IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 8, April 2023

Android Mobile Malware Detection using Machine Learning Techniques

R. Mariammal[1], Bommanaboina Haribabu[2], Konka Venkatesh[3], Korsipati Rahul Reddy[4]
Assistant Professor, Dhanalakshmi College of Engineering, Chennai, India[1]
Students, Dhanalakshmi College of Engineering, Chennai, India[2,3,4]

Abstract: *With the growth of malware and improvements in cyberattacks, malware detection is crucial to maintaining cyber security. These attacks frequently employ previously undetectable malware that is not the focus of security firms, and it is inevitable that solutions will be found to learn from unlabelled sample data. This paper introduces SHERLOCK, a deep learning model for malware detection based on self-monitoring and the Vision Transformer (ViT) architecture. Using binary representation based on images, SHERLOCK is a novel malware CA detection technology that learns distinctive traits to separate malware from the benign programmes they employ. Self-supervised learning can achieve accuracy 97% for binary malware classification, which is higher than current state-of-the-art algorithms, according to experimental results employing 1.2 million Android apps hierarchy of 47 categories and 696 families. With macro-F1 scores of 0.497 and 0.491, respectively, the suggested model can also outperform cutting-edge methods for multi-class malware classification types and families.*

Keywords: Supervised Learning, Deep Learning, Malware Detection, Android Security

I. INTRODUCTION

Since its conception, artificial intelligence has been used in several applications, gaining substantial appeal in recent years. It currently rules the imaging industry, particularly in picture categorization [1]. The accuracy and dependability of models that employ deep learning architectures have been substantially enhanced by advancements in deep learning technology. The effectiveness of Deep Neural Networks (DNNs), which benefit from the growth of big data sets as well as the increase in computer capacity, may be impressive. Although DNNs have achieved state-of-the-art performance, new research has shown that these models are vulnerable to adversarial attacks, which change the DNN's output and reduce performance [2]. Due to the nature of deep neural networks, which are extremely sensitive to minor adjustments that are invisible to the human eye, adversarial perturbation is a well-known problem. Many works have recently JiafengXie served as the manuscript's assistant editor, overseeing the review process and giving final approval for publication. have been suggested to provide a varied set of adversarial examples, which have been used as benchmarks to assess the resilience of deep learning models [3]. Like this, a number of methods are put forth in the literature to thwart such attacks [4]

Malware detection, which involves identifying dangerous software programmes by separating them from legitimate applications, offers an increasingly difficult issue in the field of software security [5]. Recent research [6] show that the prevalence of malicious software is rising alarmingly, and that some or most of it hides inside well-known programmes utilising obfuscation techniques to avoid being discovered using conventional signature-based detection techniques. The integrity of Android applications is substantially threatened by Android malware, and identification of this malware is very important. More than 3.25 million harmful Android apps were discovered in 2016 [7], according to statistics, meaning that one new malware app is discovered every 10 seconds.

To stop malware from being released in the world through Android application markets, more focus and study are needed on Android malware detection. The issue of malware detection can be viewed as a classification problem, where the algorithm must correctly classify a particular programme into one of the two categories—malware or benign—in order to identify it. The majority of currently used methods for detecting Android malware focus on extracting application data and training a machine learning classifier with prior information to differentiate between good and bad applications. Despite their excellent accuracy, these models are useless since attackers can add characteristics that are frequently found in good programmes to avoid being detected. Simple additions of benign features, including pop-up notifications and logging, can transform malicious apps into benign ones, changing the detection class.

Recent malware programmes are designed with the intention of being extremely similar to innocuous software by combining characteristics of well-known applications, with only tiny variations that are challenging for humans to perceive [9]. DNNs are able to spot such subtle changes, and with enough training data, we can identify malicious application characteristics even when they are mixed in with a lot of good ones. In fact, earlier studies [10] have demonstrated that malware may be found using image classifiers built using deep learning models. Deep learning models can be trained to recognise certain traits that can then be used to detect the presence of similar variations in other applications. Malware variants from the same family demonstrate visual similarities in byte graph images. By transforming data to pixels, these images are created using visualisation in the spatial domain. In order to identify malware, the works often transform malware binaries into digital pictures and send them to a neural network. Machine learning is frequently used to identify Android malware, whether the analysis method is static, dynamic, or hybrid. Machine-learning based approaches offer higher performance in detecting efficacy and efficiency, in addition to the capacity to detect, when compared to classic methods, such as signature-based detection. Zero-shot learning, which can be used to find malware that has not yet been seen.

In order to gain from deep learning without incurring the cost of extensive annotations and learn feature representations when data supplies the supervision, self-supervised methods are crucial in this situation. We demonstrate that, using one of the largest data sets comprised of one million Android apps carefully selected from the AndroZ-oo[11]repository, our proposed technique outperforms state-of-the-art techniques by achieving 97% accuracy in malware detection and with 87% precision in properly identifying the malware family. The study's primary contributions include the following:

- The application of a computer vision model based on Transformer that uses self-supervised learning to detect Android malware. To the best of our knowledge, this is the first study to employ vision transformers and self-supervised learning for the goal of detecting malware on a sizable data set.
- A thorough analysis of how well malware detection software can be divided into malware type and malware family categories. Our findings show that, with a macro-F1 score of 0.497 for 47 types and 0.491 for 696 families of malware, respectively, self-supervised learning can successfully categorise malware.
- Advancing knowledge of synthetic images performance of self-supervised computer vision models, particularly the Vision Transformer architectures.

The remainder of the text is structured as follows: Basic information on malware detection and self-supervised learning is provided in Section II. The approach is then explained in Section III, and the evaluation and outcomes are covered in Section IV. With Section VI discussing significant ideas, Section V provides a summary of related work. The article's conclusion is presented at the end.

II. BACKGROUND

The key background information for self-supervised virus detection utilising byte plot grayscale images is condensed in this part. First, we'll go over the procedure for creating the image, which involves converting the malware executable into a byte plot grayscale image. Automatic visual pattern recognition that can be used in static malware analysis can be performed using these transformed binary executable images. The background information on picture classification utilising machine learning and self-supervision is then provided.

2.1 BYTEPLOT VISUALIZATION

The binary executable to image transformation was first described by Conti et al [12] to represent binary data objects as grayscale images, where the pixel value of the image corresponds to a byte in the binary file. Since the byte range is 0-255, the colour of the pixels is rendered as grayscale, where zero is black and 255 is white, the other values represent intermediate shades of Gray. This representation provides a visual analysis of binary data to distinguish structurally different regions of data, thus enabling a wide range of static analysis such as file type identification, primitive data type inference, fragment classification, and other tasks that required special reverse engineering for binary data. Nataraj et al proposed to transform malware samples into byte graph images with the observation that there are significant visual similarities in malware belonging to the same family [13]. The transformation consists of reading the binary file as a vector of 8-bit unsigned integers organized into a 2D array, where the width is defined based on empirical observations and the height varies depending on the size of the file.

Freitas et al. extended this work for Android applications by creating an image representation using a DEX (bytecode) file obtained from the Android APK [14]. The extracted DEX file was converted to a 1D array (instead of 2D) of 8-bit unsigned integers. The vector is then used in a 3-phase conversion process of 1) converting the 1D field to a 2D image representation 2) reducing the image to a standard size and 3) encoding the semantic information into RGB channels. Similar to the approach presented by Nataraj et al [13], the first step is to convert the data array into a grayscale image where the width is fixed and the height varies depending on the file size. The transformed image is then scaled down to 256 × 256 using a standard Lanczos filter from the Pillow library. Once the downscaled grayscale image is obtained, the final step is to encode semantic information that can further aid the discrimination of characteristic texture patterns. By colouring each byte according to its usage in the malware executable, the image has an added layer of semantic information on top of the raw bytecode. Gennissen et al present [15] an encoding that assigns each byte to a specific RGB color channel depending on its position in the DEX file structure. The observation is that the sections in the DEX file should be distinguished for malware detection because each type of malware uses each section differently and such patterns can be easily recognized in the image visualization. The encoded image can be decoded back to grayscale by combining each of the channels.

