Self-Attentive Models for Real-Time Malware Classification

QIKAI LU1, HONGWEN ZHANG2, HUSAM KINAWI2, AND DI NIU1, (Member, IEEE)

1Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2R3, Canada
2Wedge Networks, Calgary, AB T2R 0G7, Canada

Corresponding author: Di Niu (dniu@ualberta.ca)

This work was supported in part by Wedge Networks Inc., and in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant CRDPJ 531722.

ABSTRACT Malware classification is a critical task in cybersecurity, as it offers insights into the threats that malware poses to the victim device and helps in the design of countermeasures. For real-time malware classification, due to the high network throughputs of modern networks, there is a challenge of achieving high classification accuracy while maintaining low inference latency. We first introduce two self-attention transformer-based classifiers, SeqConvAttn and ImgConvAttn, to replace the currently predominant Convolutional Neural Network (CNN) classifiers. We then devise a file-size-aware two-stage framework to combine the two proposed models, thereby controlling the tradeoff between accuracy and latency for real-time classification. To assess our proposed designs, we conduct experiments on three malware datasets: the Microsoft Malware Classification Challenge (BIG 2015) and two selected subsets from the BODMAS PE malware dataset, BODMAS-11 and BODMAS-49. We show that our transformer-based designs can achieve better classification accuracy than traditional CNN-based designs. Furthermore, we show that the proposed two-stage framework reduces the average model inference latency while maintaining superior accuracy, thereby fulfilling the requirements of real-time classification.

INDEX TERMS Malware classification, self-attention networks, multi-stage classification, cybersecurity.

I. INTRODUCTION
Malicious software, commonly known as malware, conducts malicious activities when executed on a victim device. Classification of malware is an important process for both minimizing the damage done to infected devices and designing precautionary measures to prevent future attacks. Yet, while it is critical for the classification engine to be accurate, its inference process must also be timely. One major challenge in the anti-malware industry today is the large volumes of files constantly being transferred through connected networks. To handle this volume of incoming potential malware without disrupting the operations of the target device, endpoint security systems must efficiently identify and analyze transmitted software binaries in real time [1], [2], [3]. Therefore, achieving high accuracy with low latency is the principal objective of real-time malware classification.

By methodology, malware classification can be broadly separated into two categories: dynamic and static analysis [4]. Dynamic analysis usually requires the execution of suspicious software within a sandbox environment, with the malicious activities observed at runtime. Although this approach can potentially produce detailed information on malicious activities, the analysis is slow to conduct. The requirement of a sandbox also makes this approach inconvenient in scenarios where realtime remediation is required. Furthermore, some malware can differentiate between sandbox and actual runtime environments, and accordingly behave benignly during sandbox execution to evade dynamic analysis [5], [6]. Alternatively, static analysis examines signatures (patterns in a software byte sequence) and other static features extracted from software files to deduce malicious behaviour. This approach can be significantly faster than dynamic analysis, as malware execution is not required. However, many static analysis methods require converting the file binary into assembly code, which is nevertheless time-consuming and
not suitable for real-time applications. Thus, several recent publications focused on developing malware classification networks for direct analysis of raw binaries, with minimal preprocessing or feature engineering.

Neural network classifiers for malware binaries can be categorized into two types: sequence-based and image-based. Sequence-based classifiers, such as [7] and [8], apply 1D convolution to raw byte sequences for classification. However, a major challenge of sequence-based classification is the efficient modelling of long-byte sequences [8]. Image-based classifiers, on the other hand, first convert a binary file into an image representation. Traditionally, this entails converting the byte sequence into a greyscale image [9]. However, recent research investigated converting the byte sequence into frequency domain, such as [10], [11], and [12]. Many works, such as [13], [14], [15], and [16], also designed different convolutional neural network (CNN) architectures to improve malware classification performance. Compared to byte sequences, images can be viewed as a more size-efficient representation of malware binaries, at the cost of losing some level of information. We note that, from our survey of related work, barring some exceptions, such as [17] and [18], most published malware binary classifiers are predominantly CNN architectures.

In this paper, we propose the novel application of self-attention mechanisms for both sequence-based and image-based analysis to achieve superior results in malware classification. Transformer [19], first introduced for Natural Language Processing (NLP), has achieved state-of-the-art results in machine translation, summarization, question-and-answering, etc. [20]. The transformer predominantly relies on internal self-attention mechanisms to model inter-dependencies within a sequence. This allows the transformer to capture global contexts, whereas a conventional convolution layer could only capture local contexts within the scope of its kernels. More recently, [21] introduced the Vision Transformer, showing that transformers could also be applied to image classification.

We design two transformer-based malware classification models to enhance both sequence-based and image-based classification. To reduce the inference latency, we further integrate the two models into a novel file-size-aware two-stage framework. We assessed our proposed designs on three datasets, BIG 2015 [22], and two datasets derived from the BODMAS PE malware dataset [23], BODMAS-11 and BODMAS-49. Based on these experiments, our paper makes the following contributions:

- We propose SeqConvAttn for malware classification by using the byte sequences of binary files. By supplanting the traditional 1D CNN backbone with a transformer, the model achieves superior classification accuracy when compared to baseline models in most cases.
- We then propose the model ImgConvAttn, which employs Vision Transformer [21] on malware classification over malware images. ImgConvAttn can achieve good classification accuracy while significantly reducing inference latency when compared with that of SeqConvAttn.
- We integrate both models into a file-size-aware two-stage framework. Here, ImgConvAttn occupies the first stage, and SeqConvAttn takes the second stage. During inference, the framework would check the size of the binary file and the first-stage classification uncertainty score to decide whether the malware needs to undergo second-stage reclassification. By executing the necessary models on a need-to-run basis, this framework efficiently leverages different information learnt by the two proposed models to achieve superior accuracy while reducing the overall inference latency.

The rest of the paper is organized in the following manner. Section II reviews relevant related work. Section III presents the designs of SeqConvAttn, ImgConvAttn, and the file-size-aware two-stage framework. Section IV discusses experiment designs and results. Section V offers further insights on SeqConvAttn and ImgConvAttn. Section VI addresses limitations of our designs. Section VII contains the conclusion of this paper.

II. RELATED WORK

Static analysis checks the signatures and other static features derived from the examined file to determine the malware type. Technically, such files could contain binaries, assembly code, or even human-readable programming languages. However, in most situations, only software binaries are readily available. Disassembly of binaries into assembly code can often be time-consuming [24]. To reduce the overall file-to-verdict inference latency, file preprocessing time should also be minimized. We therefore forego any feature extraction techniques involving assembly code, and focus solely on classification using raw binaries. In this section, we present surveyed literature addressing the problem of malware binary classification. We summarize the surveyed literature in Table 1. Note that throughout the paper, the term "binary" and "byte sequence" will be used interchangeably, with the former used when referring to the content of malware files, and the latter used when addressing the input to a model.

