>
<td>99</td>
<td>24</td>
<td>0.404</td>
<td>0.736</td>
<td>0.736</td>
<td>0.559</td>
<td>0.521</td>
</tr>
<tr>
<td>0.5</td>
<td>62</td>
<td>84</td>
<td>93</td>
<td>29</td>
<td>0.400</td>
<td>0.681</td>
<td>0.681</td>
<td>0.525</td>
<td>0.504</td>
</tr>
<tr>
<td>0.6</td>
<td>61</td>
<td>90</td>
<td>87</td>
<td>30</td>
<td>0.412</td>
<td>0.670</td>
<td>0.670</td>
<td>0.492</td>
<td>0.510</td>
</tr>
<tr>
<td>0.7</td>
<td>59</td>
<td>103</td>
<td>74</td>
<td>32</td>
<td>0.444</td>
<td>0.648</td>
<td>0.648</td>
<td>0.418</td>
<td>0.527</td>
</tr>
<tr>
<td>0.8</td>
<td>55</td>
<td>112</td>
<td>65</td>
<td>36</td>
<td>0.458</td>
<td>0.604</td>
<td>0.604</td>
<td>0.367</td>
<td>0.521</td>
</tr>
<tr>
<td>0.9</td>
<td>41</td>
<td>138</td>
<td>39</td>
<td>50</td>
<td>0.513</td>
<td>0.451</td>
<td>0.451</td>
<td>0.220</td>
<td>0.480</td>
</tr>
</tbody>
</table>

**Table A.22: Input file saturated by 10% with NOPs**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>91</td>
<td>42</td>
<td>135</td>
<td>3</td>
<td>0.395</td>
<td>0.967</td>
<td>0.967</td>
<td>0.763</td>
<td>0.561</td>
</tr>
<tr>
<td>0.2</td>
<td>77</td>
<td>61</td>
<td>113</td>
<td>20</td>
<td>0.386</td>
<td>0.780</td>
<td>0.780</td>
<td>0.638</td>
<td>0.516</td>
</tr>
<tr>
<td>0.3</td>
<td>67</td>
<td>75</td>
<td>102</td>
<td>24</td>
<td>0.396</td>
<td>0.736</td>
<td>0.736</td>
<td>0.576</td>
<td>0.515</td>
</tr>
<tr>
<td>0.4</td>
<td>65</td>
<td>81</td>
<td>96</td>
<td>26</td>
<td>0.404</td>
<td>0.714</td>
<td>0.714</td>
<td>0.542</td>
<td>0.516</td>
</tr>
<tr>
<td>0.5</td>
<td>60</td>
<td>87</td>
<td>90</td>
<td>31</td>
<td>0.400</td>
<td>0.659</td>
<td>0.659</td>
<td>0.508</td>
<td>0.498</td>
</tr>
<tr>
<td>0.6</td>
<td>55</td>
<td>94</td>
<td>83</td>
<td>36</td>
<td>0.399</td>
<td>0.604</td>
<td>0.604</td>
<td>0.469</td>
<td>0.480</td>
</tr>
<tr>
<td>0.7</td>
<td>52</td>
<td>105</td>
<td>72</td>
<td>39</td>
<td>0.419</td>
<td>0.571</td>
<td>0.571</td>
<td>0.407</td>
<td>0.484</td>
</tr>
<tr>
<td>0.8</td>
<td>42</td>
<td>112</td>
<td>65</td>
<td>49</td>
<td>0.393</td>
<td>0.462</td>
<td>0.462</td>
<td>0.367</td>
<td>0.424</td>
</tr>
<tr>
<td>0.9</td>
<td>35</td>
<td>147</td>
<td>30</td>
<td>56</td>
<td>0.538</td>
<td>0.385</td>
<td>0.385</td>
<td>0.169</td>
<td>0.449</td>
</tr>
</tbody>
</table>