2.2 IMAGE CLASSIFICATION

Image classification is a paradigm of machine learning involving assigning categories to images. Most image classification problems are multi-class, single label classification problems involving multiple potential categories for the images but with only a single label or category being applied to an image. However, it is possible to apply multiple labels to some imagery. For example, images that contain multiple objects (i.e. a vehicle and a horse) or images that captures multiple features (i.e. an image of a bird can be labelled based on its physical features). These labels can be explored as categories either independently (as multiple single-label, multi-class problems) or jointly (a single multi-label, multi-class problem). A key issue when performing multiple single-label, multi-class classifications on the same dataset is the computational cost associated with training a different network for each problem. Transfer learning is a commonly used technique to alleviate some of the computational cost. In transfer learning, a previously trained neural network is reused to instantiate the weights of a new neural network that is then trained on the new problem. By reusing the weights of the previous task the ‘‘knowledge’’ from the previous task is transferred over and reused to some extent. Rezende et al [10] used transfer learning to train a DNN using an ImageNet dataset containing 1.28 million images of 1000 classes (which do not include malware images) and were able to achieve an accuracy of 92.97% on a dataset containing 10,136 malware bytes. images. This approach was one of the first to show that without feature engineering, using the raw pixel values of byte images, malware could be classified with high accuracy. The DNN was initially trained for a different direction on a different data set, yet it was able to outperform then-state-of-the-art models that used specialized reverse engineering. This line of work relies on learning from already established, annotated datasets despite the difference between the initial training dataset used for the weights and the malware dataset used for testing.

Making a huge data set of malware photos with annotations on various labels it can learn from is the difficult part. The generation of such a massive data set might not be possible, as it was for ImageNet, because the manual annotation of photos in ImageNet is frequently performed by people with no specialised knowledge and in a short amount of time. To characterise a malicious binary, however, an expert analyst must spend, on average, about 10 hours [16], [17]. It is not possible to produce a huge corpus of malware photos with expert annotated data that can be used for many activities, such as malware family, malware type, malware author categorization, due to the expensive effort. Programme binaries as a whole change over time due to the evolution of software, which includes new API modifications, novel code generation techniques, and improved compiler optimisations. Similar to how they develop malicious software, adversaries create harmful binaries that are undetectable and can adapt to these changes [18]. Maintaining an up-to-date data collection with expert-labelled data that can be used to train to detect new viruses with new attributes is difficult as a result. An expert-labelled dataset for malware detection can quickly become out of date because of the way malware evolves. This idea is known as "conceptual drift."

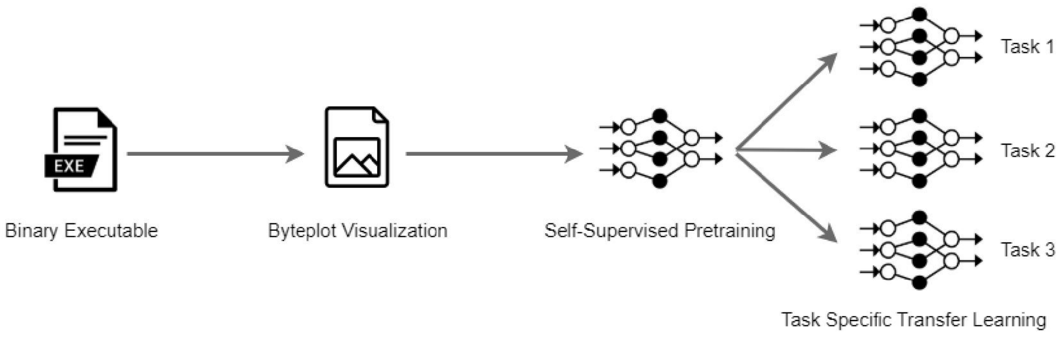


FIGURE 1. A high-level overview of SHERLOCK: self-supervised learning can be used to bootstrap different classification

2.3 Self-Supervised Learning

Self-supervised learning holds significant promise for improving representational learning when labeled data is scarce. A paradigm of representation learning known as "self-supervision" entails training a model under observation using unlabelled imagery for a pretext task. Pretext tasks are a class of tasks that can be automatically constructed from imagery in this context without human involvement. Self-supervised learning from photos aims to build semantically meaningful image representations through pretext tasks that don't call for any manual annotations. Jigsaw puzzle solving [19] for reconstructing an image after randomly permuting grid sections, relative position prediction [20] for determining the relative position and orientation of image patches with respect to one another, or image colorization [21] for removing and adding colour to an image are examples of pretext tasks. To complete these tasks and complete the pretext task, the neural network needs to learn a generic representation of the dataset. A model that has been trained to handle these pretext tasks learns representations that may be used to other interesting downstream tasks, such picture classification.

On a particular downstream objective, purely self-supervised algorithms may not produce visual representations that are as good as those produced by fully supervised techniques. Despite recent research showing positive outcomes, self-supervision alone is insufficient for practical usage. Self-supervision is combined with transfer learning or downstream job fine-tuning to ease this. This is comparable to how learning works in humans. Every new skill we learn involves an innate representation of the environment. The tagged imagery in the downstream task acts as a supervisory signal in the context of self-supervised learning, typically via propagating an error function like cross entropy, which aids in the self-supervised representation's ability to adapt to the downstream task. By converting a self-supervised learning model to a partially supervised learning model, Zhai et al. [22]

Additionally, self-supervision can enhance the robustness of a model in terms of its resistance to hostile examples, label corruptions, and typical input corruptions. Self-supervised learning can even outperform fully supervised techniques, as demonstrated by Hendrycks et al. [23], raising the question of whether or not big labelled data sets are still necessary in the age of self-supervised learning. However, in this study, we explicitly look into the advantages of using self-supervised learning in conjunction with a sizable labelled data set for malware detection and classification. We focus on the efficient processes that label-independent learning can offer for carrying out numerous single-label classifications on the same dataset.

Although self-supervised computer vision has drawn a lot of attention, the majority of recent research has been on natural imagery, or images taken from cameras of natural scenes. Humans absorb a lot of imagery, yet it often takes on a more abstract shape. Images are used to graphically express a variety of concepts. The underlying dynamics of such imagery may be very dissimilar from that of images of natural scenes. For instance, based on the size of the topic, an image of a natural scene carries semantic clues as to the proximity of an object to the sensor. More abstract imagery, however, clearly lacks or differs from such semantics. When compared to natural imagery domains, the transferability of knowledge representations between abstract imagery domains would also be further reduced for this reason, which indicates an increased benefit in creating methods like self-supervised learning for such domains. Therefore, there is a great deal of academic interest in knowing more about how self-supervised learning functions on such imagery. Prior research in this area has focused on sketch-based abstractions of images in certain cases [24, 25], while in other cases [26, 27] it has worked with map images or digital elevation outputs. To the best of our knowledge, no previous work has carried out self-supervised learning on images depicting binaries of programmes, which belong to this category of imagery since they are an abstract representation of the underlying software artefact.

III. METHODOLOGY

This section gives a general description of our model's architecture as well as our suggested approach for employing self-supervised learning to identify and categorise malware. Next, we discuss self-supervised learning to create malware images using a novel computer vision architecture. Finally, we describe how we improve the model for the problem of multi-class labelling in the context of malware classification by reusing the learned representation from the synthesis job.

3.1 OVERVIEW

We suggest formalising the Ove technique for ware identification by combining self-supervised learning with the vision transformer architecture. We interpret a malware image as a sequence of patches and analyse it using a conventional Transformer encoder as used in natural language processing, in contrast to existing systems that use image classification for malware detection. This interpretation, along with pre-training on substantial amounts of data, enables us to efficiently reuse a trained model for three separate classification tasks with little additional computational expense. An overview of our suggested technique, SHERLOCK, is shown in Figure 1. SHERLOCK's main goal is to create fully-supervised training samples for the underlying learning problem, which classification models can utilise to transfer learning bootstrap the training. Building virus images from a small number of features that capture malware semantics is a learning task. This contrasts with earlier malware detection research, which solely relies on supervised learning.

SHERLOCK is made up of two primary components that can effectively synthesise malware images using an optimised representation of the malware image and a classification component that can use the optimised representation in order to determine the proper label for a certain classification issue. A virus image is first divided into patches (tiny portions of the image) of size 16 16 pixels by SHERLOCK. This 1-dimensional vector is used to create a lower-dimensional linear embedding representation of the patches, which are then flattened or concatenated to create another 1-dimensional vector. Positional embedding is employed to retain the 2-dimensional positional relationship of the picture patches while transformers are used to model sequences. This improved embedding is utilised to fine-tune several classification models on malware picture data (such as malware detection, type-classification, and family-classification) and bootstrap the classifier component as a basic knowledge that captures the semantic encoding of a malware. As the largest publicly available cybersecurity image library (1.2 million photos) with different labels for each image corresponding to malware/dangerous (2 categories), malware type (47 categories), and malware family (696 categories), we use the MalNet dataset [14] in our research.

3.2 MASKED AUTO ENCODER

Vision Transformers (ViTs), which have recently become available, offer an alternative architecture for image processing that is both straightforward and scalable. A vision transformer starts with an image
divided into patches (such as 8 8 or 16 16 pixel-sized ones). As seen in Figure 2, these patches are then converted into an embedding using common transformer encoders. The model encodes a representation from an input image, which it then decodes to recreate the input image. As a result, an image serves as the model's input, and its output is a reconstruction of that same image using a smaller feature space. A totally accurate output is an identical replica of the input image. This idea can be used to other types of data modalities, such as Natural Language Processing (NLP), where masked language modelling as a self-supervising task has produced cutting-edge outcomes in the past [29].

