IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 13 March 2024, accepted 12 May 2024, date of publication 20 May 2024, date of current version 18 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3403458

RESEARCH ARTICLE

RMDNet-Deep Learning Paradigms for Effective Malware Detection and Classification

S. PUNEETH[1,2], SHYAM LAL[1], (Senior Member, IEEE), MAHENDRA PRATAP SINGH[3], (Member, IEEE), AND B. S. RAGHAVENDRA[1], (Member, IEEE)

[1]Department of Electronics and Communication Engineering, National Institute of Technology Karnataka (NITK) at Surathkal, Surathkal 575025, India
[2]Department of Electronics and Communication Engineering, The National Institute of Engineering, Mysuru 570008, India
[3]Department of Computer Science and Engineering, National Institute of Technology Karnataka (NITK) at Surathkal, Surathkal 575025, India

Corresponding author: B. S. Raghavendra (r.bobbi@nitk.edu.in)

ABSTRACT Malware analysis and detection are still essential for maintaining the security of networks and computer systems, even as the threat landscape shifts. Traditional approaches are insufficient to keep pace with the rapidly evolving nature of malware. Artificial Intelligence (AI) assumes a significant role in propelling its design to unprecedented levels. Various Machine Learning (ML) based malware detection systems have been developed to combat the ever-changing characteristics of malware. Consequently, there is a growing interest in exploring advanced techniques that leverage the power of Deep Learning (DL) to effectively analyze and detect malicious software. DL models demonstrate enhanced capabilities for analyzing extensive sequences of system calls. This paper proposes a Robust Malware Detection Network (RMDNet) for effective malware detection and classification. The proposed RMDNet model branches the input and performs depth-wise convolution and concatenation operations. The experimental results of the proposed RMDNet and existing DL models are evaluated on 48240 malware and binary visualization image dataset with RGB format. Also on the multi-class malimg and dumpware-10 datasets with grayscale format. The experimental results on each of these datasets demonstrate that the proposed RMDNet model can effectively and accurately categorize malware, outperforming the most recent benchmark DL algorithms.

INDEX TERMS Binary classification, concatenation, convolution, cyber security, deep learning, depthwise convolution, malware, multiclass classification.

I. INTRODUCTION

With the increasing sophistication of malware threats, the framework of cybersecurity is changing dramatically. Attackers are continually pushing the boundaries of established security procedures, equipped with innovative strategies and tools. This increase in sophistication includes not only the complexity of the malicious code itself but also the strategies used to avoid detection, exploit weaknesses, and remain within infected systems. Malware evolution is distinguished by the growth of polymorphic and metamorphic code, which makes detection and analysis more difficult. To conceal their destructive payloads and intents, cybercriminals use advanced obfuscation tactics, encryption, and anti-analysis techniques. Cybersecurity experts must use cutting-edge and flexible defense strategies in response to these growing threats. Security systems are using ML and AI at increasing rates to detect trends, abnormalities, and zero-day vulnerabilities instantly. Most objects in our present-day reality are not linked to a computer network, but this situation is rapidly transforming. Objects that were previously unconnected and exist all around us are now empowered to communicate with other objects and individuals. This advancement paves the way for novel services and enhances the efficiency of our daily lives. However, our society is embracing connected technology at a faster pace than our capacity to ensure its security. As the use of networked devices grows, protecting the security of data at rest and in transit poses considerable problems. Failure to secure network data exposes systems to the risk of malware injection and unauthorized access to

The associate editor coordinating the review of this manuscript and approving it for publication was Jemal H. Abawajy.

personal or sensitive information [1]. Particularly within the realm of Internet of Things (IoT), the need for strong security measures is paramount to uphold consumer trust. However, due to factors such as cost, size, performance, and security management often takes a lower priority in IoT deployments. Consequently, IoT becomes vulnerable to security breaches, resulting in substantial, financial and reputation damages [2]. Internet security is facing a significant menace as malware attacks continue to surge at an exponential rate. It can affect regular processes, gather sensitive information, and obtain superuser rights to carry out malicious acts. Attackers deliver it to the victim's PC by exploiting security weaknesses in operating systems or application software [3].

Traditional techniques to malware detection and analysis are struggling to keep up with the rising sophistication and diversity of malware threats. While effective against known malware variants, signature-based methods, and rule-based systems often fail to detect novel and evolving threats [4]. As a result, there is an essential need to investigate sophisticated methodologies capable of adapting to and successfully analyzing the ever-changing environment of malicious software [5]. DL is a subfield of ML that focuses on artificial neural networks with multiple layers, enabling the automatic learning and extraction of complex patterns and features from data. Inspired by the structure and function of the human brain, DL algorithms excel at tasks such as image and speech recognition, natural language processing, and pattern recognition. The depth of the neural networks allows them to autonomously learn hierarchical representations, making DL particularly powerful for tasks requiring intricate and nuanced understanding of data. [6]. This ability makes DL an appealing approach for tackling the challenges in malware analysis. The fundamental objective of malware analysis is to uncover and understand the behavior, objectives, and capabilities of malicious software [7]. This process involves static analysis, which examines the structure and content of malware samples without executing them, and dynamic analysis, which observes malware's behavior when executed in a controlled environment.

DL techniques can significantly contribute to both static and dynamic analysis by automating feature extraction, improving detection accuracy, and providing insights into the inner workings of malware. In this paper, we delve into the realm of malware analysis using DL techniques. We aim to explore the potential of various DL models in effectively analyzing and detecting malware [8]. To accomplish our objectives, we leverage large-scale datasets consisting of binary and diverse types of malware samples. These datasets encompass a wide range of malware families, such as viruses, worms, and trojans, providing a comprehensive representation of the threat landscape [9]. Discussing the pre-processing steps involved in preparing the data for training DL models, ensuring data integrity, and appropriate feature extraction. Furthermore, addressing the challenges associated with the interpretability and explainability of DL models in the context of malware analysis. While DL models have demonstrated exceptional performance, their decision-making process often lacks transparency, hindering the understanding of how and why a classification or detection decision is made [10]. Explore methods such as attention mechanisms and interpretability techniques to shed light on the reasoning behind the DL model outputs. Through extensive experimentation and evaluation, empirical evidence showcasing the effectiveness of DL techniques in malware analysis. Results obtained in this paper reveal improvements in detection accuracy, robustness against emerging malware variants, and the capacity to identify hidden patterns and characteristics that may not be detectable using standard analytic approaches. This work intends to contribute to the improvement of malware analysis by harnessing the power of DL techniques. By capitalizing on the capabilities of neural networks, we can enhance the accuracy, efficiency, and adaptability of malware detection and analysis systems.To defend against zero-day attacks, the proposed RMDNet excels at detecting anomalies and recognizing novel attack signatures, providing a proactive defense against previously unseen threats. RMDNet's ability to autonomously adapt to evolving threats without explicit programming makes it valuable for real-time detection and mitigation. The key contributions of this study are as follows:

1) A RMDNet-deep learning based malware classification algorithm is proposed to efficiently differentiate classes of malware samples while retaining high accuracy on different malware datasets and also emphasizing computational efficiency.
2) Propose a DCOCO block, that performs depth wise convolution and concatenation methods and has the ability to perform effective and efficient feature extraction.

Rest of the article is organized as follows: Section II discusses various methods for categorizing and detecting malware. The proposed RMDNet architecture model's comprehensive description and the datasets are the main topics of Section III. The proposed RMDNet model's training and implementation details are described in Section IV. The experimental results of our model are detailed in Section V along with comparisons of various benchmark models. Section VI of our paper presents the results of our research.

II. RELATED WORK

Malware detection is crucial for securing computer systems and user data in the area of computer security. To fight the constantly changing threat landscape, several malware detection techniques have been developed as shown in Figure 1. These techniques are classified as static analysis, dynamic analysis, feature extraction techniques, and DL techniques.

