Article

Memory Visualization-Based Malware Detection Technique

Syed Shakir Hameed Shah 1,*, Norziana Jamil 1 and Atta ur Rehman Khan 2

1 Institute of Energy Infrastructure, College of Computing and Informatics, Universiti Tenaga Nasional, Kajang 43000, Malaysia
2 College of Engineering and IT, Ajman University, Ajman 346, United Arab Emirates
* Correspondence: pt20914@student.uniten.edu.my

Abstract: Advanced Persistent Threat is an attack campaign in which an intruder or team of intruders establishes a long-term presence on a network to mine sensitive data, which becomes more dangerous when combined with polymorphic malware. This type of malware is not only undetectable, but it also generates multiple variants of the same type of malware in the network and remains in the system’s main memory to avoid detection. Few researchers employ a visualization approach based on a computer’s memory to detect and classify various classes of malware. However, a preprocessing step of denoising the malware images was not considered, which results in an overfitting problem and prevents us from perfectly generalizing a model. In this paper, we introduce a new data engineering approach comprising two main stages: Denoising and Re-Dimensioning. The first aims at reducing or ideally removing the noise in the malware’s memory-based dump files’ transformed images. The latter further processes the cleaned image by compressing them to reduce their dimensionality. This is to avoid the overfitting issue and lower the variance, computing cost, and memory utilization. We then built our machine learning model that implements the new data engineering approach and the result shows that the performance metrics of 97.82% for accuracy, 97.66% for precision, 97.25% for recall, and 97.57% for f1-score are obtained. Our new data engineering approach and machine learning model outperform existing solutions by 0.83% accuracy, 0.30% precision, 1.67% recall, and 1.25% f1-score. In addition to that, the computational time and memory usage have also reduced significantly.

Keywords: malware analysis; polymorphic malware; memory analysis; machine learning; denoising filters; wavelet transform; computer vision; advanced persistent threat; energy security

1. Introduction

Cybersecurity is the protection of internet-connected devices and services from malicious attacks by hackers, spammers, and cybercriminals. As billions of network devices connect and share potentially sensitive data, there is a need for greater security [1]. The practice is used by companies to protect against phishing schemes, ransomware attacks, identity theft, data breaches, and financial losses. To launch the attack, these cybercriminals use malicious files, often called malware. Malware is malicious software that is designed with the specific intention of causing disruption or leakage of information. There are many types of malware, such as spyware, viruses, bots, trojans, file-less malware, adware, ransomware, worms, rootkits, keyloggers, and many more [2]. Malware can be categorized into three types, including polymorphic, metamorphic, and obfuscation [3]. In polymorphic, the malicious file continuously modifies its identifying characteristics to evade detection. One component of polymorphic malware is constantly evolving, while the other remains constant. This type of malware is difficult to detect using conventional methods such as signature-based detection. It becomes extremely dangerous when an APT actor uses polymorphic malware that resides in the system’s main memory to avoid detection.

Malware has increased exponentially through the daily creation of new malware and variants of existing malware. According to a report [4], every day, 338,860 new malware
and potentially unwanted applications (PUA) are created. Similarly, another report [5] stated that 450,000 new malware and PUA are created every day. These malicious malware target a variety of devices, including mobile phones and computers. The study shows mobile malware attacks also significantly increased in recent years [6]. Mobile devices have detected 6,463,414 instances of malware, adware, and riskware, according to Kaspersky’s first-quarter report [7]. These malicious programs are either associated with mobile banking trojans or ransomware trojans. According to a report [8], a total of 1,216,350,437 attacks on Personal Computers (PC) are detected from online resources around the world. The statistics shows that Microsoft Windows operating system is the easiest target for malware developers [5].

