An Investigation on Self-Attentive Models for Malware Classification

by

Qikai Lu

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Malware classification is a critical task in cybersecurity. It offers insights on the threats posed to victim devices from different malware and aids in the designing of precautionary measures. In real world applications, due to the vast amount of malware present in the networks, real-time malware classification must be both accurate and fast. In this thesis, we first investigate the application of self-attentive models to classify malicious binary files from raw bytes alone. We propose two transformer-based models. The first model, SeqConvAttn, conducts sequenced-based classification using byte sequences extracted from binary files. Noting that the feedforward latency of SeqConvAttn scales poorly to input sequence length, we then experimented with converting binary files into images, and introduced the second model, ImgConvAttn, to apply self-attention to image-based classification. Next, we investigated the integration of the two models into a two-stage framework, such that the superior accuracy and low latency of the respective models can both be leveraged. Through experiments on the BIG 2015 Dataset provided by the Microsoft Malware Classification Challenge and a select subset of the BODMAS Malware Dataset, we demonstrate that self-attention can enhance the accuracy of malware classifiers for both sequence-based and image-based classification. Furthermore, we demonstrate that our two-stage framework design can reduce inference latency significantly while maintaining high accuracy.

Preface

A portion of the content presented in this thesis, including Chapter 2, 3, 4, 5, and 6 have been submitted to IEEE INFOCOM 2022 for review.

Acknowledgements

I would like to thank Dr. Niu for helping me through my research. His supervision had helped me greatly in overcoming the difficulties encountered during my research. Furthermore, his guidance offered me lasting insights on how to become a good researcher.

I would also like to thank my family, who offered me support through these years. They have offered me encouragements during these times, allowing me to pursue my research with full effort.

Table of Contents

List of Tables

List of Figures

List of Variables

Transformer-based Model

B Expansion factor in feedfoward component.

d_m, i_d Encoding dimension of X and index of encoding dimension, respectively.

d_{qkv} Encoding dimension of query, key, and value sequence.

H Number of heads in multihead-attention.

K Number of serially connected transformer blocks.

M Raw byte sequence length.

N, n Length (number of elements) of X and position index of some element in X, respectively.

S Size of both kernel and stride in convolutional networks of SeqConvAttn and ImgConvAttn.

W_1, b_1 Feedforward expansion layer weight and bias, respectively.

W_2, b_2 Feedforward restorative layer weight and bias, respectively.

W_O Multihead-attention weight that projects the concatenation of parallel self-attention outputs back into an encoding sequence.

W_q, W_k, W_v Self-attention weights that projects X into query, key, value sequence, respectively.

X Input sequence of the transformer architecture.

Two-Stage Framework

v, $p\%$ Uncertainty threshold and expected percentage of malware files undergoing second-stage reclassification, respectively.

C_{pred}, U_{pred} Predicted class and classification probability, respectively.

t_1, t_2 Inference latency of the first and second stage, respectively.

t_{spec} Specified latency constraint.

Chapter 1

Introduction

1.1 Problem Motivation

Malware, also known as malicious software, are software that conducts malicious activities when executed on a victim device. These activities include a wide-range of exploitative and destructive behaviours, such as the theft of personal information, potential corruption of stored data, providing unverified agents access to a device, etc. Classification of malware, based on behavioural or static properties, is an important measure for curtailing the damage inflicted by malware. On infected devices, identifying the malware type allows for accurate diagnostics, resulting in effective quarantining and removal of the malware. Detecting particular malware types from connected networks allows for security analysts to design effective precautionary measures, in anticipation of future attacks against endpoint devices. Yet, while it is critical that the classification engine must be accurate, its inference process must also be timely. A major challenge in the anti-virus industry is the vast amount of files constantly being transferred through the networks. For example, during the first quarter of 2020, Kaspersky reported 164,653,290 "unique malicious and potentially unwanted objects" [1]. To handle the vast quantity of malware efficiently, endpoint security software need to conduct malware classification and analysis on incoming software files from external networks in real-time. Thus, the average inference latency must be low, as otherwise the resultant slowdown will negatively impact other operations running on the endpoint devices. Therefore, achieving high accuracy with low latency is the principal

objective of real-time malware classification.

1.2 Malware Classification

By methodology, malware classification can be broadly separated into two categories: dynamic and static analysis. Dynamic analysis usually requires the execution of suspicious software within a sandbox environment, with the malware activity observed in runtime. Although this approach can produce detailed information regarding the malicious behaviour of a file, it is slow to perform. The requirement of a sandbox renders this approach inconvenient in many scenarios. Alternatively, static analysis examines static features, or signatures, extracted from software files to predict malicious behaviour. This approach can be significantly faster than dynamic analysis, as malware execution is not required. Note that, many static analyses involve the decompilation of binaries into assembly code or human readable languages, which is time-consuming [2] and not suitable to real-time applications. Thus, many recent publications on developing neural networks for malware classification focuses on directly analyzing the raw bytes of the binary, with minimal preprocessing or feature engineering.

Neural network classifiers for malware binary files can be further categorized into two types: sequence-based and image-based classification. Sequence-based classifiers, such as [3] and [4], applies 1D convolution onto raw byte sequences for classification. However, a major challenge of sequence-based classification is the modelling of long byte sequences in an efficient manner [5]. Image-based classifiers, on the other hand, first converts a binary file into an image representation. Traditionally, this entails converting the byte sequence into a greyscale image [6]. However, more recently, some research investigated reforming the byte sequence onto frequency domain, such as [7], [8], and [9]. Many works, such as [10], [11], [12] and [13], have explored using different types of convolutional neural network (CNN) models to improve malware classification performance. Compared to byte sequences, images is viewed as a more efficient representation of the binary file, at the cost of losing some level of

information. We note, from our survey of related works, that barring some exception, such as [4] and [14], most classifiers are CNNs.

1.3 Our Contribution

We propose the application of the transformer architecture in both sequence and image analysis to achieve superior classification accuracy. The transformer architecture [15] was first introduced in NLP, and have since achieved state-of-the-art results in machine translation, summarization, question-and-answering, etc. [16]. Functionally, transformers predominantly rely on the self-attention mechanism to model inter-dependencies within a sequence. This allows the capture of global contexts between all pairs of elements within the sequence, whereas conventional convolutional layers could only capture local context of elements within the scope of a kernel. We first introduce *SeqConvAttn*, a CNN-transformer architecture designed to classify malwares from long byte sequences.

We further extend the transformer classifier to image analysis. Recently, [17] introduced the Vision Transformer, demonstrating that transformers could also be applied to image classification. The underlying idea is to partition an input image into non-overlapping sub-image patches. Self-attention can then model the inter-dependencies between different pairs of patches, thereby capturing the global context of the image. To our knowledge, there are currently no published work on using transformers for image-based malware classification. Thus we propose *ImgConvAttn*, a transformer model for classifying malware from its image representation.

Finally, we present a two-stage framework to incorporate the two models, SeqConvAttn and ImgConvAttn, together, such that the inference latency and accuracy can be flexibly controlled. By design, the first-stage model generally has a lower latency and but also lower accuracy, and the second-stage model has a higher accuracy while incurring a higher latency. The idea is to rely on the first stage for classification in the majority of cases, while sparingly employ the second stage only if the first-stage classification is uncertain. Based on

experimental observations, we also added a file-size-aware mechanism which pre-emptively diverts certain binary files directly to the second stage to optimize the framework, further reducing the inference latency while maintaining high classification accuracy.

We evaluated the performance of the proposed models on two datasets, the BIG 2015 Dataset from Microsoft Malware Classification Challenge [18] and Sub-BODMAS, a select subset of the BODMAS Malware [19] Dataset. Our experiments show that SeqConvAttn attained accuracy and weighted-F1 scores supperior to most baseline models on sequence-based classification. ImgConvAttn is also shown to be superior to the CNN baseline in image-based classification. We then evaluated the two-stage framework on the Sub-BODMAS dataset and showed that as compared to independent baseline models. The two-stage framework maintained a very high accuracy while reducing inference latency significantly.

1.4 Thesis Outline

The content of this thesis is organized in the following manner. Chapter 2 is a survey of related works. Chapter 3 explains of the designs of SeqConvAttn, ImgConvAttn, and the two-stage framework. Chapter 4 details our experiment settings global to all subsequent experiments. Chapter 5 presents the experimental results of SeqConvAttn and ImgConvAttn as independent models. Chapter 6 presents the experimental results of the two-stage framework. Chapter 7 contains the conclusion and discusses potential future works.

Chapter 2

Related Work

2.1 Malware Classification by Raw Bytes

Static analysis examines the signatures, static features derived from associated files, to determine the malware type. Technically, such files could be binaries, assembly code, or even human-readable programming language. However, in most situations, only software binaries are readily available. Decompilation of binaries into assembly code or programming language can often be time consuming [2]. To reduce the overall inference latency, file preprocessing time should also be minimized. We therefore forego any feature extraction techniques that involve assembly code or human readable languages, and focus solely on classification using raw binaries.

2.1.1 Byte Sequence Classification

Intuitively speaking, the content of a binary file can be directly examined as a sequence. Based on our survey, the first malware classifier for byte sequences developed is Malconv, proposed by [3]. Given a byte sequence, Malconv first performs byte embedding, then forwards the embedded sequence through gated convolution [20]. Notably, the convolutional layer kernel size and strides are set to 512 bytes, aggressively reducing the sequence length. [21] then proposed an alternative deep architecture using multiple convolutional layers, with smaller kernel and stride sizes. However, these models are only concerned with malware detection, a binary classification problem. [22], [23], and [24] formulated several adversarial attacks on Malconv to iden-

tify the weakness of the architecture. [25] conducted activation analysis to gain further insights about the information learned from byte sequences. The analysis concluded that filters from low-level convolutional layers were able to identify salient ASCII and instruction sequences. [26] then extended Malconv to multi-class classification on the BIG 2015 Dataset [18]. Finally, [5] introduced two major improvements on the original Malconv architecture. First, a convolution-over-time scheme was introduced, allowing Malconv to process binary files of arbitrary sizes in an efficient manner. Second, to better model dependencies between distant elements in the byte sequence, they proposed the Global Channel Gating mechanism.

Alternative to Malconv, [4] and [14] have also introduced CNN-BiLSTM and CNN-BiGRU, respectively, to process raw bytes. To handle the length of the byte sequence, these approaches directly handled the byte sequences as 1D greyscale images, and resized the images to a length of 10,000 elements. More importantly, these papers demonstrate the potential of applying NLP architectures to malware classification.