A. STATIC ANALYSIS

In this approach, most of the antivirus software used for detection uses the signature-based technique. These signatures are generated by gazing at the disassembled

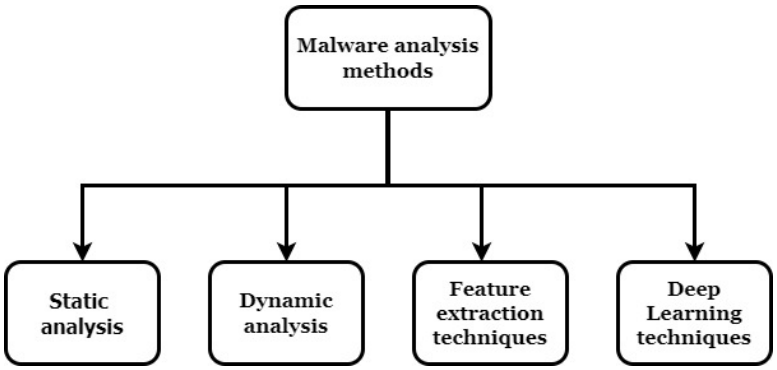

FIGURE 1. Malware detection and classification methods.

code or using the string command. Disassembling portable executables is made easier by a variety of disassemblers and debuggers. Thus, features are taken from disassembled code, and it is analyzed. Therefore, these characteristics are crucial in creating the signature of a specific malware family. Static analysis is a technique for assessing malware without running it [11]. This is often accomplished by analyzing the coding of a binary file in order to comprehend its operation and discover any malicious activities. Potential security issues in a sample can be discovered via static analysis without affecting the analysis environment. Static analysis does not involve the execution of the actual malware. It is safer because of this non-execution environment, which lowers the possibility of unintentional activation or unexpected repercussions during dynamic analysis [12]. They are quick, safe, and can readily detect multi-path malware. But they may make errors while analyzing malware that is unknown. Attackers employ several ways to find vulnerable devices and transform them into bots using infection scripts. Data botnet arrays are then leveraged to conduct network traffic attacks. Opcodes are building blocks that have historically been utilized for malware detection and statically analyzing program activity. The control flow graph method (CFG) is used to extract the executable opcodes, representing behavior characteristic executable [13], [14]. Features are chosen from CFG-based and text-based sequences, if packed malware cannot be unpacked, this strategy is rendered insignificant. Formulating a Convolution Recurrent Neural Network (CRNN) to detect malware using an N-extracted opcode sequence from a binary file without execution doesn't reflect in indirect branching instructions. Information about the program or its intended behavior is gathered from explicit and implicit observations in its binary/source code through static analysis [15]. Static analysis solutions are often created using signature-based approaches, but even with their extreme precision, they are ineffective against unknown malicious code.

B. DYNAMIC ANALYSIS

As traditional static approaches fail to keep up with the increasing sophistication of malicious software, dynamic analysis is essential in the identification and categorization of malware. Dynamic analysis involves executing malware samples in a controlled environment to observe their behavior during runtime. An effective method used to investigate malware is by analyzing the program's behavior during execution [16], [17]. This approach, also known as behavioral analysis, involves observing and collecting information from the operating system including API call sequence analysis, system call monitoring, network traffic analysis, sandboxing, and memory behavior analysis. However, dynamic analysis does come with some limitations. For example, accurately simulating the conditions necessary to trigger the malware's dangerous functionalities, especially when targeting specific vulnerable applications, can be quite challenging [18]. Additionally, determining the precise time frame required to observe harmful behavior for each malware instance remains uncertain. To accomplish categorization, the ML approach makes use of various aspects of the malware samples [19]. This technique can give great accuracy, but it involves significant effort to run the malware files and does not guarantee the execution of the malware's entire code, thus the harmful section of the malware code may not be identified. Executing malware in controlled environments poses risks of unintentional infection and contamination if proper isolation measures are not in place. Dynamic analysis can be computationally intensive, requiring substantial resources and potentially impacting system performance during analysis [20]. Some extremely sophisticated malware can identify the analysis environment and change their behavior to avoid detection, decreasing the effectiveness of dynamic analysis.

C. FEATURE EXTRACTION TECHNIQUES

Several efforts have been made to adapt feature extraction techniques for the classification of malware. There have been surveys of numerous visualization approaches, including image processing for malware analysis. The visualization of malware as images [21], which presented the first studies on the viewing of binary data as images, improved the capabilities of text-based hex editors, and provided a method

for converting binary files into images called byteplots. A method for visualizing static malware samples as grayscale images, discovering that images from the same malware family appear very similar in structure and texturing for many malware families [22]. Figure 2 depicts the process of converting malware binary data to images. Malware binary files, which are in the form of bits are grouped into eight-bit vectors and then converted to grayscale images.

A recurring element of these initiatives is the transformation of binary malware samples into various image formats, followed by the implementation of image classification algorithms to categorize based on the image representation of the malware. Using local gray level cooccurrence matrices and global color moments, features from both grayscale and color byteplots are extracted and then sent to classifiers. This was tested on fifteen malware families and showed that scoring 97 % in accuracy, the combined feature sets outperformed either local or global features alone. There hasn't been a lot of study on malware classification using space-filling curves, in contrast to the byteplot related work that was mentioned [23]. An approach using Hilbert curves and a Self-Organizing Incremental Neural Network were employed to classify malware, very small sample size was used and it is insufficient to properly show the benefits of classifying malware using the Hilbert curve, Overfitting makes it difficult for the model to generalize to new samples [24]. SimHash keeps the malware's unique characteristics while encoding them to identical lengths. When converting SimHash bits to grayscale images, each SimHash value may be viewed as a pixel. SimHash can be improved by employing multiple cascade hash functions rather than a single hash result [25]. Bitmap Image Converter, a technique that accepts binary files from Windows Portable Executables (PE) as input and converts them into bitmap images in order to visualize them. In order to assess the similarity of the original binary files, each line of bitmap images has an entropy value, which is calculated by the entropy graph generator, and these values are used to create entropy graphs, it incorrectly classified malware binary files belonging to few families of malware [26]. The bytes transfer probability matrix based Markov images are fixed-size pixel matrices. It ignores the scaling issue when compared to grayscale images. Malware binaries are seen as a stream of bytes that may be visualized as a stochastic process [27].

D. DEEP LEARNING TECHNIQUES

Due to the improved feature learning ability of convolution neural network (CNN) from malware images, several researchers have tried to contribute elegant DL techniques to work on malware analysis. Figure 3 shows the basic block diagram or workflow diagram of the DL method. The malware datasets are pre-processed and split into training and testing data. Training data is applied on different DL algorithms and its performance is evaluated by applying test datasets. Models are built on the continuous evaluation of the performance of the model using quality metrics.

Malware detection and classification using a deep random forest approach, and a sliding window were proposed [28]. However, because it uses smaller versions of the input image for each sliding window, this uses more memory. To investigate informative aspects from the one-dimensional structure of binary executables, a byte-level 1D CNN model was presented [29]. While binary executables were being converted and resized to larger images, such as 128 × 128, this 1D CNN did not always perform better. And also suggest learning the useful characteristics from larger images may require a more complex model, like ResNet or EfficientNet. Deep Image Mal Detect (DIMD) model was proposed, consisting of Deep Neural Nework (DNN)and Long Short Term Memory (LSTM). The highest accuracy of this CNN-LSTM model was 96% with a cross-validation of 10 fold. Results in this model suggest a scope of improvement is required to develop DL complex model [30]. Experiments were conducted on different ResNet models and transfer learning for malware classification, with more complicated ResNet variants that did not yield a meaningful improvement in results [31]. An Alex Net and Resnet hybrid DNN was presented, integrating the two pre-trained networks to provide a feature vector and fully connected layers for categorization. The limitations of the model were the adversary's attacks were not tested using crafted inputs, and complexity in rises more hidden layers [32]. A model on Convolution Recurrence (ConRec), based on VGG16 and BiLSTM was used along with image augmentation on the malware samples. Model performance without image augmentation was less [33]. It was suggested to employ transfer learning-based architecture rather than class balancing techniques to identify malware from different families [34]. It utilized spatial attention created by CNN, as well as feedforward and dropout layers with less trainable parameters. On the Malimg benchmark dataset, the performance was examined and an accuracy of 97.68 % was obtained. Malware classification utilizing Co-Lab image, VGG 16, and Support vector machine (MalCVS) presented, with image feature extraction using a fine-tuned VGG16 model. Following that, the retrieved features are used to construct a multi-class SVM mode. Due to malware's ability to change or confuse the header field of PE files after packing, this visualization technique cannot classify malware that has been packaged [35]. Using transfer learning using ShuffleNet and DenseNet-201, in the final classification layer, an ensemble configuration of Support Vector Machines (SVM) with Optimal Error Correction Output Coding (ECOC). This model reported an accuracy of 99.14 % on the malimg dataset and 96.62 % on dumpware 10 datasets [37]. VGG16 and ResNet-50 ensemble of CNN architectures to extract malware image bottleneck characteristics, which were subsequently used to train SVM classifiers. The computational complexity of this model was high [41].