A. BINARY SEQUENCE CLASSIFICATION

Intuitively, the content of a binary file can be directly examined as a sequence. Malware detection on byte sequence was first proposed by [7] using the Malconv model. Given a byte sequence, Malconv first performs byte embedding, then forwards the embedded sequence through gated convolution. To handle the length of the binary input, the convolution kernel and stride size are set to 512 bytes, aggressively reducing the resultant encoding length. Reference [25] then proposed an alternative deep architecture using multiple convolutional layers, with smaller kernel and stride sizes. References [31] and [32] formulated several adversarial attacks on Malconv to identify weaknesses of the architecture. Reference [33]
TABLE 1. Summary of surveyed literature on sequence-based and image-based malware classification.

| Literature | Year | Feature | Model(s) |
|------------|------|---------|----------|
| [7]        | 2018 | Binary Sequence | Malconv (Custom Gated Convolutional Network) |
| [25]       | 2018 | Binary Sequence | Custom CNN |
| [8]        | 2020 | Binary Sequence | Malconv + Convolution Over Time + Global Channel Gating |
| [17]       | 2018 | 1D Greyscale | CNN + BiLSTM |
| [18]       | 2019 | 1D Greyscale | CNN + BiGRU |
| [13]       | 2016 | 2D Greyscale | 3C2D and other shallow CNNs |
| [14]       | 2017 | 2D Greyscale | ResNet50 |
| [15]       | 2018 | 2D Greyscale | VGG16 |
| [26]       | 2021 | 2D Greyscale | ResNet50 |
| [27]       | 2019 | 2D Greyscale | CNN + Attention Mechanism |
| [28]       | 2021 | 2D Greyscale | VGG-19 + Custom Attention-CNN Design |
| [29]       | 2021 | 2D Greyscale | CNN + Residual Attention Mechanism |
| [30]       | 2021 | 2D Greyscale | Custom Attentional Residual Network |
| [10]       | 2018 | Markov Transition | ResNet50 |
| [11]       | 2020 | Markov Transition | Custom DCNN (similar to VGG16) |
| [16]       | 2021 | Greyscale, RGB, Markov Transition | Custom CNN variants and ResNet50 |
| [12]       | 2021 | Greyscale, Bigram, Bigram+DCT | 3C2D, ResNet18, ResNet50 |

Conducted activation analysis to gain further insights to the information learned from the byte sequences. Reference [34] then extended Malconv from malware detection, a binary problem, to malware classification, a multi-class problem. Finally, [8] introduced two major improvements on the original Malconv architecture. First, a convolution-over-time scheme was introduced, allowing Malconv to process byte sequences of arbitrary length in an efficient manner. Second, to better model the interactions between distant elements in the byte sequence, the paper proposed the Global Channel Gating architecture. Aside from CNN, architectures using recurrent neural networks, such as CNN+BiLSTM [17] and CNN+BiGRU [18], have also been explored as alternatives for sequence-based classification. Based on our survey, we note that works on sequence-based analyzers for malware binaries are somewhat limited. We attribute this issue to the fact that conventional sequence modelling architectures, such as CNN, RNN (Recurrent Neural Network), and transformer, have difficulties handling the extreme length of malware binaries, thereby discouraging research in sequence-based classification.

B. BINARY IMAGE CLASSIFICATION

Reference [9] was one of the first to investigate malware classification based on image representations of binaries. The approach applied K-nearest neighbour classification based on texture features extracted from malware greyscales. More importantly, this laid the groundwork for binary-to-image conversion. For each binary file, an appropriate image width is selected according to the file size. The binary file is then reconstructed by line-breaking the content over the image width. Reference [13] pioneered the application of CNN classifiers to malware images. Reference [14] proposed integrating deep networks with transfer-learning for malware classification, where ResNet50 [35] bottleneck output is taken to train shallow classifier networks. To accommodate the input constraint of ResNet50, the greyscale is converted into RGB image and resized to the specified input dimensions. Reference [15] repeated this approach, but replaced ResNet50 with VGG16 [36]. Subsequently, [26] further investigated the application of ResNet50 for malware classification. We note that there is a general trend of employing deep models for malware classification. To achieve low inference latency, inferences by deep networks has to be conducted on dedicated GPU. However, since GPU resources are often not available in many deployment scenarios, our research thus focused on the CPU inference time. We show in Table 2 that inferences made by common deep models on GPU results in high per-file latency, which makes them unsuitable for real-time application.

In addition to convolution, some publications also integrated attention mechanisms [37] into the model architecture, helping the classifier to focus on important regions in the malware image. Reference [27] employed an attentional convolutional network to visually identify regions-of-interest on a malware grayscale. Thereafter, binary sequences corresponding to these regions could be reassembled into assembly code, providing insight on how the malware operates. Later, [29] introduced a similar network, but employed a residual attention mechanism instead. Reference [28] investigated applying attention onto the VGG19 [36] bottleneck output to enhance classification accuracy. Reference [30] integrated Attentional Residual (AR) modules onto a ResNeXt50 backbone [38]. During classification, intermediate outputs from the ResNeXt-50 model are retrieved, passed through independent AR modules, and concatenated into a feature map. This feature map then passes through additional network layers to generate the classification verdict.

Aside from greyscale images, some recent works have also proposed generating malware images using alternative means. Reference [39] proposed generating a hash map based on the byte size associated to the opcodes found in the malware assembly script. However, this method requires a disassembly process, which would incur unacceptably high latency for classification. Reference [10] proposed generating malware Markov image (transition matrix). The Markov image has a dimension of $256 \times 256$, with each pixel representing the transition probability from one byte value to
TABLE 2. Feedforward time of common deep models for image classification in TorchVision.

| Model     | CPU Latency [ms] |
|-----------|------------------|
| VGG-16    | 116.8            |
| ResNet50  | 89.7             |
| Inception-v3 | 115.3         |

another in the binary. Reference [11] further experimented with training a deep CNN model from scratch using Markov images. Reference [16] devised an architecture to accept Markov image along with greyscale and RGB images, diversifying the information extracted from the binary file. Reference [12] investigated an alternative by directly recording the frequency count of each byte bigram, and then performing Discrete Cosine Transform (DCT) to desparsify the image.

C. TRANSFORMERS IN MALWARE CLASSIFICATION

The application of transformers to malware classification is recent. Reference [40] devised I-MAD, a hierarchical transformer-based framework for classification of assembly code. The work defined three different hierarchies of content in an assembly file: assembly instructions, basic blocks, and assembly functions. The architecture employed three transformers, with each transformer responsible for the encoding of its respective data hierarchy. For example, the basic block is encoded by feeding its component assembly instructions as the input to the basic-block level transformer. Reference [41] presented an alternative hierarchical transformer classifier, again for assembly code. Reference [42] proposed a non-hierarchical transformer model, where the input is a sequence of high frequency words extracted from the assembly code. Reference [43] investigated using variants of BERT [20] to detect Android malware based on the contents of AndroidManifest.xml files extracted from Android APKs. Note that most of these works require the disassembly of malware binary into assembly code. To our knowledge, there are currently no published works which investigates the application of transformers in malware classification using raw binaries.

III. PROPOSED METHOD

In this section, we first explain the design of the transformer module. Afterwards, we present our novel classifiers, SeqConvAttn and ImgConvAttn. Finally, we present our two-stage framework that efficiently integrate SeqConvAttn and ImgConvAttn to achieve fast and accurate malware classification.