2.1.2 Malware Image Classification

The earliest research in image-based classification by malware was conducted by [6]. The approach used K-nearest neighbour classification on texture features extracted from malware greyscale image. More importantly, the approach laid the groundwork in binaries-to-image conversion. For each binary file, an image width is assigned according to the file size. The byte sequence from the file is then restructured by line-breaking over the image width. Interpreting the each byte value as a brightness value, the binary file is thus converted into a greyscale image. From our knowledge, the first known research on applying CNN classifier to malware images was published by [10]. Their investigated architectures are shallow, with the deepest model consisting of 3 convolutional layers and 2 fully connected layers. [11] proposed integrating deep networks with transfer-learning for malware classification, where the bottleneck feature from ResNet50 [27] is taken to train shallow classifier networks. To accommodate the input constraint of ResNet50, the greyscale is

directly converted into RGB image, and re-sized to the specified input dimensions. [12] repeated this approach, but replacing ResNet50 with VGG16 [28]. [29] further proposed integrating attention mechanism into CNN, both to improve classification accuracy and to identify salient sections in the greyscale image for analysis. [30] applied Inception-v3 [31] and Inception-ResNet-v2 [32] for android malware detection. [33] then further investigated transfer learning with ResNet50. In general, there is a clear trend of employing deep network for achieving greater accuracy. However, this also translates to higher per-file inference latency. On Table 2.1, we report the CPU latency for single image feedforward through common deep models based on their TorchVision implementation. While the latency could be significantly reduced by running these models on sufficiently powerful GPU, the environment firewall and end-point security software are installed in may lack such hardware, rendering them unsuitable for many applications.

Alternative to greyscales, some recent works have also proposed generating malware images by analyzing byte frequency. [7] proposed generating malware image through Markov image (transition matrix) rather than greyscales. The markov image has a dimension of 256×256, with each pixel representing the transition probability from one byte value to another in the byte sequence. [8] further experimented with training a deep CNN model from scratch using Markov images. [13] devised an architecture to accept the Markov image along with the greyscale and RGB images, augmenting the information extracted from the binary file. [9] investigated an alternative by directly recording the frequency count of each byte bigram, and then performing discrete cosine transform to desparsify the image.

We curtly note the existence of other byte-level feature engineering techniques for extracting texture statistics from greyscale images. For example, [34] experimented with using Gabor filter, histogram of oriented gradient, and local binary pattern analysis to enhance texture information of the greyscale. [35] proposed a texture partitioning and extraction technique to omit non-important regions of a greyscale image representation. [2] devised second order texture features by statistical methods. However, these methods currently do

not appear to be widespread.

Table 2.1: FeedForward Time of Image-based Deep Networks in TorchVision

Setting	**CPU Latency [ms]**
VGG-16 (Input of 256×256)	116.8
ResNet50 (Input of 256×256)	89.7
Inception-v3 (Input of 300×300)	115.3

2.2 Transformers in Malware Classification

Research on applying transformers to malware classification is fairly recent, with published works being relatively scarce. [36] devised I-MAD, a hierarchical transformer-based framework for classification on assembly code. The work defined three different hierarchy of content in an assembly file: assembly instructions, basic blocks, and assembly functions. The architecture employs three transformers, with each transformer responsible for computing the encoding of its respective data hierarchy. For example, the encoding of the basic block would be computed by using its component assembly instructions as the input of the corresponding transformer. [37] presented an alternative hierarchical transformer classifier, again for assembly code. [38] proposed a non-hierarchical transformer model, where the input is a sequence of high frequency words extracted from the assembly code. [39] investigated using variants of BERT [16] to detect Android malware based on files from decompiled APKs. Note that these works all require the decompilation of malware binaries to obtain the assembly code. From our knowledge, no works regarding the application of transformers to raw binaries for malware classification have yet been published.

2.3 Two-Stage Framework for Malware Classification

A number of works have investigated using two-stage frameworks for malware classification. However, the purpose of the stages differs greatly between designs. [40] implemented the framework to perform tiered classifications: the first stage detects malware from benignware, the second stage classifies the malware identified from the first stage. The designs of [41] and [42] are functionally similar, but specialized their second tier to only discerning whether the malware is a ranswomware. Echelon, developed by [43], uses the second stage to double-check software that are predicted benign by the first stage, reducing the overall false negative rate, where actual malware are predicted as benign. TuningMalconv, [44], uses the second stage to reclassify a malware only if the first stage classification is uncertain, with the intent of boosting the accuracy of the first stage without significantly increasing the average latency. TAMD [45] is functionally similar, but architecturally more comprehensive. The design employs model ensembles, rather than single models, in both stages. Additionally, the framework is designed to facilitate both efficient model training and classification.

Chapter 3

Model Designs

In this section, we first provide a background on the transformer architecture. Afterwards, the designs of our sequence-based and image-based classifiers are described separately. Finally, the design two-stage framework is given.

3.1 Background on Transformer

Transformer [15] refers to a family of architectures that relies self-attention to model inter-dependencies between elements in a sequence. The underlying idea of is that the encoding of any target element can be computed as the aggregate of the encodings of all source elements within the sequence. The extent that a particular source element contributes to the aggregation process is determined by an attentional weight. Colloquially, the weight is the amount of attention the target pays to the source. Mathematically, the entire aggregation process is referred to as scaled dot-product attention, defined by [15] as Equation 3.1.

$$SA(X) = softmax(\frac{XW_Q(XW_K)^T}{\sqrt{d_{kqv}}})XW_V \tag{3.1}$$

Note that $X \in \mathbb{R}^{N \times d_m}$ is the initial sequence encoding of length N and an encoding dimension of d_m. Learnable weights $W_Q, W_K, W_V \in \mathbb{R}^{d_m \times d_{kqv}}$ are used to project the X into, respectively, the query XW_Q, key XW_K, and value XW_V, with d_{kqv} as the encoding dimension of these sequences. From the perspective of the target element, scaled dot-product attention computes its encoding by conducting a weighted sum over all source elements in the value, with the attentional weights determined by the similarity of between

its corresponding query element and the every key element. By carrying out the entire self-attention computation as a series of matrix multiplication, all target element encodings are thus computed concurrently.

To learn the different types of inter-dependencies that may exist within the sequence, [15] proposed employing multiple scaled dot-product attention heads in parallel. As the weight of each attention head is initialized differently, different heads could potentially capture different type of inter-dependency in the sequence. To combine the information learned from the parallel heads, their outputs are concatenated along the encoding dimension and re-projected to a final encoding. The entire design is referred as multihead attention, with the mathematical definition is presented by 3.2.

$$MHA(X) = Concat([SA_h(X)]_{h=1}^{H})W^O \tag{3.2}$$

Here, H refers to the number of parallel attention heads, and $W_O \in \mathbb{R}^{Hd_{kqv} \times d_m}$ the post-concatenation projection weight. Note that in most transformer designs, the model dimension stays invariant after undergoing multihead attention. Thus, for all subsequent models, $d_m = Hd_{kqv}$.

A conventional transformer architecture is composed of multiple encoder blocks connected in a serial fashion, Each block contains a multihead attention component followed by a feedforward component, with interjecting residual connection after each component. The feedforward blocks consists of a ReLU activated expansion layer followed by restorative layer, as shown by Equation 3.3 [15].

$$FF(X) = ReLU(XW_1 + b_1)W_2 + b_2 \tag{3.3}$$

Note that the expansion layer parameters $W_1 \in \mathbb{R}^{d_m \times Bd_m}$, $b_1 \in \mathbb{R}^{Bd_m}$ and the restorative layer parameters $W_2 \in \mathbb{R}^{Bd_m \times d_m}$, $b_2 \in \mathbb{R}^{d_m}$. Here, B is referred to as an expansion factor.

Transformers have no implicit awareness of the positions of elements in the sequence. To remedy this, [15] proposed adding positional encoding to the initial sequence encoding before undergoing self-attention. The positional encodings of the two proposed models differ. Thus, further details are deferred to description about the individual models.

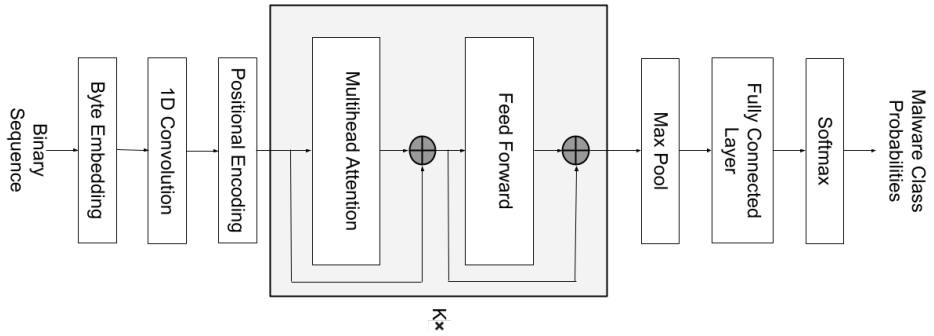

Figure 3.1: Note that the K transformer blocks are connected serially.

3.2 SeqConvAttn: Byte Sequence Classifier

Intuitively, the content of a binary file can be represented as a byte sequence. We devise the SeqConvAttn to classify malware from byte sequence alone, with the architecture shown on Figure 3.1. Given a byte sequence of length M, where $M >> N$, the input byte sequence first undergoes byte embedding. The embedding layer maps each of the 256 byte values (and a unique padding "byte") to a corresponding vector. The embedded sequence then undergoes 1D convolution. Following [3], the convolutional layer is designed with d_m large kernels with size S and stride of S. The purpose of this layer is to compress every segment of S bytes into a single sequence element, aggressively reducing the output sequence length to $N = \frac{M}{S}$. The length reduction is necessary as the computation of the scaled dot-product attention from Equation 3.1 scales quadratically to the sequence length N. Exceedingly long input sequence would therefore require too much computation resources and incur an unacceptably long feedforward time.