Static approach remains invaluable in identifying known threats and lays the groundwork for subsequent stages of analysis. Although dynamic malware analysis provides

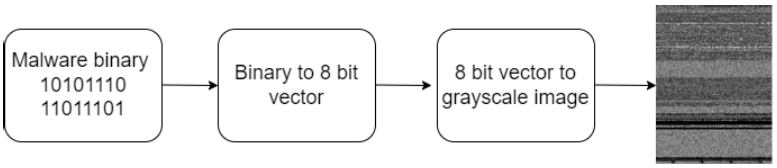

FIGURE 2. Visualizing malware as grayscale image.

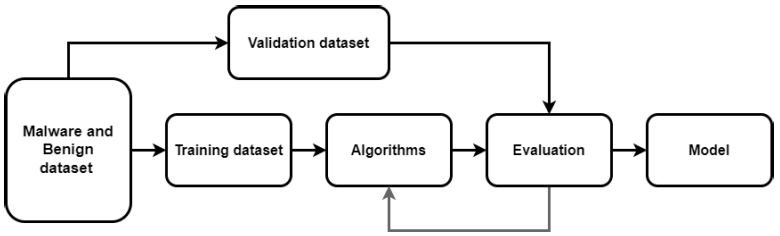

FIGURE 3. DL method of malware detection and classification.

useful insights into the behavior of harmful software, it is not a solution. Its limitations, including evasion techniques, time-dependent behavior, and the complexity of real-world environments. However, the true innovation of this study lies in the fusion of DL techniques with the established practices. DL with its ability to learn feature hierarchies independently from malware data, overcomes the constraints of handcrafted features. To overcome these limitations, a robust malware detection and classification model is required, and to address this issue, we propose the RMDNet architecture for efficiently differentiate classes of malware samples while retaining high accuracy on different malware datasets. Detailed description about proposed RMDNet is presented in Section III.

III. PROPOSED ARCHITECTURE

The proposed architecture, RMDNet for malware detection and categorization is described in this section. This section outlines the detailed architecture and provides insights into each layer's purpose and functionality.

A. PROPOSED RMDNet MODEL

The schematic diagram of proposed RMDNet CNN model is shown in Figure 4. This model has 3 stages, performing Convolution 2D and depthwise convolution in several layers. Conv2D captures spatial hierarchies and local patterns efficiently, allowing the network to learn the hierarchical representation of features in images. This is crucial for tasks like image recognition, where local features combine to form more complex patterns. Whereas separable convolution may not capture global relationships as effectively as Conv2D. Hence, Conv2D is used during the initial stages of convolution operations in the RMDNet.The model starts with an input layer that expects images of size 224 × 224 with three color channels (RGB) or one grayscale, depending on the input image channels. In stage 1 the model is branched into two sections, section I and section II. Section I starts with a convolutional layers (Conv2D) with 64 filters, each having a kernel size of 3 × 3 and ReLU activation. These layers are responsible for extracting high-level features from the input images. Batch normalization is applied after each Conv2D layer to accelerate training and stabilize the learning process of the model. Following batch normalization, max pooling is performed to downsample the feature maps spatially while preserving essential information. The output equation of Section I is given in equation1, the ReLU activation function is represented as φ, batch normalization is represented as $\mathbb{BN}$, and the max pooling layer as ϑ.

$$X_{sec1} = \vartheta(\mathbb{BN}(\varphi(X_{in} * W_{3\times3}))) \tag{1}$$

Section II begins with a Conv2D with 64 filters, followed by a depth-wise convolution. The depth-wise convolution performs separate convolutions for each input channel and then continues them. Batch normalization and max pooling are applied subsequently. Equation for section II is given in equation 2

$$X_{sec2} = \vartheta(\mathbb{BN}(\varphi(\varphi(X_{in} * W_{3\times3})) * W_{3\times3depthwise})) \tag{2}$$

The output of these two sections is concatenated together, represented in equation 3.

$$X_{stage1} = X_{sec1} \copyright X_{sec2} \tag{3}$$

The output of stage 1 is fed as an input to stage 2. It also has two sections, namely sections III and IV. Section III is identical to Section I of Stage 1 and is represented in equation 4. Here Conv2D is performed with a filter size of 128.

$$X_{sec3} = \vartheta(\mathbb{BN}(\varphi(X_{stage1} * W_{3\times3depthwise}))) \tag{4}$$

Section IV resembles Section II of the first stage, but with a deliberate omission of one convolutional layer. This

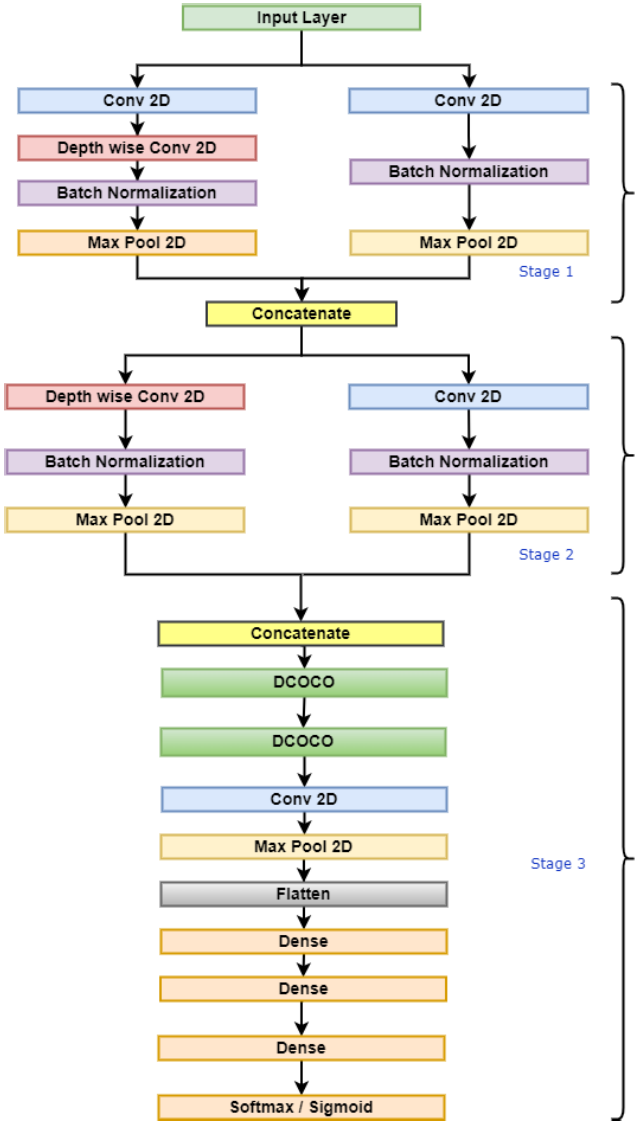

FIGURE 4. Proposed RMDNet architecture.

adjustment is made to effectively decrease the total number of parameters within the module, as described in the equation 5

$$X_{sec4} = \vartheta(\mathbb{BN}(\varphi(X_{stage1} * W_{3\times3}))) \tag{5}$$

The output of these two sections is concatenated together, represented in equation 6.

$$X_{stage2} = X_{sec3} \copyright X_{sec4} \tag{6}$$

In Conv2D, each filter is applied to all input channels, resulting in a large number of operations. However, in depthwise convolution, each channel is convolved separately, significantly reducing the number of operations. This makes depthwise convolution computationally more efficient than Conv2D. Depthwise convolution requires fewer parameters compared to Conv2D. There are a lot of learnable parameters in Conv2D since each filter has the same amount of parameters as the input channels. A single filter is used for each input channel in depthwise convolution, which minimizes the number of parameters and improves model performance. Depthwise convolution preserves spatial information better than Conv2D. In Conv2D, filters apply the same weights to all input channels, potentially mixing different types of features. Conversely, depth-wise convolution applies different filters to individual channels, which helps retain channel-specific information. However, Conv2D is adept at preserving spatial hierarchies and local patterns efficiently. This is crucial for tasks like image recognition, where local features combine to form more complex patterns. Hence, Conv2D is effectively used in different stages of the proposed model. Stage 3 consists of two DCOCO layers, followed by a convolution layer with 256 filters, with a kernel size of 3×3, max pooling. The output from the last max pooling layer is then flattened to 1d vectors and passed through two fully connected (dense) layers with 4096 units each, and ReLU activation is applied to learn high-level representations from the flattened features. Unprocessed neural network outputs are transformed into a vector of probabilities. Essentially, a probability distribution over the input classes using the equation (7) softmax activation function. where x_i is a standard exponential function for the input vector and x_j is the standard exponential function of the output vector.

$$softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{7}$$

The sigmoid $\sigma(\dot{a})$ given in (8) is used for binary datasets, and softmax is used as an activation function for malimg and dumpware 10 datasets to classify the images accordingly.