A. BACKGROUND ON TRANSFORMER ARCHITECTURE

Transformer [19] refers to a family of architectures that relies on self-attention to model inter-dependencies between elements in a sequence. The underlying idea is that the encoding of any target element can be computed as the aggregate over 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, this aggregation process is referred to as scaled dot-product attention, defined by [19] as

$$SA(X) = \text{Softmax} \left( \frac{XW_Q(XW_K)^T}{\sqrt{d_{kv}}} \right) XW_V$$.

(1)

Note that $X \in \mathbb{R}^{N \times d_m}$ is the initial sequence encoding of length $N$ and an encoding dimension of $d_m$. The learnable weights $W_Q, W_K, W_V \in \mathbb{R}^{d_m \times d_{kv}}$ are used to project the $X$ into, respectively, the query $XW_Q$, key $XW_K$, and value $XW_V$ sequences, with $d_{kv}$ as the encoding dimension of these sequences. From the perspective of a target element, the scaled dot-product attention computes its encoding through a weighted summation over the encodings of all source elements in the value, with the attentional weights determined by the similarity between its corresponding query element and every key element. By carrying out the entire self-attention computation as a series of matrix multiplications, all element encodings are computed concurrently. Note that the scaled dot-product attention has a computation complexity of $O(N^3)$, scaling quadratically with respect to the sequence length. However, the computations can be parallelized, meaning that given sufficient computation resources, the time complexity can become constant, or $O(1)$.

To learn the different types of inter-dependencies within the sequence $X$, multiple scaled dot-product attention heads are employed in parallel. As the weight of each attention head is initialized differently, different heads would capture different types of inter-dependency. 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, which is mathematically defined as

$$\text{MHA}(X) = \text{Concat}(\{SA_h(X)^H\}_{h=1}^H) W_O$$.

(2)

Here, $H$ refers to the number of parallel attention heads, and $W_O \in \mathbb{R}^{hd_{kv} \times d_m}$ the post-concatenation projection weight. Note that in most transformer designs, the model dimension stays invariant after undergoing multihead attention. Thus, in this paper, $d_m = H d_{kv}$.

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 connections. The feedforward block consists of a ReLU activated expansion layer followed by a restorative layer, as shown by

$$\text{FF}(X) = \text{ReLU}(XW_1 + b_1)W_2 + b_2$$.

(3)

Note that the expansion layer parameters are $W_1 \in \mathbb{R}^{d_m \times d_{1u}}$, $b_1 \in \mathbb{R}^{d_{1u}}$ and the restorative layer parameters $W_2 \in \mathbb{R}^{d_{1u} \times d_m}$, $b_2 \in \mathbb{R}^{d_m}$. Here, $B$ is referred to as an expansion factor.

Transformers do not have implicit awareness of the positions of elements in the sequence. To remedy this, positional encoding is added to the initial sequence encoding before
undergoing self-attention. The positional encodings of the two proposed models differ. Thus, this detail is deferred to the descriptions of the proposed models.

**B. SeqConvAttn: SEQUENCE-BASED CLASSIFIER**

We first devised the SeqConvAttn to classify malware files based on the raw binary alone, with the architecture shown on the left of Figure 1. Given a byte sequence of length \( M \), where \( M \gg N \), the input byte sequence first undergoes byte embedding. The embedding layer maps each of the distinct 256 byte values (ranging from 0x00 to 0xFF) and a special padding “byte” to corresponding vectors. The embedded sequence then undergoes 1D convolution. Following [7], the convolution layer is designed with \( d_m \) large kernels with size \( S \) and stride \( S \). This compresses every consecutive segment of \( S \) bytes into a single sequence element, aggressively reducing the output encoding length to \( N = \frac{M}{S} \). The length reduction is necessary because the computation of the scaled dot-product attention from Equation 1 scales quadratically with respect to sequence length. Thus, an exceedingly long input would either incur an unacceptably long feedforward time or require too much computation resources.

After propagating through the 1D convolution layer, positional 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 design by [19], as shown by

\[
\text{PE}(n, i_d) = \begin{cases} 
\sin \left( \frac{n}{10000 \cdot \frac{d_m}{2^d}} \right), & i_d \mod 2 = 0 \\
\cos \left( \frac{n}{10000 \cdot \frac{d_m}{2^d}} \right), & i_d \mod 2 = 1 
\end{cases}
\]

where \( n \in \mathbb{Z} \cap [1, N] \) indicates the positional index of 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 the positional context through a set of alternating sinusoids. After adding positional encoding, the resultant sequence \( X \) propagates through a series of \( K \) transformer blocks. Upon obtaining the transformer output encoding, it is then max-pooled elementwise into a vector of dimension \( d_m \). This vector then proceeds through additional fully-connected layers. The final output then undergoes softmax, yielding the classification probability.

During experimentation, we found that reducing the encoding sequence length by 1D convolution alone is inadequate in decreasing the computation resource usage and incurred latency to an acceptable degree. We found it necessary to further truncate the byte sequence to a preset limit before conducting SeqConvAttn inference. However, overly truncating a byte sequence may remove salient byte sections critical to malware class identification, which could cause
misclassification. For example, [8] presented a case where
the malware author simply inserted a malicious payload
after the truncation index, thereby bypassing classifier detection.
To overcome this issue, we then explored converting the entire content of a binary file into an image, as an
efficient malware representation, and then applying another transformer-based architecture for classification.

C. ImgConvAttn: IMAGE-BASED CLASSIFIER
For the ImgConvAttn design, we first present the classifier architecture. Afterwards, we present the methods employed
in the malware detection process.

1) MODEL DESIGN
Reference [21] recently presented the Vision Transformer, a transformer model specialized for image classification. Following
the theme of leveraging self-attention, we further implementedImgConvAttn, a Vision Transformer model specialized for image-based malware classification. This design is illustrated on the right of Figure 1. Given a malware image, ImgConvAttn first processes the image through a single 2D convolutional layer. The 2D convolution consists of \( d_m \) kernels with size \( S \times S \) and stride of \( S \) both horizontally and vertically. This partitions the input image into nonoverlapping
sub-image patches and projects each patch into its corresponding encoding vector. Afterwards, the patch encodings are flattened into an encoding sequence. A placeholder vector, referred as \([SOS]\) (start-of-sequence), is appended to the front of this sequence. The purpose of this \([SOS]\) token is to efficiently encapsulate information within the entire sequence into a single vector of length \( d_m \). Positional encodings are then added to preserve the positional context of the patches. According to [21], the method of positional encoding used is not important, as long as some type of positional encoding exists. Thus, we followed the implementation of [44], where the positional encoding of ImgConvAttn is designed as a set of learnable parameters. The sequence encoding then passes through a series of transformer encoder layers. Upon obtaining the final encoding sequence, only the \([SOS]\) encoding is kept as the latent image representation, and the rest of the sequence elements discarded. To generate the final classification, the \([SOS]\) encoding propagates through additional fully connected layers, and then undergoes softmax to generate the classification probability.

2) MALWARE IMAGE GENERATION
We investigate two methods to generate malware images: greyscale and bigram frequency. For greyscale generation, the
malware byte sequence is first broken into consecutive lines of 256 bytes. Here, each byte is considered as a brightness value. To ensure consistent image dimension, the image is then resized into a 256 \( \times \) 256 matrix using bilinear interpolation.