**Table A.23: Input file saturated by 20% with NOPs**

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>91</td>
<td>42</td>
<td>135</td>
<td>0</td>
<td>0.403</td>
<td>1.000</td>
<td>1.000</td>
<td>0.763</td>
<td>0.574</td>
</tr>
<tr>
<td>0.2</td>
<td>77</td>
<td>61</td>
<td>116</td>
<td>14</td>
<td>0.399</td>
<td>0.846</td>
<td>0.846</td>
<td>0.655</td>
<td>0.542</td>
</tr>
<tr>
<td>0.3</td>
<td>72</td>
<td>74</td>
<td>103</td>
<td>19</td>
<td>0.411</td>
<td>0.791</td>
<td>0.791</td>
<td>0.582</td>
<td>0.541</td>
</tr>
<tr>
<td>0.4</td>
<td>69</td>
<td>80</td>
<td>97</td>
<td>22</td>
<td>0.416</td>
<td>0.758</td>
<td>0.758</td>
<td>0.548</td>
<td>0.537</td>
</tr>
<tr>
<td>0.5</td>
<td>68</td>
<td>90</td>
<td>87</td>
<td>23</td>
<td>0.439</td>
<td>0.747</td>
<td>0.747</td>
<td>0.492</td>
<td>0.553</td>
</tr>
<tr>
<td>0.6</td>
<td>64</td>
<td>94</td>
<td>83</td>
<td>27</td>
<td>0.435</td>
<td>0.703</td>
<td>0.703</td>
<td>0.469</td>
<td>0.538</td>
</tr>
<tr>
<td>0.7</td>
<td>58</td>
<td>102</td>
<td>75</td>
<td>33</td>
<td>0.436</td>
<td>0.637</td>
<td>0.637</td>
<td>0.424</td>
<td>0.518</td>
</tr>
<tr>
<td>0.8</td>
<td>46</td>
<td>113</td>
<td>64</td>
<td>45</td>
<td>0.418</td>
<td>0.505</td>
<td>0.505</td>
<td>0.362</td>
<td>0.458</td>
</tr>
<tr>
<td>0.9</td>
<td>27</td>
<td>136</td>
<td>41</td>
<td>64</td>
<td>0.397</td>
<td>0.297</td>
<td>0.297</td>
<td>0.232</td>
<td>0.340</td>
</tr>
</tbody>
</table>

**Table A.24: Input file saturated by 30% with NOPs**

### Appendix A. Raw Experiment Results

#### Table A.25: Input file saturated by 40% with NOPs

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>87</td>
<td>42</td>
<td>135</td>
<td>4</td>
<td>0.392</td>
<td>0.956</td>
<td>0.956</td>
<td>0.556</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>73</td>
<td>67</td>
<td>110</td>
<td>18</td>
<td>0.399</td>
<td>0.802</td>
<td>0.802</td>
<td>0.533</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>67</td>
<td>78</td>
<td>99</td>
<td>24</td>
<td>0.404</td>
<td>0.736</td>
<td>0.736</td>
<td>0.521</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>64</td>
<td>88</td>
<td>89</td>
<td>27</td>
<td>0.418</td>
<td>0.703</td>
<td>0.703</td>
<td>0.525</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>60</td>
<td>91</td>
<td>86</td>
<td>31</td>
<td>0.411</td>
<td>0.659</td>
<td>0.659</td>
<td>0.506</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>56</td>
<td>99</td>
<td>78</td>
<td>35</td>
<td>0.418</td>
<td>0.615</td>
<td>0.615</td>
<td>0.498</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>52</td>
<td>107</td>
<td>70</td>
<td>39</td>
<td>0.426</td>
<td>0.571</td>
<td>0.571</td>
<td>0.488</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>44</td>
<td>119</td>
<td>58</td>
<td>47</td>
<td>0.431</td>
<td>0.484</td>
<td>0.484</td>
<td>0.456</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>24</td>
<td>142</td>
<td>35</td>
<td>67</td>
<td>0.407</td>
<td>0.264</td>
<td>0.264</td>
<td>0.320</td>
<td></td>
</tr>
</tbody>
</table>