$$\sigma(\dot{a}) = \frac{1}{1 + e^{-\dot{a}}} \tag{8}$$

B. DCOCO BLOCK

The proposed DCOCO module has two paths shown in Figure 5. Let Xin be the input to the DCOCO block derived from the stage 2 output Xstage2, as indicated in equation (6). The input xin is processed using depthwise convolution, and the output of path 1 is given in equation (9). Depthwise convolution encourages feature separability by learning distinct features in each input channel. This can be particularly helpful when dealing with diverse and complex feature representations in multi-channel data like RGB images, where different channels represent different color information. When adapting pre-trained models to new tasks or datasets, depthwise convolution can be advantageous. Due to its parameter efficiency, depthwise convolution allows for faster fine-tuning and adaptation to new data, reducing the risk of overfitting when the target dataset is small.

$$Out_{path1} = \vartheta(\mathbb{BN}(\varphi(X_{in} * W_{3\times3\ depthwise}))) \tag{9}$$

In the second path, input X_{in} is processed through two depth-wise convolution layers, Let X_{2conv} be the output given in (10). Applying depthwise convolution twice allows the

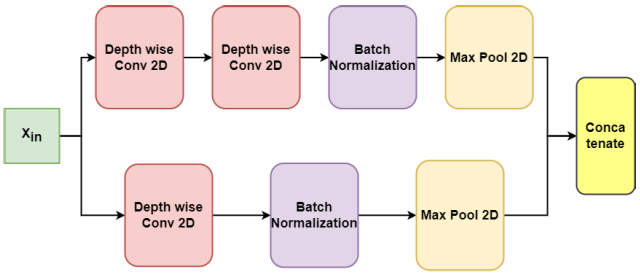

FIGURE 5. Block diagram of proposed DCOCO module.

model to capture hierarchical features from the input images.

$$X_{2conv} = \varphi(\varphi(X_{in} * W_{3\times3\ depthwise}) * W_{3\times3\ depthwise}) \tag{10}$$

The output of two depth-wise convolutions is processed through a sequence of ReLU, batch normalization, and max-pooling layers, as given in (11). BN is applied only once that is, after the second depth wise convolution, but it still contributes to training stability. Batch normalization normalizes activations within the feature maps, which helps in faster convergence and reduces the risk of vanishing/exploding gradients during training. This stabilization enhances the overall training process. performing the max pooling operation downsamples the feature maps, reducing their spatial dimensions. This downsampling reduces computational complexity and focuses the model's attention on the most important features, leading to more efficient feature extraction and faster inference.

$$Out_{path2} = \vartheta(\mathbb{BN}(X_{2conv})) \tag{11}$$

The outputs of the two paths, Out_{path1} and Out_{path2} are concatenated, and the output of the DCOCO model is represented in equation 12, The concatenation (©) operation combines the feature maps from both paths, enabling the model to learn from different levels of feature representations. This model is part of the DL architecture, where depth-wise convolutional layers, batch normalization, max-pooling, and concatenation operations are skillfully utilized to extract and learn relevant and hierarchical features from input feature maps. These operations are essential for the model's ability to perform sophisticated image processing tasks, such as malware classification and detection.

$$Out_{DCOCO} = (Out_{path1} © Out_{path2}) \tag{12}$$

This combination creates an efficient and effective DL architecture for malware image classification tasks. The model becomes adept at feature extraction, providing a robust and accurate representation for classification.

IV. TRAINING AND IMPLEMENTATION

A. DATASET

Binary and multiclass datasets are the two kinds that are employed. 48240 malware samples and binary visualization of images (dataset-1), Malimg (dataset-2), and Dumpware 10

TABLE 1. Description of malware datasets.

No.	Dataset	Number of classes	Total number of samples
1	Binary visualization of images	2	24,109
2	Malimg	25	9339
3	Dumpware 10	11	4294

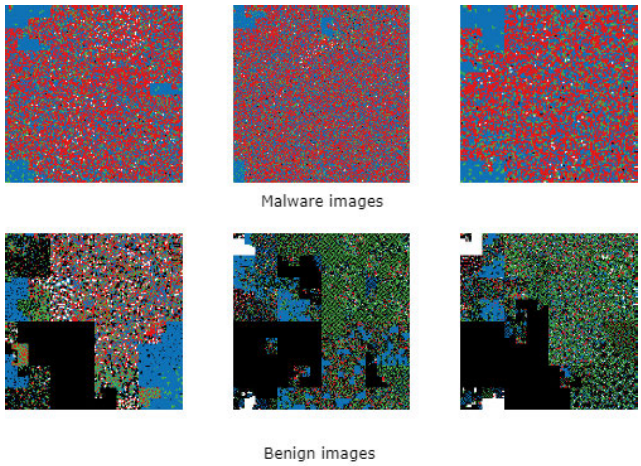

FIGURE 6. Binary malware samples (dataset 1).

(dataset-3) are used to carry out the work and Figures 6, 7, and 8 show the visualization of these datasets respectively. Dataset-1 contains 24,109 images out of which 11,919 images are malicious, and 12,190 images are benign and contain other infected files [36]. Dataset-2 consists of 9339 malicious images from 8 malware families, sub-categorized into 25 malware families, which are contained in dataset-2. The dataset-2 was constructed by converting malware binaries into a matrix. This matrix contains an unsigned, 8-bit integer and is seen as a grayscale image with values in the [0, 255] range, where 0 denotes black and 255 denotes white [30]. The dataset encompasses a diverse array of malware types, providing comprehensive coverage across a wide spectrum of malicious software, including viruses, trojans, and other malicious software. This diversity enables researchers and practitioners to analyze and classify various types of malware. Among these families, the largest one is the Allaple.A family, comprising a total of 2949 malware images. In contrast, the smallest malware class in the dataset is the Skintrim.N class, which includes 80 malware images. Consequently, dataset-3 [37] has 4294 images, comprising 3686 malware and 608 benign image samples. each malware family has a sample with a 224 × 224 final size and a single channel, containing ten malware and one benign class.

B. EVALUATION METRICS

The study evaluates RMDNet efficiency in classifying malware using five evaluation metrics: precision, recall, F1-score, accuracy and confusion matrix. It also compares the

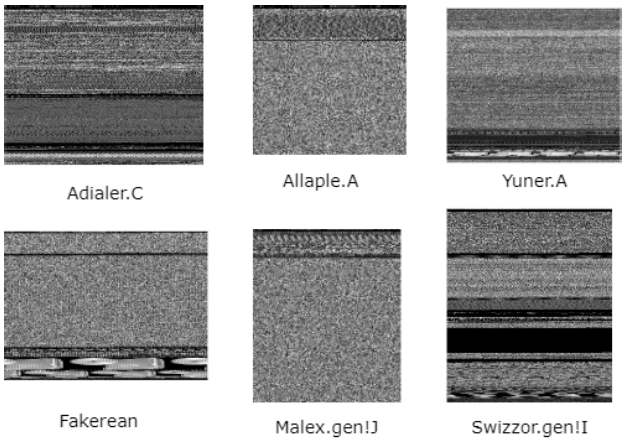

FIGURE 7. Malimg malware samples (dataset 2).

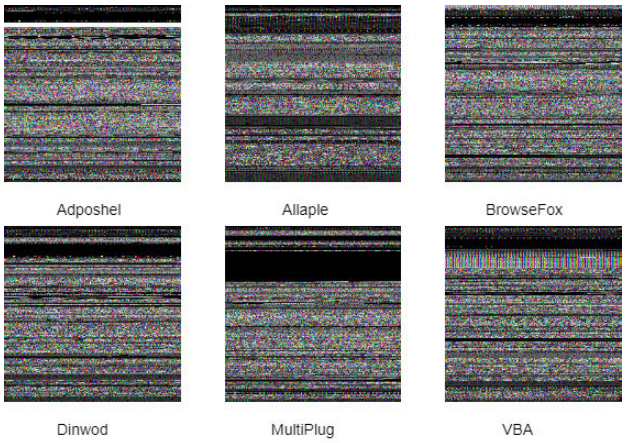

FIGURE 8. Dumpware 10 malware samples (dataset 3).

model's performance using four computational complexity metrics and computes the total number of trainable parameters. The confusion metrics predict True Positive (TP), False Positive (FP), True Negative (TN), and False Negatives (FN). Accuracy is a metric used to assess a classification model's correctness, given in equation (13). With precision, indicating the proportion of accurately predicted positive observations relative to the total number as shown in equation (14). Recall quantifies the percentage of correctly predicted positive outputs, evaluating the architecture's ability to acquire all positive outputs without missing any represented in equation (15). F1-score calculates the ratio of accurately predicted positive observations to the actual number of positive observations in the class represented in equation (16).

$$Acc = \frac{TNC + TPC}{FPC + FNC + TPC + TNC} \tag{13}$$

where TPC = True Positive Calculated, TNC = True Negative Calculated, FPC = False Positive Calculated and FNC = False Negative Calculated.

$$Pr = \frac{TPC}{FPC + TPC} \tag{14}$$

$$Re = \frac{TP}{FN + TP} \tag{15}$$

$$F1 = \frac{2 * Re * Pr}{Re + Pr} \tag{16}$$

where Re = Recall and Pr = Precision.