To generate a bigram frequency image, the generation process consists of first tallying the occurrence count of each of the 65,536 distinct byte bigrams within the malware binary into a histogram. Subsequently, the histogram is rearranged into a matrix of size 256 \( \times \) 256, where an entry in row \( i \) and column \( j \) indicates the number of occurrences of the bigram \((i, j)\).

Recall that the computation complexity of the transformer module is \( O(N^2) \). Comparing the design of ImgConvAttn against SeqConvAttn, we expect that the model feedforward time should be much faster. Specifically, based on experimental settings later presented in Section IV-C, the ImgConvAttn transformer module input length is \( N = 257 \), while the SeqConvAttn transformer module input length is \( N = 800 \). However, the malware image generation process itself can incur significant latency. As shown later in Figure 3, the time needed to generate a bigram frequency image scales linearly to the malware binary length. Thus, it is not guaranteed that the overall file-to-verdict inference time of ImgConvAttn will always be faster than that of SeqConvAttn. This information is relevant to the design of the subsequent file-size-aware two-stage framework, as discussed in Section III-D2.

D. TWO-STAGE FRAMEWORK
Here, we first explain the design of the standard two-stage framework. Afterwards, we present the improved design by implementing the file-size-aware mechanism.

1) STANDARD TWO-STAGE DESIGN
Given that ImgConvAttn avoids the direct handling of long byte sequences, this design should have a shorter feedforward time when compared to SeqConvAttn. This satisfies the low latency requirement of real-time classification. However, as the two models learn to identify different types of features, classification accuracy could be improved by considering the verdicts from both models. To augment model accuracy while minimizing the overall classification latency, we incorporate both ImgConvAttn and SeqConvAttn into a two-stage framework, as shown by the design on the top diagram of Figure 2. We assign ImgConvAttn as the first-stage classifier, with an expected per-file inference latency of \( t_1 \). SeqConvAttn is then assigned as the second-stage, with a latency of \( t_2 \). To classify a malware binary, ImgConvAttn first conducts an initial classification. The classification uncertainty is then compared against a threshold value \( \nu \). If the uncertainty is below the \( \nu \), the classification process concludes. However, if the uncertainty exceeds the \( \nu \), the binary file then undergoes reclassification by SeqConvAttn. Thus, this framework avoids unnecessarily running the slower SeqConvAttn if ImgConvAttn is sufficiently certain in its classification. Assuming that ImgConvAttn classification is sufficiently certain most of the time, the majority of binary files should only incur an inference latency of \( t_1 \), while only a minority would incur a latency of \( t_1 + t_2 \). In our design, the classification uncertainty is defined as

\[
U_{pred} = 1 - P(C_{pred}). \quad (5)
\]
Here, $C_{\text{pred}}$ and $U_{\text{pred}}$ refer to, respectively, the predicted class and the classification uncertainty.

Tuning the $\nu$ hyperparameter offers control on the tradeoff between latency and accuracy, since it controls the approximate proportion, $p\%$, of malware files undergoing second-stage reclassification. This fact is important if a maximum average latency $t_{\text{spec}}$ is specified, since the latency can be related to reclassification proportion $p\%$ as

$$p\% = \frac{t_{\text{spec}} - t_1}{t_2}. \quad (6)$$

Thus we present a simple approach to selecting the optimal $\nu$ based on $t_{\text{spec}}$, or rather, $p\%$. First, prepare a set-aside (validation) set that is sampled from the test environment and disjoint with respect to the model training set. Second, conduct classification on the set-aside set using the first-stage classifier, and record the classification uncertainty score for each malware file. Finally, the corresponding $\nu$ is determined as the $(100 - p\%)^{\text{th}}$ percentile over the recorded set of uncertainty scores. Assuming that the set-aside set is sufficiently representative of the distribution of malware files encountered in the test environment, the two-stage framework should only perform reclassification for approximately $p\%$ of the time, thereby meeting the maximum latency requirement.

For subsequent experiments and discussions, we refer to the standard two-stage framework as TwoStage-$p\%$. The $p\%$ refers to the fact that the uncertainty threshold $\nu$ is set with the expectation that about $p\%$ of examined malware undergoes reclassification.

2) FILE-SIZE-AWARE TWO-STAGE DESIGN
Recall from Section III-C2 that the latency of the image generation process scales linearly to the malware file size. This entails that for sufficiently large malware binaries, the overall file-to-verify inference latency of ImgConvAttn will exceed that of SeqConvAttn. To handle these instances, we upgrade the two-stage framework by adding the file-size-possible mechanism. This mechanism preemptively redirects large binary files, whose expected latency fulfills the condition $t_1 \geq t_2$, to the second-stage classifier. This design is shown by the bottom diagram of Figure 2. Note that by preemptively redirecting larger malware files to the second-stage classifier, the actual proportion of files undergoing SeqConvAttn classification during runtime will likely be greater than $p\%$. However, the average inference latency should also be reduced.

For subsequent discussions, we refer to the file-size-aware design as TwoStage-fsa-$p\%$, again with $p\%$ indicative of the selected uncertainty threshold $\nu$ determined over an set-aside set.

IV. EXPERIMENTS

A. DATASET
We conducted experiments on three malware datasets, BIG 2015, BODMAS-11 and BODMAS-49.

1) BIG 2015
The original BIG 2015 dataset [22] consists of a “train” and a “test” partition. However, only the “train” set is labelled. Thus, we partitioned the 10,868 labelled malware binaries in the “train” set into disjoint train, validation, and test sets. Note that the average size of malware binaries in BIG 2015 is 1.29 MB, with only two outliers cases being larger than 3.8 MB (the largest file is 15 MB). The statistics of partitioned dataset are presented in Table 3.

The original dataset stores each malware binary as a sequence (string) of hexadecimal value in a text file.
To acquire the binary file, we converted the hexadecimals into bytes. For some files, some hexadecimals are of the value `??`s as `00` during the conversion process. Note that the resultant binary files are sterilized, as the portable executable headers were removed by the original vendor before distribution.

2) BODMAS-11 AND BODMAS-49
The original BODMAS dataset [23] contains of 57,293 malware binaries, belonging to one of the 581 malware families. The average size of a BODMAS malware binary file is 2.81MB, with the maximum file size being 214 MB. However, the majority of malware classes possess insufficient number of files for meaningful assessment. Consequently, only a subset are selected for the experiment. Specifically, we first retrieved only files timestamped on and after January 1, 2020. Afterwards, we composed the BODMAS-11 dataset from the 11 malware classes which contains more than 1000 instances. This dataset is then partitioned into disjoint train, validation, and test sets. Statistics on BODMAS-11 are presented in Table 4.

### Table 4. Statistics of BODMAS-11 dataset.

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

In addition, we also composed a BODMAS-49 dataset, consisting of the 49 malware classes that contains at least 100 instances. This dataset is also partitioned into disjoint train, validation, and test sets, as shown by Table 5. This dataset is highly imbalanced in the number of instances between different classes. Hence, BODMAS-49 is used to assess the limitations of our proposed models.

