IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Self-Supervised Vision Transformers for Malware Detection

SACHITH SENEVIRATNE[1], RIDWAN SHARIFFDEEN[2], SANKA RASNAYAKA[2], AND NURAN KASTHURIARACHCHI[3].

[1]University Of Melbourne, Australia (e-mail: sachith.seneviratne@unimelb.edu.au)
[2]School of Computing, National University of Singapore (e-mail: ridwan@u.nus.edu, sanka@nus.edu.sg)
[3]University of Moratuwa, Sri Lanka (e-mail: nuran.11@cse.mrt.ac.lk)

Corresponding author: Sachith Seneviratne (e-mail: sachith.seneviratne@unimelb.edu.au).

This work was supported by Australian NHMRC Grant GA80134.

ABSTRACT Malware detection plays a crucial role in cyber-security with the increase in malware growth and advancements in cyber-attacks. Previously unseen malware which is not determined by security vendors are often used in these attacks and it is becoming inevitable to find a solution that can self-learn from unlabeled sample data. This paper presents SHERLOCK, a self-supervision based deep learning model to detect malware based on the Vision Transformer (ViT) architecture. SHERLOCK is a novel malware detection method which learns unique features to differentiate malware from benign programs with the use of image-based binary representation. Experimental results using 1.2 million Android applications across a hierarchy of 47 types and 696 families, shows that self-supervised learning can achieve an accuracy of 97% for the binary classification of malware which is higher than existing state-of-the-art techniques. Our proposed model is also able to outperform state-of-the-art techniques for multi-class malware classification of types and family with macro-F1 score of .497 and .491 respectively.

INDEX TERMS Self-Supervised Learning, Deep Learning, Malware Detection, Android Security

I. INTRODUCTION

ARTIFICIAL intelligence has gained significant popularity over the recent years, serving in many applications since its inception. Currently, it dominates in the imaging field - in particular, image classification [1]. Advancements in deep learning technology has further improved the accuracy and reliability of models that use deep learning architectures. One can be impressed by the performance of Deep Neural Networks (DNNs) which takes advantage of the proliferation of large data-sets in addition to the increase in computational power. Despite state-of-the-art performance achieved by DNNs, recent research reveals that such models are prone to adversarial attacks which alters the output of the DNN and degrades the performance [2]. Adversarial perturbation is a well-known problem, due to the nature of deep neural networks which are highly sensitive to slight modifications that are imperceptible to human eye. Recently, many works have been proposed to generate diverse set of adversarial examples, which have been used as benchmarks to evaluate the robustness of deep learning models [3]. Similarly, several strategies are also bought forward in the literature to thwart such attacks [4].

In the realm of software security, malware detection poses an increasingly challenging task [5], which is to identify malicious software programs by differentiating such programs apart from benign applications. According to recent studies [6], malicious software is increasing at an alarming rate, where some or most hide inside popular applications using obfuscation techniques that evades traditional signature-based detection methods. In particular, Android malware detection plays a vital role which seriously threatens the integrity of Android applications. Statistics shows that more than 3.25 million malicious Android apps have been found in 2016 [7], indicating that a new malware app is found every 10 seconds. Android malware detection needs more attention and research to prevent malware from being released in the wild through Android application marketplaces. Malware authors use common obfuscation techniques such as encryption, oligomorphic, polymorphic, metamorphic, stealth and packaging methods to make the task of detection difficult [8]. The malware detection problem can be seen as a classification problem whereby the algorithm needs to correctly categorize a given program into one of the classes: malware or benign. Most existing approaches to detect Android malware rely on extraction of application features and using prior knowledge to train a machine learning classifier to distinguish between benign and malicious applications. Despite the high accuracy, these models are ineffective as attackers bypass such detection by adding features commonly used by benign apps. Such simple addition of benign functionality to malicious apps, such as pop-up messages and logging, can change the detection class from malicious to benign.

Recent malware programs are written with the objective of being highly similar to benign applications by incorporating features of popular applications, having only slight differences which is difficult for humans to perceive [9]. DNNs can detect such imperceptible changes, and by training with a large data-set we can detect features of malicious applications despite having a large amount of benign features embedded. In fact, previous research has shown that malware can be detected using image classifiers made of deep learning models [10]. Malware variants belonging to the same family exhibits visual similarity in the byteplot images, which can be used to train deep learning models to detect specific features which can then be used to detect the existence of similar variant in other applications. These images are generated based on visualization in the spatial domain by converting bytes to pixels. The works typically convert malware binaries to digital images and pass them into a neural network in order to detect malware. Machine learning is widely applied in the detection of Android malware, whether based on static, dynamic or hybrid analysis approaches. Compared with traditional methods, such as signature-based detection, machine-learning based approaches provide better performance in detection efficacy and efficiency, in addition to the ability to detect previously unseen types of malware i.e. zero-shot learning.

In this paper, we present a novel technique building on self-supervised representation learning to detect malware with high-accuracy surpassing state-of-the-art techniques on one of the largest malware data-sets containing 1.2 million binary images across a hierarchy of 47 types and 696 families of malware. Self-supervised learning is capable of adopting self-defined pseudo-labels as supervision and use the learned representations to avoid the cost of annotating large-scale data-sets. Traditional supervised learning methods heavily depend on the availability of annotated large-scale training data, which is impractical with the ever-evolving nature of malware programs. Even though there are plethora of data available in a limited scale, the learning can suffer from issues such as generalization error, spurious correlations and adversarial attacks. This is where self-supervised methods play a vital role in reaping the benefits of deep learning without the expense of heavy annotations and learn feature representations where data provides the supervision. We show that our proposed technique is able to achieve 97% accuracy in detecting malware and with 87% precision in correctly identifying the malware family, outperforming state-of-the-art techniques for one of the largest data-set consisting of million Android apps curated from AndroZoo [11] repository. The main contributions of this study are as follows:

- Implementation of a Transformer based computer vision model utilizing self-supervised learning for Android malware detection. To the best of our knowledge this is the first work to utilize self-supervised learning and vision transformers for the purpose of detecting malware on a large-scale data-set.
- A comprehensive evaluation of the effectiveness in detecting a malware and multi-class malware classification into malware type and malware family. Our results demonstrate that self-supervised learning can effectively classify 47 types and 696 families of malware with a macro-F1 score of 0.497 and 0.491 respectively.
- Contributing to the understanding of self-supervised computer vision models, specifically the Vision Transformer architectures, and their performance on synthetic imagery.

The rest of the article is organized as follows: Section II provides a background to malware detection and self-supervised learning. Next, Section III details the methodology while Section IV elaborates on the evaluation and the results. Section V gives an overview of related work with Section VI discussing important insights. Finally, the conclusion is given at the end of the article.

II. BACKGROUND

In this section, we summarize the important background details for self-supervised malware detection using byteplot grayscale images. First, we describe the process for the image generation where the malware executable is converted to a byteplot grayscale image. These converted images of binary executables can be used to automatically identify visual patterns that can be used in static malware analysis. We then describe the necessary background on image classification using machine learning and self-supervision for image classification.

A. BYTEPLOT VISUALIZATION

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

Freitas et al extended this work for Android applications by constructing the image representation using DEX file (bytecode) 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 array to a 2D image representation 2) scaling the image to a standard size and 3) encoding semantic information into the 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 height is varied based on the file size. The transformed image is then scaled to 256×256 using a standard Lanczos filter from the Pillow library. Once the scaled grayscale image is obtained the final step is encoding semantic information that can further assist the differentiation of distinctive texture patterns. By coloring each byte according to its usage in the malware executable file, the image has an added layer of semantic information on top of the raw bytecode. Gamut et al present [15] an encoding that assigns each byte to a particular RGB color channel depending on its position in the DEX file structure. The observation is that the sections in a DEX file should be distinguished for malware detection as each malware type utilize each section differently, and such patterns can be easily recognized in an image visualization. The encoded image can be decoded back to grayscale by combining each of the channels.