FLOPs measure the computational capability of a computing entity, while training and testing time represent the time required to train a DL architecture. Trainable parameters represent the bulkiness of the architecture. True Positive Rate (TPR) measures the proportion of positive instances correctly classified as positive by a model, presented in equation (17). While False Positive Rate (FPR) quantifies the ratio of negative instances incorrectly classified as positive given in equation (18).

$$TPR = \frac{TP}{FN + TP} \tag{17}$$

$$FPR = \frac{FP}{TN + FP} \tag{18}$$

C. TRAINING SETUP

This section gives a detailed overview of the training setup used for conducting experiments of on the proposed RMDNet model and benchmark models. The training process was performed on Kaggle's cloud based environment with a preconfigured Linux distribution, utilizing the GPU Kernel-Tesla P100. The NVIDIA Tesla P100 is a high-performance GPU with 16 GB High Bandwidth Memory (HBM-2), based on NVIDIA Pascal architecture having 3584 CUDA cores. The model is implemented using the Keras API framework and Tensorflow 2.11.0. The proposed and benchmark models utilized in this study are trained with a batch size of 32 for 40 epochs on the binary dataset, 100 epochs on the mailing dataset, and 100 epochs on the dumpware 10 dataset. It uses the Adam optimizer with early stopping. The best of these values are reported below after all of these models were trained five times on each of the three datasets. No data augmentation was performed on any of these datasets.

D. ABLATION STUDY

Ablation study is the systematic analysis of the impact of eliminating or changing certain components, features, or parameters inside a model to understand their individual contributions to the model's performance. The purpose is to determine the importance of each component in overall operation of the RMDNet.

1) Intermediate stage 3: In this abliation study elimanating the stage 1 and stage 2 of the RMDNet the model performance was studies on all the three datasets and the number of parameters were high.
2) Intermediate stage 2 and 3: In this abliation study only the stage 1 of the RMDNet was elimated to observe the model performance on all the three datasets, the number of parameters were reduced but their was no significant improvements were found in the performance metrics. The stage 1 of the proposed model initially perform the Conv2D operation. From the results, it is seen that eliminating the Conv2D operation reduces the accuracy and other performance

TABLE 2. Ablation study on binary dataset.

Model	Acc	Pre	Rec	F1	Train time	Test time	Flops	Parameters
Intermediate stage 3	0.9811	0.9914	0.9702	0.9807	55min	3.81 ms	0.91G	838,905,432
Intermediate stage 2 and 3	0.9885	0.9907	0.9861	0.9884	30m17s	5.1ms	2.87G	223,539,288
Proposed RMDNet	0.9915	0.9936	0.9890	0.9913	54min24s	4.5ms	5.31G	70,713,346

TABLE 3. Ablation study on malimg dataset.

Model	Acc	Pre	Rec	F1	Train time	Test time	Flops	Parameters
Intermediate stage 3	0.9778	0.9468	0.94	0.9439	9min55s	2.8ms	0.91G	838,980,991
Intermediate stage 2 and 3	0.9862	0.9680	0.97	0.9675	12min30s	4ms	2.6G	223,612,555
Proposed RMDNet	0.9926	0.9837	0.9812	0.9825	23min34s	4.61ms	5.08G	70,805,273

TABLE 4. Ablation study on dumpware 10 dataset.

Model	Acc	Pre	Rec	F1	Train time	Test time	Flops	Parameters
Intermediate stage 3	0.9589	0.9642	0.9585	0.9608	8min18s	3.3ms	0.91G	838,923,645
Intermediate stage 2 and 3	0.9638	0.9587	0.9599	0.9592	9min28s	5.1ms	2.6G	223,555,197
Proposed RMDNet	0.9819	0.9796	0.9811	0.9857	10min28s	4.77ms	5.08G	70,751,755

TABLE 5. Performance metrics on binary dataset.

Algorithms	Acc	Pre	Rec	F1
ResNeXt [39]	0.9440	0.9167	0.9752	0.9451
VGG 19 [34]	0.9825	0.9893	0.9752	0.9822
LiverNET [38]	0.9740	0.9677	0.9802	0.9739
EfficientNet B0 [39]	0.9483	0.9717	0.9223	0.9464
DenseNET 121 [40]	0.9431	0.9048	0.9890	0.9450
Proposed RMDNet	0.9915	0.9936	0.9890	0.9913

TABLE 6. Performance metrics on malimg dataset.

Algorithms	Acc	Pre	Rec	F1
ResNeXt [39]	0.9904	0.9778	0.9759	0.9768
VGG 19 [34]	0.9756	0.9550	0.9382	0.9465
LiverNET [38]	0.9823	0.9777	0.9802	0.9789
EfficientNet B0 [39]	0.9788	0.9494	0.9474	0.9484
DenseNET 121 [40]	0.9883	0.9720	0.9706	0.9713
Proposed RMDNet	0.9926	0.9837	0.9812	0.9825

metrics of the model. To capture the spatial hierarchies and local patterns efficiently the Conv2D opertaion must be performed in the initial layers over the seperable convolution method. The table 2 gives the ablation study on binary dataset, table 3 and 4 gives the ablation results of the malimg and dumpware 10 datasets respectively.

V. RESULTS AND DISCUSSIONS

In this section, we present the results of our comprehensive study. Our study aimed to develop a robust malware detection & classification model on malware datasets. Compare the results with the state-of-the-art models using different performance metrics, as detailed in sections V-A and V-B.

A. COMPARISON WITH BENCHMARK MODELS

Performance metrics play a crucial role and DL pipeline, providing valuable insights into progress and quantifying it numerically. Regardless of the type of model employed, be it a statistical model or a neural network approach such as DNN or CNN, an appropriate metric is indispensable for evaluating performance. Numerous evaluation metrics exist for DL problems, and this discussion will explore some of the popular ones and delve into the insights they provide regarding model performance. knowing models perceive the data is crucial for gaining valuable insights into its strengths, weaknesses, and overall effectiveness. By examining these metrics, a deeper understanding of the model's behavior and decisions about its optimization and potential enhancements can be determined. Using the evaluation metrics, the performance of the proposed RMDNet model is compared to ResNeXt [39], VGG 19 [34], LiverNET [38], EfficientNet-B0 [39], and DenseNET 121 [40]. After training and testing on all the mentioned algorithms for all three datasets obtained accuracy, precision, recall, and f1-score are listed in Table 5 for binary dataset, Table 6 for malimg dataset, and Table 7 for dumpware 10 dataset.

TABLE 7. Performance metrics on dumpware 10 dataset.

Algorithms	Acc	Pre	Rec	F1
ResNeXt [39]	0.9299	0.9080	0.9133	0.9090
VGG 19 [34]	0.9299	0.9367	0.9187	0.9250
EfficientNet B0 [39]	0.9118	0.9015	0.9012	0.8946
DenseNET 121 [40]	0.9751	0.9663	0.9676	0.9666
Proposed RMDNet	0.9819	0.9796	0.9811	0.9857

Confusion matrix for binary, malimg, and dumpware 10 datasets are shown in figures 9, 10, and 11 respectively.

Table 8 and 9 presents the TPR and FPR corresponding to each malware class for different DL algorithms trained on the Malimg dataset and dumpware 10 datasets respectively. Loss and accuracy are essential values to consider while training DL models. We can check whether our model

TABLE 8. TPR and FPR of different models on malimg dataset.