As our pre-text task, masked auto encoding is used. In general, masked auto encoding involves deleting a portion of the data and predicting the removed section using the data that is still present. As a result, before self-supervised training, some of the image's pixels are hidden, and the neural network has to learn how to retrieve the hidden pixels using the visible pixels. Recent research [30] has demonstrated that masking a significant number of the pixels in natural images (like ImageNet) results in a difficult self-supervisory task that can produce helpful representations for subsequent tasks. Since there is not any prior research using masked autoencoding for abstract or synthetic imagery, we experiment with a high percentage of masked pixels (75% of the image), forcing the model to try to recover the image using only the 25% of the image that isn't masked [30].

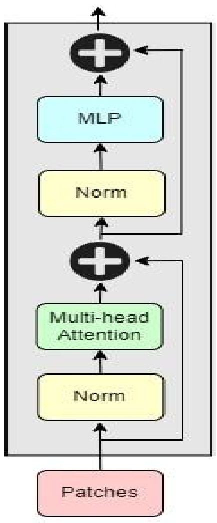


FIGURE 2. Illustration of a transformer encoder [28] that takes a sequence of patches as input and transforms into an encoding.

High percentages of masked pixels have been shown to work well for natural imagery. The outcomes of the picture synthesis for two different malware strains are shown in Figure 3. We employ Vision Transformers [28] in our analysis, which is significant as they can efficiently combine 2-Dpositional embeddings to pinpoint a patch's location inside an image. In convolutional neural networks, such incorporation is not straightforward.

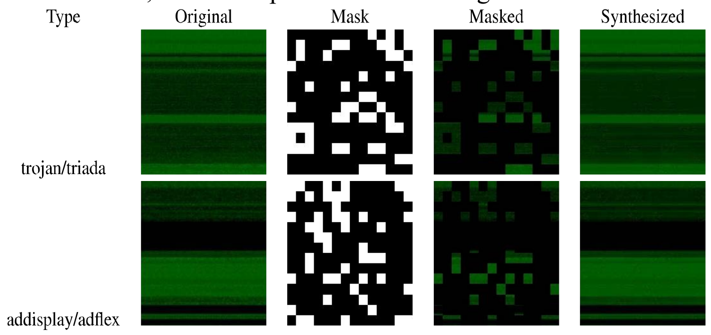


FIGURE 3. Illustration of the image synthesis using images of two malware types. Using an masked image which masks 75% of the image our learning model is able to synthesise images almost identical to the original image.

Figure 4 provides a high-level overview of the SHERLOCK auto-encoder, which employs self-supervised learning to provide an optimised embedding for more accurate reconstruction of masked pictures. By evenly selecting 25% of the image to keep, a mask is created. The created mask is then applied to the original image to create the masked input image. To create embedding for each patch, the input picture is flattened after being masked. Each patch is 16 16, resulting in a total of 14 14 = 196 patches. Each embedding is a vector of dimension 3 16 16 = 768. After shuffling the embeddings, only unmasked patches are passed through the encoder architecture to distinguish masked from unmasked patches. In order to create an embedding of size 768 for each patch, the unmasked embeddings are passed through 12 architecturally similar blocks of varying weights.The flattened masked patches, which are represented by a typical, learnable mask token signalling the presence of a masked patch, are coupled with the unmasked decoder embeddings. This tells the model to anticipate similar patches during the decoding phase. The decoder, which consists of 4 architecturally identical blocks with various weights, decodes the embeddings and mask tokens. To produce a final embedding (size 384) for each patch. A linear projection is used to translate this embedding to the number of output

pixels in a patch (31616 = 768). The original image and the synthesised image are compared at this point, and a pixel-wise loss is computed between them. This loss is then propagated back through the network.
The decoder is only utilised during self-supervised pre-training, allowing for decoder design freedom. Particularly, since less information and parameters would need to be stored within the decoder, simpler decoders compel a better representation in the other components of the architecture. In addition, the embedding dimension for the encoder is 768 pixels while that for the decoder is 384 pixels. The encoder contains 12 blocks while the decoder only has 4. Each pixel in a patch is first normalised by removing the patch's mean from it and dividing the result by the patch's standard deviation in order to calculate the reconstruction error. After normalization, the mean standard error over the pixels in the masked fields is calculated to determine the reconstruction loss.

3.3 CLASSIFICATION MODELS

The ability to reuse the same self-supervised representation for all 3 downstream tasks—malware classification (binary), malware type classification (47 categories), and malware family classification (696 categories)—while requiring little additional training is one of the main benefits of using self-supervised learning in this way. Simple supervised models, on the other hand, need to train independently from scratch, which would need a significant increase in computational processing. In this sense, self-supervised algorithms get more effective as more tasks are developed on the same dataset.

Algorithm 1 Malware Detection Algorithm
Execution:

```
Input: Binary Image, $I_{raw}$Output: Boolean value true/false
$I = resize(I_{raw})$
$L_{patches} = (I)$
$encoding = encode(L_{patches})$
$P = Compute\ Probability(encoding)$
if $p > 50\%$ then
    return ‘Malicious’’
else
    return ‘Benign’’
end
```

Algorithm 1 depicts the entire process of our proposed method for identifying/classifying an image as malware. The first step is to resize the image to 224224 pixels. When we offer our pre-trained models the feature encoding, they compute the probability P for the encoding that may contain detrimental features. If the P is substantial (i.e., p > 50%), the method returns Malware; otherwise, it returns Benign. The number of categories distinguishes malware detection workflows (which use binary classification) from type and family inference workflows (which use multi-class classification).

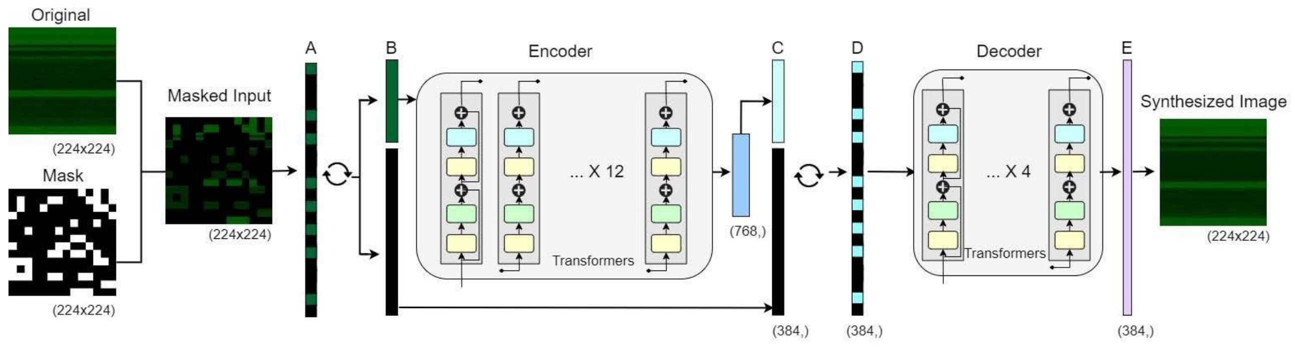


FIGURE 4. Overview of a masked autoencoder that takes masked images as input and learns to reconstruct the original image using supervised learning. Layer A is a linearization of the fields, which is then shuffled in Layer B to separate

the masked and unmasked pixels. Layer C shows the embedding of masked and unmasked pixels. Layer D represents the combination coding, which is the input layer that the decoder learns. Layer E is the final input for the reconstruction of the original image.

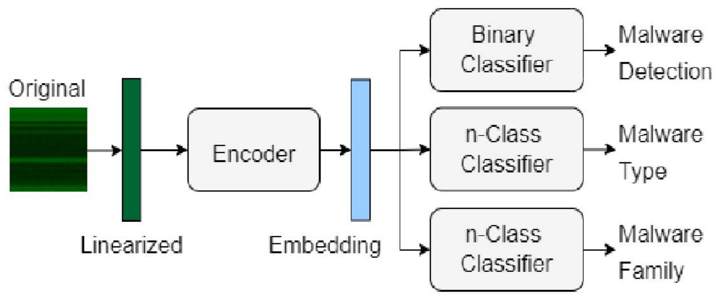


FIGURE 5. Overview of the classification models in SHERLOCK, which reuse the optimized embedding generated via self-supervised learning to bootstrap 3 different classification tasks.