After propagating through the 1D convolutional layer, position encoding is added onto the post-convolution sequence, imbuing each element with information about its position in the sequence. The positional encoding of SeqConvAttn follows the original approach proposed by [15], as shown in Equation 3.4.

$$PE(n, i_d) = \begin{cases} \sin \frac{n}{10000^{\frac{2i}{d_m}}}, & i_d \bmod 2 = 0 \\ \cos \frac{n}{10000^{\frac{2i}{d_m}}}, & i_d \bmod 2 = 1 \end{cases} \tag{3.4}$$

Here $n \in \mathbb{Z} \cap [1, N]$ indicate the positional index of the an element in the sequence, and $i_d \in \mathbb{Z} \cap [1, d_m]$ the index of the encoding dimension. Essentially, this positional encoding expresses positional context through a set of alternating sinusoids. After adding positional encoding, the resultant sequence X then propagates through a series of K transformer blocks. We refrain from further exposition about transformers here, as details are already presented in Section 3.1. Upon obtaining the transformer output encoding, it is then max-pooled element-wise into a fixed dimension vector of dimension d_m. This vector then proceeds through additional fully-connected layers. The final output then undergoes softmax, yielding the classification probability.

One potential limitation of this architecture is that the SeqConvAttn, once trained, cannot adapt to the classification new malware classes without modification. The reason lies with the final layer of the fully connected-layers block, whose width needs to match the number of classes specified by the user. However, the actual modification necessary to accommodate changes in the number of malware classes is relatively minimal. Specifically, the final layer in the fully-connected layers block is replaced with a layer whose width matches the number of malware classes. Optionally, all parameters in the fully-connected layer are then re-initialized. Finally, the modified SeqConvAttn model is then finetuned with additional data to learn to predict the newly specified malware classes.

3.3 ImgConvAttn: Malware Image Classifier

As self-attention computation scales quadratically to input length, reducing the input length should result in latency reduction for malware classification. The most direct approach to reduce sequence length is truncation. However, overly truncating a byte sequence may remove salient sections critical to malware identification. For example, [5] presented a case where the malware author simply inserted malicious payload after the truncation index, thereby bypassing classifier detection. To overcome this issue, we explored converting the contents of binary files into images to obtain an efficient representation

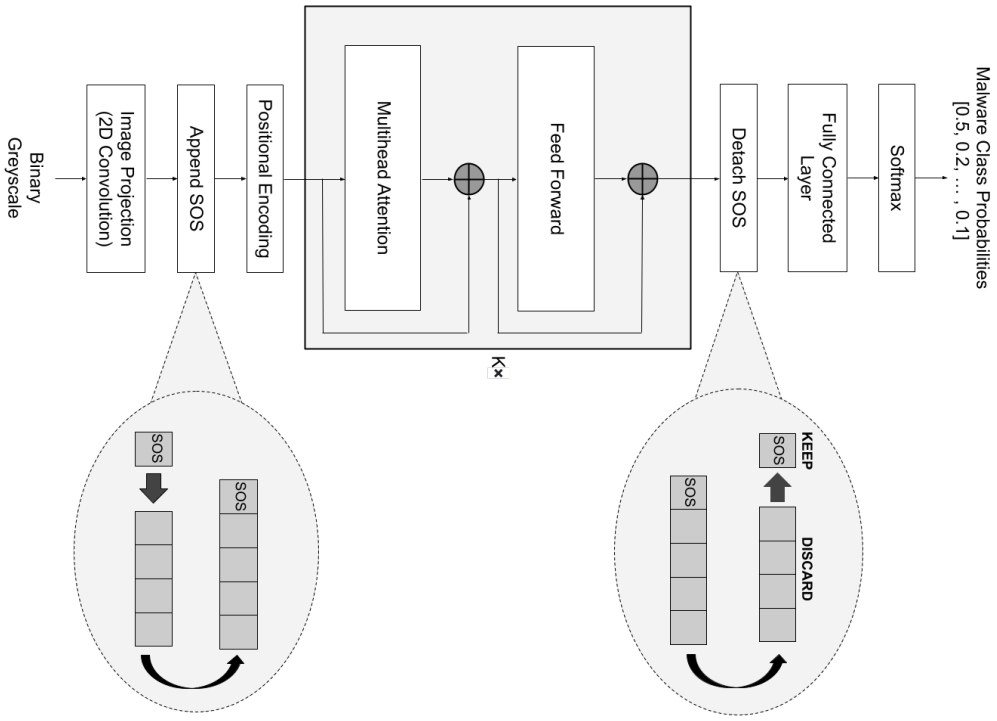

Figure 3.2: Note that the append and detach blocks have no learnable parameter, but used to indicate modifications to the sequence encoding.

of malware in its entirety. [17] recently devised the Vision Transformer, a transformer model specialized for image classification. Following the theme of leveraging self-attention, we further implemented ImgConvAttn, a Vision Transformer model specialized for image-based malware classification. In this section, we first describe the structure of ImgConvAttn. Afterwards, we describe the different binary-to-image conversion methods investigated.

3.3.1 Model Architecture

As illustrated in Figure 3.2, given a malware image, ImgConvAttn first process the image through a single 2D convolutional layer. The 2D convolution consists of d_m kernels with size $S \times S$ and stride of S in both horizontally and vertically. This step effectively partitions the input image into non-overlapping sub-image patches, and projects each sub-image into its corresponding encoding vector. Afterwards, the patch encodings are flattened into a sequence. A placeholder vector, referred to as *[SOS]* (start-of-sequence), is appended to the front of

the sequence. The purpose of this *[SOS]* token is to efficiently encapsulate information within of the entire sequence into a single vector of length d_m. Position encoding are then added to preserve positional context of the patches. According to [17], the existence of positional encoding is more important than its type. Their experiments did not show noticeable advantage of using one particular type of positional encoding over another, so long as some type of positional encoding is used. Thus, we followed the implementation of [46], where the positional encoding of ImgConvAttn is designed to be learnable parameters, such that it can adapt to the model during training.

The resultant sequence encoding then passes through a number of a transformer encoder blocks. We refrain from further exposition about transformers here, as further details are present in Section 3.1. Upon obtaining the transformer output sequence, only the *[SOS]* encoding is kept as the latent image representation, and the rest of the sequence discarded. The idea is that after multiple layers of self-attention, the finalized *[SOS]* encoding should retain sufficient context about the original image. To generate the final classification, the *[SOS]* encoding propagates through additional fully connected layers, and then undergoes softmax to generate classification probability.

As with SeqConvAttn, ImgConvAttn can be modified and finetuned with minimal effort to adapt to changes in the malware classes. The exact procedures are exactly the same as those introduced for SeqConvAttn in Section 3.2.

3.3.2 Image Generation

Based on our survey, there are predominantly two approaches to convert binaries to images. The traditional approach, proposed by [6], converts a binary into a greyscale image. Recent works such as [7], [8], and [13] explored generating Markov image. Influenced by [9], we further investigate generating malware image using the bigram occurrence count histogram, referred subsequently as bigram frequency image, as a representation. We elaborate on the specific image generation procedure here.

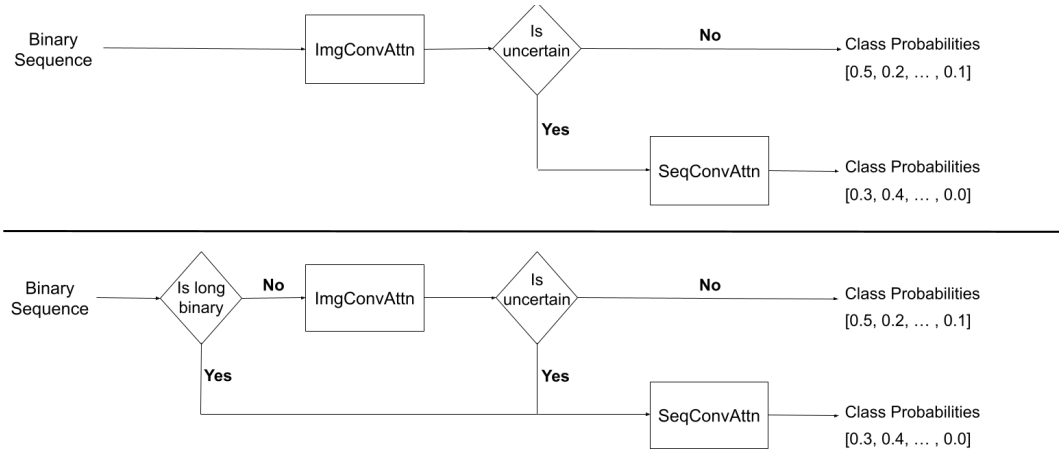

Figure 3.3: **Top:** The basic two-stage framework proposed. **Bottom:** A variant with an additional file-size-aware mechanism, which pre-emptively diverts large binary files directly to the second-stage.

- To generate the bigram frequency image, the occurrence count of each of the 65,536 distinct bigrams within a byte sequence are first tallied into a histogram. Afterwards, the histogram is re-arranged into a matrix of size 256×256, where an entry at row i and column j indicates the number of occurrence of bigram (i, j).

- Markov Image: The Markov image records the transition probability between distinct byte values within the byte sequence. Specifically, for an entry at row i and column j, its value indicate the probability of byte j following immediately after byte i in the byte sequence.

- Greyscale: To convert a byte sequence to a greyscale, we enforce a fixed width of 256 bytes, and line-break the byte sequence into consecutive lines. Here, each byte is considered as a brightness value. To ensure consistent image dimension with respect to the bigram frequency and markov image, the image is resized into a 256×256 matrix bilinear interpolation.

3.4 Two-Stage Framework

Theoretically, SeqConvAttn and ImgConvAttn are designed to be functionally complementary. Whereas SeqConvAttn should be more accurate, ImgConvAttn should be faster. We draw inspiration from [44] and [45], and devise a two-stage framework to leverage the advantage of both models, as shown by the design on the upper portion of Figure 3.3. The underlying intent is to avoid unnecessarily running the slower SeqConvAttn if ImgConvAttn can generate a confident prediction. We assign ImgConvAttn as the first-stage, with an expected per-file inference latency of t_1. SeqConvAttn is then assigned as the second-stage, with a latency of t_2. Given a binary file, ImgConvAttn would conduct the initial classification. The classification uncertainty is then checked against a threshold value υ. If the uncertainty is below the threshold, the binary file is assigned to the class predicted by ImgConvAttn, concluding the classification process. However, if the uncertainty exceeds the threshold, the binary is subjected to reclassification by SeqConvAttn. Assuming that the ImgConvAttn is sufficiently confident in its predictions most of the time, the majority of binary files should only incur a inference latency of t_1, while the minority would incur a latency of $t_1 + t_2$. In our design, classification uncertainty is defined as Equation 3.5.

$$U_{pred} = 1 - \mathbb{P}(C_{pred}) \tag{3.5}$$

Here, C_{pred} and U_{pred} refers to, respectively, the predicted class and the classification uncertainty. $\mathbb{P}$ refers to the probability "operator".