B. 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 labeled based on its physical features). In the context of malware analysis, a byteplot image generated from a malware program may contain different labels such as, if its benign or malware, type of malware, author of the program and malware family. 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 ImageNet dataset which contains 1.28 million images of 1000 classes (which does not include malware images) and was able to achieve an accuracy of 92.97% on a dataset comprising 10,136 malware byteplot images. This approach was one of the first to show that, without feature engineering, using raw pixel values of byteplot images, malware can be classified with high accuracy. The DNN was initially trained for a different task on a different dataset, yet was capable to outperform at the time state-of-the-art models that used specialized reverse engineering efforts. This line of work depend on the learning from already established, annotated data-sets despite the disparity between the initial training data-set used for the weights and the malware data-set used for testing. However, Freitas et al [14] shows that transfer learning does not scale when there is a significant disparity between the training data-set and testing data-set. Using a ResNet18 model pre-trained on ImageNet and fine tuning for malware images, they show that the pre-trained model on ImageNet is less effective than a model trained from scratch on a malware data. In order to achieve the full potential of image classification for malware analysis, the model must learn features from a data-set consisting of malware images.

The challenge is to generate a large data-set for malware images with annotations on different labels it can learn. Generating such a large-scale data-set may not be feasible as it was for ImageNet because the manual annotation for images in ImageNet can be often done by individuals with no special expertise and with minimal time. However, the classification of a malware requires an expert analyst on average around 10 hours to characterize a malicious binary [16], [17]. The expensive effort makes it impractical to generate a large corpus of malware images that consist of expert-labeled data that can be used for different tasks such as classification for malware family, type of malware, author of malware etc. Given the evolution of software with new API changes, new code generation techniques and improved compiler optimizations, program binaries as a whole changes over time. Similarly, adversaries design malicious binaries to be adaptable to these changes while also avoiding detection [18]. This makes it challenging to maintain an up-to-date data-set consisting of expert-labeled data, that can be used to train to detect new malware with new features. This is a concept known a “conceptual drift” where an expert-labeled data-set for malware detection can quickly be outdated due to the nature of malware evolution.

C. SELF-SUPERVISED LEARNING

When labeled data is scarce, self-supervised learning holds great promise for improving representation learning. Self supervision is a paradigm of representation learning which involves training a model on a pretext task with unlabeled imagery, but in a supervised manner. The pretext tasks in this context are a class of tasks which can be automatically generated from imagery without human intervention. The

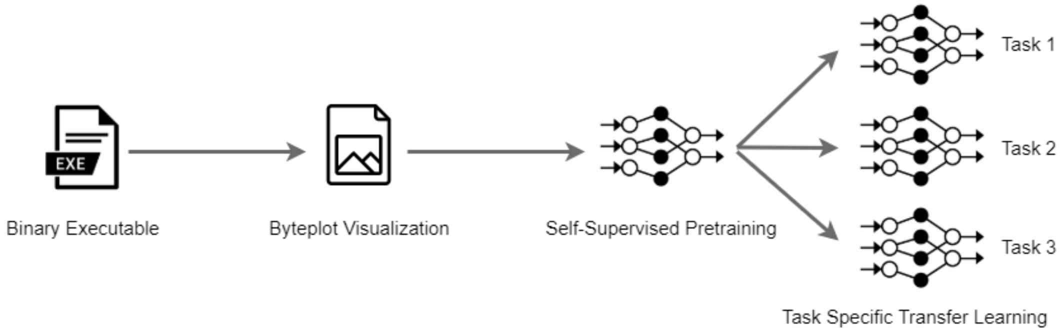

FIGURE 1. A high-level overview of SHERLOCK: self-supervised learning can be used to bootstrap different classification problems using malware image data

goal of self-supervised learning from images is to construct image representations that have semantic meaning via pretext tasks that does not require any manual annotations. Examples of pretext tasks include jigsaw puzzle solving [19] for reconstructing an image after randomly permuting grid sections, relative position prediction [20] for predicting the relative position and orientation of image patches with respect to each other or image colorization [21] for removing colour from an image and restoring it. These tasks require the neural network to learn a generic representation of the dataset in order to solve the pretext task. A model trained for solving these pretext tasks learn representations that can be reused for solving other downstream tasks of interest, such as image classification.

Purely self-supervised techniques learn visual representations that may be inferior to those achieved via fully-supervised techniques on a given downstream task. Thus, self-supervision alone is insufficient for practical use despite demonstrations of encouraging results in recent work. To alleviate this, self-supervision is combined with transfer learning or fine-tuning on the downstream task. This is similar to how humans perform learning. We bring an innate representation of the world to any new task we learn. In the self-supervised learning setting, the labelled imagery in the downstream task provides a supervisory signal (usually by propogating an error function such as cross entropy) which helps the self-supervised representation to adapt to the downstream task. Zhai et al [22] shows that self-supervised learning can dramatically benefit from a small amount of labeled examples by turning self-supervised to semi-supervised learning model. Such a self-supervised learning can alleviate the need for continuously generating manually annotated dataset for malware classification. Moreover, self-supervision can improve several aspects of model robustness, including robustness to adversarial examples, label corruptions and common input corruptions. Hendrycks et al [23] shows that self-supervised learning can even surpass fully supervised methods, raising the question of whether self-supervised learning render large labeled data-sets needless. However, in this work, we investigate the benefits of self-supervised learning combined with a large-scaled labeled data-set, specifically for malware detection and classification. In particular, we explore how label-independent learning can provide efficient workflows for performing multiple single label classifications on the same dataset.

While self-supervised computer vision has received considerable attention, most current work is focused on natural imagery (imagery captured from a camera of a natural scene). However, many forms of imagery consumed by humans take on a more abstract form. There are many concepts that are visually represented as images. The underlying dynamics of such imagery can be drastically different from that of natural scene imagery. For example, an image of a natural scene contains semantic cues as to the proximity of an object to the sensor based on the size of the subject. However, such semantics are noticeably absent or different in more abstract imagery. Additionally the transferability of knowledge representations would be further lowered between abstract imagery domains due to this reason, when compared with natural imagery domains, which points to an increased utility in developing approaches such as self-supervised learning for such domains. Further understanding of how self-supervised learning operates on such imagery is therefore of considerable academic interest. Some prior work in this regard [24], [25] focuses on sketch based abstractions of images, while others operate on images of maps [26] or digital elevation outputs [27]. To the best of our knowledge, no prior work has performed self-supervised learning on images representing binaries of programs which falls into this category of imagery as it is an abstract representation of the underlying software artifact.

III. METHODOLOGY

In this section we describe the overall architecture of our model and an overview of our proposed method to detect and classify malware using self-supervised learning. Next, we describe self-supervised learning to synthesise malware images using a new architecture in computer vision. Finally, we detail how we fine-tune the model by reusing the learnt representation from the synthesis task, for the problem of multi-class labelling in the context of malware classification.

A. OVERVIEW

We propose a novel technique for malware detection using the vision transformer architecture with self-supervised learning. Compared to existing techniques that use image classification for malware detection, we interpret a malware image as a sequence of patches and process it by a standard Transformer encoder as used in natural language processing. This interpretation combined with pre-training on large datasets allows us to re-purpose a trained model effectively for 3 different classification tasks with minimal computation cost. Figure 1 depicts an overview of our proposed method, SHERLOCK. The core idea of SHERLOCK is to generate training samples for the underlying learning task in a fully-supervised manner, which then can be used by classification models to bootstrap the training via transfer learning. The learning task is to synthesise malware images from a limited set of features that captures the semantics of a malware. This is contrary to the previous work on malware detection which is entirely based on supervised learning.