Model	ResNeXt [43]		VGG 19 [34]		LiverNET [38]		ENetB0 [39]		DNet121 [40]		RMDNet	
Malware class	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
1	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
2	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
3	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.001	1.0	0.0	1.0	0.0
4	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
5	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
6	0.454	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
7	0.7333	0.006	0.933	0.002	0.933	0.003	0.867	0.005	0.7333	0.006	0.866	0.001
8	0.85	0.002	1.0	0.0	0.95	0.0	0.9	0.002	0.95	0.03	1.0	0.0
9	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
10	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
11	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
12	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
13	1.0	0.002	1.0	0.002	1.0	0.002	1.0	0.002	0.991	0.0	1.0	0.0022
14	0.894	0.0	0.894	0.0	0.895	0.001	0.895	0.001	0.894	0.0	0.894	0.001
15	1.0	0.0	1.0	0.001	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
16	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
17	0.928	0.0	0.928	0.0	0.929	0.0	0.857	0.001	0.928	0.0	0.929	0.0
18	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
19	1.0	0.0	1.0	0.001	1.0	0.0	1.0	0.001	1.0	0.0	1.0	0.0
20	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
21	1.0	0.002	1.0	0.003	0.846	0.002	0.692	0.004	0.846	0.002	1.0	0.001
22	0.642	0.004	0.642	0.001	0.714	0.003	0.5	0.003	0.714	0.003	0.857	0.001
23	0.951	0.0	1.0	0.0	1.0	0.0	0.976	0.0	1.0	0.0	1.0	0.0
24	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
25	1.0	0.006	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0

TABLE 9. TPR and FPR of different models on dumpware 10 dataset.

Model	ResNeXt [43]		VGG 19 [34]		ENetB0 [39]		DNet121 [40]		RMDNet	
Malware class	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
1	1.00	0.00	1.00	0.00	1.00	0.01	1.00	0.00	1.00	0.00
2	0.93	0.01	0.93	0.00	0.93	0.01	0.98	0.00	0.98	0.00
3	0.98	0.01	0.96	0.01	0.96	0.01	0.98	0.00	0.98	0.00
4	0.86	0.02	0.86	0.01	0.90	0.03	0.90	0.00	1.00	0.00
5	0.89	0.00	1.00	0.00	0.79	0.00	0.95	0.01	1.00	0.00
6	0.71	0.01	0.71	0.00	0.71	0.00	0.93	0.00	0.93	0.00
7	0.94	0.01	0.94	0.00	0.96	0.02	0.98	0.00	0.98	0.00
8	0.88	0.00	0.84	0.02	0.84	0.01	0.98	0.00	0.96	0.01
9	0.87	0.01	0.89	0.03	0.79	0.01	0.97	0.01	0.98	0.01
10	0.98	0.00	0.98	0.00	0.98	0.00	0.98	0.00	0.98	0.00
11	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00	1.00	0.00

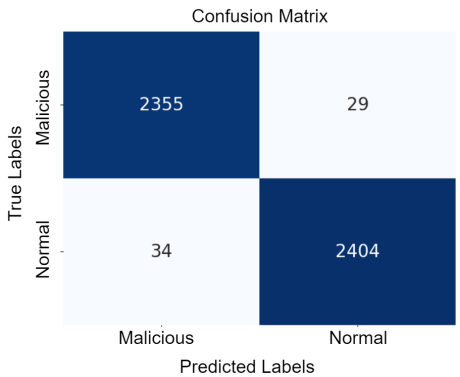

FIGURE 9. Confusion matrix of binary dataset.

is over-fitting, under-fitting, or better-fitting by taking a closer look at the train and validation accuracy plot of the models. The confusion matrix also plays a critical role in finding accuracy, robustness, and many more performance parameters. Figures 12, 13, and 14 show the learning curves of RMDNet on binary, malimg, and dumpware 10 datasets respectively. In the confusion matrix of malimg dataset shown in figure 10, and also in table 8 refer T = True, P = Predicted, columns/rows 1 correspond to Adialer.C, 2 = Agent.FY1, 3 = Allaple.A, 4 = Allaple.L, 5 = Alueron.gen!J, 6 = Autorun.K, 7 = C2Lop.gen!G, 8 = C2Lop.P, 9 = Dialplatform.B, 10 = Dontovo.A, 11 = Fakerean, 12 = Instantaccess, 13 = Lolyda.AA 1, 14 = LolydaAA 2, 15 = LolydaAA 3, 16 = LolydaAT, 17 = Malex.gen!J, 18 = Obfuscator.AD, 19 = Rbot!gen, 20 = Skintrim.N, 21 = Swizzor.gen!E, 22 = Swizzor.gen!I, 23 = VB.AT, 24 = Wintrim.BX, 25 = Yuner.A. Similarly, in the confusion matrix of dumpware 10 dataset shown in figure 11, and also in table 9 refer T = True, P = Predicted, columns/ rows 1 correspond to Malware class Adposhel, 2 = Allaple, 3 = Amonetize, 4 = AutoRun, 5 = BrowseFox, 6 = Dinwod, 7 = InstallCore, 8 = MultiPlug, 9 = Other, 10 = VBA, 11 = Vilsel.

B. COMPUTATION COMPLEXITY AND ANALYSIS

The evaluation of model complexity is important in understanding the performance characteristics and resource

T/P	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	296	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	160	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	13	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	44	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	22	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	2	17	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	13	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0	0	0
22	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	12	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80

FIGURE 10. Confusion matrix of malimg dataset.

T/P	1	2	3	4	5	6	7	8	9	10
1	47	0	0	0	0	0	0	0	0	0
2	0	44	0	0	0	0	1	0	0	0
3	0	0	44	0	0	0	0	1	0	0
4	0	0	0	21	0	0	0	0	0	0
5	0	0	0	0	19	0	0	0	0	0
6	0	0	0	0	0	13	0	0	1	0
7	0	0	0	0	0	0	47	1	0	0
8	0	0	0	1	0	0	0	48	1	0
9	0	0	0	0	0	1	0	0	61	0
10	0	0	0	0	0	0	0	0	1	50

FIGURE 11. Confusion matrix of dumpware 10 dataset.

TABLE 10. Complexity matrices of different models on binary dataset.

Algorithms	Flops	Training Time	Testing Time (per sample)	Total Trainable Parameter
ResNeXt [39]	7.77 G	1h 39min	5.37 ms	56,305,594
VGG 19 [34]	31.0 G	1h 11min	5.57 ms	134,268,738
LiverNET [38]	7.8 G	5h 15min	14.51 ms	2,088,702
EfficientNet B0 [39]	0.8 G	50min 19s	4.85 ms	4,010,110
DenseNET 121 [40]	5.7 G	1h 16min	8.33 ms	6,955,906
Proposed RMDNet	5.31G	54min 24s	4.50 ms	70,713,346

TABLE 11. Complexity matrices of different models on malimg dataset.

Algorithms	Flops	Training Time	Testing Time (per sample)	Total Trainable Parameter
ResNeXt [39]	7.61 G	1hr 25min	5.52 ms	56,322,345
VGG 19 [34]	30.8 G	20min 30s	5.82 ms	134,361,817
LiverNET [38]	7.75 G	3h 30min	5.60 ms	2,094,037
EfficientNet B0 [39]	0.79 G	36min 41s	4.57 ms	4,038,997
DenseNET 121 [40]	5.54 G	18min 45s	5.67 ms	6,973,209
Proposed RMDNet	5.08 G	23min 34s	4.61 ms	70,805,273

requirements of a model. For having a comprehensive understanding of RMDNet model capabilities, Conducted

TABLE 12. Complexity matrices of different models on dumpware 10 dataSet.

Algorithms	Flops	Training Time	Testing Time (per sample)	Total Trainable Parameter
ResNeXt [39]	7.61 G	43min 23s	7.57 ms	56,308,331
VGG 19 [34]	30.8 G	3min 55s	8.64 ms	134,304,459
EfficientNet B0 [39]	0.79 G	17min 18s	7.76 ms	4,021,063
DenseNET 121 [40]	5.54 G	20min 13s	11.69 ms	6,958,859
Proposed RMDNet	5.08 G	10min 28s	4.77 ms	70,751,755

an in-depth analysis and calculated several key complexity metrics. The results of these calculations on binary, malimg, and dumpware 10 datasets are given in table 10, 11 and 12 respectively. There are some minor differences in complexity matrices when used to train on the malimg dataset and dumpware 10 datasets because malimg and dumpware 10 consist of grayscale, which takes less computation while training and binary classification dataset is a three-channel RGB image takes more computation, hence the complexity matrices are high. The training time and testing time columns represent the duration it takes to train the DL algorithms and calculate a prediction for one sample image, respectively on all datasets. The FLOPs column indicates the number of floating point operations performed during the model execution. Finally, the trainable parameters column denotes the total count of adjustable parameters in the architecture during training. These complex metrics provide valuable insights into the size, computational demands, and efficiency of each algorithm. By considering these metrics, decisions can be made regarding the suitability and scalability of the algorithms for our specific use case. By incorporating these complexity metrics, we aim to provide a comprehensive overview of our model capabilities and the computational resources required for its training and testing phases.