Proper setting of uncertainty threshold υ offers meaningful control in the tradeoff between latency and accuracy of the two-stage framework. Thus we introduce a simple approach to determining the υ for a specified latency requirement using a set-aside development (validation) set. Consider an arbitrary uncertainty threshold υ_p, such that $p\%$ of files are expected to undergo SeqConvAttn reclassification. For a given latency constraint t_{spec}, we could solve for the percentage of the files permitted for reclassification by Equation

3.6.

$$p\% = \frac{t_{spec} - t_1}{t_2} \tag{3.6}$$

Once p is solved, the corresponding υ_p could then be experimentally determined by assessing the classification uncertainties of ImgConvAttn on the development set. Specifically, we set υ_p to the $(100 - p)$th percentile, such that only the $p\%$ most uncertain cases proceed to SeqConvAttn. Assuming that the development set is sufficiently representative of the test environment, a framework with uncertainty threshold υ_p should approximately meet the latency constraint of t_{spec}.

We curtly note that the lower Figure 3.3 is a variant of the two-stage framework. This variant contains a supplementary conditional that pre-emptively redirects large binaries files that are likely to incur a latency $t_1 \geq t_2$ to directly undergo second-stage classification. Note that this variant is devised based on experimental observation. Thus, we defer further discussion of this variant to Section 6.1.

Chapter 4

Common Experiment Settings

4.1 Datasets

4.1.1 BIG 2015

The BIG 2015 [18] was originally provided for the Microsoft Malware Classification Challenge. While the original dataset consists of a "train" and "test" partition, only the "train" set is labelled. We partitioned the 10868 labelled malware binaries in "train" set into disjoint train, validation, and test set. We present the statistics of partitioned datasets on Table 4.1.

Table 4.1: Statistics of BIG 2015 Dataset

Malware	**Train**	**Valid**	**Test**
Ramnit	1216	143	182
Lollipop	1979	249	250
Kelihos_ver3	2410	273	259
Vundo	368	54	53
Simda	36	2	4
Tracur	616	63	72
Kelihos_ver1	327	39	32
Obfuscator.ACY	967	129	132
Gatak	814	103	96
All	8733	1055	1080

For each malware, the original dataset provided its hexadecimal representa-

tion. Thus, we converted the hexadecimals into bytes, generating the binary file. For some of these files, some hexadecimals are of the value "??". These instances were dealt with by interpreting all "??"s as "00"s during the conversion process. Note that the resultant binary files are sterilized, with the PE headers removed by the original vendor before distribution.

For subsequent experiments, we do not directly record the reported performance on Big 2015 from surveyed publication into the results table. As the test sets differs between different publications and our work, we consider direct comparisons infeasible. However, where necessary, we will reference results from surveyed publications for explanation purposes.

4.1.2 Sub-BODMAS

The original BODMAS dataset [19] contains of 57,293 malware binaries, belonging to one of the 581 malware families. However, the majority of malware classes possess insufficient number of malware samples for meaningful assessment. Consequently, only a subset of the malware files are selected for experimentation. Specifically, we only retrieved files whose malware class contains more than 1000 instances timestamped from and after January 1, 2020. The resultant dataset contains 23065 malware binaries in total, drawn from 11 classes. The subset is then partitioned into disjoint train, validation, and test sets. We refer to the resultant dataset as Sub-BODMAS. Statistics on Sub-BODMAS is presented on Table 4.2.

4.2 Test Metrics

We use three metrics to assess the quality of our model, accuracy, weighted-F1, and latency.

- Accuracy measures the proportion of test samples whose predicted class matches the ground class.
- Weighted-F1 is a weighted average of the one-vs-all F1-score of each class, with the weights the number of test instances belonging to each

Table 4.2: Statistics of Sub-BODMAS Dataset

Malware	**Train**	**Valid**	**Test**
sfone	3618	362	543
wacatac	1524	153	228
upatre	2013	202	302
wabot	2522	253	378
small	2000	201	300
ganelp	1632	164	244
dinwod	1106	111	166
mira	1042	105	156
berbew	1112	112	166
sillyp2p	1048	105	157
ceeinject	832	84	124
All	18449	1852	2764

class.

- Latency is the average duration between when the binary file content is loaded into RAM and when the predicted classification is obtained. For assessment of independent models, this accounts for the byte sequence preprocessing (such as truncation, image conversion) and model feedforward time. For the two-stage framework, this also includes time taken by the latency control measures. Note that to address portability issues, the latency assessment is done by running the classifiers on CPUs only.

4.3 Test Environment

As our experiment involves time assessment, we list the relevant specifications of our test environment on Table 4.3. Note again that while the GPU is listed, it is only used for model training, and not for testing.

Table 4.3: Test Environment Specification

Setting	**Specification**
Python	3.6.9
Pytorch	1.7.1+cu110
Cuda	11.2
OS	Ubuntu 18.04.5 LTS
CPU	AMD Ryzen 9 3900X 12-Core Processor
Memory	64 GB
GPU	GeForce RTX 2080 TI

Chapter 5

Independent Model Experiments

5.1 Sequence-based Classification Experiments

5.1.1 Experiment Design

For preprocessing, we standardize all byte sequence length to 400,000. If the original byte sequence is longer than this limit, it is truncated to preserve only the first 400,000 bytes. If the sequence has fewer than 400,000 bytes, padding "bytes" are appended to the malware until the specified length is reached.

For SeqConvAttn, the initial embedding dimension is 8. The 1D convolutional layer consists of 128 kernels, each with the width and stride of to 500. The transformer section consists of 3 blocks. Each transformer block has a model dimension of 128, matching the post-convolution encoding dimension. The multihead attention consists of 8 parallel scaled dot-product attention heads. The feedforward components of each transformer block has an expansion factor of 4. The final classifier consists of two additional linear layers, with the first layer having a width of 128, and the width of the second layer determined by the number of classes.

For model training, the training batch size is 25. During training, Adam optimization is used, with a learning rate of 1e-4, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The training process lasts for 25 epochs, with the checkpoint yielding the highest validation accuracy selected as the optimal version.

Our baseline for comparison against SeqConvAttn are listed below.

- Malconv [3]: Our Malconv implementation is taken from existing Github code [47]. Note that for analysis purposes, we adjusted the original convolutional kernel and stride size from 512 to 500.

- Malconv+GCG [5]: This model introduced Global Convolution Gating to better learn inter-dependencies between distant elements. The original code is taken from [47], but extensively modified to remove the convolution-over-time mechanism. Again, the convolutional kernel and stride size are set to 500.

- CNN+BiLSTM [Ours]: We drew inspiration from [4], and implemented our version of CNN+BiLSTM model. Compared to the proposed SeqConvAttn model, the only difference is the replacement of the transformer section with a single biLSTM layer.

- CNN+BiGRU [Ours]: This model is very similar with the CNN+BiLSTM [Ours] baseline, with the only difference being that the BiLSTM layer is exchanged for a BiGRU model.

- CNN+BiLSTM* [14]: Technically, this CNN-BiLSTM design is based on [4]. However, as the paper did not offer sufficient information to replicate the original model, we follow the replication given by [14], with one major difference. We replaced the max-pooling layer with ReLU activation, and adjusted the convolution layers accordingly to maintain the approximate intermediate feature map dimensions. Cursory experiments demonstrated this improves the model accuracy. Additionally, this design likely better adheres to the design of [4], which did not employ max-pooling. Note also that the input sequence of this model is obtained by re-sizing the full byte sequence (without truncation) to 10,000 elements as a 1D greyscale image.

- CNN+BiGRU* [14]: This is very similar to CNN+BiLSTM*, with the only difference being the replacement of the BiLSTM with BiGRU. Again, the max-pooling layers are replaced with ReLU activation.

5.1.2 Experiment Results

Table 5.1: Results of Sequence-based Classification on BIG 2015

Model	**Accuracy**	**Weighted-F1**	**CPU Latency [ms]**
Malconv	95.28	95.11	19.1
Malconv+GCG	95.37	95.34	35.7
CNN+BiLSTM [Ours]	97.04	97.01	60.0
CNN+BiGRU [Ours]	95.46	95.31	54.5
CNN+BiLSTM*	93.61	93.60	6.2
CNN+BiGRU*	93.61	93.60	6.0
SeqConvAttn	97.22	97.22	34.1

On Big 2015, Compared to the baseline models, SeqConvAttn attained superior accuracy and weighted-F1 score when compared to the baseline. The next best model, CNN+BiLSTM, achieved a comparable accuracy score of 97.04. However, its CPU latency is about 25 ms longer than SeqConvAttn. This is because the encoding process of LSTM is sequential, with the encoding element dependent on the previous state of the LSTM. SeqConvAttn lacks this flaw, as the self-attention concurrently computes the encoding of all elements in the sequence. Compared to Malconv, there is an apparent trade-off between accuracy and latency, with SeqConvAttn being 2% more accurate, while Malconv being 15 ms faster. Unexpectedly, Malconv+GCG also attained an inferior accuracy when compared to SeqConvAttn, while incurring a comparable latency. As [5] designed Malconv+GCG to improve the modelling of inter-depednecies between distant elements in a sequence, it is surprising that SeqConvAttn achieved superior accuracy and weighted-F1 score. We surmise that this may be caused by the byte sequence truncation done during preprocessing. Potentially, using longer byte sequences as input could offset the difference in accuracy between SeqConvAttn and Malconv+GCG. However, that is beyond the scope of this investigation.

We note that SeqConvAttn also significantly surpassed CNN+BiLSTM*

in accuacy and weighted-F1. [4] reported an accuracy of 98.20% using the CNN+BiLSTM model. However, we were not able to replicate such performance using our implementation of CNN+BiLSTM*. Potentially, this is caused by difference in sequence resizing algorithm or model designs, as information from the original publication is insufficient for model replication. However, unless further information becomes available, we consider SeqConvAttn as superior for classification accuracy.