SHERLOCK consists of two main components that can efficiently synthesise malware images using an optimized representation of the malware image and a classification component that can reuse the optimized representation to identify the correct label for a specific classification problem. Given a malware image, SHERLOCK first splits the image into patches (small pieces of the image) of size 16 × 16 pixels. These patches are then flattened or concatenated to form a 1-dimensional vector, which is used to generate a lower dimensional linear embedding representation of the patches. As transformers are used for modelling sequences, positional embedding is used to maintain the 2-dimensional positional correspondence of the image patches. Using such an embedding we train a model to encode the semantic features of a malware image in a fully-supervised manner to obtain an optimized version of the embedding. This optimized embedding is used to bootstrap the classifier component as a base knowledge that captures the semantic encoding of a malware, and fine-tune different classification models on malware image data (i.e. malware detection, type-classification, family-classification). We use the MalNet dataset [14] in our analysis as it is the largest publicly available cybersecurity image database (1.2 million images) with multiple labels for each image corresponding to malware/benign (2 categories), malware type (47 categories) and malware family (696 categories).

B. MASKED AUTO ENCODER

The recent advent of Vision Transformers [28] (ViTs) provide an alternate architecture for image processing that is both simple and scalable. In a Vision Transformer, an image is first split into patches (of sizes such as 8×8 or 16×16 pixels). These patches are then transformed into an embedding using standard transformer encoders as shown in Figure 2. Given an image, the model encodes a representation which it then decodes to regenerate the original image. Therefore, the input to the model is an image, and the output is a reconstruction of the same image, from a reduced feature space with a perfectly accurate output being an exact copy of the original image. Encoding the image into a bottleneck representation (which is usually a fixed size vector of size 256, 512, 1048 etc.) enforces the self-supervisory task to learn useful semantic features for the decoder which is used to reconstruct the image. This concept can be generalized to other modalities of data, for example, Natural Language Processing (NLP) where masked language modelling as a self-supervisory task has previously led to state of the art results [29].

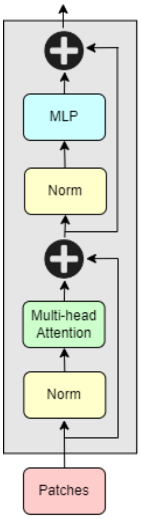

FIGURE 2. Illustration of the Transformer Encoder [28] that takes a sequence of patches as input and transforms into an encoding.

We utilize masked auto encoding as our pre-text task. In general masked auto encoding is associated with removing a portion of the data, and using the remaining data to predict the removed portion. Thus, given an image, some pixels of the image are masked prior to self-supervised training and the neural network must learn how to recover the masked pixels based on the visible pixels. Recent work [30] has shown that masking a large portion of pixels in natural images (such as ImageNet) leads to a challenging self-supervisory task capable of generating useful representations for downstream tasks. In the absence of prior work applying masked auto-encoding to abstract/synthetic imagery, we explore a large masked pixel percentage (75% of the image), thus forcing the model to attempt to recover the image based only on the unmasked 25% of the image, such high percentages have been shown to work well for natural imagery [30]. Figure 3 depicts the results of the image synthesis for two different malware types. Importantly, we use Vision Transformers [28] in our analysis, which are able to effectively incorporate 2-D positional embeddings in order to indicate the location of a patch within an image. Such incorporation is not straightforward in convolutional neural networks.

Figure 4 gives an overview of the auto-encoder component of SHERLOCK that use self-supervised learning to obtain an

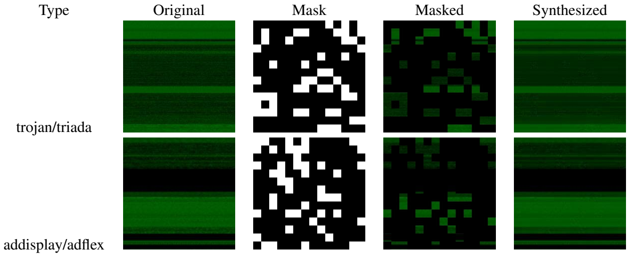

FIGURE 3. Illustration of the image synthesis using images of two malware types. Using an masked image which masks 75% of the image our learning model is able to synthesise images almost identical to the original image.

optimized embedding to reconstruct masked images with a higher accuracy. A mask is generated by uniformly sampling 25% of the image to retain. The original image is then filtered using the generated mask to generate the masked input image. The masked input image is flattened to generate embedding for each patch. Each image is 224×224 with 3 RGB channels, each patch is 16×16 leading to $14 \times 14 = 196$ patches in total. Each embedding is a vector of size $3 \times 16 \times 16 = 768$. The embeddings are shuffled to separate the masked vs unmasked patches, with only the unmasked patches being passed through the encoder architecture. The unmasked embeddings are passed through 12 architecturally identical blocks which have different weights in order to generate an embedding of size 768 for each patch. The unmasked encoder embeddings (size 768) are converted into unmasked decoder embeddings (size 384) using a linear projection layer to match the width of the decoder. The unmasked decoder embeddings are combined with the flattened masked patches, which are represented by a common, learnable mask token indicating the presence of a masked patch. This indicates to the model that such patches are to be predicted in the decoding process. The embeddings and mask tokens are decoded by the decoder, which contains 4 architecturally identical blocks with different weights. In order to generate a final embedding for each patch (size 384). This embedding is passed through a linear projection which maps it to the number of output pixels in a patch ($3 \times 16 \times 16 = 768$. At this stage, a pixel-wise loss is computed between the original image and the synthesized image, which is then backpropagated throughout the network.

Importantly, the ability to only encode the unmasked inputs enables more computationally efficient training in a self-supervised manner. The decoder is only used during self-supervised pre-training which allows flexibility in decoder design. In particular, simpler decoders enforce a better representation in the other parts of the architecture as less knowledge/parameters would be stored within the decoder. The encoder containers 12 blocks while the decoder only has 4, additionally the embedding dimension for the encoder has size 768 while the decoder has size 384. To evaluate reconstruction error, each pixel in a patch is first normalized by subtracting the mean of pixels in the patch and dividing the result by the patch standard deviation. The reconstruction loss is computed by calculating mean standard error over pixels in the masked patches after normalization.

C. CLASSIFICATION MODELS

One of the major advantage in utilizing a self-supervised learning in this manner is the possibility to reuse the same self-supervised representation with minimal additional training in all 3 downstream tasks - in this case malware classification (binary), malware type classification (47 categories) and malware family classification (696 categories). In contrast, simple supervised models would need to separately train from scratch, which would require considerable additional computational processing. In this regard, as more tasks that are formulated on the same dataset, self-supervised methods become more efficient. The self-supervised representations generated are thus fine-tuned separately on each individual task with accuracy and macro f1 scores generated for each task.

Algorithm 1: Malware Detection algorithm

```
Input: Binary Image, $I_{raw}$
Output: Boolean value true/false
Execution:
1. $I = resize(I_{raw})$
2. $L_{patches} = (I)$
3. $encoding = encode(L_{patches})$
4. $P = ComputeProbability(encoding)$
5. if $p > 50\%$ then
    return "Malicious"
else
    return "Benign"
end
```

Illustrated in Algorithm 1 is the overall workflow in our proposed technique to detect/classify an image as a malware. The first step is to resize the image to a size of 224×224. Once the standard size image is generated $GeneratePatches$ method takes as input an image of size 224×224 and generates linearized patches of size 16×16. The list of linearized patches is then provided to our encoder which generates an embedding of the patches $L_{patches}$ capturing complex features. The feature encoding $encoding$ is provided to our pre-trained models which computes the probability P for the encoding which may contain malicious features. If the P is significant (i.e. p > 50%) the algorithm returns Malware else returns Benign. Similar workflows are followed for type and family inference, with the main difference arising from the number of categories (binary classification for malware detection and multi-class classification for type and family).