### B. TEST SETTINGS

We used three metrics to assess the performance of our model, accuracy, weighted-F1, and (file-to-verdict) latency. The former two metrics are used to measure the correctness of the model predictions. For each design, we report the average accuracy and weighted-F1 score over five repetitions. Latency refers to the average duration between when the binary file content is loaded into RAM and when the model prediction is obtained. For the assessment of single model designs, this consists of the binary file preprocessing time and the model inference time. For the two-stage frameworks, this also includes the time expended to check first-stage classification uncertainty and, for TwoStage-fsa-p%, to run the file-size-aware mechanism. Note that while we use GPU for model training, to address portability issues, only CPU is used to assess classification latency. As our experiment involves latency assessment, we list the relevant specifications of our test environment in Table 6.

### Table 5. Statistics of BODMAS-49 dataset.

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

### Table 6. Test environment specification.

| Setting | Specification |
|---------|---------------|
| Python  | 3.6.9         |
| Pytorch | 1.7.1+cu110   |
| Cuda    | 11.2          |
| OS      | Ubuntu 18.04 LTS |
| CPU     | AMD Ryzen 7 3900X 12-Core Processor |
| Memory  | 64 GB         |
| GPU     | GeForce RTX 2080 Ti |
C. MODEL SETTINGS

Here, we present the model dimensions and training parameters for SeqConvAttn and ImgConvAttn. We also introduce the baselines models for comparison of experiment results.

1) SeqConvAttn

For preprocessing, we standardize the byte sequence length to 400,000. If a byte sequence is longer, it is truncated to preserve only the first 400,000 bytes. If a sequence is shorter than 400,000 bytes, padding “bytes” are appended until the specific 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 500. The transformer section consists of 3 blocks, based on the PyTorch implementation of the TransformerEncoderLayer. 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 component 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 class.

For model training, the 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 baselines for comparison against SeqConvAttn are defined below.

- Malconv [7]: The authorial implementation of this model can be found on Github [45]. Note that for analysis purposes, we adjusted the original convolutional kernel and stride size from 512 to 500. Note that as Malconv relies primarily on 1D convolution to analyze the malware binaries, its computation complexity is linear with respect to the input length, or $O(N)$. However, as the convolution process can be parallelized, the potential time complexity can be constant, or $O(1)$, if the sufficient computation resources are available.

- Malconv+GCG [8]: This model introduced Global Convolution Gating to better learn inter-dependencies between distant elements. The original code is taken from [45], but extensively modified to remove the convolution-over-time mechanism. Again, the convolutional kernel and stride size are set to 500. Similar to Malconv, its computation complexity is $O(N)$, but its time complexity can be $O(1)$.

- CNN+BiLSTM: While [17] already presented a version of CNN+BiLSTM, the paper did not provide sufficient information for replication. Thus, we implemented our own version of CNN+BiLSTM model. Compared to the proposed SeqConvAttn model, the only difference is the replacement of the transformer section with a single layer of BiLSTM network. Note that while the computation complexity is $O(N)$, the BiLSTM encoding process is recurrent (performed iteratively over the elements in a sequence), and is thus not parallelizable. Thus, its time complexity remains $O(N)$.

- CNN+BiGRU: This model is very similar to CNN+BiLSTM baseline, with the only difference being that the BiLSTM layer is exchanged for a BiGRU network. Similar to BiLSTM, the recurrent BiGRU mechanism has a computation and time complexity of $O(N)$.

2) ImgConvAttn

The ImgConvAttn code is based on an implementation of the Vision Transformer [21] by [44]. The initial convolutional layer consists of 64 kernels of dimensions $16 \times 16$ 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 feedoward component has an expansion factor of 4. The final classifier consists of a single fully connected layer, whose width is determined by the number of classes.

We assign a training batch size of 50. During training, the same Adam optimizer settings 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, implemented based on the description provided by [12]. The original design, proposed by [13], is a shallow CNN consisting of 3 convolution-and-max-pooling layers and 2 fully connected layers with training dropout. Reference [12] added dropouts to the fully connected layer to improve model robustness. For each model, we experimented with two image types: bigram frequency and greyscale. We forego using deep models as done by [14], [15], and [26], and others. The reason is that, based in Table 2, these models incur very large latency, which is too slow for the purpose of real-time classification. This renders comparison to deep models pointless.

3) TWO STAGE FRAMEWORK

For the two-stage frameworks, we employ the ImgConvAttn model with bigram frequency malware images on the first stage, and the SeqConvAttn model on the second stage. The uncertainty thresholds $\nu$ corresponding to the expected proportion of reclassified file $p\%$ are determined through the validation set of each dataset. As experiments on the two-stage framework are also conducted with five repetitions (each times using a different set of ImgConvAttn-Frequency and SeqConvAttn), separate $\nu$ values are used for each of the five runs. However, the magnitude of $\nu$ remain broadly consistent throughout the runs. Thus, we report the averaged $\nu$ in Table 8 for reference purposes. Note that for TwoStage-fsa-$p\%$ experiment, the file size threshold is set to 5 MB, since that is where the per-file latency of ImgConvAttn-Frequency and the average latency of SeqConvAttn are about equal, as shown on Figure 3 and
D. DISCUSSION OF RESULTS

From the results in Table 7, the following general observations are made. First, SeqConvAttn is shown to achieve superior accuracy in most cases when compared to baseline sequence-based classifiers. Second, ImgConvAttn-Frequency achieves the best accuracy when compared to all alternative image-based classification designs. Finally, the Two-Stage-fsa-p% framework is shown to be able to achieve superior accuracy when compared to all single model designs, while incurring relatively low average classification latency. Further discussions are separated into three parts, corresponding to the BIG 2015, BODMAS-11, and BODMAS-49 datasets.

1) BIG 2015 RESULTS

On Big 2015, for all sequence-based classifiers, SeqConvAttn achieved the best accuracy score of 96.59%. Although CNN+BiLSTM also achieve the same accuracy, its CPU latency is about 25 ms longer than SeqConvAttn. Recall that this is because the BiLSTM feedforward process is conducted recurrently and is thus not parallelizable. The self-attention of SeqConvAttn, on the other hand, concurrently computes the encodings of all elements of the post-convolution input sequence. Compared to Malconv, SeqConvAttn is approximately 2% more accurate, while Malconv is about 18 ms faster. Interestingly, SeqConvAttn also outperformed Malconv+GCG, which is an upgrade over the original Malconv model. This suggests that SeqConvAttn can identify certain salient features, possibly inter-dependencies within the binaries, that Malconv-based models cannot detect.

Among the image-based classifiers, ImgConvAttn is demonstrated to be superior to the 3C2D model when given the same image type. Although ImgConvAttn is generally 2-3 ms slower than 3C2D, the average inference latency is nevertheless quite fast. Additionally, the bigram frequency images are demonstrated as superior to greyscale images on ImgConvAttn. Note also that the 98.17% accuracy score achieved by ImgConvAttn-Frequency is higher than that of SeqConvAttn, while the former also incurred significantly lower latency. This may be explained by the information loss caused by the truncation of input binary for SeqConvAttn, which potentially removed salient features in the byte sequence after the truncation limit.

2) BODMAS-11 RESULTS

On BODMAS-11, SeqConvAttn attained the best accuracy and weighted-F1 score of any single model designs. However, the differences in accuracy between SeqConvAttn and the other baselines are quite small. For example, the second best model, CNN+BiGRU is only about 0.09% worse than SeqConvAttn. We suspect that this is because all sequence-based classifiers likely identified similar features for classification. When compared to Malconv, SeqConvAttn is 0.16% better in accuracy, but its latency is also 14 ms longer. Most likely, the marginal accuracy discrepancy between different models is caused by the truncation of malware binaries. As BODMAS binary files are generally larger than that of BIG 2015, the truncation process thus removes more salient features, including the distant inter-dependencies within malware binaries that SeqConvAttn is designed to identify.