Table 5.2: Results of Sequence-based Classification on Sub-BODMAS

Model	**Accuracy**	**Weighted-F1**	**CPU Latency [ms]**
Malconv	96.71	96.74	18.4
Malconv+GCG	96.92	96.94	36.4
CNN+BiLSTM [Ours]	96.89	96.91	55.4
CNN+BiGRU [Ours]	96.56	96.60	55.5
CNN+BiLSTM*	89.91	89.23	8.9
CNN+BiGRU*	89.91	89.58	8.0
SeqConvAttn	96.92	96.92	33.7

On Sub-BODMAS, the advantage of SeqConvAttn appears greatly diminished, with difference in accuracy and weighted-F1 score between different models under 0.5%. For example, when compared to Malconv+GCG, the accuracy and weighted-F1 of SeqConvAttn are comparable. Compared to CNN+BiLSTM, and CNN+BiGRU, the most significant advantage of SeqConvAttn is its latency, with a difference of 25 ms. When comparing SeqConvAttn with Malconv, the tradeoff between accuracy and latency appears to favour towards Malconv, as the slight improvement of 0.21% accuracy may not justify a latency increase of 15 ms in many situation. Nevertheless, by accuracy alone, SeqConvAttn still attained one of the highest scores. Furthermore, compared to models of comparable accuracy, Malconv+GCG and CNN+BiLSTM, the inference latency is still, respectively, comparable and shorter.

5.1.3 Visualization of SeqConvAttn Attention

We further investigate the features learned by the SeqConvAttn model. Recall that the attentional weights correspond to the $softmax(\frac{XW_Q(XW_K)^T}{\sqrt{d_{kqv}}})$ matrix in Equation 3.1. Additionally, note that each element in the post-convolution sequence corresponds to a segment of 500 bytes. A high attention value at row i and column j of the matrix thus suggests a salient dependency of byte segment i on byte segment j. To visualize such inter-dependencies between segments, attention maps of all attention heads in the final transformer block are elementwise averaged into a single attention map. The resultant maps for two malware samples are presented at the top of Figure 5.1. Note that the softmax of Equation 3.1 is applied horizontally across the attention map. Furthermore, natural log was applied to all values in the matrix before image display for better visualization effect.

From the SeqConvAttn attention maps, several vertical highlights, or bright green streaks, can be identified. Additionally, several faint horizontal lines are also noted. That the column index of the vertical highlights and the row index of the horizontal lines are the same is no coincidence, as they both address the same sequence byte segments. Based on Equation 3.1, the vertical highlights indicate that most byte segments in the post-convolution sequence have strong dependency to the highlighted byte segment. On the other hand, the faint horizontal highlights indicate that for these byte segments, there are no strong dependencies with respect to other byte segments in the sequence, as the attentional weight is approximately equal across the sequence. Summarily, the highlighted byte segments likely contain salient information critical to malware identification. To some extent, the predominance of these highlights for different byte sequences is somewhat unexpected, as it may imply that few inter-dependencies exists between distant byte segments. However such implication is against intuition. Consider that in assembly code, conditional and jump branches allow programs to execute instructions that are nonconsecutive on the binary level, which can be interpreted as a form of dependency. The more likely reason for the absence of inter-dependency is that by compressing

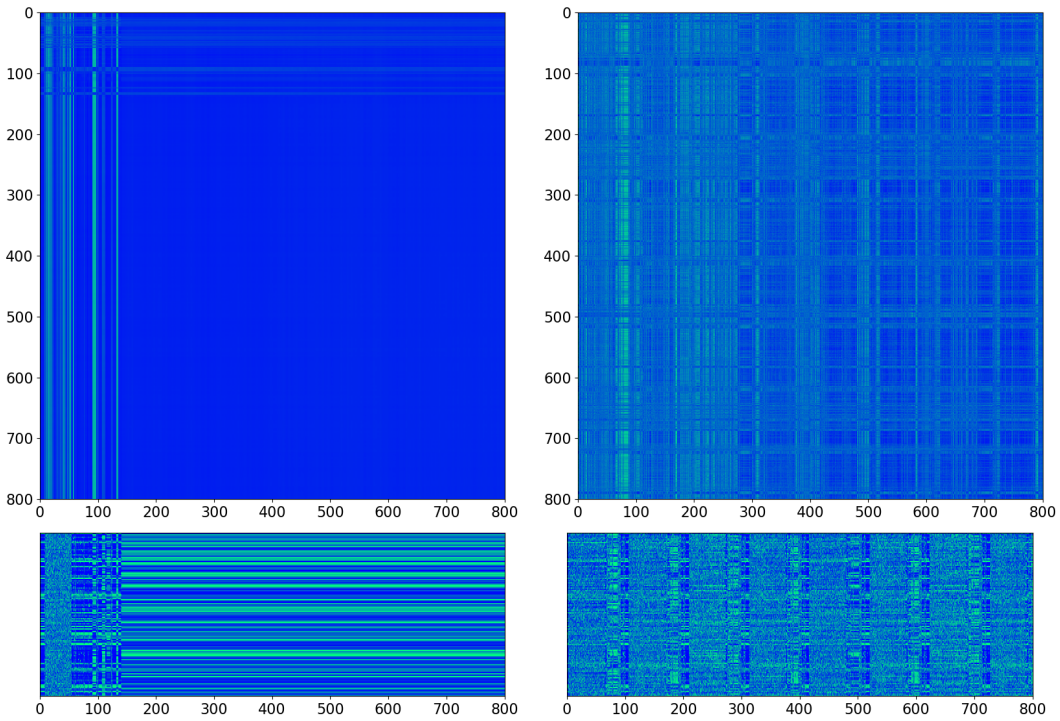

Figure 5.1: **Left:** The SeqConvAttn Attention Map (Top) and Malconv Gating Map (Top) of a SillyP2P file. **Left:** The SeqConvAttn Attention Map (Top) and Malconv Gating Map (Top) of a Berbew file.

segments of 500 bytes into single elements through 1D convolution, the resultant element encodings likely lost inter-dependencies information contained in the byte segments.

We further compared the attention map of SeqConvAttn to the gating map in Malconv. Akin to the attention weights in SeqConvAttn, Malconv employs gated convolution [20] to filter byte segment information for classification. The bottom of Figure 5.1 displays the gating maps computed for the same malware samples as that of the attention maps. Note that it is difficult to divulge from the gating map the emphasis or suppression of information from a particular byte segment. Unlike attention maps, whose value suppresses or emphasize the entire element encoding, values of gating map can independently suppress specific sections (along the dimension) of the element encoding. This hinders the interpretability of Malconv, as salient byte segments cannot easily identified from the gating map. The best can be discerned from the gating maps are the presence of different byte sections, indicated by the different texture

patterns, Potentially, this is indicative of different types of information being presented by different byte sections. However, unlike SeqConvAttn attention maps, it is difficult to identify individual byte segments that are salient for classification. From this, we note another advantage of SeqConvAttn, that its attention map is more readily interpretable for human analysis.

Comparing attention maps against gating maps, it is noted that dense sections of high attention byte segments in the former often corresponds to particular binary region of the latter. We suspect then, that the underlyng features learned by ConvSeqAttn and Malconv are likely similar for many cases. This potentially explains the close accuracy score attained on the Sub-BODMAS dataset between the two models, as shown in Table 5.2. The fact that both SeqConvAttn and Malconv appear to retrieve similar information, despite the feature extraction mechanism being relatively different, suggests the limitation of using convolution-based dimension reduction approach. Essentially, the ability of transformers to model inter-dependencies between any pair of elements is not applicable in most instances due to information lost during 1D convolution. Improvements in feature engineering, such that dependencies between features are better enhanced, should be subjected to future investigation.

5.2 Image-based Classification Experiments

5.2.1 Experiment Design

The ImgConvAttn is taken from an existing implementation of the Vision Transformer [17] by [46]. The initial convolutional layer consists of 64 kernels of dimensions 16×16 and a stride of 16. The subsequent transformer consists of 4 blocks, each having a model dimension of 64. Each multihead attention contains 4 parallel attention heads. The feedfoward component has an expansion factor of 4. The final classifier consists of a single linear layer, whose width is determined by the number of class.

We assign a training batch size of 50, During training, the same Adam optimization setting for the SeqConvAttn experiment is used here. Each model

is trained for 100 epochs, with the checkpoint yielding the highest validation accuracy selected as the optimal version.

The baseline for comparison against ImgConvAttn is the 3C2D model [9]. The original design, proposed by [10], is a shallow CNN consisting of 3 convolution-and-max-pooling layers and 2 fully connected layers with training dropout. [9] added dropouts to the fully connected layer to improve model robustness. This baseline is implemented based on description provided by [9]. Note that we did not implement additional baselines with deep models to minimize the inference latency. In addition, for each model, we experiment with three different types of image: bigram frequency, Markov, and greyscale.

5.2.2 Experiment Results

Table 5.3: Results of Imaged-based Classification on BIG 2015

Model	**Accuracy**	**Weighted-F1**	**CPU Latency [ms]**
3C2D-Greyscale	91.30	91.38	2.8
ImgConvAttn-Greyscale	93.06	93.09	5.1
3C2D-Markov	96.67	96.63	6.0
ImgConvAttn-Markov	97.04	96.97	8.7
3C2D-Frequency	95.83	95.80	5.9
ImgConvAttn-Frequency	98.61	98.60	8.2

Two observations can be taken from experimental results of the BIG 2015. First, the ImgConvAttn model is superior to the 3C2D model in classification accuracy for all image types. While ImgConvAttn is generally 2-3 ms slower than 3C2D, the average inference latency is nevertheless quite fast. Second, on ImgConvAttn, between the different malware image types, bigram frequency generates the best results, significantly outperforming greyscale and surpassing Markov images in accuracy and weighted-F1. The synergy of these two observations is demonstrated by the ImgConvAttn-Frequency model, which impressively achieved a higher accuracy than SeqConvAttn, while only incurring a quarter of the latency of the latter.