We use the transformer based ViT-Base architecture with a patch size of 16 (ViT-B/16). The ViT-Base architecture contains 12 model layers or 'blocks' (as in figure 2, and 86 million parameters, corresponding to the encoder architecture used for self-supervised learning as described in Section III-B. The self-supervised encoder from the self-supervised learning step is reused as-is for each of the 3

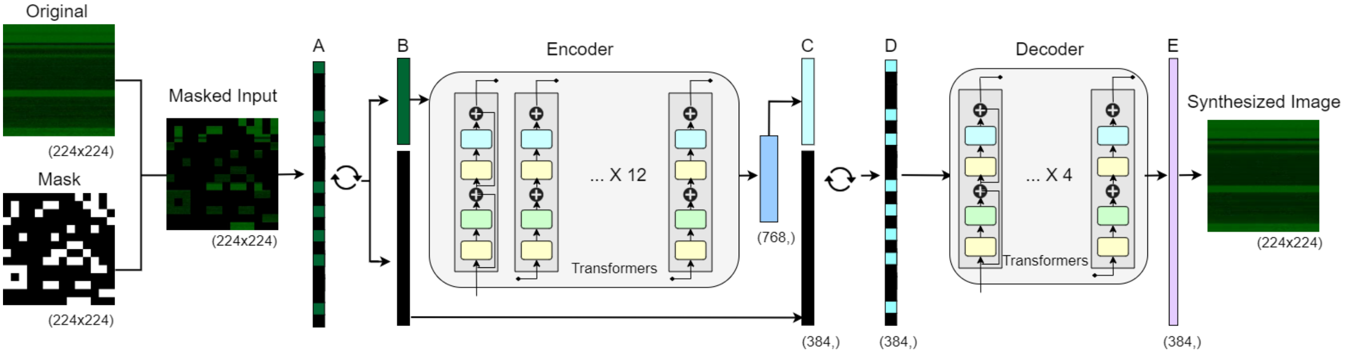

FIGURE 4. Overview of the masked auto encoder which takes masked images as input and learns to reconstruct the original image using self-supervised learning. Layer A is the linearization of the patches which is then shuffled in Layer B to separate masked and unmasked pixels. Layer C depicts the embedding of the masked and unmasked pixels. Layer D represents combined encoding which is the input layer for the decoder to learn. Layer E is the final embedding for the reconstruction of the original image.

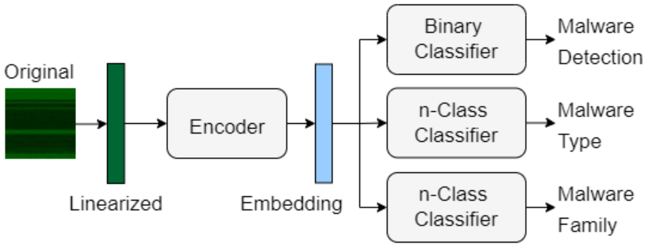

FIGURE 5. Overview of the classification models in SHERLOCK, which reuse the optimized embedding generated via self-supervised learning to bootstrap 3 different classification tasks.

downstream classifications (binary, type and family as per Figure 5). Thus, all the features and weights learned by the self-supervised model are re-used to initialize most of the supervised neural network, which is then further trained to learn improved task-specific features. For each analysis, a linear layer is constructed mapping from the embedding of the encoder (768 in size) to the number of output classes (2, 47 or 696 respectively). The resultant network is fine-tuned end to end (backpropogating gradients and updating weights of every parameter of the neural network), thus enabling the network to learn more specialized features suitable for classifying imagery based on task-specific categories, but building upon the features learned during the self-supervised task of filling in masked patches.

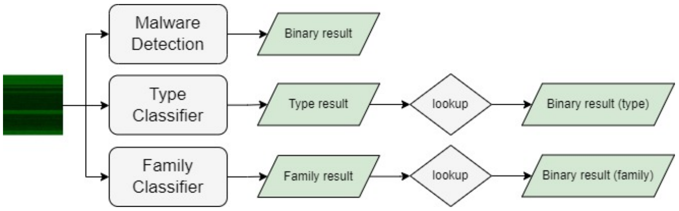

FIGURE 6. Using finer granularity tasks to predict coarse granularity tasks.

The classes for malware-type and malware-family has a direct one-to-one relationship to the binary class malware-benign. Using this hierarchical relationship we can infer the classification of a given android app as malicious or benign, by training the model to classify for a fine granular class such as malware-type or malware-family. In particular, as shown in Figure 6 we leverage the training from finer granularity classification task to infer a classification for coarse granular class. However, a similar inference cannot be established from malware-family to malware-type in this data-set since there is a many-to-many relationship between these two categories.

IV. EVALUATION

In this section, we evaluate the efficiency of our proposed malware detection method SHERLOCK on the largest open-source dataset for Android malware [14]. First, we describe the configuration setup for our experiment, the data set usage and evaluation criteria. Next, we evaluate the performance against state-of-the-art machine learning algorithms, show the confusion matrix of malware classification and discuss the comparison results.

A. EXPERIMENTAL SETUP

We evaluate the efficacy of our proposed self-supervised learning method SHERLOCK on three tasks using the largest open-source dataset for Android malware [14]. First, we evaluate the effectiveness for the pre-training task to synthesise malware images. Second, we evaluate the performance for the classification task of correctly identifying the malware label. Third, we compare our classification performance with existing state of the art tools, and finally we analyze the sensitivity of the synthesis process on the 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 conducted in the Spartan Cluster [31] on Dell R630 servers (6 Cores, 256GB RAM, Nvidia Tesla K80).

For our experiments we use MalNet dataset [14] which:

- includes 1,262,024 malware images extracted from real-world Android applications in AndroZoo [11]

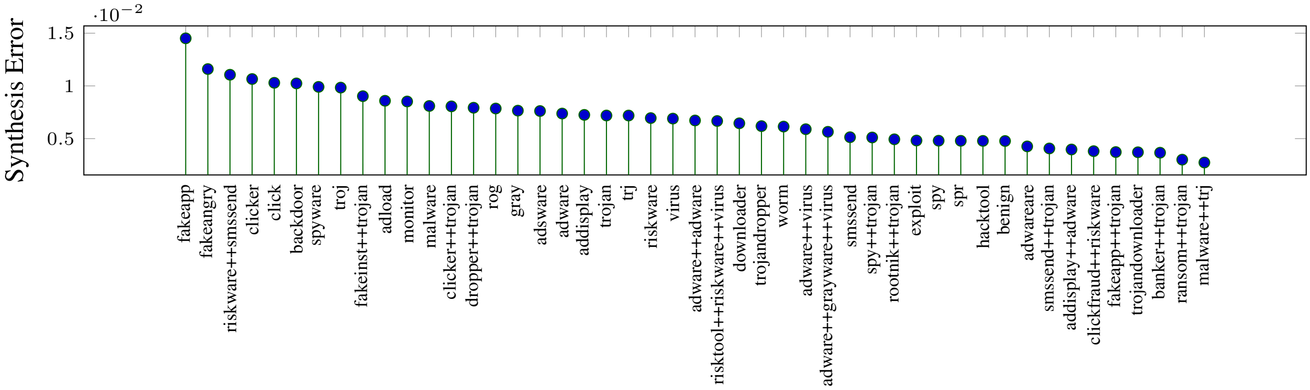

FIGURE 7. Quality of the self-supervised image synthesis process where the reconstruction error is aggregated across malware type

- contains the largest diverse open-source dataset comprising of 47 malware types and 696 malware families including benign applications

Table 1 provides a detailed breakdown of the number of images and families in each malware type. Notably, the class distribution is highly imbalanced across both image type and family, which is a common property of many real-world datasets, where a few of the classes contain a majority of the examples. For all tasks, we report Accuracy, macro Precision, macro Recall and macro F1 scores, and compare against prior work by utilizing the same train/test splits from this dataset.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3}$$

Here, TP stands for True Positive, which represents the number of samples belonging to class c which are correctly classified as class c; FP stands for False Positive, which represents the number of samples not belonging to class c which are erroneously classified as class c; FN stands for False Negative, which is the number of samples that belong to class c which are erroneously classified as a different class.