ViT-Base architecture with patch size 16 (ViT-B/16) is used. The self-supervised encoder from the supervised learning step is used for each of the three subsequent classifications (as shown in Figure 5). As a result, the self-supervised model's features and weights are re-used to initialise the bulk of the supervised neural network, which is then trained to learn improved task-specific features. A linear layer is generated for each research that translates the encoder embedding (768 in size) to the number of output classes (2, 47, or 696, respectively). The resulting network is finetuned from beginning to end (backpropagating gradients and updating weights of every neural network parameter), allowing the network to learn more specialised features suitable for classifying imagery based on task-specific categories while building on the features learned during the self-supervised task of filling in masked patches. The malware class and malware family have a one-to-one relationship with the binary class.

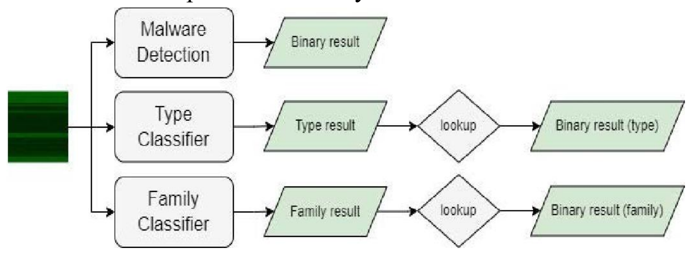


FIGURE 6. Using finer granularity tasks to predict coarse granularity tasks.

In Figure 6, we use the training from the finer-grained classification challenge to derive a coarse-grained classifier. However, because there is a many-to-many relationship between these two categories, a similar inference from malware-family to malware-type cannot be drawn in this data-set.

IV. EVALUATION

In this section, we test the performance of our proposed malware detection system SHERLOCK against the largest opensource dataset of Android malware [14]. First, we detail the experiment's structure, data set utilisation, and evaluation criteria. Following that, we compare the performance to state-of-the-art machine learning algorithms, display the malware classification confusion matrix, and examine the results.

4.1 EXPERIMENTAL SETUP

SHERLOCK, our proposed self-supervised learning technique, is tested on three tasks using the largest open-source Android malware dataset [14]. First, we evaluate the pre-training assignment's performance in creating malware images. Second, we evaluate the classification task's ability to correctly identify the malware label. Third, we compare our classification performance to that of existing cutting-edge technologies, and fourth, we investigate the sensitivity of the synthesis process to overall classification performance. For comparison with state-of-the-art deep learning architectures we consider; ResNet, DensNet and MobileNetV2 with different configurations. All our experiments were carried out on a single Spartan Cluster [31] node with 24 cores (Intel Xeon CPU E5-2650 v4 @ 2.20GHz) and 4 P100 GPUs with 12GB of GPU RAM.

For our experiments we use MalNet dataset [14] which: contains 1,262,024 malware pictures collected from real-world Android apps in AndroZoo [11] contains the largest diverse open-source dataset comprising of 47 malware types and 696 malware families including benign applications

Table 1 details the number of pictures and families found in each malware kind. Notably, the class distribution is significantly lopsided across image type and family, which is a common feature of many real-world datasets in which a few classes include most of the samples. We give Accuracy, macro-Precision, macro-Recall, and macro F1 scores for all tasks and compare them to past work using the identical train/test splits from this dataset.

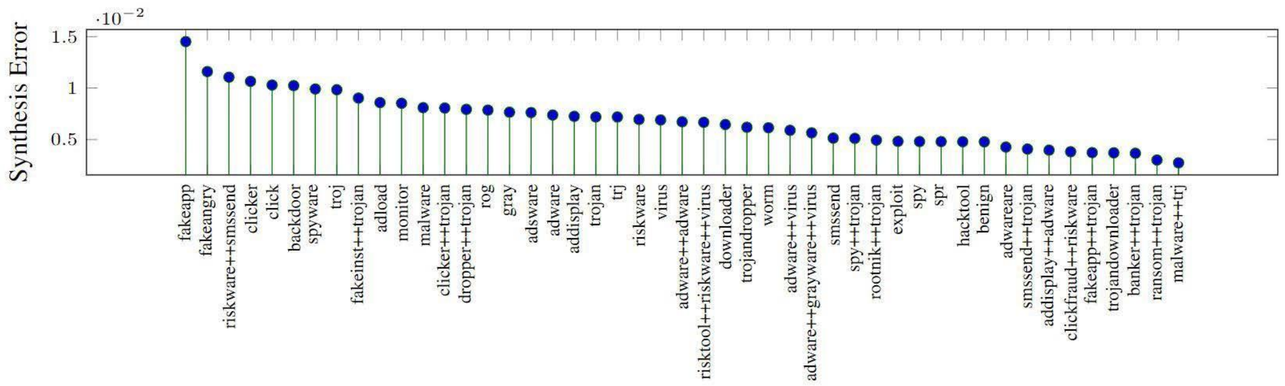


FIGURE 7. Quality of the self-supervised image synthesis process where the reconstruction error is aggregated across malware type.

TP Precision = TP +FP/ TP (1)

Recall = TP + FN 2 × Precision × Recall (2)

F1 = Precision + Recall (3)

True Positive (TP) denotes the number of samples from class c that are correctly identified as class c. False Positive (FP) is the number of samples that do not belong to class c but are incorrectly classified as such. For the classifications of malware, type and family, we report the

Macro-Precision (MP), Macro-Recall (MR) and Macro-F1 score.

^{Pc} *Precision*$_c$

Macro − Precision $^=$ (4)

Macro − Recall =(5)

*C*2 × *MP* × *MR*

*Macro − F*1 = ———— (6)

MP + *MR*

The macro criteria are calculated by taking a class-wise average of the metrics, as shown in Equations 4, 5, and 6. C denotes the number of classes in each multi-class classification task in this scenario. Visualisations will also be utilised to better understand the performance of our models.

1 − 0.999

Wc= 1 − 0.999$_{nc}$ (7)

For the training procedure, we used simple cross entropy loss with class re-balancing. Classes in the dataset with fewer training instances are weighted more heavily in the loss function. As shown in Equation 7, the weights provided in [14] are utilised, where nc is the number of photographs in classc. These weights cause a significantly higher loss to be attributed to mis categorizations in samples originating from under-represented groups. As a result, the neural network will prefer learning a representation that can more successfully identify such underrepresented classes over a typical (unweighted) cross-entropy loss.

4.2 ACCURACY OF MALWARE IMAGE SYNTHESIS

Because this is a direct evaluation of the self-supervision job, it can be interpreted as a preliminary assessment of the representation's and decoder's ability to capture the semantic features associated with the binary of the considered categories.

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.301

Volume 3, Issue 8, April 2023

We run each image through for each malware-type class, generating 10 random masks for 75% of the image and generating 10 masked input images with only 25% of the original image's pixels. For each masked input image, we construct an image that is comparable to the original image using the trained self-supervised model. We average the absolute error for each pixel in comparison to the original image for each synthesised image.

TABLE 1. The number of images and families in each type of malware in MalNet [14]

Type	Img.	Fam.
Adware	884K	250
Trojan	179K	441
Benign	79K	1
Riskware	32K	107
Addisplay	17K	38
Spr	14K	46
Spyware	7K	19
Exploit	6K	13
Downloader	5K	7
Smssend+Trojan	4K	25
Troj	3K	36
Smssend	3K	12
Clicker+Trojan	3K	3
Adsware	3K	16
Malware	3K	19
Adware+Adware	3K	2
Rog	2K	22
Spy	2K	7
Monitor	1K	5
Ransom+Trojan	1K	7
Banker+Trojan	1K	6
Trj	940	18
Gray	922	10
Adware+Grayware+Virus	835	4
Fakeinst+Trojan	718	10
Malware+Trj	609	1
Backdoor	602	10
Dropper+Trojan	592	8
Trojandownloader	568	7
Hacktool	542	7
Fakeapp	425	5
Clickfraud+Riskware	369	5
Adload	333	4
Addisplay+Adware	294	1
Adware+Virus	274	9
Clicker	265	5
Fakeapp+Trojan	256	1
Riskware+Smssend	247	7
Rootnik+Trojan	223	5
Worm	220	7
Fakeangry	211	2
Virus	191	3
Trojandropper	178	4
Adwareare	152	3
Risktool+Riskware+Virus	152	3
Spy+Trojan	119	5
Click	113	1

The fault over all 10 randomly produced masked input photos. Finally, for each class, the error is averaged across all such photos. Figure 7 depicts the pooled reconstruction error across malware types.

When the reconstruction error is averaged among malware categories, it ranges from 0.27% to 1.45%, with a global average of 0.68%. The scale of the pixel values in the image (ranging between [0, 255] or [0.0, 1.0] per channel, as we

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

IJARSCT

Impact Factor: 7.301

Volume 3, Issue 8, April 2023

use in this work) and the percentage of masked pixels in the image—as indicated in the issue footnote—determine the extent of the inaccuracy. The same average was used for all categories a strategy is used the final averages we provide are comparable across categories since the masked pixel percentage is fixed for each image (and they differ from other comparable evaluations by only a scalar component). The absolute inaccuracy between the original image's masked pixels and the synthesised image would be needed, though, if the masked percentage varied between photos.