1.1. Background

To develop a detection mechanism, it is vital to understand the malware behaviors that determine the intention of the malware developer. Much research conducted in the past analyzed suspicious files and extracted important artifacts that perform distrustful activities. The process of understanding malware behavior is called malware analysis [9]. There are different analysis techniques called static, dynamic, hybrid, and memory forensics/analysis [10]. In static analysis, it is not necessary to run the code to perform the analysis [11]. Instead, the static analysis looks through the file to see if it contains any indicators of malicious intent. Identifying malicious infrastructure, libraries, or packed files can be facilitated by using this technique. File names, hashes, strings, IP addresses, domain names, and information contained within file headers are some of the technical indicators that can be used to determine whether or not a file contains malicious content. In addition, tools such as disassemblers and network analyzers may be used to examine malware without executing it to gain information about its activity. Since static analysis does not execute the code, complex malware with dangerous runtime behavior might elude detection [12,13]. For instance, if a file emits a dynamic string that downloads a malicious file, it would be undetectable by simple static analysis. To get a better understanding of the behavior of a file, businesses have turned to dynamic analysis.

In a sandbox, suspicious harmful code is executed in a protected environment [14]. This closed system enables security experts to monitor the activity of malware without the danger of infection or egress into the company network. The dynamic analysis gives greater visibility to threat hunters and incident responders, allowing them to assess the exact nature of a threat. Automated sandboxing reduces the time necessary to reverse-engineer a file to find harmful code. The challenge with dynamic analysis is that opponents are smart and know of the existence of sandboxes [15]; therefore, they are good at spotting them. To fool a sandbox, opponents hide latent code into their bodies.

Basic static analysis is not a reliable way for identifying advanced malicious code and may sometimes circumvent sandbox protection. By integrating static and dynamic analysis, the hybrid analysis offers security teams the advantages of both techniques [10]. It may first identify malicious code that is trying to obscure itself and then extract a large number of indicators of compromise (IOCs) through dynamic analysis. Hybrid analysis facilitates the detection of unknown threats, particularly those provided by the most sophisticated malware. Hybrid analysis, for example, uses static analysis to data provided by behavioral analysis, such as when malicious code runs and causes memory modifications.

Memory malware analysis is commonly employed for digital investigation and malware analysis. It refers to the process of analyzing a dumped memory image from a targeted system after malware execution to obtain a variety of artifacts [16], such as network information, running processes, API hooks, kernel-loaded modules, and Bash history, etc. This phase is crucial because it is always advisable to have a better grasp of the malware’s capabilities. Volatility tools are frequently employed to extract relevant artifacts from memory images. Volatility is an open-source memory forensics framework designed for incident response and malware examination. This is a very powerful tool that can interact with memory dump files in numerous ways. However, it requires domain expertise to
extract relevant artifacts from the malware memory dump and also takes a considerable amount of time. Therefore, researchers proposed several visualization-based techniques for detecting and categorizing malware classes.

1.2. Problem Statement

Numerous researchers have proposed a visualization-based detection technique capable of analyzing the behavior of malware to overcome the obstacles posed by other detection techniques. Visualizing malware as a color or in grayscale image provides the advantage of differentiating different components of the malware; however, it is crucial to use appropriate artifact extraction techniques for the malware images. Numerous techniques are employed, including global image descriptor (GIST), histogram of oriented gradient (HOG), principal component analysis (PCA), Kernel PCA, and many others. However, these feature-engineering techniques are either too complicated or require a great deal of computing and memory power. To the best of our knowledge, no researchers have removed noise added to malware images during the transformation and coding process. These noises are artifacts produced by random pixel values in an image that does not originate from the source, and they have a direct impact on the performance [17]. Hence, it is a need to understand, investigate, and develop an efficient technique that can remove the noise from the malware image.

1.3. Objectives and Contribution

In this study, we convert all malicious and benign memory dump files into 224 × 224 pixels of grayscale images. In image processing, noise reduction is one of the most crucial steps [18,19]. These noises may be introduced during the transformation [20] from computer memory to dump files or from dump files to grayscale images. We propose the Non-Linear Means denoising technique for removing noise from images to restore the original state. Since noise, edge, and texture are high-frequency components, it is difficult to distinguish them during the denoising process, and denoised images may lose detail. The second important step is to compress the denoised images of malware to reduce the irrelevance and redundancy of the data. Image compression serves primarily to reduce the number of bits required to represent an image. Finally, the compressed images are fed into machine learning classifiers for testing and training in order to detect and classify various malware classes.