Table 5.4: Results of Imaged-based Classification on Sub-BODMAS

Model	**Accuracy**	**Weighted-F1**	**CPU Latency [ms]**
3C2D-Greyscale	89.22	88.27	8.4
ImgConvAttn-Greyscale	91.03	90.74	10.2
3C2D-Markov	95.12	95.07	20.8
ImgConvAttn-Markov	96.02	95.94	23.1
3C2D-Frequency	95.04	94.97	20.3
ImgConvAttn-Frequency	95.98	95.98	22.6

The same experiment is repeated on the Sub-BODMAS dataset, with experimental results again showing ImgConvAttn as the more accurate model. Additionally, as all models are slower on Sub-BODMAS, the slight latency disparity between 3C2D and ImgConvAttn becomes further insignificant. Note that the cause of average latency increase on Sub-BODMAS is because Sub-BODMAS binary files are generally larger than binary files in BIG 2015, which slows down the image conversion step. Comparing the results for the bigram frequency and Markov images, the resultant accuracy appears to be comparable between the two image types on both ImgConvAttn and 3C2D. In any case, when compared to greyscale images, both bigram frequency and Markov images yielded significantly better accuracy and weighted-F1 score, but incurred a significantly longer latency. The latency disparity between greyscale and other image types is caused by different image generation procedures, as both bigram frequency and Markov image both requires the tallying of bigram occurrence, whereas greyscale does not. Figure 5.2 plots the relationship between binary file size (byte sequence length) and inference latency for ImgConvAttn-Greyscale and ImgConvAttn-Frequency on the Sub-BODMAS validation set. From the plots, strong linear relationships could be observed for both image generation methods. Note that the presence of two lines for the ImgConvAttn-Greyscale plot is caused by the conditional application of byte sequence padding during the greyscale generation process.

Finally, we note that comparing the ImgConvattn with SeqConvAttn on

Sub-BODMAS, the selection of one architecture over another reflects a trade-off between accuracy and speed. This is the expected scenario based on the original design philosophy. Thus, we then experimented leveraging the advantage of both using the two-stage framework.

5.2.3 Comparing Gresycale and Bigram Frequency Images

Noting that the the classification accuracy from bigram frequency images outperforms that of greyscale images significantly, we further investigated the reasons behind these results. On Figure 5.3 and 5.4, the greyscale and bigram frequency image of three samples of Upatre malware from Sub-BODMAS test set are presented. Note that for bigram frequency, we project the image as a heatmap over the grey spectrum. Furthermore, certain pixel value in the image are capped for viewing purposes. Additionally, we also present the corresponding ImgConvAttn attention maps for analysis purposes. To visualize each attention maps, attention maps from the attention heads of the last transformer block are first averaged elementwise. The resultant map is then further averaged across the sequence, perpendicular to the dimension of softmax. This yields a vector, with each value the average attention over the entire sequence. The *[SOS]*-associated attentional value is then removed, and the resultant 256-length vector reshaped to a 16×16 image. Thus, each value in the attention map can be interpreted as the saliency of its corresponding sub-image patch in the image. Note that grid lines are added to the greyscales and bigram frequency images for visualization purposes.

The greyscale images, when compared with each other, are significantly varied despite belonging to the same malware class. This is because the underlying byte sequences from different file instances can differ greatly. Such variation between the images likely make acquiring the salient features for classification difficult. As observed from the corresponding attention maps, the salient patches identified from the three gresycale images have little in common. However, the more fundamental cause for the poor accuracy from greyscale image is two fold. First, converting byte sequence to 2D image in-

troduces relationship between bytes that are not existent in the original 1D format. Such relationship could be easily altered by shifting the position of certain byte sections or interjecting bytes at some locations, resulting in significant variation in the resultant greyscale. Second, the byte value cannot be interpreted as a greyscale value. This is because distinct byte values, such as 12 (0x0C) and 255 (0xFF) do note have a comparative relationship that defines one value as greater than the other. This issue becomes relevant when resizing the greyscale, as the interpolation from image pixels has no meaningful explanation. Note that from our survey, some publications did achieve very high accuracy using greyscale only. For example, the transfer-learned MCFT-CNN [33], based on Resnet50 [27], reported an accuracy of 98.63%. Thus, the greyscale-based classification is not necessarily discredited as an effective type of malware image. However, its lack of an intuitive justification must be acknowledged.

Compared to greyscale images, the bigram frequency images from different Upatre samples appear, at least superficially, to be relatively consistent. Overlaying the malware ImgConvAttn attention maps onto their respective frequency images, a number of salient patches are appear common between the instances. For example, using notation [row, column], patches [0, 3], [0, 7], [0, 10], [0, 12], are highlighted to some extent in all three samples. As each patch encapsulate the frequency of 256 distinct bigrams, this suggests that certain bigrams within the highlighted sub-images are salient for malware identification, with the more frequent bigrams being the priority candidates. The potential correlation between bigrams and malware class can also be intuitively justified. The execution of malicious activity may rely on particular instruction sets, which remains consistent between different samples. The presence of these instruction sets may be reflected by particular occurrence frequency of specific bigrams. As the bigram frequency is a matrix, the entry corresponding to a particular bigram occupies the same pixel for all images. Thus, salient information can be captured for more easily by ImgConvAttn, leading to higher classification accuracy.

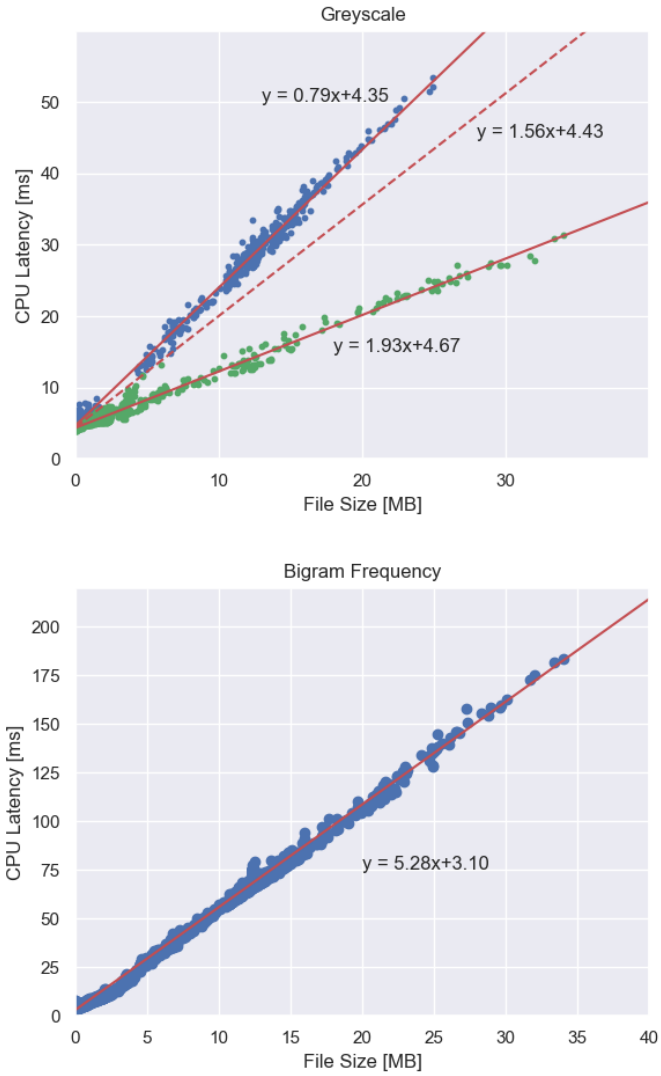

Figure 5.2: **Top:** The relation between file size and inference latency for ImgConvAttn-Greyscale. Note that the dashed red line separates the data-points into two disjoint subsets. A linear equation is then derived for each subset. **Bottom:** The relation between file size and inference latency for ImgConvAttn-Frequency.

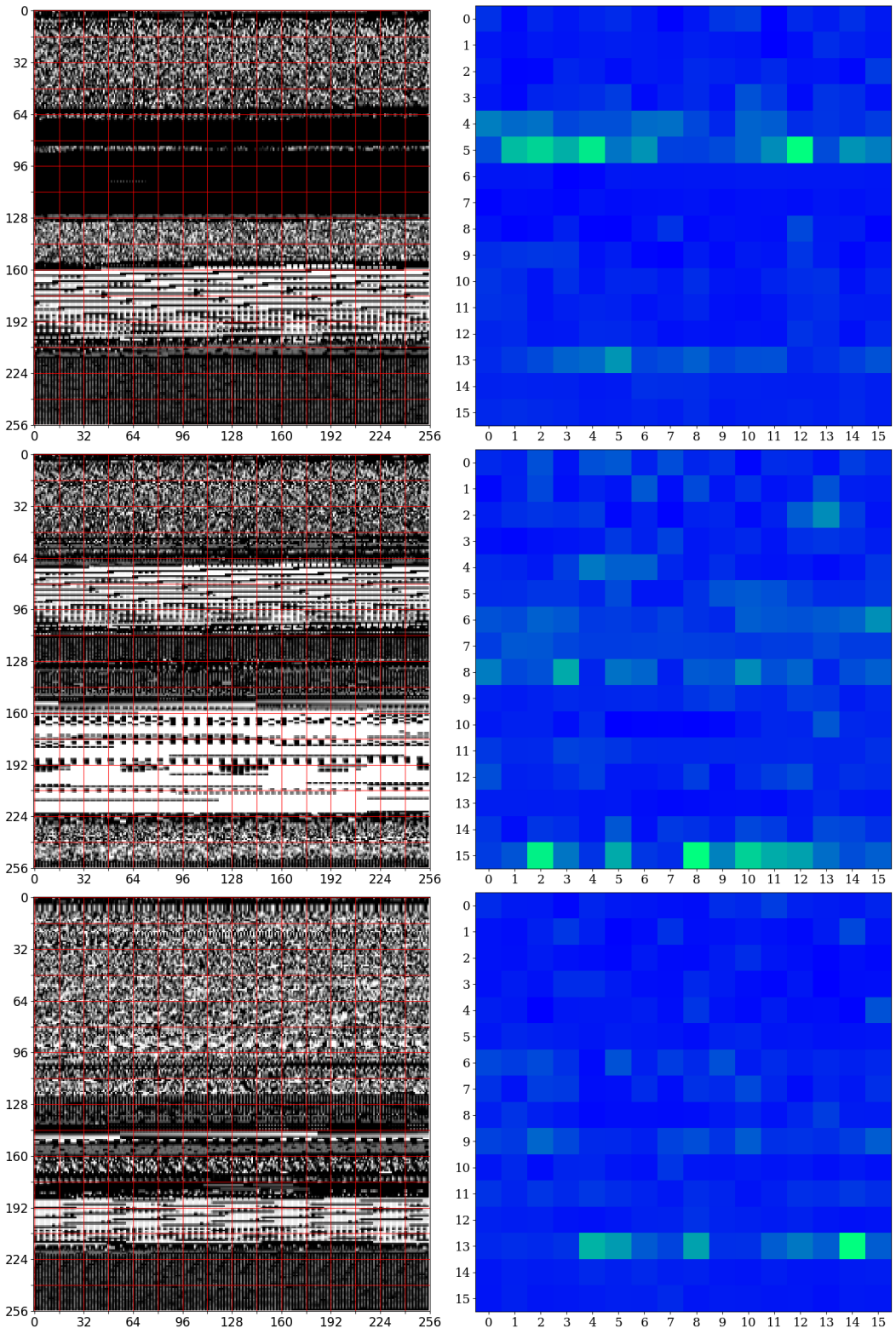

Figure 5.3: **Left:** Generated Greyscale Image. **Right:** Visualization of Img-ConvAttn Attention Map. Note that reach row corresponds to a different Upatre sample.