For the classifications of malware, type and family, we report the Macro-Precision (MP), Macro-Recall (MR) and Macro-F1 score.

$$Macro - Precision = \frac{\sum_c Precision_c}{C} \tag{4}$$

$$Macro - Recall = \frac{\sum_c Recall_c}{C} \tag{5}$$

$$Macro - F1 = \frac{2 \times MP \times MR}{MP + MR} \tag{6}$$

The macro criteria are calculated using a class wise average of the metrics as shown in Equations 4, 5 and 6. Here C represents the number of classes in each multi-class classification task. Additionally, we will also make use of visualizations to better understand the performance of our models. The confusion matrix as well as the Receiver Operating Characteristic (ROC) curve will be used to understand the nuances of our model performance.

$$W_c = \frac{1 - 0.999}{1 - 0.999^{n_c}} \tag{7}$$

For the training process, we used simple cross entropy loss with class re-balancing. The classes which have fewer training examples in the dataset are weighted higher in the loss function. We use the weights suggested in [14] as shown in Equation 7, where n_c is the number of images in class c. These weights cause a relatively higher loss to be assigned to miscategorizations for samples originating from under-represented classes. Therefore, the neural network will prioritize learning a representation that can more accurately classify such under-represented classes, compared to a standard (unweighted) cross-entropy loss.

B. ACCURACY OF MALWARE IMAGE SYNTHESIS

We quantify the ability of the self-supervised representation to regenerate the images in the test set. As this is a direct evaluation of the self-supervisory task, it can be interpreted as an initial evaluation of the ability of the representation and decoder to capture the semantic features associated with the binaries of the considered categories.

For each class in malware-type we iterate over each image and generate 10 random masks for 75% of the image and generate 10 masked input images containing only 25% of the pixels from the original image. For each masked input image, using the trained self-supervised model we generate an image that is similar to the original image provided. For each synthesised image we compute the absolute error for each pixel in comparison to the original image, and average the error across all 10 randomly generated masked input images. Finally, the error is averaged across all such images for each class. The aggregated reconstruction error across malware type is shown in Figure 7.

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 reconstruction error aggregated over malware-type ranges from 0.27%-1.45% with an overall average of 0.68%. It is useful to note that the scale of the error computed depends on the scale of pixel values in the image (ranging between [0, 255] or [0.0, 1.0] per channel, as we use in this work) and the proportion of masked pixels in the image. The same averaging procedure is used across categories and the masked pixel percentage is constant for each image, the final averages we generate are comparable across categories (and only differ from other similar evaluations by some scalar factor). However, if the masked percentage were to differ between images, it would be necessary to take the absolute error between only the masked pixels of the original image and the synthesized image.

> **Malware Image Synthesis Accuracy:** ViT-Base architecture we used in SHERLOCK was able to accurately synthesise malware images of 75% masking with an average reconstruction error of 0.68%.

C. MALWARE CLASSIFICATION

We evaluate the efficacy of our proposed self-supervised model SHERLOCK in correctly identifying the label for each malware-image in 3 different categories. For each category we train a separate classifier for which semantic learning from the supervised-learning is transferred as shown in Figure 5. For each category we report the macro-F1, macro-Precision and macro-Recall as shown in Table 2. For comparison we use 3 popular deep learning architectures–ResNet, DenseNet and MobileNetV2. The results for the compared architectures are referenced from the MalNet dataset [14].

SHERLOCK was able to correctly classify the malware-type and malware-family across 47 and 696 classes respectively. Our model was able to classify both classes correctly with an accuracy of 83.7% for malware-type and 80.2% for malware-family, outperforming the state-of-the-art deep learning models. Furthermore, SHERLOCK recorded the highest macro-F1 score, macro-precision and macro-recall in both categories indicating the superiority of using a self-supervised learning model over existing supervised learning models. One of the contrasting differences between the compared models and SHERLOCK is the source for transfer learning. SHERLOCK is able to directly learn the semantics of a malware image due to its self-supervised pre-training task while the compared models are trained from scratch. Prior work [14] has used transfer learning from ImageNet [32] to pretrain a model and then fine tune to the training data, however it performs significantly worse than the one trained from scratch due to the difference of the semantics between ImageNet images and MalNet images.

Despite the performance for malware-type and malware-family, our model SHERLOCK does not perform well for the binary classification task. Our model reports the best performance in precision while the worse performance in recall for the binary classification task, resulting in worse overall performance in macro-F1 score. Cross comparison with the performance of our model in all 3 tasks, indicates the precision of the results are improved from the semantics learnt from the self-supervised learning, while recall can be improved with additional fine-tuning. However since the class imbalance is high (i.e. 2:31) for the binary classification task the fine-tuning does not have access to enough training data to accurately learn features for the benign class. How-

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	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

ever, with more classes introduced in the tasks for classifying malware-type and malware-family the imbalance effect is distributed over multiple classes, and improves overall recall. This indicates a strong influence from the pre-training task which learns semantic information for the secondary classification task. Since the pre-training task in our model is optimized to synthesise images it learns necessary semantic features relevant for the synthesis. Additionally due to the class imbalance problem in our dataset the semantic learning is skewed towards malware images, resulting in a higher number of misclassifications as malware.

Malware Classification Accuracy: SHERLOCK was able to outperform existing state-of-the-art deep learning models for malware-type and malware-family with an macro F1-Score of .497 and .491 respectively.

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

Model	F1 Score	Precision	Recall
SHERLOCK	.854	**.920**	.810
SHERLOCK (type)	.876	.891	.862
SHERLOCK (family)	**.878**	.867	**.890**
ResNet18	.862	.893	.837
ResNet50	.854	.907	.814
DensetNet121	.864	.900	.834
DenseNet169	.864	.890	.841
MobileNetV2(x.5)	.857	.894	.827
MobileNetV2(x1)	.854	.889	.825

D. MALWARE DETECTION

We evaluate the overall effectiveness of SHERLOCK to accurately detect a malware application using inference based on the classification label it generates as described in Figure 6. Importantly, generating an image from the corresponding binary takes on average 0.479 seconds, while inferring a label using Sherlock takes on average 0.003 seconds, for a total inference time of 0.482 seconds with a single 4 GPU node containing 24 cores(Intel Xeon CPU E5-2650 v4 @ 2.20GHz) and 4 P100 GPU with 12GB of GPU RAM each on Spartan [31]. A comparative analysis of our inference result with the state-of-the-art deep learning models is presented in Table 3, where the results for state-of-the-art is extracted from recent work [14]. Our model SHERLOCK has the best performance in all three dimensions in macro F1-score, macro precision and macro recall. However, inference using malware-family classifier has the overall best-performance with an macro F1-score of 0.878, indicating that pre-trained model with finer granularity can learn distinct features to differentiate an image between the benign class and malware class.