We make the following contributions in this paper,

- We examine various image noises, including Gaussian, Salt Pepper, Poisson, and Speckle, and apply various filtering techniques to minimize the distortion in these images.
- We present a technique using discrete wavelet transform to compress the malware images to reduce the irrelevant and redundant data in the images, without compromising important information.
- We present various machine learning classifiers and experiments with their appropriate hyperparameters to tune them to obtain a significant result.

1.4. Paper Structure

The structure of the subsequent sections is as follows: Section 2 describes the relevant work. In Section 3, a thorough summary of the methodology is presented. Section 4 displays the experiment, result, and findings. Section 5 delivers the discussion, and Section 6 provides the conclusion.

2. Related Work

Researchers employed a variety of methodologies to investigate and identify malware. A number of researchers employed static analysis in their hunt for malicious software [21–27]. However, due to limitations, such as vulnerability to unseen malware, time consumption, and manual feature extraction, etc., it is not feasible to use. Apart from static analysis, dynamic malware analysis is also widely used by various researchers [28–32].
Similarly, researchers also proposed the integration of static and dynamic analysis approaches for malware detection and classification [9,33–35]. However, these analysis techniques also have some limitations and, as of today, the most widely used approach for malware analysis is memory analysis.

In [36], researchers present a memory forensics-based malware detection technique to detect and classify various malware classes. The authors transformed the memory dump file into grayscale images. Two important techniques—contrast limited adaptive histogram equalization and wavelet transform of level two—are used to remove the noise and compress the image. Various machine learning classifiers are used, and SVM with RBF kernel obtained the best result, with an accuracy of 97.01%, precision of 97.36%, recall of 95.65%, and f1-score of 96.36%.

In [16], researchers present malware detection based on an integrated approach of memory forensics and dynamic analysis. The suspicious characteristics are extracted using memory and dynamic analysis. The combined feature characteristics are fed into the machine learning classifiers, such as support vector machine (SVM), decision tree (DT), naïve bayes (NB), k-nearest neighbor (KNN), and random forest (RF). The SVM outperformed other classifiers, with an accuracy of 98.50% and a false positive rate (FPR) of 1.7%, respectively.

In [37], researchers present a malware detection and classification approach based on memory analysis. In their approach, authors extract features, such as API calls, DLLs, process information, hidden code injection, and communication log, from the malware memory images. A feature selection technique such as information gain (IG) and correlation is used to select the most relevant artifacts. Finally, the highest weight of artifacts is fed into the machine learning classifiers, such as NP, SVM, KNN, DT, and RF. Among all other classifiers, SVM obtained the highest accuracy of 98.5% and a false positive rate of 1.24%.

In [38], researchers present malware detection and classification based on the malware’s memory dump. The memory dump files are converted into grayscale images with a fixed size and dimensions. A histogram of oriented gradient (HOG) is used to extract the vital artifacts from grayscale images. Various classifiers, such as SPAM-GIST, HPC, KNN, RF, and multilayer perceptron (MLP), are used. Among all, an MLP obtained the highest accuracy of 95.2%, precision of 95.10%, recall of 93.20%, and f1-score of 94.10%.

In [13], researchers present malware detection and classification based on memory forensics, manifold, and computer vision. Authors convert the memory dump files into color images that consist of three channels r (red), g (green), and b (blue) with various dimension sizes such as 224 pixels, 300 pixels, 4096 pixels, and a square root scheme. Two important feature extraction techniques, HOG and global image descriptor (GIST), are used to extract the most relevant features from the color images. The combined features of HOG and GIST are later fed into machine learning classifiers such as RF, SVM (RBF, Linear), XGBoost, and J48. Among all other classifiers, SVM with RBF kernel obtained the highest accuracy of 96.36%, precision of 96.40%, recall of 96.40%, and f1-score of 96.40% for a classification problem.

In [39], researchers describe how memory dump files can be used as a heuristic environment for detecting malware. In their approach, the authors collect features such as registry activities, import libraries, and API function calls from the memory dump files. Before feeding the extracted features into the machine learning classifiers, they evaluate each feature in terms of significance values. The SVM classifier achieved the best accuracy of 96%.