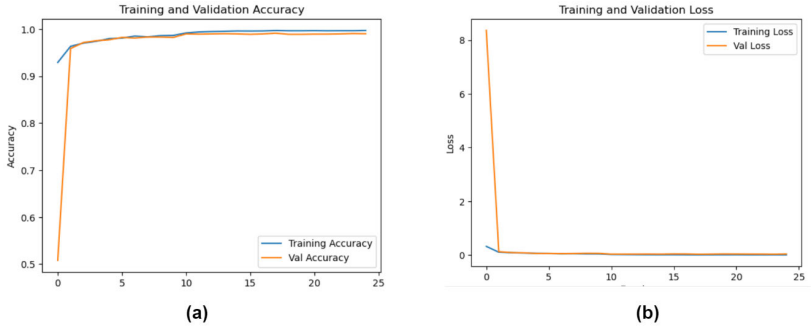

FIGURE 12. Learning curve of the binary dataset. Training and validation (a) accuracy graph and (b) loss graph.

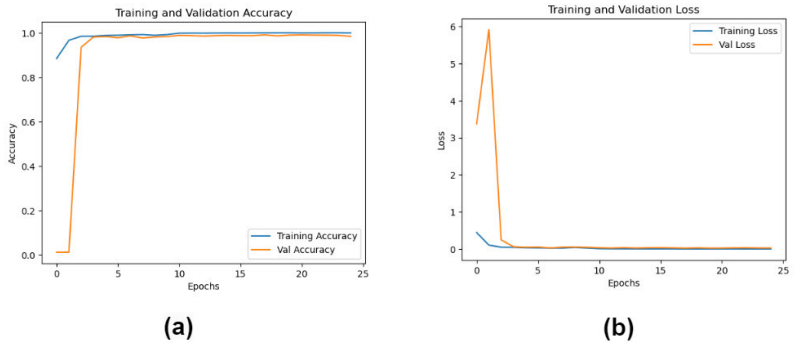

FIGURE 13. Learning curve of mailing dataset. Training and validation (a) accuracy graph and (b) loss graph.

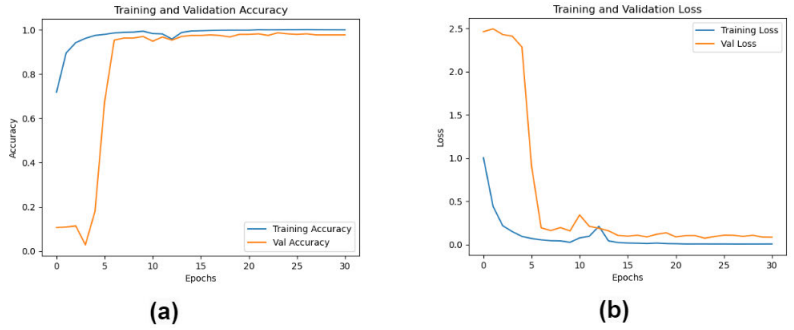

FIGURE 14. Learning curve of dumpware 10 dataset. Training and validation (a) accuracy graph and (b) loss graph.

Figures 12, 13, and 14 show the learning curves of RMDNet on binary, malimg, and dumpware 10 datasets respectively. Training, validation accuracy and loss of dataset 1 and 2 are similar compared to dataset 3. The model is converging prior 15 epochs.

VI. CONCLUSION

A novel RMDNet model was proposed for accurately identification of malware variants. The performance of several existing DL models, including VGG19, ResNeXt, LiverNet, EfficientNet B0, and DenseNet was compared with the proposed RMDNet. After conducting a thorough evaluation and analysis, the results demonstrated that the presented RMDNet model excelled overall in terms of quality metrics irrespective of the malware datasets. This demonstrates that proposed RMDNet architecture is capable of identifying and classifying image based malware in more accurately. The second best result for the binary dataset was with VGG 19, for the malimg dataset was with ResNeXt, and for dumpware 10 was with DenseNET 121. By leveraging domain-specific knowledge and experimenting with different architectural components, it was possible to design and develop a robust deep learning model which was able to effectively captured the essential features and designs of malware images. The results obtained from proposed RMDNet model highlighted the importance of exploring custom architectures

and leveraging domain expertise when dealing with complex classification tasks, such as image-based malware detection. While existing DL models provide strong baselines, tailoring the architecture to the specific task can lead to significant performance improvements. It was worth noting that the self-created model's success does not diminish the significance of the existing DL models. These models have undergone considerable study and validation in a variety of domains, making them useful tools for categorization tasks. However, in order to obtain the optimum performance, the special needs of image-based malware detection demanded the development of a customized DL architecture. This RMDNet shows the effectiveness of a self-designed DL architecture in binary and multi-class classification of image-based malware. The superior performance of the model highlighted the importance of tailoring the architecture to the specific task at hand. The findings of this work contribute to the ongoing research and development of more accurate and efficient methods for malware recognition and categorization using DL techniques.

REFERENCES

[1] D. B. Rawat, R. Doku, and M. Garuba, "Cybersecurity in big data era: From securing big data to data-driven security," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 2055–2072, Nov. 2021, doi: 10.1109/TSC.2019.2907247.

[2] Y. Lu and L. D. Xu, "Internet of Things (IoT) cybersecurity research: A review of current research topics," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2103–2115, Apr. 2019, doi: 10.1109/JIOT.2018.2869847.

[3] M. Roopak, G. Yun Tian, and J. Chambers, "Deep learning models for cyber security in IoT networks," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2019, pp. 0452–0457, doi: 10.1109/CCWC.2019.8666588.

[4] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-Centric Comput. Inf. Sci.*, vol. 8, no. 1, Dec. 2018, doi: 10.1186/s13673-018-0125-x.

[5] S. Kumar, P. Tiwari, and M. Zymbler, "Internet of Things is a revolutionary approach for future technology enhancement: A review," *J. Big Data*, vol. 6, no. 1, Dec. 2019, doi: 10.1186/s40537-019-0268-2.

[6] Y. Li and Q. Liu, "A comprehensive review study of cyber-attacks and cyber security; emerging trends and recent developments," *Energy Rep.*, vol. 7, pp. 8176–8186, Nov. 2021, doi: 10.1016/j.egyr.2021.08.126.

[7] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Comput. Secur.*, vol. 81, pp. 123–147, Mar. 2019, doi: 10.1016/j.cose.2018.11.001.

[8] Y. Ye, L. Chen, S. Hou, W. Hardy, and X. Li, "DeepAM: A heterogeneous deep learning framework for intelligent malware detection," *Knowl. Inf. Syst.*, vol. 54, no. 2, pp. 265–285, Feb. 2018, doi: 10.1007/s10115-017-1058-9.

[9] K. Shaukat, S. Luo, and V. Varadharajan, "A novel deep learning-based approach for malware detection," *Eng. Appl. Artif. Intell.*, vol. 122, Jun. 2023, Art. no. 106030, doi: 10.1016/j.engappai.2023.106030.

[10] M. S. Akhtar and T. Feng, "Detection of malware by deep learning as CNN-LSTM machine learning techniques in real time," *Symmetry*, vol. 14, no. 11, p. 2308, Nov. 2022, doi: 10.3390/sym14112308.

[11] M. E. Ahmed, S. Nepal, and H. Kim, "MEDUSA: Malware detection using statistical analysis of system's behavior," in *Proc. IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, Nepal, Oct. 2018, pp. 272–278, doi: 10.1109/CIC.2018.00044.

[12] M. Alazab, S. Venkataraman, and P. Watters, "Towards understanding malware behaviour by the extraction of API calls," in *Proc. 2nd Cybercrime Trustworthy Comput. Workshop*, Jul. 2010, pp. 52–59, doi: 10.1109/CTC.2010.8.

[13] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on OpCode patterns," *Secur. Informat.*, vol. 1, no. 1, Dec. 2012, doi: 10.1186/2190-8532-1-1.

[14] S. Jeon and J. Moon, "Malware-detection method with a convolutional recurrent neural network using opcode sequences," *Inf. Sci.*, vol. 535, pp. 1–15, Oct. 2020, doi: 10.1016/j.ins.2020.05.026.

[15] Y. Ding, W. Dai, S. Yan, and Y. Zhang, "Control flow-based opcode behavior analysis for malware detection," *Comput. Secur.*, vol. 44, pp. 65–74, Jul. 2014, doi: 10.1016/j.cose.2014.04.003.

[16] 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, doi: 10.1155/2015/659101.

[17] C.-Y. Wang, C.-Y. You, F.-H. Hsu, C.-H. Lee, C.-H. Liu, and Y. Zhuang, "SMS observer: A dynamic mechanism to analyze the behavior of SMS-based malware," *J. Parallel Distrib. Comput.*, vol. 156, pp. 25–37, Oct. 2021, doi: 10.1016/j.jpdc.2021.05.004.