For an in-depth analysis we plot the results of our inference models in Figure 9. Binary classification of a given byte-plot image trained with coarse-granularity achieves highest precision but lowest recall. For applications such as Malware detection where precision is most important, a model pre-trained with self-supervised learning can achieve state-of-the-art performance, with the final model trained on coarse-grained task such as two-class labeling. Whereas the same pre-trained model with the final model trained on a finer-grained task such as identifying the malware-type/malware-family (N classes) will be more effective in improving recall and overall F1-score. An observation we make is that with the increasing number of classes for the final model, the recall is improved while precision is decreased. Additionally, both malware-type based inference and malware-family based inference is able to achieve similar macro F1-score with the contrasting differences in its precision and recall values. While malware-type based inference has the better precision in comparison with malware-family, recall is better for the latter. To further understand the significance of our results, we also plot the Receiver Operating Characteristics (ROC) curve in Figure 10. Despite having a better F1-score, malware-type based inference has a low Area Under Curve (AUC), indicating the classifier will not perform well for different 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 able to correctly detect a malware with an accuracy of 97% for an highly imbalanced dataset while maintaining 86.7% precision and 89% recall.

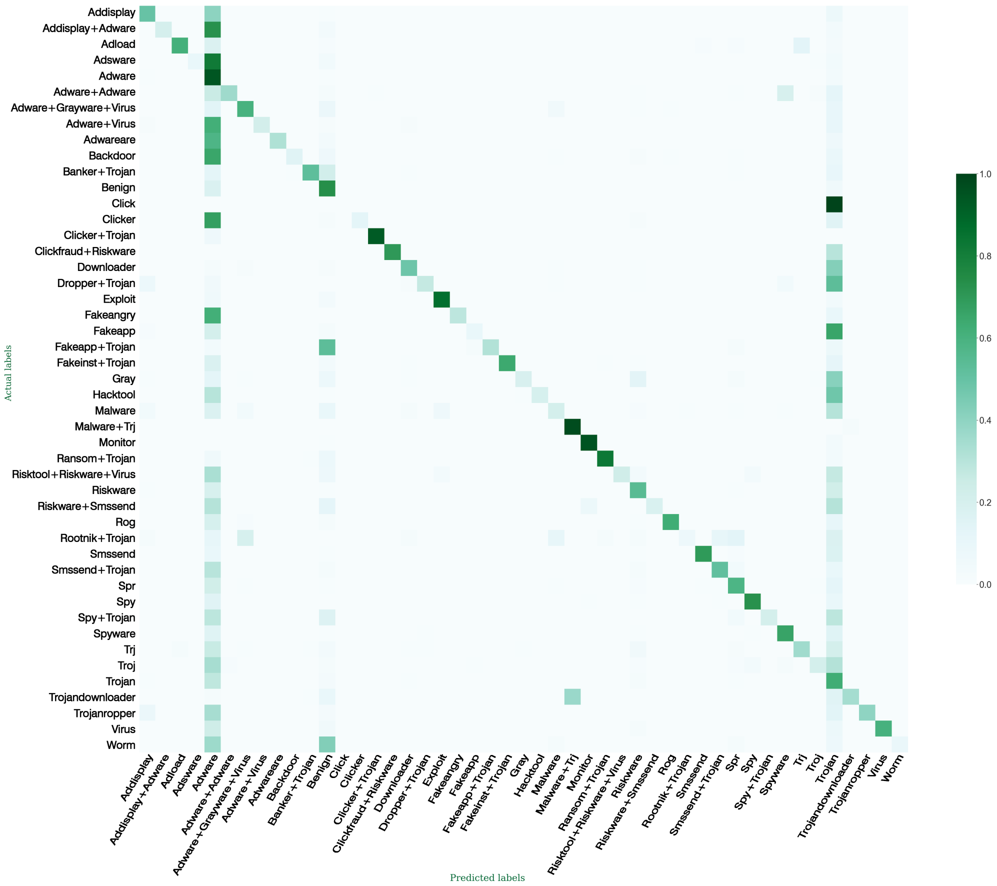

FIGURE 8. Confusion Matrix for malware type classification, which visualize the performance in correctly identifying the actual class label. The darker the shade the stronger the classifier performance and lighter-shade indicates poor performance. Our model results depict more darker shades for the diagonal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

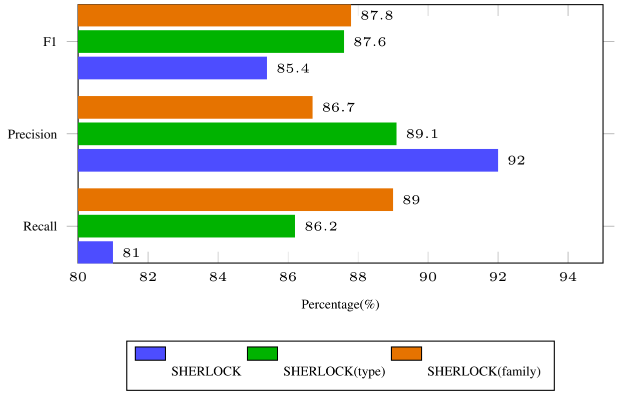

FIGURE 9. In-depth analysis of the performance metrics of SHERLOCK with 3 binary classification models trained on different granularity

V. RELATED WORK

A. MALWARE DETECTION AND CLASSIFICATION

Malware detection and classification has become a crucial task due to the increasing complexity of malware and the commonality of computing systems. Abusitta et. al. [33] categorized malware detection and classification approaches based on features and algorithms used, as shown in Figure 11.

Features used for the analysis can be classified as static or dynamic. Static features are extracted from the executable files while dynamic feature extraction methods make use of a sandbox environment to run the program and collect the memory image or the behaviors of the program execution to extract features [34]. Some of the features in literature include printable strings [35]–[37], byte code [38]–[41], assembly code [42]–[44], API/DLL system calls [44]–[46], control flow graphs [47]–[49] and function level features [50]. Algorithm used for detection and classification approaches can be broadly categorized into signature based approaches and Artificial Intelligence based approaches. Signature based approaches are the de facto of current antivirus providers [51]. These signatures are created by humans and the malware detectors performs a matching between programs and signatures to detect or classify malware [52], [53]. One can argue this method could only detect known

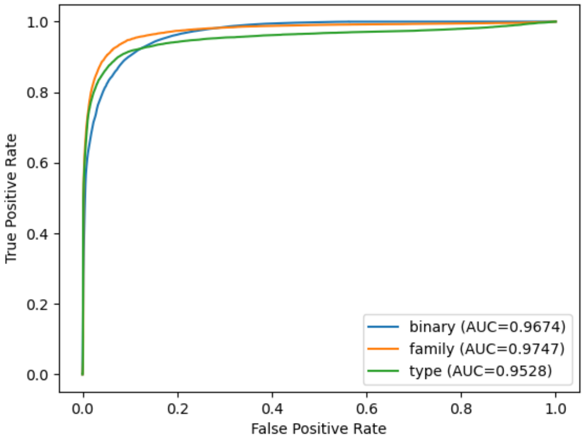

FIGURE 10. Receiver Operating Characteristics (ROC) curve for malware detection with 3 binary classification models trained on different granularity

malware and dependent on the manual signature generation. There is a tendency to move towards AI based malware detection and classification which can mainly be categorized into supervised [54], [55], unsupervised [56]–[58] and semi supervised [59]–[61] methods. In supervised malware detection the model learns features from a labeled dataset, while unsupervised methods extract patterns from unlabeled data. Semi-supervised methods make use of both labeled and unlabeled data. While traditional methods like Naïve Bayes, Decision Trees, K-Nearest Neighbors and SVM have been extensively used in the past. An analysis of these classical methods showed SVM performed well for malware detection [62]. An android specific malware detection survey by Liu et. al looks at the current ML methods used for malware detection [63]. More recently ReLU, LSTM and CNN based methods have received more attention. A deep learning based detection method which works with image data was explored by Yadav et. al. [64]. Due to the ability to recognize previously unseen malware and the higher performance over traditional methods, these machine learning based malware detection techniques are becoming widely researched [65].