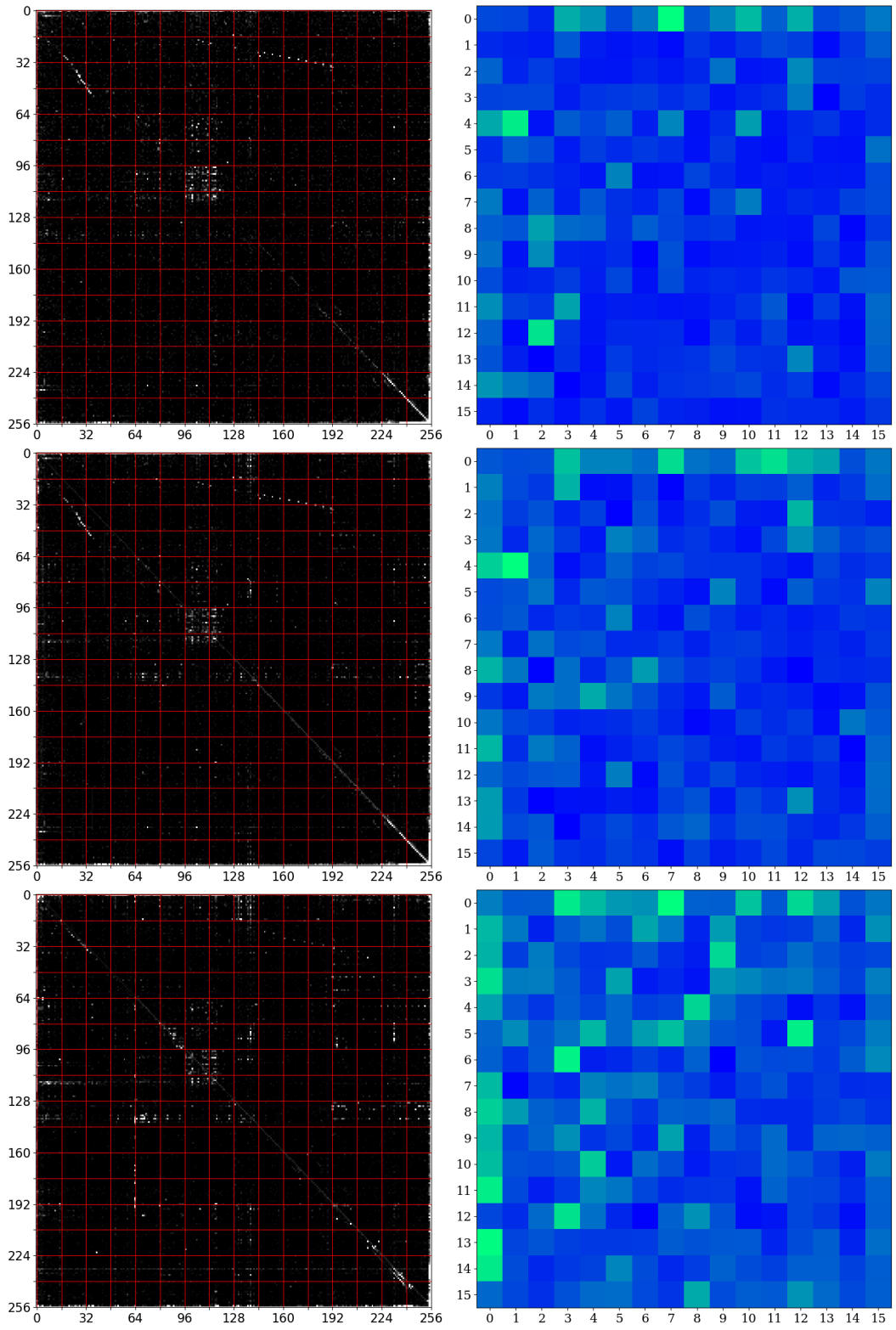

Figure 5.4: **Left:** Generated Bigram Frequency Image. **Right:** Visualization of ImgConvAttn Attention Map. Note that reach row corresponds to a different Upatre sample.

Chapter 6

Two-Stage Experiments

6.1 Experiment Design

We experimented with two two-stage framework designs, as shown by Diagram 3.3. The first design is a basic design of the two-stage framework. The second contains an additional file-size-aware mechanism, such that binary files of sufficiently large size are sent directly to the second-stage, bypassing first-stage classification altogether. The intent of this mechanism is to reduce the average inference latency of the framework. Results from image-based classification, as shown by Figure 5.2, demonstrate that larger files (longer byte sequences) incur larger latency due to the image generation process. For several binary files, the incurred latency of ImgConvAttn exceeded the average latency of SeqConvAttn, resulting in $t_1 \geq t_2$. Diverting these large binary files directly to SeqConvAttn would reduce the latency and potentially improve prediction accuracy. Given the linear equation modelling the relationship between file size and latency, we set the expected latency to 30 ms (approximately the latency of SeqConvAttn), and determine the approximate file size threshold. Note that for ImgConvAttn-Greyscale, we employed the linear equation with the shallower slope.

We refer to the two-stage frameworks as ⟨Stage1⟩-⟨Stage2⟩-[fsa]-⟨p⟩%. ⟨Stage1⟩ and ⟨Stage2⟩ are the incorporated models. The optional term [fsa] is abbreviation of "file-size-aware", indicating that the two-stage framework design contains such mechanism to divert large binaries directly to the second-stage. Value p is the expected percentage of files directed to the second-stage based

on uncertainty threshold setting, which maps to specific uncertainty thresholds determined from the validation set. Note that while in real-world application, we expect that the uncertainty threshold v is selected based on a specified latency constraint t_{spec} using Equation 3.6. However, for this experiment, uncertainty threshold v is selected based on $p\%$, based on development (validation) set performance, for ease of comparison. We list these uncertainty thresholds on Table 6.1.

The model pairs integrated in the two-stage framework are the following:

- ICAGrey+SCA: ImgConvAttn-Greyscale as first-stage, SeqConvAttn as second-stage.
- ICAFreq+SCA: ImgConvAttn-Frequency as first-stage, SeqConvAttn as second-stage.

Note that a two-stage framework that incorporates ImgConvAttn-Markov was not implemented, as it is demonstrated comparable to ImgConvAttn-Frequency in both accuracy and latency. For the file-size-aware two-stage variants, the computed file size threshold for ICAGrey+SCA-fsa is 13 MB. For ICAFreq+SCA-fsa, the threshold is 5 MB.

6.2 Results on Two-Stage Framework

Table 6.1: Uncertainty Threshold v for Two-Stage Framework on Sub-BODMAS

Percentage to Second Stage	**ICAGrey-SCA**	**ICAFreq-SCA**
p = 5	0.35199	0.21816
p = 10	0.26004	0.04455
p = 15	0.17975	0.01322
p = 20	0.05142	0.00808
p = 25	0.01374	0.00587
p = 30	0.00541	0.00455

Table 6.2: Results of Two-Stage Framework Classification on Sub-BODMAS

Model	**Accuracy**	**Weighted-F1**	**CPU Latency [ms]**
ICAGrey-SCA-5%	92.76	92.59	12.1
ICAGrey-SCA-10%	93.60	93.48	13.9
ICAGrey-SCA-15%	94.54	94.43	15.8
ICAGrey-SCA-20%	95.51	95.41	17.8
ICAGrey-SCA-25%	95.84	95.75	19.4
ICAGrey-SCA-30%	96.06	95.98	20.8
ICAFreq-SCA-5%	96.67	96.66	24.0
ICAFreq-SCA-10%	96.81	96.80	25.4
ICAFreq-SCA-15%	96.92	96.91	27.2
ICAFreq-SCA-20%	96.96	96.95	29.8
ICAFreq-SCA-25%	96.96	96.95	30.97
ICAFreq-SCA-25%	96.96	96.95	32.52
ICAGrey-SCA-fsa-5%	93.45	93.32	11.1
ICAGrey-SCA-fsa-10%	93.92	93.80	12.0
ICAGrey-SCA-fsa-15%	94.72	94.61	13.5
ICAGrey-SCA-fsa-20%	95.62	95.52	15.2
ICAGrey-SCA-fsa-25%	95.84	95.75	16.5
ICAGrey-SCA-fsa-25%	96.06	95.98	19.0
ICAFreq-SCA-fsa-5%	96.78	96.77	14.4
ICAFreq-SCA-fsa-10%	96.85	96.84	14.8
ICAFreq-SCA-fsa-15%	96.96	96.95	17.0
ICAfreq-SCA-fsa-20%	97.00	96.99	17.4
ICAFreq-SCA-fsa-25%	97.00	96.99	21.2
ICAFreq-SCA-fsa-25%	97.00	96.99	22.1

For all designs, it is observed that the adjustment of the uncertainty threshold υ does offer control between the accuracy and latency for the two-stage frameworks, with both metrics monotonically increasing with smaller uncer-

tainty threshold (larger $p\%$). However, at some point, there is always a clear effect of diminishing return observed. This effect appears when $p \in \{15, 20, 25, 30\}$. At this range, further increase in latency corresponds to successively smaller gain in accuracy. The reason for this phenomenon is that with smaller uncertainty thresholds, binary files with more certain classification are also subjected to second-stage reclassification. As the first-stage classification with higher certainty are more likely to be correct, reclassification by the second-stage would likely yield the same prediction. This redundancy thus results in the observed diminishing returns.