#### Table A.26: Input file saturated by 50% with NOPs

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>88</td>
<td>42</td>
<td>135</td>
<td>3</td>
<td>0.395</td>
<td>0.967</td>
<td>0.967</td>
<td>0.561</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>81</td>
<td>63</td>
<td>114</td>
<td>10</td>
<td>0.415</td>
<td>0.890</td>
<td>0.890</td>
<td>0.566</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>76</td>
<td>76</td>
<td>101</td>
<td>15</td>
<td>0.429</td>
<td>0.835</td>
<td>0.835</td>
<td>0.567</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>73</td>
<td>84</td>
<td>93</td>
<td>18</td>
<td>0.440</td>
<td>0.802</td>
<td>0.802</td>
<td>0.568</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>70</td>
<td>87</td>
<td>90</td>
<td>21</td>
<td>0.438</td>
<td>0.769</td>
<td>0.769</td>
<td>0.558</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>67</td>
<td>99</td>
<td>78</td>
<td>24</td>
<td>0.462</td>
<td>0.736</td>
<td>0.736</td>
<td>0.568</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>63</td>
<td>104</td>
<td>73</td>
<td>28</td>
<td>0.463</td>
<td>0.692</td>
<td>0.692</td>
<td>0.555</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>54</td>
<td>118</td>
<td>59</td>
<td>37</td>
<td>0.478</td>
<td>0.593</td>
<td>0.593</td>
<td>0.529</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>35</td>
<td>138</td>
<td>39</td>
<td>56</td>
<td>0.473</td>
<td>0.385</td>
<td>0.385</td>
<td>0.220</td>
<td></td>
</tr>
</tbody>
</table>

#### A.3.3 Grayscale with Plain Network

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>48</td>
<td>159</td>
<td>18</td>
<td>43</td>
<td>0.727</td>
<td>0.527</td>
<td>0.527</td>
<td>0.611</td>
<td></td>
</tr>
<tr>
<td>0.2</td>
<td>36</td>
<td>164</td>
<td>13</td>
<td>55</td>
<td>0.735</td>
<td>0.396</td>
<td>0.396</td>
<td>0.514</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>31</td>
<td>169</td>
<td>8</td>
<td>60</td>
<td>0.795</td>
<td>0.341</td>
<td>0.341</td>
<td>0.477</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>24</td>
<td>171</td>
<td>6</td>
<td>67</td>
<td>0.800</td>
<td>0.264</td>
<td>0.264</td>
<td>0.397</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>17</td>
<td>173</td>
<td>4</td>
<td>74</td>
<td>0.810</td>
<td>0.187</td>
<td>0.187</td>
<td>0.304</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>11</td>
<td>173</td>
<td>4</td>
<td>80</td>
<td>0.733</td>
<td>0.121</td>
<td>0.121</td>
<td>0.208</td>
<td></td>
</tr>
<tr>
<td>0.7</td>
<td>9</td>
<td>175</td>
<td>2</td>
<td>82</td>
<td>0.818</td>
<td>0.099</td>
<td>0.099</td>
<td>0.176</td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>6</td>
<td>176</td>
<td>1</td>
<td>85</td>
<td>0.857</td>
<td>0.066</td>
<td>0.066</td>
<td>0.122</td>
<td></td>
</tr>
<tr>
<td>0.9</td>
<td>1</td>
<td>177</td>
<td>0</td>
<td>90</td>
<td>1.000</td>
<td>0.011</td>
<td>0.011</td>
<td>0.022</td>
<td></td>
</tr>
</tbody>
</table>
### Appendix A. Raw Experiment Results