Our approach can be classified as a self-supervised learning based algorithm that make use of static features of the application. Contrast to existing work, this is the first self-supervised based learning method that has been studied with the largest openly-available dataset for Android malware.

B. IMAGE CLASSIFICATION

Imagery such as those generated to represent code or program binaries broadly fall into the category known as synthetic imagery. Such imagery is distinct from natural imagery, which is the standard type of data used in computer vision in datasets such as ImageNet [32], incorporating images of real world scenes captured through a camera. Exploration of the performance of self-supervised methods on complex synthetic imagery remains under-explored compared to natural imagery. Some prior work in this area has explored representation learning from sketches [24], [25], altitude imagery [27], synthetic scene imagery [66], and Google map imagery [26]. Analysis in such domains contributes useful knowledge regarding the performance of self-supervised methods on synthetic imagery, which have different visual features to natural imagery.

VI. PERSPECTIVES

A key advantage of using self-supervision in this imagery domain (malware imagery) is the ability to initialize a single general representation in a task-independent manner. This representation can then be specialized to perform specific analysis (such as malware detection, malware type categorization and malware family categorization, as well as potentially other tasks). In essence, the representation can be easily specialized for any different set of labels associated with the imagery, for example future work could look at labels associated with malware authors and attempt to identify unique "signatures" corresponding to authors from the imagery.

Interestingly, the use of coarser labels has improved performance in the malware detection task (classification of malware images into benign vs malicious). Both malware family and type predictions can be converted into a malware detection prediction by performing a lookup. We found that the models for family and type prediction, once converted into the corresponding malware detection category ("malicious" or "benign") provided superior performance when compared with the malware detection prediction, when all models were trained with the same number of images and identical analytical settings. Since the only difference between the analyses was the granularity of the labels (696 for family, 47 for type, 2 for malware detection), we can conclude that the coarser labels have led to a better characterization of the images for the malware detection task. In this aspect, while both models trained on more granular labels show significant improvement in F1 score over the malware detection model, they both have similar performance to each other (in terms of F1 score). This suggests that the improvement gained via increased label granularity has diminishing returns, which may be interesting for future work to explore.

Upon further consideration, the reason for this improvement can be induced. On a fixed dataset, with fine-grained labels, the model is forced to create multiple decision boundaries to separate instances from different categories from each other. The number of potential required boundaries will grow quadratically in the number of categories (n*(n-1)/2), as each category may be adjacent to every other in the feature space. This leads to a more granular clustering of images with a larger number of clusters containing fewer images in the feature space (as the total number of images is fixed in all analyses). Additionally, this provides further reasoning for the muted improvement (in terms of malware detection results) of increasing the number of categories from 47 to 696. The malware detection task has 1 decision boundary (benign vs malicious), whereas type prediction

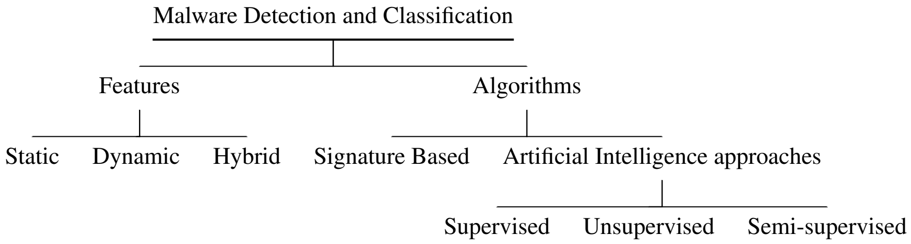

FIGURE 11. Categorization of malware detection and classification [33]

47*46/2 = 1081 potential boundaries and family prediction has 676*675/2 = 228150. Based on the observed results the granularity of type labels is sufficient to capture the level of detail required for the binary classification required in malware detection, and the added granularity of the family labels does not improve results further. The impact of the label granularity is also apparent in the ROC curves for malware detection, with the more granularly labelled family prediction model showing superior performance to the other more coarsely labelled models. Future work could explore this aspect further by exploring the added impact of having a more complex baseline task (with more than 2 categories). As the type and family labels do not have a one to one mapping, we are unable to explore this any further at this stage. The impact of label granularity of classification has previously been studied in convolutional neural networks [67]. The findings in our work further generalize such results to the Vision Transformer architecture.

VII. CONCLUSION

In this paper, we present a system based on self-supervised learning for malware classification, SHERLOCK which is a transformer based computer vision model that utilizes self-supervised learning to detect Android malware. We evaluate SHERLOCK on a large-scale data-set of 1.2 million Android apps consisting of 47 malware types and 696 malware families. Compared with state-of-the-art deep learning architectures ResNet, DenseNet and MobileNetV2, our proposed technique SHERLOCK outperforms all techniques in malware detection and classification with 97% accuracy for detection. SHERLOCK was also able to correctly classify malware type and malware family with a macro-F1 score of .497 and .491 respectively. This work demonstrates the efficacy of self-supervised computer vision models, specifically the Vision Transformer architectures in the use for malware classification. Through this work, we improve static analysis based malware detection and classification, which we believe can be further improved with by augmenting additional features derived from dynamic analysis.

To allow the research community to better replicate and reproduce the findings in our studies and to extend the work further, we have open-sourced our model to the community in Github: https://github.com/sachith500/Sherlock.

ACKNOWLEDGMENTS

This project was supported by Australian NHMRC Grant GA80134. This research was undertaken using the LIEF HPC-GPGPU Facility hosted at the University of Melbourne. This Facility was established with the assistance of LIEF Grant LE170100200.

REFERENCES

[1] A. A. M. Al-Saffar, H. Tao, and M. A. Talab, "Review of deep convolution neural network in image classification," in 2017 International conference on radar, antenna, microwave, electronics, and telecommunications (ICRAMET). IEEE, 2017, pp. 26–31.

[2] A. Subramanya, V. Pillai, and H. Pirsiavash, "Fooling network interpretation in image classification," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 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 Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 321–331.

[4] S. Qiu, Q. Liu, S. Zhou, and C. Wu, "Review of artificial intelligence adversarial attack and defense technologies," Applied Sciences, 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 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 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 2022 24th International Conference on Advanced Communication Technology (ICACT). IEEE, 2022, pp. 297–301.

[7] M. A. Omer, S. Zeebaree, M. Sadeeq, B. W. Salim, S. Mohsin, Z. N. Rashid, and L. M. Haji, "Efficiency of malware detection in android system: A survey," Asian Journal of Research in Computer Science, pp. 59–69, 2021.

[8] X. Deng and J. Mirkovic, "Polymorphic malware behavior through network trace analysis," in 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS). IEEE, 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 communications surveys & tutorials, vol. 17, no. 2, pp. 998–1022, 2014.

[10] E. Rezende, G. Ruppert, T. Carvalho, F. Ramos, and P. de Geus, "Malicious software classification using transfer learning of resnet-50 deep neural network," in 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), 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 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), 2016, pp. 468–471.