Examining between ICAGrey-SCA and ICAFreq-SCA frameworks, former is significantly faster. However, the accuracy of ICAGrey-SCA is left wanting, with ICAGrey-SCA-25% not even matching the accuracy of ICAFreq-SCA-5%, or that of most of the sequence-based classifiers. This observation is also noted with the file-size-aware two-stage frameworks, where ICAGrey-SCA-fsa is still inferior to ICAFreq-SCA-fsa on accuracy. Examining the latency difference between ICAGrey-SCA and ICAGrey-SCA-fsa, a moderate latency decrease of 1-3 ms is achieved by the latter design. However, a significant latency reduction of about 10 ms is noted between the ICAFreq-SCA and ICAFreq-SCA-fsa. This makes sense, as the latency of bigram frequency image generation scales worse against file size when compared to greyscale, this results more files diverted directly to the second-stage. Comparing the latency ICAGrey-SCA-fsa with ICAFreq-SCA-fsa, the speed advantage of the former is effectively nullified. We conclude that despite the two-stage model being capable of offsetting the inferior accuracy of the first-stage model through occasional reclassification of uncertain cases by the second-stage model, the inherent accuracy of the first-stage model nevertheless greatly affects the overall performance of the two-stage framework.

A peculiarity noted from the ICAFreq-SCA and its file-size-aware variant is that when uncertainty threshold υ is sufficiently low, the accuracy attained by the two-stage framework exceeds the accuracy of either of its component model. Recall that the accuracy attained by ImgConvAttn and SeqConvAttn on BODMAS are, respectively, 95.98 and 96.92. ICAFreq-SCA-20% slightly

surpassed those models with an accuracy of 96.96, and ICAFreq-SCA-fsa-20% surpassed that still with 97.00. We suspect that this improvement reflects that the two-stage model, in some instances, operates as an ensemble model. Specifically, for binary files with uncertain initial classification, both models become involved in generating the final prediction.

Chapter 7

Conclusions and Future Work

In this investigation, we introduced the application of self-attentive models to conduct low latency classification of malware through raw binaries. To achieve this, we implemented two models to interpret the binaries in two different ways. First, we proposed SeqConvAttn to handle the raw binary as a byte sequence. Intending to reduce the inference latency, we then investigated representing binary files as images, and applied ImgConvAttn, a self-attentive model based on the Vision Transformer [17], to classify malware images. We further proposed the integration of the two models into a two-stage framework, such that accuracy and latency advantage of the respective models could be leveraged in a controllable manner, thereby achieving superior classification while reducing per-file inference latency. From experimental results on the BIG 2015 dataset [18], and Sub-BODMAS, a select subset of the BODMAS malware dataset [19], we demonstrated that both the following:

- SeqConvAttn is superior to most of the baseline models tested in classification accuracy, with its overall performance comparable to Malconv+GCG [3].
- ImgConvAttn consistently attains superior accuracy over the CNN baseline. For the different images, bigram frequency image is able to achieve superior accuracy over greyscale image, and is generally comparable to Markov image.
- The two-stage framework can effectively control the average per-file in-

ference latency. Additionally, by minimizing occasional slowdowns of the first-stage model through the file-size-aware mechanism, superior accuracy could be maintained while lowering the inference latency.

We also recognize that our investigation have a number of limitations. Thus, we propose the followup research for future works:

- Self-attention computation in transformer scales quadratically to the length of the input sequence. This entails that long byte sequences need to be truncated before classification, which renders the model vulnerable to malware files whose malicious payload are located after the truncation limit. Investigations in designing a self-attentive model capable of reading entire byte sequences of arbitrary lengths is thus necessary.

- We suspect that sequence length reduction of by convolution with large kernels causes significant information loss. Investigation on acquiring features on a finer granularity in the byte sequence may further improve classification accuracy.

- We have conducted analysis of SeqConvAttn and ImgConvAttn attention maps to identify salient features for malware classification. Further investigations could be conducted by matching such features to instruction sets in the assembly file, to offer further explainability for model prediction.

- For our two-stage framework, the prescribed uncertainty threshold selection is only feasible in a static test environment, where the underlying distribution of malware encountered remains the same. A process to adjust the uncertainty threshold in a dynamic environment, where such distribution drifts over time, would be useful for real-world application.

Bibliography

[1] V. Chebyshev, F. Sinitsyn, D. Parinov, O. Kupress, E. Lopatin, and A. Kulaev, *It threat evolution q1 2020. statistics*, Accessed: 2021-08-14. [Online]. Available: https://securelist.com/it-threat-evolution-q1-2020-statistics/96959/.

[2] V. Verma, S. K. Muttoo, and V. Singh, "Multiclass malware classification via first-and second-order texture statistics," *Computers & Security*, vol. 97, p. 101 895, 2020.

[3] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[4] Q. Le, O. Boydell, B. Mac Namee, and M. Scanlon, "Deep learning at the shallow end: Malware classification for non-domain experts," *Digital Investigation*, vol. 26, S118–S126, 2018.

[5] E. Raff, W. Fleshman, R. Zak, H. S. Anderson, B. Filar, and M. McLean, "Classifying sequences of extreme length with constant memory applied to malware detection," *arXiv preprint arXiv:2012.09390*, 2020.

[6] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

[7] S. Xia, Z. Pan, Z. Chen, W. Bai, and H. Yang, "Malware classification with markov transition field encoded images," in *2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, IEEE, 2018, pp. 1–5.

[8] B. Yuan, J. Wang, D. Liu, W. Guo, P. Wu, and X. Bao, "Byte-level malware classification based on markov images and deep learning," *Computers & Security*, vol. 92, p. 101 740, 2020.

[9] T. M. Mohammed, L. Nataraj, S. Chikkagoudar, S. Chandrasekaran, and B. Manjunath, "Malware detection using frequency domain-based image visualization and deep learning," *arXiv preprint arXiv:2101.10578*, 2021.

[10] D. Gibert, "Convolutional neural networks for malware classification," *University Rovira i Virgili, Tarragona, Spain*, 2016.

[11] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. De Geus, "Malicious software classification using transfer learning of resnet-50 deep neural network," in *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2017, pp. 1011–1014.

[12] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. de Geus, "Malicious software classification using vgg16 deep neural network's bottleneck features," in *Information Technology-New Generations*, Springer, 2018, pp. 51–59.

[13] A. Pinhero, M. Anupama, P. Vinod, C. A. Visaggio, N. Aneesh, S. Abhijith, and S. AnanthaKrishnan, "Malware detection employed by visualization and deep neural network," *Computers & Security*, p. 102 247, 2021.

[14] S. Venkatraman, M. Alazab, and R. Vinayakumar, "A hybrid deep learning image-based analysis for effective malware detection," *Journal of Information Security and Applications*, vol. 47, pp. 377–389, 2019.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[17] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[18] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, "Microsoft malware classification challenge," *arXiv preprint arXiv:1802.10135*, 2018.

[19] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "Bodmas: An open dataset for learning based temporal analysis of pe malware," in *Proceedings of Deep Learning and Security Workshop (DLS), in conjunction with IEEE Symposium on Security and Privacy (IEEE SP)*, 2021.

[20] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International conference on machine learning*, PMLR, 2017, pp. 933–941.

[21] M. Krčál, O. Švec, M. Bálek, and O. Jašek, "Deep convolutional malware classifiers can learn from raw executables and labels only," 2018.

[22] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2019, pp. 8–14.

[23] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European signal processing conference (EUSIPCO)*, IEEE, 2018, pp. 533–537.

[24] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Explaining vulnerabilities of deep learning to adversarial malware binaries," *arXiv preprint arXiv:1901.03583*, 2019.

[25] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," in *2019 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2019, pp. 21–27.

[26] M. A. Kadri, M. Nassar, and H. Safa, "Transfer learning for malware multi-classification," in *Proceedings of the 23rd International Database Applications & Engineering Symposium*, 2019, pp. 1–7.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, "Malware analysis of imaged binary samples by convolutional neural network with attention mechanism," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 127–134.

[30] J. Jung, J. Choi, S.-j. Cho, S. Han, M. Park, and Y. Hwang, "Android malware detection using convolutional neural networks and data section images," in *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, 2018, pp. 149–153.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[32] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[33] S. Kumar *et al.*, "Mcft-cnn: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in internet of things," *Future Generation Computer Systems*, 2021.

[34] N. Kumar and T. Meenpal, "Texture-based malware family classification," in *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, IEEE, 2019, pp. 1–6.

[35] Y. Zhao, C. Xu, B. Bo, and Y. Feng, "Maldeep: A deep learning classification framework against malware variants based on texture visualization," *Security and Communication Networks*, vol. 2019, 2019.

[36] M. Q. Li, B. Fung, P. Charland, and S. H. Ding, "I-mad: A novel interpretable malware detector using hierarchical transformer," *arXiv preprint arXiv:1909.06865*, 2019.

[37] X. Hu, R. Sun, K. Xu, Y. Zhang, and P. Chang, "Exploit internal structural information for iot malware detection based on hierarchical transformer model," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2020, pp. 927–934.

[38] Y. Ding, S. Wang, J. Xing, X. Zhang, Z. Oi, G. Fu, Q. Qiang, H. Sun, and J. Zhang, "Malware classification on imbalanced data through self-attention," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2020, pp. 154–161.

[39] A. Rahali and M. A. Akhloufi, "Malbert: Using transformers for cybersecurity and malicious software detection," *arXiv preprint arXiv:2103.03806*, 2021.

[40] F. Yang, Y. Zhuang, and J. Wang, "Android malware detection using hybrid analysis and machine learning technique," in *International Conference on Cloud Computing and Security*, Springer, 2017, pp. 565–575.

[41] M. A. Salah, M. F. Marhusin, and R. Sulaiman, "A two-stage malware detection architecture inspired by human immune system," in *2018 Cyber Resilience Conference (CRC)*, IEEE, 2018, pp. 1–4.

[42] J. Hwang, J. Kim, S. Lee, and K. Kim, "Two-stage ransomware detection using dynamic analysis and machine learning techniques," *Wireless Personal Communications*, vol. 112, no. 4, pp. 2597–2609, 2020.

[43] A. D. Raju and K. Wang, "Echelon: Two-tier malware detection for raw executables to reduce false alarms," *arXiv preprint arXiv:2101.01015*, 2021.

[44] L. Yang and J. Liu, "Tuningmalconv: Malware detection with not just raw bytes," *IEEE Access*, vol. 8, pp. 140 915–140 922, 2020.

[45] A. Yan, Z. Chen, R. Spolaor, S. Tan, C. Zhao, L. Peng, and B. Yang, "Network-based malware detection with a two-tier architecture for online incremental update," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, IEEE, 2020, pp. 1–10.

[46] R. Wightman, *Pytorch image models*, https://github.com/rwightman/pytorch-image-models, 2019. DOI: 10.5281/zenodo.4414861.

[47] *Malconv2*, https://github.com/NeuromorphicComputationResearchProgram/MalConv2, 2020.