#### Table A.28: Input file saturated by 20% with NOPs

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>44</td>
<td>155</td>
<td>22</td>
<td>47</td>
<td>0.667</td>
<td>0.484</td>
<td>0.484</td>
<td>0.124</td>
<td>0.561</td>
</tr>
<tr>
<td>0.2</td>
<td>33</td>
<td>164</td>
<td>13</td>
<td>58</td>
<td>0.717</td>
<td>0.363</td>
<td>0.363</td>
<td>0.073</td>
<td>0.482</td>
</tr>
<tr>
<td>0.3</td>
<td>25</td>
<td>167</td>
<td>10</td>
<td>66</td>
<td>0.714</td>
<td>0.275</td>
<td>0.275</td>
<td>0.056</td>
<td>0.397</td>
</tr>
<tr>
<td>0.4</td>
<td>18</td>
<td>171</td>
<td>6</td>
<td>73</td>
<td>0.750</td>
<td>0.198</td>
<td>0.198</td>
<td>0.034</td>
<td>0.313</td>
</tr>
<tr>
<td>0.5</td>
<td>15</td>
<td>173</td>
<td>4</td>
<td>76</td>
<td>0.789</td>
<td>0.165</td>
<td>0.165</td>
<td>0.023</td>
<td>0.273</td>
</tr>
<tr>
<td>0.6</td>
<td>14</td>
<td>177</td>
<td>0</td>
<td>77</td>
<td>1.000</td>
<td>0.154</td>
<td>0.154</td>
<td>0.000</td>
<td>0.267</td>
</tr>
<tr>
<td>0.7</td>
<td>10</td>
<td>177</td>
<td>0</td>
<td>81</td>
<td>1.000</td>
<td>0.110</td>
<td>0.110</td>
<td>0.000</td>
<td>0.198</td>
</tr>
<tr>
<td>0.8</td>
<td>5</td>
<td>177</td>
<td>0</td>
<td>86</td>
<td>1.000</td>
<td>0.055</td>
<td>0.055</td>
<td>0.000</td>
<td>0.104</td>
</tr>
<tr>
<td>0.9</td>
<td>2</td>
<td>177</td>
<td>0</td>
<td>89</td>
<td>1.000</td>
<td>0.022</td>
<td>0.022</td>
<td>0.000</td>
<td>0.043</td>
</tr>
</tbody>
</table>

#### Table A.29: Input file saturated by 30% with NOPs

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>38</td>
<td>157</td>
<td>20</td>
<td>53</td>
<td>0.655</td>
<td>0.418</td>
<td>0.418</td>
<td>0.113</td>
<td>0.510</td>
</tr>
<tr>
<td>0.2</td>
<td>30</td>
<td>166</td>
<td>11</td>
<td>61</td>
<td>0.732</td>
<td>0.330</td>
<td>0.330</td>
<td>0.062</td>
<td>0.455</td>
</tr>
<tr>
<td>0.3</td>
<td>19</td>
<td>170</td>
<td>7</td>
<td>72</td>
<td>0.731</td>
<td>0.209</td>
<td>0.209</td>
<td>0.040</td>
<td>0.325</td>
</tr>
<tr>
<td>0.4</td>
<td>17</td>
<td>173</td>
<td>4</td>
<td>74</td>
<td>0.810</td>
<td>0.187</td>
<td>0.187</td>
<td>0.023</td>
<td>0.304</td>
</tr>
<tr>
<td>0.5</td>
<td>14</td>
<td>175</td>
<td>2</td>
<td>77</td>
<td>0.875</td>
<td>0.154</td>
<td>0.154</td>
<td>0.011</td>
<td>0.262</td>
</tr>
<tr>
<td>0.6</td>
<td>10</td>
<td>177</td>
<td>0</td>
<td>81</td>
<td>1.000</td>
<td>0.110</td>
<td>0.110</td>
<td>0.000</td>
<td>0.198</td>
</tr>
<tr>
<td>0.7</td>
<td>8</td>
<td>177</td>
<td>0</td>
<td>83</td>
<td>1.000</td>
<td>0.088</td>
<td>0.088</td>
<td>0.000</td>
<td>0.162</td>
</tr>
<tr>
<td>0.8</td>
<td>7</td>
<td>177</td>
<td>0</td>
<td>84</td>
<td>1.000</td>
<td>0.077</td>
<td>0.077</td>
<td>0.000</td>
<td>0.143</td>
</tr>
<tr>
<td>0.9</td>
<td>4</td>
<td>177</td>
<td>0</td>
<td>87</td>
<td>1.000</td>
<td>0.044</td>
<td>0.044</td>
<td>0.000</td>
<td>0.084</td>
</tr>
</tbody>
</table>