Malware Image Synthesis Accuracy: ViT-Base a
ture we used in SHERLOCK was able to accurate
thesise malware images of 75% masking with an
reconstruction error of 0.68%.

4.3 MALWARE CLASSIFICATION

We put our proposed self-supervised model SHERLOCK to the test in three different domains to see how well it can identify the label for each malware image. For each category, we train a different classifier to which the semantic learning from the supervised-learning is transferred, as shown in Figure 5.

Malware kinds and families were successfully categorised by SHERLOCK spanning 47 and 696 classes, respectively. Modern deep learning models could not compare to our model's performance, which correctly classified both classes with an accuracy of 83.7% for malware-type and 80.2% for malware-family. Additionally, SHERLOCK had the highest macro-F1 score, macro-precision, and macro-recall in both categories, demonstrating the benefits of using a self-supervised learning model over current supervised learning model. One of the contrasts between the compared models and SHERLOCK and the learning transfer resource. Although it performs significantly worse than the trained model, earlier work [14] used ImageNet [32] transfer learning to pretrain a model and then fine tune to the training data. because the semantics of the various ImageNet images differ, from scratch and MalNet images.

For the binary classification problem, our model reports the best performance in precision but the worst performance in recall, resulting in inferior overall performance in macro-F1 score. Cross-comparison of our model's performance across all 3 tasks shows that the semantics improve the precision of the findings.

Model	Binary			Type			Family		
	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall
ViT-B (SHERLOCK)	.854	**.920**	.810	**.497**	**.628**	**.447**	**.491**	**.568**	**.461**
ResNet18	.862	.893	.837	.467	.556	.424	.454	.538	.423
ResNet50	.854	.907	.814	.479	.566	.441	.468	.541	.443
DenseNet121	.864	.900	.834	.471	.558	.428	.461	.529	.438
Densenet169	**.864**	.890	**.841**	.477	.573	.433	.462	.545	.434
MobileNetV2(x.5)	.857	.894	.827	.460	.547	.424	.451	.528	.423
MobileNetV2(x1)	.854	.889	.825	.452	.527	.419	.438	.532	.405

TABLE 2. Results on malware classification on 3 classes against 3 popular architectures ResNet, DenseNet and MobileNetV2—on macro-F1, macro-precision, and macro-recall. Results for compared architectures are referenced from MalNet dataset [14].

Model	F1 Score	Precision
SHERLOCK	.854	**.920**
SHERLOCK (type)	.876	.891
SHERLOCK (family)	**.878**	.867
ResNet18	.862	.893
ResNet50	.854	.907
DensetNet121	.864	.900
DenseNet169	.864	.890
MobileNetV2(x.5)	.857	.894
MobileNetV2(x1)	.854	.889

TABLE 3. Efficacy of malware detection against 3 popular architectures ResNet, DenseNet and MobileNetV2—on its accuracy, macro-F1, macro-precision, and macro-recall.

learned by self-supervised learning, and memory can be enhanced with further amplification. The imbalance effect is diffused over numerous classes and improves overall recollection when more classes are added to the tasks for classifying malware-type and malware-family. This suggests that the pre-training task, which teaches semantic information for the subsequent categorization test, had a significant impact. Our model learns the necessary semantic features necessary for the synthesis because the pre-training job is designed to optimise picture synthesis. Additionally, the semantic learning is biased towards images of malware as a result of the class imbalance issue in our dataset, leading to a higher rate of misclassifications as malware.

Malware Classification Accuracy: SHERLOC able to outperform existing state-of-the-art deep l models for malware-type and malware-family v macro F1-Score of .497 and .491 respectively.

4.4 MALWARE DETECTION

We assess SHERLOCK's overall performance to correctly identify a malicious application using inference based on the categorization label it produces, as shown in Figure 6. A single 4 GPU node with 24 cores (Intel Xeon CPU E5-2650 v4 @2.20GHz) and 4 P100 GPU with 12GB of GPU RAM each on Spartan takes an average of 0.479 seconds to generate an image from the corresponding binary, while Sherlock takes an average of 0.003 seconds to infer a label [31]. In Table 3, where the findings for state-of-the-art are taken from previous work [14], a comparison of our inference result with the most advanced deep learning models is shown. A pre-trained model with finer granularity can learn distinct characteristics to discriminate an image between the benign class and malware class, as evidenced by the fact that inference utilising malware-family classifier has the overall greatest performance with a macro F1-score of 0.878.

We present the results of our inference models in Figure 9 for a more thorough study. A model pre-trained with self-supervised learning can provide state-of-the-art performance for applications like malware detection where accuracy is crucial. The final model is trained on a coarse-grained task like two-class labelling. The recall and total F1-score will be improved more by using the same pre-trained model with the final model trained on a finer-grained task, such as recognising the malware type/malware-family (N classes). We note that while the precision decreases as the number of classes in the final model increases, recall improves. Additionally, with contrasting variances in their precision and recall values, malware-type-based inference and malware-family-based inference are both able to obtain similar macro F1-scores. Malware-type based inference has a higher F1-score, but it also has a lower Area Under Curve (AUC), which means it will struggle with varying thresholds. However, malware-family based inference model has the highest AUC score as well as for the macro F1-score.

Malware Detection Efficacy: SHERLOCK was al correctly detect a malware with an accuracy of 97' an highly imbalanced dataset while maintaining 8 precision and 89% recall.

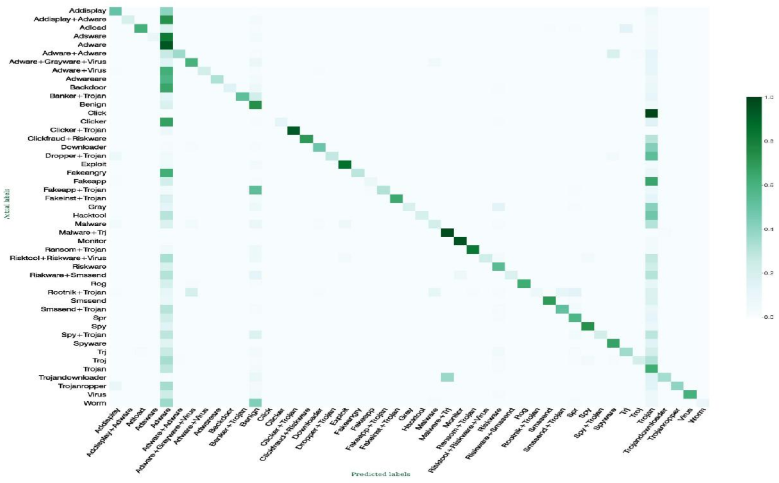


FIGURE 8. Confusion Matrix for malware type classification, which visualize the performance in correctly identifying the actual class label. The darker the shade the stronger the classifier performance and lighter-shade indicates poor performance. Our model results depict more darker shades for the diagonal entries while lighter-shade for non-diagonal

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
Impact Factor: 7.301
Volume 3, Issue 8, April 2023

entries indicating good performance.

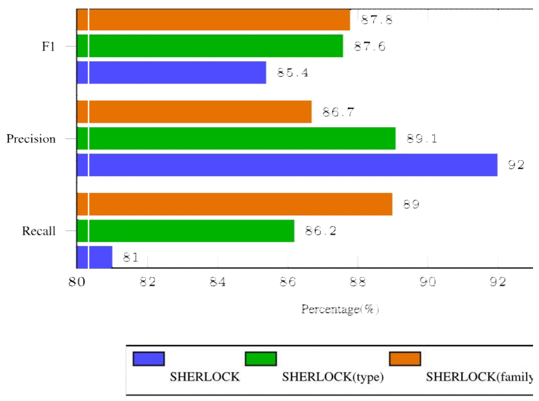


FIGURE 9. In-depth analysis of performance metrics SHERLOCK with 3 binary classification models trained at different granularities.

V. RELATED WORK

5.1 MALWARE DETECTION AND CLASSIFICATION

Due to the growing complexity of malware and the widespread use of computing systems, the identification and categorization of malware has become a critical issue. Figure 11 illustrates how Abusitta et al.'s [33] classification of malware detection and classification techniques is dependent on the features and methods employed.