For image-based classifiers, ImgConvAttn-Frequency is the most accurate of the image-based classification designs. Note that the latency disparity between 3C2D and ImgConvAttn becomes insignificant on BODMAS-11, as the average classification latency for all models on BODMAS-11 are significantly larger than that of BIG 2015. This is caused by the fact that BODMAS-11 malware binaries are generally larger in size. Referring again to Figure 3, this slows down the bigram-frequency image generation process. Note that this also causes a significant latency disparity between greyscale and bigram frequency, likely since greyscale generation time does not scale as strongly to the length of malware binaries.

For both TwoStage-p% and TwoStage-fsa-p%, adjustment of the uncertainty threshold $\nu$ results in a tradeoff between accuracy and latency for the two-stage frameworks, as both metrics generally increases with smaller uncertainty threshold (larger $p$%). However, at approximately after $p = 15$, there is a clear effect of diminishing return, such that further increase in latency no longer corresponds to gains in accuracy. This phenomenon is caused by the fact that, with smaller uncertainty thresholds, binary files with more confident first-stage classification are subjected to second-stage reclassification. Since files with higher confidence for first-stage classification are more likely to be correct, reclassification by the second-stage becomes increasingly redundant.
Comparing TwoStage-p% against TwoStage-fsa-p%, their classification accuracy scores are similar, with TwoStage-fsa-p% being slightly superior. Furthermore, due to the file-size-aware mechanism, TwoStage-fsa-p% is able to reduce the classification latency by 10 ms. Since ImgConvAttn is slower than SeqConvAttn if malware binary exceeds 5 MB, it is more efficient to reroute these instances directly to SeqConvAttn rather than continuing with the conventional two-stage classification process.

Finally, comparing TwoStage-fsa-p% with all other single model designs, as long as $\nu$ is properly set, the file-size-aware two-stage framework would yield superior accuracy over all other models, while only incurring a latency comparable to Malconv. This is exemplified by TwoStage-fsa-25%, which achieved the best test accuracy score on BODMAS-11 of 96.96% while incurring a latency of just 18.5 ms. Note that while the latency reduction is the primary intent of the two-stage design, the slight accuracy increase over its component SeqConvAttn model is a secondary effect. We explain the accuracy increase by emphasizing that the component model SeqConvAttn and ImgConvAttn identifies different types of features. Thus, there exists malware binaries which would be misclassified on SeqConvAttn while being correctly (and confidently) classified by ImgConvAttn. The uncertainty-based reclassification process in the two-stage framework thus filters-out a portion of these instances. Essentially, the two models somewhat covers the weaknesses of each other, resulting in the accuracy of the two-stage framework to exceed that of either of its component models.

3) BODMAS-49 RESULTS

We experimented with BODMAS-49 to determine potential limitations of our designs in an environment with significant class imbalance. Examining the single model designs, ImgConvAttn-Frequency accuracy and weighted-F1 are significantly better than the other image-based schemes. However, while SeqConvAttn performance are better than most of the baseline sequence-based classifiers, its accuracy is equal to the accuracy of Malconv. We explain this observation as SeqConvAttn being inadequately trained on the minority malware classes, as these classes has insufficient number of training files. Despite that SeqConvAttn is able to learn more complex features when compared to Malconv, this advantage cannot be leveraged if there are too few examples to reliably generalize the salient and discriminating features of the minority classes.

Despite the limitations of SeqConvAttn, the proposed two-stage framework nevertheless achieved superior accuracy when compared to all single models. Specifically, we consider TwoStage-fsa-25% as the best design. This design achieved an accuracy of 93.42% and incurred latency of 17.2 ms, which is slightly better than the accuracy of 93.33% and latency of 18.4 ms from Malconv. Note that while TwoStage-20% technically achieved the overall best
accuracy of 93.43%, the improvement over TwoStage-fsa25% is insignificant, while the lack of file-size-aware mechanism also resulted in a significantly larger latency of 27.7 ms. Thus, we demonstrate on BODMAS-49 again that the file-size-aware two-stage framework can achieve superior accuracy while incurring a relatively low latency.

V. ADDITIONAL INSIGHTS
In this section, we present some insights gained from additional analyses of our SeqConvAttn and ImgConvAttn classifiers.

A. ANALYSIS OF ATTENTION MAPS OF SeqConvAttn
We further investigated the features learned by the SeqConvAttn model by examining the attention maps on different malware binaries. For ease of interpretation, for each input byte sequence, the attention maps from the different attention heads in the final transformer block are averaged element-wise. Note also that only the attention on the first 200 elements is shown. The resultant maps for two malware samples are presented at the top of Figure 4. Note that application of softmax in Equation 1 corresponds to rows across the attention maps.

From the attention maps, we notice that there exists several vertical highlights (bright yellowish-green streaks). These highlights indicate strong attention paid to the corresponding elements. Recall that each element in the byte sequence corresponds to a segment of 500 bytes. Thus, the corresponding elements can be interpreted as salient sections in the binary indicative of the malware class.

Furthermore, we compared the attention map of SeqConvAttn with the gating map in Malconv. Akin to the attention weights in SeqConvAttn, Malconv employs gated convolution [46] to suppress or highlight information. The bottom images of Figure 4 displays the gating maps of the respective samples, with the columns corresponding to the sequence elements. We note that it is difficult to visually divulge from the gating map the emphasis or suppression of particular information. Unlike attention maps, whose value suppresses the information of entire elements, values of gating maps can independently suppress specific sections of the encoding of a sequence element. This hinders the interpretability of Malconv, as we cannot easily detect the binary sections deemed salient by the model. The best we can discern is the presence of different binary regions. This is shown by the different textures exhibited on the gating map, which are potentially indicative of the different types of information present in different regions.

We also observed that in most cases, the vertical sections of attention maps, identified by similar highlight densities, tend to correspond to their respective binary regions on the gating map. This suggests that, though by different means, the
underlying features learnt by SeqConvAttn and Malconv are likely similar. We consider the capability of SeqConvAttn to generate attention maps as a further advantage over Malconv. As the vertical attention highlights the salient binary blocks, it renders SeqConvAttn more interpretable when compared to Malconv gating map.

**B. COMPARISON OF GREYSCALE AND BIGRAM FREQUENCY WITH ImgConvAttn**

Noting that the classification accuracy from bigram frequency images significantly outperforms that of greyscale images, we further investigated the reasons behind these results. In Figures 5 and 6, the greyscale and bigram frequency images of three samples of the Upatre malware class from the BODMAS-11 test set are presented. Note that for bigram frequency, we project the image as a heatmap over the grey spectrum. Furthermore, certain pixel values in the image are clipped for viewing purposes. Additionally, we also present the corresponding ImgConvAttn attention maps for analysis purposes. For visualization, attention maps from the attention heads of the last transformer block are first averaged elementwise. The resultant map is then further averaged across the dimension perpendicular to the softmax in Equation 1, producing a 257-length vector. The attentional value associated to the [SOS] vector is then removed, and the resultant 256-length vector is reshaped to a $16 \times 16$ image. Thus, each value in the attention map can be interpreted as the saliency of its corresponding sub-image patch. Note that red grid lines are added to the greyscales and bigram frequency images for visualization purposes.