[12] G. Conti, E. Dean, M. Sinda, and B. Sangster, "Visual reverse engineering of binary and data files," in International Workshop on Visualization for Computer Security. Springer, 2008, pp. 1–17.

[13] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: Visualization and automatic classification," in Proceedings of the 8th International Symposium on Visualization for Cyber Security, ser.

VizSec '11. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/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, 2021. [Online]. Available: https://arxiv.org/abs/2102.01072

[15] J. Gennissen, L. Cavallaro, V. Moonsamy, and L. Batina, "Gamut: sifting through images to detect android malware," Bachelor thesis, Royal Holloway University, London, UK, 2017.

[16] A. Mohaisen, A. G. West, A. Mankin, and O. Alrawi, "Chatter: Classifying malware families using system event ordering," in 2014 IEEE Conference on Communications and Network Security. IEEE, 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 29th USENIX Security Symposium (USENIX Security 20), 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 Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, 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: Springer International Publishing, 2016, pp. 69–84.

[20] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in 2015 IEEE International Conference on Computer Vision (ICCV). Santiago: IEEE, 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: Springer International Publishing, 2016, pp. 649–666.

[22] X. Zhai, A. Oliver, A. Kolesnikov, and L. Beyer, "S4l: Self-supervised semi-supervised learning," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1476–1485.

[23] D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song, "Using self-supervised learning can improve model robustness and uncertainty," Advances in Neural Information Processing Systems, vol. 32, 2019.

[24] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, "Unsupervised data augmentation for consistency training," arXiv preprint arXiv:1904.12848, 2019.

[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 Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 5672–5681.

[26] S. Seneviratne, K. A. Nice, J. S. Wijnands, M. Stevenson, and J. Thompson, "Self-supervision. remote sensing and abstraction: Representation learning across 3 million locations," in 2021 Digital Image Computing: Techniques and Applications (DICTA). IEEE, pp. 01–08.

[27] S. Seneviratne, "Contrastive representation learning for natural world imagery: Habitat prediction for 30,000 species," CLEF working notes, 2021.

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

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, 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," arXiv preprint arXiv:2111.06377, 2021.

[31] G. Lev Lafayette, L. Vu, and B. Meade, "Spartan performance and flexibility: an hpc-cloud chimera," OpenStack Summit, Barcelona, Spain. doi, vol. 10, p. 49, 2016.

[32] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition. Ieee, 2009, pp. 248–255.

[33] A. Abusitta, M. Q. Li, and B. C. Fung, "Malware classification and composition analysis: A survey of recent developments," Journal of Information Security and Applications, vol. 59, p. 102828, 2021.

[34] A. Damodaran, F. D. Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, "A comparison of static, dynamic, and hybrid analysis for malware detection," Journal of Computer Virology and Hacking Techniques, 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 International conference on detection of intrusions and malware, and vulnerability assessment. Springer, 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 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 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 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, 2020, pp. 775–784.

[38] B. Anderson, C. Storlie, and T. Lane, "Improving malware classification: bridging the static/dynamic gap," in Proceedings of the 5th ACM workshop on Security and artificial intelligence, 2012, pp. 3–14.

[39] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in 2015 10th international conference on malicious and unwanted software (MALWARE). IEEE, 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 generation computer systems, vol. 102, pp. 112–126, 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 International Workshop on Deployable Machine Learning for Security Defense. Springer, 2021, pp. 81–106.

[42] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," Journal in computer Virology, 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," Journal in computer virology, vol. 7, no. 2, pp. 107–119, 2011.

[45] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining api calls," in Proceedings of the 2010 ACM symposium on applied computing, 2010, pp. 1020–1025.

[46] Z. Salehi, A. Sami, and M. Ghiasi, "Using feature generation from api calls for malware detection," Computer Fraud & Security, 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. 21 235–21 245, 2019.

[48] J. Yan, G. Yan, and D. Jin, "Classifying malware represented as control flow graphs using deep graph convolutional neural network," in 2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 2019, pp. 52–63.

[49] D. Bruschi, L. Martignoni, and M. Monga, "Detecting self-mutating malware using control-flow graph matching," in International conference on detection of intrusions and malware, and vulnerability assessment. Springer, 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, 2021.

[51] M. Al-Asli and T. A. Ghaleb, "Review of signature-based techniques in antivirus products," in 2019 International Conference on Computer and Information Sciences (ICCIS). IEEE, 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 Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001. IEEE, 2000, pp. 38–49.

[53] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in 12th USENIX Security Symposium (USENIX Security 03), 2003.

[54] J. Singh and J. Singh, "Assessment of supervised machine learning algorithms using dynamic api calls for malware detection," International Journal of Computers and Applications, vol. 44, no. 3, pp. 270–277, 2022.

[55] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on api call sequence analysis," International Journal of Distributed Sensor Networks, vol. 11, no. 6, p. 659101, 2015.

[56] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in International Workshop on Recent Advances in Intrusion Detection. 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 Generation Computer Systems, vol. 120, pp. 91–108, 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 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019, pp. 771–782.

[59] I. Santos, J. Nieves, and P. G. Bringas, "Semi-supervised learning for unknown malware detection," in International Symposium on Distributed Computing and Artificial Intelligence. 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, and H. Xie, "Malware classification for the cloud via semi-supervised transfer learning," Journal of Information Security and Applications, vol. 55, p. 102661, 2020.

[62] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," Human-centric Computing and Information Sciences, vol. 8, no. 1, pp. 1–22, 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. 124 579–124 607, 2020.

[64] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, "A two-stage deep learning framework for image-based android malware detection and variant classification," Computational Intelligence, 2022.

[65] M. T. Ahvanooey, Q. Li, M. Rabbani, and A. R. Rajput, "A survey on smartphones security: software vulnerabilities, malware, and attacks," arXiv preprint arXiv:2001.09406, 2020.

[66] Z. Ren and Y. J. Lee, "Cross-domain self-supervised multi-task feature learning using synthetic imagery," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 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 2018 IEEE international conference on data mining workshops (ICDMW). IEEE, 2018, pp. 895–904.

SACHITH SENEVIRATNE received his B.Sc. degree in computer science and engineering from the University of Moratuwa, Sri Lanka. He completed his PhD in machine learning from Monash University, Australia. Currently, he is working as a research fellow at the University of Melbourne. Sachith's research work revolves around 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 is a PhD Candidate in Department of Computer Science at School of Computing, National University of Singapore. His current focus of research is on automated program repair, software security and software engineering automation. He received his Bachelor of Science(BSc) Honors (Hons) degree from the Computer Science Engineering Department at University of Moratuwa, Sri Lanka. From 2016-2018 he worked in the industry as a Senior Software Engineer for System Security and Automation Technology. Mr. Ridwan's 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 School of Computing, National University of Singapore in 2021 and CINTEC Award for the best Computer Science and Engineering graduand for Research by University of Moratuwa, Sri Lanka in 2016.

SANKA RASNAYAKA is a lecturer at the School of Computing in the National University of Singapore. His research interests lie in applications of AI and computer vision. He mainly works in the fields of Biometrics and authentication. In 2021, he received his PhD in Computer Science from the School of Computing in the National University of Singapore, for his work on Continuous Authentication for mobile devices. Prior to that in 2016 he received his Bachelor of Science Honors degree in Computer Science and Engineering from the Engineering Faculty of the University of Moratuwa, Sri Lanka.

NURAN KASTHURIARACHCHI is an Artificial Intelligence Engineer at the Singapore Press Holdings. He has multiple years of experience in the industry working in the fields of Aritifical Intelligence and Machine Learning. He obtained his Bachelor of Science (BSc) Honors degree from the Computer Science and Engineering department of the University of Moratuwa, Sri Lanka in 2016.