While dynamic feature extraction methods use a sandbox environment to execute the programme and gather the memory image or the behaviours of the programme execution to extract features, static feature extraction methods extract features from executable files [34]. Printable strings [35, 36, 37], byte code [38, 39, 40, 41], assembly code [42, 43, 44], API/DLL system calls [44, 45, 46], control flow charts [47, 48, 49] and feature-level functions [50] are several examples of characteristics reported in the literature. The two main types of algorithms used for detection and classification are those based on signatures and those based on artificial intelligence. Current antivirus companies de facto use signature-based methods [51]. These signatures were made by humans, and malware detectors compare programmes with signatures to find or categorise malware.

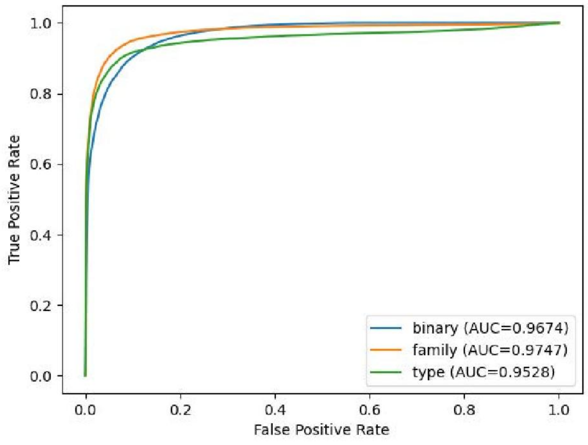


FIGURE 10: Receiver operating characteristic (ROC) curve for malware detection with 3 binary classification models trained at different granularities.

malware is [52, 53]. One could argue that this technology is reliant on manual signature generation and can only detect known malware. AI-based malware detection and classification techniques are increasingly being used, and they may be broadly divided into supervised [54, [55], unsupervised [56, [57], and [58] and semi-supervised [59, 60, and [61] methods. Unsupervised methods identify patterns from unlabelled data, whereas supervised methods use a model to learn characteristics from a labelled dataset to detect malware.While conventional techniques like Naive Bayes, Decision Trees, K-Nearest Neighbours, and SVM have long been in use. A comparison of these traditional techniques

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal

Impact Factor: 7.301

Volume 3, Issue 8, April 2023

revealed that SVM was effective at detecting malware [62]. Liu et al. examine at the existing ML approaches utilised for malware detection in their survey on android-specific malware detection [63]. Recently, approaches based on ReLU, LSTM, and CNN have drawn greater interest. Yadav et al. [64] studied a deep learning-based detection approach that utilises picture data. These machine learning-based malware detection techniques are increasingly being studied because of their increased performance over conventional methods and their capacity to identify previously undiscovered malware [65].

Our strategy can be categorised as a self-supervised learning-based technique that makes advantage of the application's static properties. This is the first self supervised based learning approach that has been investigated using the largest openly-available dataset for Android malware, in contrast to previous work.

5.2 IMAGE CLASSIFICATION

The term "synthetic imagery" refers to imagery that is produced to depict code or programme binaries in general. Such imagery is different from natural imagery, which includes photographs of actual real-world scenes taken with a camera and is the common sort of data used in computer vision datasets like ImageNet [32]. Compared to natural photography, the performance of self-supervised algorithms on complicated synthetic images is still being studied. The learning of representations from sketches has been investigated in some past research in this field [24],
[25], aerial photographs [27], photographs of artificial scenes [66], and images from Google maps [26]. The performance of self-supervised algorithms on synthetic imagery, which has various visual properties from natural imagery, can be better understood by analysis in such domains.

VI. PERSPECTIVES

The capacity to initialise a single generic representation in a task-independent way is one of the fundamental benefits of adopting self-supervision in this imaging domain (malware imagery). Then, this representation can be tailored to carry out certain analysis (including malware detection, malware type categorization, malware family categorization, and perhaps other functions as well). In essence, the representation may be readily customised for any unique set of labels connected to the imagery. For instance, future research could look at labels connected to malware authors and try to find distinctive "signatures" related to authors from the imagery.

Interestingly, the classification of malware images into benign and harmful has performed better when coarser labels are used. Lookups can be used to transform malware family and type predictions into predictions for malware detection. When all models were trained with the same number of images and the same analytical parameters, we discovered that the models for family and type prediction performed better than the malware detection prediction once they were converted into the corresponding malware detection category ("malicious" or "benign"). We can infer that the coarser labels have resulted in a better characterisation of the images for the malware detection task since the sole distinction between the studies was the granularity of the labels (696 for family, 47 for type, and 2 for malware detection). While both models, when trained on more specific labels, significantly outperform the malware detection model in terms of F1 score, their performance is comparable to one another. This implies that the improvement brought about by more precise labelling has diminishing benefits, which may be an intriguing area for future research.

The cause of this improvement can be inferred with more thought. The model is compelled to develop numerous decision boundaries to distinguish cases from various categories from one another on a given dataset with fine-grained labels. As each category may be next to every other in the feature space, the number of probable needed borders will increase quadratically as the number of categories (n*(n-1)/2) increases. Because the total number of photos is fixed across all analyses, this results in a more fine-grained clustering of images with a greater number of clusters having fewer images in the feature space. Additionally, this offers more justification for the little gain (in terms of malware detection outcomes) brought about by going from 47 to 696 categories. In contrast to type prediction, which has 1081 potential borders (47*46/2), and family prediction, which has 676*675/2 = 228150, the malware detection task just has one decision boundary (benign vs malicious). According to the observed results, the extra granularity of the family labels does not further improve outcomes since the amount of detail required for the binary classification required in malware detection is captured by type labels with adequate granularity. The effect of label granularity is also visible in the ROC curves for malware detection, with the more finely labelled family prediction model outperforming the other more coarsely labelled models in terms of performance. Future research could investigate this aspect further by examining the additional effects of using a baseline task that is more complicated (with more than two categories). We are unable to investigate this further at this time since the type and family designations do not map one to one. Convolutional neural networks have previously been used to examine the effects of label granularity on classification [67]. The results of our research further generalise these conclusions to the architecture of the Vision Transformer.

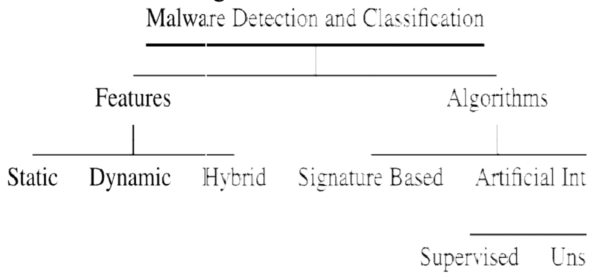
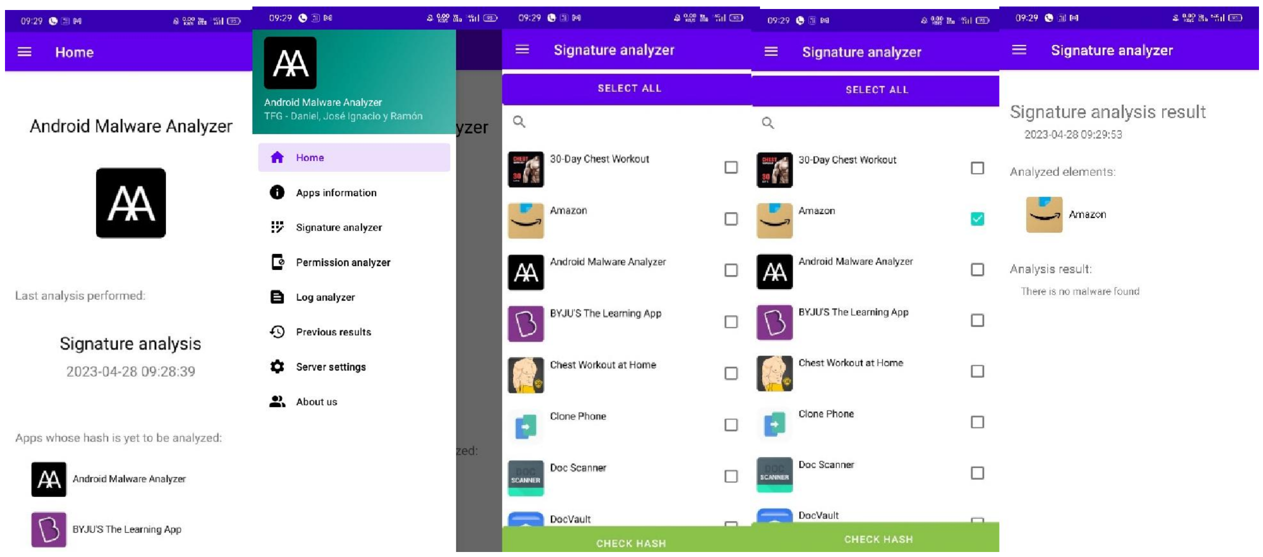


FIGURE 11. Categorization of malware detection and classification [33].