The greyscale images, when compared to each other, visually differ greatly despite the samples belonging to the same malware class. This is because the underlying byte sequences from different file instances differ greatly. This difference likely makes acquiring the salient features for classification difficult. As observed from the corresponding attention maps, the salient patches identified from the three greyscale images have little in common. However, the more fundamental cause for the poor accuracy on greyscale image is two-fold. First, converting from 1D byte sequence to 2D image introduces vertical relationship between bytes that does not exist in the original 1D format. Such relationship could be easily altered by shifting the position of certain byte sections or interjecting additional bytes at some location. Second, the byte value cannot be interpreted as a greyscale value. This is because distinct byte values, such as 12 ($0 \times 0C$) and 255 ($0xFF$) do not have a comparative relationship that defines one value as
FIGURE 6. Top: Generated Bigram Frequency Image. Bottom: Visualization of ImgConvAttn Attention Map. Note that each column corresponds to a different Upatre sample. Notice that the attention maps of the three samples generally looks similar. For example, in the first row of (d), (e), and (f), the sub-image patches [0, 3], [0, 7], [0, 10], and [0, 12] are all attended on to some extent.

VI. LIMITATIONS
Regarding the model designs, we identify two limitations of SeqConvAttn that require further investigation. First, the fact that transformer complexity scales quadratically to the input length necessitates the truncation of malware binaries, as otherwise the feedforward time and required computation resources would grow unacceptably large. However, the byte sequence lost from truncation may cause SeqConvAttn to misclassify malware files. Thus, it is necessary to develop a process to feed an entire malware binary to SeqConvAttn without truncation, while overcoming the issue of computation complexity. Second, as inferred from the results on BODMAS-49, despite the higher model complexity of SeqConvAttn, superior classification performance cannot be achieved if minority classes have insufficient number of training examples. Thus, better training schemes need to be developed to help SeqConvAttn when learning within an environment with significant class imbalance and scarce examples for each class.
We also note that the scope of our work could be expanded to improve the utility of our designs. First, we may investigate incorporating dynamic analysis on the proposed file-size-aware two-stage framework. Specifically, additional stages may be created to support the execution of a dynamic analysis engine for malware files with uncertain classifications. This would allow dynamic analysis to be conducted occasionally under a real-time classification setting, potentially improving overall accuracy. Second, we should consider the scenario where the two-stage framework is executed on a dynamic environment, with the distribution of different malware classes changing overtime. This would necessitate the adaptation of the uncertainty threshold $\nu$ to the malware class distribution in order to control the overall inference latency.

VII. CONCLUSION

In this paper, we investigated the use of transformers on raw binaries for real time malware classification. To achieve this, we proposed two models to interpret the binaries in two different ways. First, we proposed SeqConvAttn for handling the raw binary as a byte sequence. Noting that the classification latency scales quadratically with the length of the input sequence, we then proposed ImgConvAttn, an alternative low-latency model that uses Vision Transformer to classify malware based on preprocessed images. To leverage both models in an efficient manner, we integrated the two models into a file-size-aware two-stage framework. We conducted extensive experiments on the BIG 2015, BODMAS-11, BODMAS-49 dataset, and demonstrated that both SeqConvAttn and ImgConvAttn are superior to their respective baselines in accuracy and weighted-F1. Additionally, we demonstrated on BODMAS-11 and BODMAS-49 that the proposed file-size-aware two-stage framework can effectively control the average inference latency in real-time application while achieving superior classification accuracy through correct setting of the uncertainty threshold.

REFERENCES

[1] S. Alam, R. N. Horpool, I. Traore, and I. Sogukpinar, “A framework for metamorphic malware analysis and real-time detection,” Comput. Secur., vol. 48, pp. 212–233, Feb. 2015.

[2] R. Vinayakumar, M. Alazab, K. Soman, P. Poonachandran, and S. Venkatraman, “Robust intelligent malware detection using deep learning,” IEEE Access, vol. 7, pp. 46717–46738, 2019.

[3] T. Rezaei and A. Hamze, “An efficient approach for malware detection using PE header specifications,” in Proc. 6th Int. Conf. Web Res. (ICWR), Apr. 2020, pp. 234–239.

[4] A. Abusita, M. Q. Li, and B. C. M. Fung, “Malware classification and composition analysis: A survey of recent developments,” J. Inf. Secur. Appl., vol. 59, Jun. 2021, Art. no. 102828.

[5] E. Raff and C. Nicholas, “A survey of machine learning methods and challenges for Windows malware classification,” 2020, arXiv:2006.09271.

[6] O. Aslan and R. Samet, “A comprehensive review on malware detection approaches,” IEEE Access, vol. 8, pp. 6249–6271, 2020.

[7] E. Raff, J. Barker, J. Sylvester, B. Brandon, B. Catanzaro, and C. K. Nicholas, “Malware detection by eating a whole EYE,” in Proc. Workshops 32nd AAAI Conf. Artif. Intell., 2018, pp. 1–9.

[8] E. Raff, W. Fleschner, R. Zak, H. S. Anderson, B. Filat, and M. McLean, “Classifying sequences of extreme length with constant memory applied to malware detection,” 2020, arXiv:2012.09350.

[9] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: Visualization and automatic classification,” in Proc. 8th Int. Symp. Visualiz. Cyber Secur., 2011, pp. 1–7.

[10] S. Xia, Z. Pan, Z. Chen, W. Bai, and H. Yang, “Malware classification with Markov transition field encoded images,” in Proc. 8th Int. Conf. Instrum. Meas., Comput., Commun. Control (IMCC), Jul. 2018, pp. 1–5.

[11] B. Yuan, J. Wang, D. Liu, W. Guo, P. Wu, and X. Ban, “Byte-level malware classification based on Markov images and deep learning,” Comput. Secur., vol. 92, May 2020, Art. no. 101740.

[12] T. Manhur Mohammed, L. Nataraj, S. Chikgogudar, S. Chandrasekaran, and B. S. Manjunath, “Malware detection using frequency domain-based image visualization and deep learning,” 2021, arXiv:2101.10578.

[13] D. Gilbert, “Convolutional neural networks for malware classification,” Dept. Escola Técnica Superior d’Enginyeria, Univ. Rovira i Virgili, Tarragona, Spain, Tech. Rep., 2016.

[14] 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 Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA), Dec. 2017, pp. 1011–1014.

[15] 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 (Advances in Intelligent Systems and Computing). Cham, Switzerland: Springer, 2018, pp. 51–59.

[16] A. Pinhero, M. L. Anupama, P. Vinod, C. A. Visaggio, N. Aneesh, S. Abhijith, and S. AnanthaKrishnan, “Malware detection employed by visualization and deep neural network,” Comput. Secur., vol. 105, Jun. 2021, Art. no. 102247.

[17] Q. Le, O. Boydell, B. M. Namee, and M. Scanlon, “Deep learning at the shallow end: Malware classification for non-domain experts,” Digit. Invest., vol. 26, pp. S118–S126, Jul. 2018.