[18] M. Tang and Q. Qian, "Dynamic API call sequence visualisation for malware classification," *IET Inf. Secur.*, vol. 13, no. 4, pp. 367–377, Jul. 2019, doi: 10.1049/iet-ifs.2018.5268.

[19] E. Amer and I. Zelinka, "A dynamic windows malware detection and prediction method based on contextual understanding of API call sequence," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101760, doi: 10.1016/j.cose.2020.101760.

[20] J. Ragaventhiran, P. Vigneshwaran, M K. Mallikarjun, S T. Ahmed, R. Prabu, and P. Megantoro, "An unsupervised malware detection system for windows based system call sequences," *Malaysian J. Comput. Sci.*, pp. 79–92, 2022, doi: 10.22452/mjcs.sp2022no2.7.

[21] J. Homer, A. Varikuti, X. Ou, and M. A. McQueen, "Improving attack graph visualization through data reduction and attack grouping," 2008, doi: 10.1007/978-3-540-85933-8_7.

[22] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images," in *Proc. 8th Int. Symp. Visualizat. Cyber Secur.*, Jul. 2011, doi: 10.1145/2016904.2016908.

[23] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, "Malware visualization for fine-grained classification," *IEEE Access*, vol. 6, pp. 14510–14523, 2018, doi: 10.1109/ACCESS.2018.2805301.

[24] I. Baptista, "Binary visualisation for malware detection," *Plymouth Student Scientist*, vol. 11, no. 1, pp. 223–237, 2018.

[25] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *Int. J. Inf. Secur.*, vol. 14, no. 1, pp. 1–14, Feb. 2015, doi: 10.1007/s10207-014-0242-0.

[26] S. O'Shaughnessy and S. Sheridan, "Image-based malware classification hybrid framework based on space-filling curves," *Comput. Secur.*, vol. 116, May 2022, Art. no. 102660, doi: 10.1016/j.cose.2022.102660.

[27] B. Yuan, J. Wang, D. Liu, W. Guo, P. Wu, and X. Bao, "Byte-level malware classification based on Markov images and deep learning," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101740, doi: 10.1016/j.cose.2020.101740.

[28] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent vision-based malware detection and classification using deep random forest paradigm," *IEEE Access*, vol. 8, pp. 206303–206324, 2020, doi: 10.1109/ACCESS.2020.3036491.

[29] W.-C. Lin and Y.-R. Yeh, "Efficient malware classification by binary sequences with one-dimensional convolutional neural networks," *Mathematics*, vol. 10, no. 4, p. 608, Feb. 2022, doi: 10.3390/math10040608.

[30] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46717–46738, 2019, doi: 10.1109/ACCESS.2019.2906934.

[31] J. Kim, A. Sim, J. Kim, K. Wu, and J. Hahm, "Transfer learning approach for botnet detection based on recurrent variational autoencoder," in *Proc. 3rd Int. Workshop Syst. Netw. Telemetry Analytics*, Jun. 2020, doi: 10.1145/3391812.3396273.

[32] Ö. Aslan and A. A. Yilmaz, "A new malware classification framework based on deep learning algorithms," *IEEE Access*, vol. 9, pp. 87936–87951, 2021, doi: 10.1109/ACCESS.2021.3089586.

[33] A. Mallik, A. Khetarpal, and S. Kumar, "ConRec: Malware classification using convolutional recurrence," *J. Comput. Virol. Hacking Techn.*, vol. 18, no. 4, pp. 297–313, Feb. 2022, doi: 10.1007/s11416-022-00416-3.

[34] M. J. Awan, O. A. Masood, M. A. Mohammed, A. Yasin, A. M. Zain, R. Damaševičius, and K. H. Abdulkareem, "Image-based malware classification using VGG19 network and spatial convolutional attention," *Electronics*, vol. 10, no. 19, p. 2444, Oct. 2021, doi: 10.3390/electronics10192444.

[35] M. Xiao, C. Guo, G. Shen, Y. Cui, and C. Jiang, "Image-based malware classification using section distribution information," *Comput. Secur.*, vol. 110, Nov. 2021, Art. no. 102420, doi: 10.1016/j.cose.2021.102420.

[36] B. Saridou, J. R. Rose, S. Shiaeles, and B. Papadopoulos, "SAGMAD—A signature agnostic malware detection system based on binary visualisation and fuzzy sets," *Electronics*, vol. 11, no. 7, p. 1044, Mar. 2022, doi: 10.3390/electronics11071044.
[37] D. Vasan, M. Alazab, S. Wassan, B. Safaei, and Q. Zheng, "Image-based malware classification using ensemble of CNN architectures (IMCEC)," *Comput. Secur.*, vol. 92, May 2020, Art. no. 101748, doi: 10.1016/j.cose.2020.101748.
[38] W. K. Wong, F. H. Juwono, and C. Apriono, "Vision-based malware detection: A transfer learning approach using optimal ECOC-SVM configuration," *IEEE Access*, vol. 9, pp. 159262–159270, 2021, doi: 10.1109/ACCESS.2021.3131713.
[39] J. H. Go, T. Jan, M. Mohanty, O. P. Patel, D. Puthal, and M. Prasad, "Visualization approach for malware classification with ResNeXt," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2020, pp. 1–7, doi: 10.1109/CEC48606.2020.9185490.
[40] A. A. Aatresh, K. Alabhya, S. Lal, J. Kini, and P. P. Saxena, "LiverNet: Efficient and robust deep learning model for automatic diagnosis of sub-types of liver hepatocellular carcinoma cancer from H&E stained liver histopathology images," *Int. J. Comput. Assist. Radiol. Surg.*, vol. 16, no. 9, pp. 1549–1563, Sep. 2021, doi: 10.1007/s11548-021-02410-4.
[41] R. Chaganti, V. Ravi, and T. D. Pham, "Image-based malware representation approach with EfficientNet convolutional neural networks for effective malware classification," *J. Inf. Secur. Appl.*, vol. 69, Sep. 2022, Art. no. 103306, doi: 10.1016/j.jisa.2022.103306.
[42] J. C. Kimmell, M. Abdelsalam, and M. Gupta, "Analyzing machine learning approaches for online malware detection in cloud," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, Aug. 2021, pp. 189–196, doi: 10.1109/SMARTCOMP52413.2021.00046.

S. PUNEETH received the B.E. degree from VVIET, Mysuru, Karnataka, India, and the M.Tech. degree in VLSI design and embedded systems from the P. E. S. College of Engineering, Mandya, Karnataka, in 2013. He is currently a Research Scholar (External Registrant) with the Department of Electronics and Communication Engineering, National Institute of Technology Karnataka at Surathkal, Surathkal, India. He is also an Assistant Professor with the Department of Electronics and Communication Engineering, The National Institute of Engineering, Mysuru, Karnataka. His research interests include the Internet of Things, machine learning, deep learning, and cyber security.

SHYAM LAL (Senior Member, IEEE) received the M.Tech. degree in electronics and communication engineering from the National Institute of Technology, Kurukshetra, Haryana, India, in 2007, and the Ph.D. degree in image processing from the Birla Institute of Technology, Mesra, Ranchi, India, in 2013. He has been an Associate Professor with the Department of Electronics and Communication Engineering, National Institute of Technology Karnataka (NITK) at Surathkal, Surathkal, India. He has published around 100 research papers in reputed journals and conferences. His research interests include machine learning, deep learning, cyber security, digital image processing, satellite remote sensing, and medical image processing.

MAHENDRA PRATAP SINGH (Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur (IIT Kharagpur). He is currently an Assistant Professor with the Department of Computer Science and Engineering, National Institute of Technology Karnataka (NITK) at Surathkal, Surathkal, India. He has published more than 21 research papers in reputed international journals and conferences. His research interests include network security, information security, and privacy.

B. S. RAGHAVENDRA (Member, IEEE) received the B.E. degree from the R.V. College of Engineering (RVCE), Bangalore University, Bengaluru, India, the M.Tech. degree from the National Institute of Technology Karnataka (NITK) at Surathkal, Surathkal, India, and the Ph.D. degree from ECE, Indian Institute of Science, Bangalore, in 2011. He was a member of the Research Staff at Samsung India Bangalore. He joined the Department of ECE, NITK Suarthkal, as an Assistant Professor in the year 2013, and currently is an Associate Professor. His research interests include applied signal processing, sensor signal processing, pattern recognition, machine learning, data analytics, and deep learning, with applications in biomedical and remote sensing fields.