VII. RESULT



Android Malware Analyzer

Last analysis performed:

Signature analysis
2023-04-28 09:28:39

Apps whose hash is yet to be analyzed:

Android Malware Analyzer
BYJU'S The Learning App

Android Malware Analyzer
TFG - Daniel, José Ignacio y Ramón

30-Day Chest Workout
Amazon
Android Malware Analyzer
BYJU'S The Learning App
Chest Workout at Home
Clone Phone
Doc Scanner
DocVault

Signature analysis result
2023-04-28 09:29:53

Analyzed elements:
Amazon

Analysis result:
There is no malware found

VIII. CONCLUSION

In this research, we introduce SHERLOCK, a computer vision model based on transformers that makes use of self-supervised learning to identify Android malware. SHERLOCK is a system based on self-supervised learning for malware classification. A massive data collection of 1.2 million Android apps, including 47 different malware kinds and 696 malware families, is used to analyse SHERLOCK. With a macro-F1 score of.497 for malware kind and.491 for malware family, SHERLOCK was also capable of classifying malware correctly. In particular, the Vision Transformer architectures are used in this work to show the effectiveness of self-supervised computer vision models in the usage of malware categorization. Through this work, we improve malware detection and classification based on static analysis, which we believe can be further improved by integrating additional features derived from dynamic analysis. We have open-sourced our model to the community through GitHub so that the research community can more effectively duplicate and recreate the findings in our studies and to continue the work further: https://github.com/sachith500/Sherlock.

REFERENCES

[1]. A. M. Al-Saffar, H. Tao, and M. A. Talab, ''Review of deep convolution neural network in image classification,'' in Proc. Int. Conf. Radar, Antenna, Microw., Electron., Telecommun. (ICRAMET), Oct. 2017, pp. 26–31.

[2]. Subramanya, V. Pillai, and H. Pirsiavash, ''Fooling network interpretation in image classification,'' in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 2020–2029.

[3]. Y. Dong, Q.-A. Fu, X. Yang, T. Pang, H. Su, Z. Xiao, and J. Zhu, ''Benchmarking adversarial robustness on image classification,'' in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 321–331.

[4]. S. Qiu, Q. Liu, S. Zhou, and C. Wu, ''Review of artificial intelligence adversarial attack and defense technologies,'' Appl. Sci., vol. 9, no. 5, p. 909, 2019.

[5]. Z. Zhu and T. Dumitraş, ''FeatureSmith: Automatically engineering features for malware detection by mining the security literature,'' in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 767–778.

[6]. S.-J. Lee, H.-Y. Shim, Y.-R. Lee, T.-R. Park, S.-H. Park, and I.-G. Lee, ''Study on systematic ransomware detection techniques,'' in Proc. 23rd Int. Conf. Adv. Commun. Technol. (ICACT), Feb. 2021, pp. 297–301.

[7]. M. A. Omer, S. R. M. Zeebaree, M. A. M. Sadeeq, B. W. Salim, S. X. Mohsin, Z. N. Rashid, and L. M. Haji, ''Efficiency of malware detection in Android system: A survey,'' Asian J. Res. Comput. Sci., vol. 7, no. 4, pp. 59–69, Apr. 2021.

[8]. X. Deng and J. Mirkovic, ''Polymorphic malware behavior through network trace analysis,'' in Proc. 14th Int. Conf. Commun. Syst. Netw. (COMSNETS), Jan. 2022, pp. 138–146.

[9]. P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, ''Android security: A survey of issues, malware penetration, and defenses,'' IEEE Commun. Surveys Tuts., vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015.

[10]. E.Rezende,G.Ruppert,T.Carvalho,F.Ramos,andP.deGeus,''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.

[11]. K. Allix, T. F. Bissyandé, J. Klein, and Y. L. Traon, ''Androzoo: Collecting millions of Android apps for the research community,'' in Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories (MSR), May 2016, pp. 468–471.

[12]. G. Conti, E. Dean, M. Sinda, and B. Sangster, ''Visual reverse engineering of binary and data files,'' in Proc. Int. Workshop Vis. Comput. Secur. Berlin, Germany: Springer-Verlag, 2008, pp. 1–17.

[13]. L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, ''Malware images: Visualization and automatic classification,'' in Proc. 8th Int. Symp. Vis. Cyber Secur. (VizSec). New York, NY, USA: Association for Computing Machinery, 2011, pp. 1–7, doi: 10.1145/2016904.2016908.

[14]. S. Freitas, R. Duggal, and D. H. Chau, ''MalNet: A large-scale cybersecurity image database of malicious software,'' CoRR, vol. abs/2102.01072, pp. 1–7, Jan. 2021.

[15]. J. Gennissen, L. Cavallaro, V. Moonsamy, and L. Batina, ''Gamut: Sifting through images to detect Android malware,'' Bachelor thesis, Inst. Comput. Inf. Sci., Roy. Holloway Univ., London, U.K., 2017.

[16]. A. Mohaisen, A. G. West, A. Mankin, and O. Alrawi, ''Chatter: Classifying malware families using system event ordering,'' in Proc. IEEE Conf. Commun. Netw. Secur., Oct. 2014, pp. 283–291.

[17]. D. Votipka, S. Rabin, K. Micinski, J. S. Foster, and M. L. Mazurek, ''An observational investigation of reverse engineers' processes,'' in Proc. 29th USENIX Secur. Symp. (USENIX Security), 2020, pp. 1875–1892.

[18]. A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar, ''Better malware ground truth: Techniques for weighting anti-virus vendor labels,'' in Proc. 8th ACM Workshop Artif. Intell. Secur., Oct. 2015, pp. 45–56.

[19]. M. Noroozi and P. Favaro, ''Unsupervised learning of visual representations by solving jigsaw puzzles,'' in Computer Vision—ECCV 2016, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Amsterdam, The Netherlands: Springer, 2016, pp. 69–84.

[20]. Doersch, A. Gupta, and A. A. Efros, ''Unsupervised visual representation learning by context prediction,'' in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Santiago, Chile, Dec. 2015, pp. 1422–1430.

[21]. R. Zhang, P. Isola, and A. A. Efros, ''Colorful image colorization,'' in Computer Vision—ECCV 2016, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham, Switzerland: Springer, 2016, pp. 649–666.

[22]. X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer, ''S4L: Self-supervised semi-supervised learning,'' in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Oct. 2019, pp. 1476–1485.

[23]. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, ''Using selfsupervised learning can improve model robustness and uncertainty,'' in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019, pp. 1–12.

[24]. Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, ''Unsupervised data augmentation for consistency training,'' 2019, arXiv:1904.12848.

[25]. A. K. Bhunia, P. N. Chowdhury, Y. Yang, T. M. Hospedales, T. Xiang, and Y.-Z. Song, ''Vectorization and rasterization: Self-supervised learning for sketch and handwriting,'' in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2021, pp. 5672–5681.

[26]. S.Seneviratne,K.A.Nice,J.S.Wijnands,M.Stevenson,andJ. Thompson, ''Self-supervision. Remote sensing and abstraction: Representation learning across 3 million locations,'' in Proc. Digit. Image Comput., Techn. Appl. (DICTA), Nov. 2021, pp. 01–08.

[27]. S. Seneviratne, ''Contrastive representation learning for natural world imagery: Habitat prediction for 30,000 species,'' in Proc. CLEF Work. Notes, 2021, pp. 1639–1648.

[28]. 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 16 × 16 words: Transformers for image recognition at scale,'' 2020, arXiv:2010.11929.

[29]. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ''BERT: Pre-training of deep bidirectional transformers for language understanding,'' in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., vol. 1. Minneapolis, MN, USA: Association for Computational Linguistics, Jun. 2019, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[30]. K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, ''Masked autoencoders are scalable vision learners,'' 2021, arXiv:2111.06377.

[31]. G. Lev Lafayette, L. Vu, and B. Meade, ''Spartan performance and flexibility: An HPC-cloud chimera,'' in Proc. OpenStack Summit, vol. 10, Barcelona, Spain, 2016, p. 49.

[32]. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, ''ImageNet: A large-scale hierarchical image database,'' in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2009, pp. 248–255.

[33]. A. Abusitta, 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.

[34]. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, ''A comparison of static, dynamic, and hybrid analysis for malware detection,'' J. Comput. Virol. Hacking Techn., vol. 13, no. 1, pp. 1–12, 2017.

[35]. W. Huang and J. W. Stokes, ''MtNet: A multi-task neural network for dynamic malware classification,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Berlin, Germany: Springer-Verlag, 2016, pp. 399–418.

[36]. G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, ''Large-scale malware classification using random projections and neural networks,'' in Proc. IEEE Int. Conf. Acoust., Speech Signal Process., May 2013, pp. 3422–3426.