[18] S. Venkatraman, M. Alazab, and R. Vinayakumar, “A hybrid deep learning image-based analysis for effective malware detection,” J. Inf. Secur. Appl., vol. 47, pp. 377–389, Aug. 2019.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5998–6008.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” 2018, arXiv:1810.04805.

[21] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2020, arXiv:2010.11929.

[22] R. Ronen, M. Rudu, C. Feuerstein, E. Yom-Tov, and M. Ahmadi, “Microsoft malware classification challenge,” 2018, arXiv:1802.10135.

[23] L. Yang, A. Ciptadi, I. Laziu, A. Ahmadzadeh, and G. Wang, “BODMAS: An open dataset for learning based temporal analysis of PE malware,” in Proc. IEEE Secur. Privacy Workshops (SPW), May 2021, pp. 78–84.

[24] V. Verma, S. K. Muttoo, and V. B. Singh, “Multiclass malware classification via first- and second-order texture statistics,” Comput. Secur., vol. 97, Oct. 2020, Art. no. 101895.

[25] M. Krčál, O. Švec, M. Bálek, and O. Jásek, “Deep convolutional classifiers can learn from raw executables and labels only,” in Proc. ICLR, 2018, pp. 1–4.

[26] Sudhakar and S. Kumar, “MCFT-CNN: Malware classification with fine-tune convolution neural networks using traditional and transfer learning in Internet of Things,” Future Gener. Comput. Syst., vol. 125, pp. 354–351, Dec. 2021.

[27] H. Yakura, S. Shinozaki, R. Nishimura, Y. Oyama, and J. Sakuma, “Neural malware analysis with attention mechanism,” Comput. Secur., vol. 87, Nov. 2019, Art. no. 101592.

[28] M. J. Awan, O. A. Masood, M. A. Mohammed, A. Yasin, A. M. Zain, R. Damaschevicius, and K. H. Abdullahkareem, “Image-based malware classification using VGG 19 network and spatial convolutional attention,” Electronics, vol. 10, no. 19, p. 2444, Oct. 2021.

[29] S. Ganesan, V. Ravi, M. Krichen, V. Sowmya, A. Alroobaea, and K. Soman, “Robust malware detection using residual attention network,” in Proc. IEEE Int. Conf. Consum. Electron. (ICCE), Jan. 2021, pp. 1–6.

[30] D. B. Tariang, S. C. Birudaraju, R. Naskar, V. Khare, and R. S. Chattobory, “Malware classification through attention residual network based visualization,” in Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST), Dec. 2020, pp. 1–6.
[31] O. Suciu, S. E. Coull, and J. Johns, “Exploring adversarial examples in malware detection,” in Proc. IEEE Secur. Privacy Workshops (SPW), May 2019, pp. 8–14.

[32] 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 Proc. 26th Eur. Signal Process. Conf. (EUSIPCO), Sep. 2018, pp. 533–537.

[33] S. E. Coull and C. Gardner, “Activation analysis of a byte-based deep neural network for malware classification,” in Proc. IEEE Secur. Privacy Workshops (SPW), May 2019, pp. 21–27.

[34] M. A. Kadri, M. Nassar, and H. Safa, “Transfer learning for malware multi-classification,” in Proc. 23rd Int. Database Appl. Eng. Symp. (IDEAS), 2019, pp. 1–7.

[35] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.

[36] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2014, arXiv:1409.1556.

[37] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in Proc. Int. Conf. Mach. Learn., 2015, pp. 2048–2057.

[38] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He, “Aggregated residual transformations for deep neural networks,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 1492–1500.

[39] B. Jung, T. Kim, and E. G. Im, “Malware classification using byte sequence information,” in Proc. Conf. Res. Adapt. Convergent Syst., Oct. 2018, pp. 143–148.

[40] M. Q. Li, B. C. M. Fung, P. Charland, and S. H. H. Ding, “I-MAD: Interpretable malware detector using galaxy transformer,” 2019, arXiv:1909.06605.

[41] 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 Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom), Dec. 2020, pp. 927–934.

[42] Y. Ding, S. Wang, J. Xing, X. Zhang, Z. Qi, G. Fu, Q. Qiang, H. Sun, and J. Zhang, “Malware classification on imbalanced data through self-attention,” in Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom), Dec. 2020, pp. 154–161.

[43] A. Rahali and M. A. Akhloufi, “MalBERT: Using transformers for cybersecurity and malicious software detection,” 2021, arXiv:2103.03806.

[44] R. Wightman. (2019). Pytorch Image Models. [Online]. Available: https://github.com/rwightman/pytorch-image-models

[45] (2020). MalConv2. [Online]. Available: https://github.com/NeuronorphicComputationResearchProgram/MalConv2

[46] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, “Language modeling with gated convolutional networks,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 933–941.

QIKAI LU received the B.Eng. and M.Sc. degrees in computer engineering from the University of Alberta, in 2018 and 2021, respectively, where he is currently pursuing the Ph.D. degree. His current research interests include integrating machine learning into malware detection and analysis and natural language processing.

HONGWEN ZHANG received the M.Sc. and Ph.D. degrees in computer science from the Universities of Calgary, Canada, and London, U.K. Within his role at Wedge Networks as the President and the Chief Scientist, he works with Wedge’s customers, Research and Development, and engineering teams to cater to customer requirements and demands, while ensuring that the company’s product and IP portfolio is continually updated and relevant within the fast moving security industry. He had co-founded M power Technologies Inc.—a wireless telecommunications software company, in 1997, and ActiveIq.com (NASDAQ: AIQT), a Boston based e-Business applications firm, in 1999. He is a Co-Founder of Wedge Networks. He is an innovator with more than 17 years of research and development experience working with industry leaders such as Newbridge (Alcatel), Siemens, United Technologies, and Apple in the areas of distributed information systems, embedded applications, and wireless internet solutions.

HUSSAM KINAWI received the M.Sc. and Ph.D. degrees in computer science from the Universities of Calgary, Canada, and London, U.K. Within his role at Wedge Networks as the President and the Chief Scientist, he works with Wedge’s customers, Research and Development, and engineering teams to cater to customer requirements and demands, while ensuring that the company’s product and IP portfolio is continually updated and relevant within the fast moving security industry. He had co-founded M power Technologies Inc.—a wireless telecommunications software company, in 1997, and ActiveIq.com (NASDAQ: AIQT), a Boston based e-Business applications firm, in 1999. He is a Co-Founder of Wedge Networks. He is an innovator with more than 17 years of research and development experience working with industry leaders such as Newbridge (Alcatel), Siemens, United Technologies, and Apple in the areas of distributed information systems, embedded applications, and wireless internet solutions.

DI NIU (Member, IEEE) received the B.Eng. degree from Sun Yat-sen University, in 2005, and the M.Sc. and Ph.D. degrees from the University of Toronto, in 2009 and 2013, respectively. He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Alberta, specialized in the interdisciplinary areas of distributed systems, data mining, machine learning, text mining, and optimization algorithms. He was a recipient of the Extraordinary Award of the CCF-Tencent Rhino Bird Open Grant 2016 for his research on natural language processing and machine learning for web document understanding at scale. Since 2017, he has been involved with Wedge Networks in the research and application of machine learning models for commercial cybersecurity services.