#### Table A.30: Input file saturated by 40% with NOPs

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>40</td>
<td>157</td>
<td>20</td>
<td>51</td>
<td>0.667</td>
<td>0.440</td>
<td>0.440</td>
<td>0.113</td>
<td>0.530</td>
</tr>
<tr>
<td>0.2</td>
<td>31</td>
<td>164</td>
<td>13</td>
<td>60</td>
<td>0.705</td>
<td>0.341</td>
<td>0.341</td>
<td>0.073</td>
<td>0.459</td>
</tr>
<tr>
<td>0.3</td>
<td>20</td>
<td>168</td>
<td>9</td>
<td>71</td>
<td>0.690</td>
<td>0.220</td>
<td>0.220</td>
<td>0.051</td>
<td>0.333</td>
</tr>
<tr>
<td>0.4</td>
<td>11</td>
<td>171</td>
<td>6</td>
<td>80</td>
<td>0.647</td>
<td>0.121</td>
<td>0.121</td>
<td>0.034</td>
<td>0.204</td>
</tr>
<tr>
<td>0.5</td>
<td>7</td>
<td>174</td>
<td>3</td>
<td>84</td>
<td>0.700</td>
<td>0.077</td>
<td>0.077</td>
<td>0.017</td>
<td>0.139</td>
</tr>
<tr>
<td>0.6</td>
<td>6</td>
<td>175</td>
<td>2</td>
<td>85</td>
<td>0.750</td>
<td>0.066</td>
<td>0.066</td>
<td>0.011</td>
<td>0.121</td>
</tr>
<tr>
<td>0.7</td>
<td>5</td>
<td>176</td>
<td>1</td>
<td>86</td>
<td>0.833</td>
<td>0.055</td>
<td>0.055</td>
<td>0.006</td>
<td>0.103</td>
</tr>
<tr>
<td>0.8</td>
<td>4</td>
<td>176</td>
<td>1</td>
<td>87</td>
<td>0.800</td>
<td>0.044</td>
<td>0.044</td>
<td>0.006</td>
<td>0.083</td>
</tr>
<tr>
<td>0.9</td>
<td>3</td>
<td>177</td>
<td>0</td>
<td>88</td>
<td>1.000</td>
<td>0.033</td>
<td>0.033</td>
<td>0.000</td>
<td>0.064</td>
</tr>
</tbody>
</table>
# Appendix A. Raw Experiment Results

<table>
<thead>
<tr>
<th>Threshold</th>
<th>TP</th>
<th>TN</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
<th>TPR</th>
<th>FPR</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>38</td>
<td>156</td>
<td>21</td>
<td>53</td>
<td>0.644</td>
<td>0.418</td>
<td>0.418</td>
<td>0.119</td>
<td>0.507</td>
</tr>
<tr>
<td>0.2</td>
<td>24</td>
<td>165</td>
<td>12</td>
<td>67</td>
<td>0.667</td>
<td>0.264</td>
<td>0.264</td>
<td>0.068</td>
<td>0.378</td>
</tr>
<tr>
<td>0.3</td>
<td>17</td>
<td>168</td>
<td>9</td>
<td>74</td>
<td>0.654</td>
<td>0.187</td>
<td>0.187</td>
<td>0.051</td>
<td>0.291</td>
</tr>
<tr>
<td>0.4</td>
<td>15</td>
<td>169</td>
<td>8</td>
<td>76</td>
<td>0.652</td>
<td>0.165</td>
<td>0.165</td>
<td>0.045</td>
<td>0.263</td>
</tr>
<tr>
<td>0.5</td>
<td>11</td>
<td>170</td>
<td>7</td>
<td>80</td>
<td>0.611</td>
<td>0.121</td>
<td>0.121</td>
<td>0.040</td>
<td>0.202</td>
</tr>
<tr>
<td>0.6</td>
<td>9</td>
<td>174</td>
<td>3</td>
<td>82</td>
<td>0.750</td>
<td>0.099</td>
<td>0.099</td>
<td>0.017</td>
<td>0.175</td>
</tr>
<tr>
<td>0.7</td>
<td>7</td>
<td>176</td>
<td>1</td>
<td>84</td>
<td>0.875</td>
<td>0.077</td>
<td>0.077</td>
<td>0.006</td>
<td>0.141</td>
</tr>
<tr>
<td>0.8</td>
<td>6</td>
<td>177</td>
<td>0</td>
<td>85</td>
<td>1.000</td>
<td>0.066</td>
<td>0.066</td>
<td>0.000</td>
<td>0.124</td>
</tr>
<tr>
<td>0.9</td>
<td>4</td>
<td>177</td>
<td>0</td>
<td>87</td>
<td>1.000</td>
<td>0.044</td>
<td>0.044</td>
<td>0.000</td>
<td>0.084</td>
</tr>
</tbody>
</table>

Table A.31: Input file saturated by 50% with NOPs