[37]. Y.-T. Lee, T. Ban, T.-L. Wan, S.-M. Cheng, R. Isawa, T. Takahashi, and D. Inoue, ''Cross platform IoT-malware family classification based on printable strings,'' in Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom), Dec. 2020, pp. 775–784.

[38]. Anderson, C. Storlie, and T. Lane, ''Improving malware classification: Bridging the static/dynamic gap,'' in Proc. 5th ACM Workshop Secur. Artif. Intell., 2012, pp. 3–14.

[39]. J. Saxe and K. Berlin, ''Deep neural network based malware detection using two dimensional binary program features,'' in Proc. 10th Int. Conf. Malicious Unwanted Softw. (MALWARE), Oct. 2015, pp. 11–20.

[40]. M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, ''Static malware detection and attribution in Android byte-code through an end-to-end deep system,'' Future Gener. Comput. Syst., vol. 102, pp. 112–126, Jan. 2020.

[41]. N. Daoudi, J. Samhi, A. K. Kabore, K. Allix, T. F. Bissyandé, and J. Klein, ''DEXRAY: A simple, yet effective deep learning approach to Android malware detection based on image representation of bytecode,'' in Proc. Int. Workshop Deployable Mach. Learn. Secur. Defense. Cham, Switzerland: Springer, 2021, pp. 81–106.

[42]. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, ''Graph-based malware detection using dynamic analysis,'' J. Comput. Virol., vol. 7, no. 4, pp. 247–258, 2011.

[43]. J. Dai, R. K. Guha, and J. Lee, ''Efficient virus detection using dynamic instruction sequences,'' J. Comput., vol. 4, no. 5, pp. 405–414, 2009.

[44]. M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, ''Malware detection using assembly and API call sequences,'' J. Comput. Virol., vol. 7, no. 2, pp. 107–119, 2011.

[45]. A. Sami, B. Yadegari, N. Peiravian, S. Hashemi, and A. Hamze, ''Malware detection based on mining API calls,'' in Proc. ACM Symp. Appl. Comput., 2010, pp. 1020–1025.

[46]. Z. Salehi, A. Sami, and M. Ghiasi, ''Using feature generation from API calls for malware detection,'' Comput. Fraud Secur., vol. 2014, no. 9, pp. 9–18, 2014.

[47]. Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, ''A combination method for Android malware detection based on control flow graphs and machine learning algorithms,'' IEEE Access, vol. 7, pp. 21235–21245, 2019.

[48]. J. Yan, G. Yan, and D. Jin, ''Classifying malware represented as control flow graphs using deep graph convolutional neural network,'' in Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN), Jun. 2019, pp. 52–63.

[49]. Bruschi, L. Martignoni, and M. Monga, ''Detecting self-mutating malware using control-flow graph matching,'' in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Berlin, Germany: SpringerVerlag, 2006, pp. 129–143.

[50]. M. Cai, Y. Jiang, C. Gao, H. Li, and W. Yuan, ''Learning features from enhanced function call graphs for Android malware detection,'' Neurocomputing, vol. 423, pp. 301–307, Jan. 2021.

[51]. M. Al-Asli and T. A. Ghaleb, ''Review of signature-based techniques in antivirus products,'' in Proc. Int. Conf. Comput. Inf. Sci. (ICCIS), Apr. 2019, pp. 1–6.

[52]. M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, ''Data mining methods for detection of new malicious executables,'' in Proc. IEEE Symp. Secur. Privacy (SP), May 2001, pp. 38–49.

[53]. M. Christodorescu and S. Jha, ''Static analysis of executables to detect malicious patterns,'' in Proc. 12th USENIX Secur. Symp. (USENIX Security), 2003, pp. 169–186.

[54]. J. Singh and J. Singh, ''Assessment of supervised machine learning algorithms using dynamic API calls for malware detection,'' Int. J. Comput. Appl., vol. 44, no. 3, pp. 270–277, Mar. 2022.

[55]. Y. Ki, E. Kim, and H. K. Kim, ''A novel approach to detect malware based on API call sequence analysis,'' Int. J. Distrib. Sensor Netw., vol. 11, no. 6, Jun. 2015, Art. no. 659101.

[56]. A. Tang, S. Sethumadhavan, and S. J. Stolfo, ''Unsupervised anomalybased malware detection using hardware features,'' in Proc. Int. Workshop Recent Adv. Intrusion Detection. Cham, Switzerland: Springer, 2014, pp. 109–129.

[57]. Z. Liu, R. Wang, N. Japkowicz, D. Tang, W. Zhang, and J. Zhao, ''Research on unsupervised feature learning for Android malware detection based on restricted Boltzmann machines,'' Future Gener. Comput. Syst., vol. 120, pp. 91–108, Jul. 2021.

[58]. M. Fan, X. Luo, J. Liu, M. Wang, C. Nong, Q. Zheng, and T. Liu, ''Graph embedding based familial analysis of Android malware using unsupervised learning,'' in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE), May 2019, pp. 771–782.

[59]. Santos, J. Nieves, and P. G. Bringas, ''Semi-supervised learning for unknown malware detection,'' in Proc. Int. Symp. Distrib. Comput. Artif. Intell. Springer, 2011, pp. 415–422.

[60]. A. Mahindru and A. Sangal, ‘‘Feature-based semi-supervised learning to detect malware from Android,’’ in Automated Software Engineering: A Deep Learning-Based Approach. Springer, 2020, pp. 93–118.
[61]. X.Gao,C.Hu,C.Shan,B.Liu,Z.Niu,andH.Xie,‘‘Malwareclassification for the cloud via semi-supervised transfer learning,’’ J. Inf. Secur. Appl., vol. 55, Dec. 2020, Art. no. 102661.
[62]. A. Souri and R. Hosseini, ‘‘A state-of-the-art survey of malware detection approaches using data mining techniques,’’ Hum.-Centric Comput. Inf. Sci., vol. 8, no. 1, pp. 1–22, Dec. 2018.
[63]. K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, ‘‘A review of Android malware detection approaches based on machine learning,’’ IEEE Access, vol. 8, pp. 124579–124607, 2020.
[64]. P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, ‘‘A twostage deep learning framework for image-based Android malware detection and variant classification,’’ Comput. Intell., May 2022.
[65]. M. T. Ahvanooey, Q. Li, M. Rabbani, and A. R. Rajput, ‘‘A survey on smartphones security: Software vulnerabilities, malware, and attacks,’’ 2020, arXiv:2001.09406.
[66]. Z. Ren and Y. J. Lee, ‘‘Cross-domain self-supervised multi-task feature learning using synthetic imagery,’’ in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 762–771.
[67]. Z. Chen, R. Ding, T.-W. Chin, and D. Marculescu, ‘‘Understanding the impact of label granularity on CNN-based image classification,’’ in Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW), Nov. 2018, pp. 895–904.

BIOGRAPHY

SACHITH SENEVIRATNE received the B.Sc. degree in computer science and engineering from the University of Moratuwa, Sri Lanka, and the Ph.D. degree in machine learning from Monash University, Australia. He is currently working as a Research Fellow with the University of Melbourne. His research interests include deep learning, with a focus on contrastive representation learning and applications. He is broadly interested in self-supervised deep-learning approaches across various disciplines such as computer vision, NLP, and reinforcement learning.

RIDWAN SHARIFFDEEN, He received the B.Sc. degree (Hons.) from the Computer Science and Engineering Department, University of Moratuwa, Sri Lanka. He is currently pursuing the Ph.D. degree with the Department of Computer Science, School of Computing, National University of Singapore. His research interests include automated program repair, software security, and software engineering automation. From 2016 to 2018, he worked in the industry as a Senior Software Engineer for System Security and Automation Technology. His research has been recognized by the National Research Foundation Singapore with an invitation to the 10th Global Young Scientist Summit (GYSS’22). He was also awarded the Research Achievement Award by the School of Computing, National University of Singapore, in 2021, and also the CINTEC Award for the Best Computer Science and Engineering gradians for Research by the University of Moratuwa, in 2016.

SANKA RASNAYAKA he received the B.Sc. degree (Hons.) in computer science and engineering from the Engineering Faculty, University of Moratuwa, Sri Lanka, in 2016, and the Ph.D. degree in computer science from the School of Computing, National University of Singapore, for his work on continuous authentication for mobile devices, in 2021. He is currently a Lecturer with the School of Computing, National University ofSingapore. His research interests include applications of AI and computer vision. He mainly works in the fields of biometrics and authentication.

NURAN KASTHURIARACHCHI received the B.Sc. degree (Hons.) from the Computer Science and Engineering Department, University of Moratuwa, Sri Lanka, in 2016. He is an Artificial Intelligence Engineer at the Singapore Press Holdings. He has multiple years of experience in the industry working in the fields of artificial intelligence and machine learning.