In [40], researchers present malware detection techniques based on memory dumps and static analysis. The authors claim that, using their approach, one can reveal the hidden characteristics of suspicious files. The authors first used static analysis for packed malware samples and obtained 90% of the data. In addition, they claim that their approach can be used in a variety of strategies to improve the malware detection rate.

In [41], researchers present a novel solution using trigger-based memory analysis. With this approach, memory dumps are taken place when a specific or desired event occurs.
There are many problems such as user-mode and kernel-mode root kits, and calling native functions that are not hooked, which can be treated with the approach claimed by the authors. Combining this approach with other memory analysis can improve security and provide important information for malware detection.

In [42], researchers present malware detection based on trustworthy volatile memory dumps from a virtual machine in the private cloud of the organization. The authors used the MinHash technique to examine the data in the dump files. This approach is further accessed in terms of efficacy by using various machine learning classifiers and similarity-based techniques. The authors claim their proposed framework can identify the known and unseen suspicious files with a high true positive rate (TPR) and low false positive rate. Their framework is tested and obtained a TPR of 100% and FPR of 1.8% for the ransomware and RATS, respectively.

In [43], researchers present an agentless sandbox solution independent of the virtual machine’s (VM) hypervisor for malware detection. Using the VM’s introspection method, it kept an eye on malware running in memory data outside of the VM and analyzed its system behaviors, such as process, file, registry, and network activities. The authors used 20 advanced and eight script-based malware samples to test their approach and determine the viability. The results show that their approach could be used to analyze suspicious malware activities that are helpful in cyber security.

In [44], researchers present a visualization method of malware detection using deep learning. In this approach, in Cuckoo, the API trigger is replaced with hooking, which indicates that API calls and memory dumps are synchronized. This is due to the encrypted IP address being stored in the memory. The authors claim that it is feasible to retrieve the plain text and identify the hidden information before malware hides it again.

In [45], researchers present a novel approach to capturing a memory from a system infected by ransomware malware such as NotPetya, Bad Rabbit, and Phobos hybrid. The contents of the memory are examined with live forensic tools to determine the symmetric encryption keys in use. Two critical steps are used to recover the AES keys. First, a timeline is manually created by combining data from multiple sources to show ransomware’s behavior and when encryption keys are in the memory and for how long. Second, encrypted files are decrypted using the found keys. The authors stated that the generated timelines are an excellent way to visualize ransomware behavior and encryption key management practices from a forensic investigation and mitigation perspective when the encryption keys are in the memory.

In [46], researchers propose a VMI, MFA, and hypervisor machine learning combination to construct an upgraded VMM-based A-Introspection Extraction (A-InExt) system. They employ VMI to examine digital artifacts from a live guest operating system to comprehend the processes. A-InExt features an intelligent cross-view analyzer (ICVA) that searches VMI data for hidden, dead, and suspicious processes, while predicting the early execution of malware on the guest OS. Malicious MFA executables are discovered by methods using machine learning. The A-InExt system is put to the test by running malicious and benign executables on the live guest OS. It identifies unknown malware on a produced dataset with 99.55% accuracy and a 0.004% false positive rate. A-InExt outperformed VMM’s detection of malware by 6.3%.

In [47], researchers present signature matching and string pattern matching criteria for the YARA scanner to identify malware in RAM dumps. To reduce time, a GUI-based automated forensics toolkit is employed for process analysis rather than the command-line method of the volatile memory forensic toolkit. A built-in software examines the RAM image to discover and uninstall potentially harmful applications. According to the research, malware forensics may be used in the recorded processes to analyze the attack’s sources and identify the root cause.
3. Methodology

The methodology that we propose includes the following five primary steps: (i) data collection; (ii) binary to visualization; (iii) denoising; (iv) image compression/contraction; (v) machine learning classifiers; and (vi) classification. The step of data collection is comprised of a couple of sub-steps, such as the execution of portable executables (PE) within a virtual box. This is to collect an image of the system’s random-access memory (RAM), which contains all malicious activities. In some of the literature, these memory-based images are also referred to as memory dump files. Essentially, memory dump files are binary files. In the second step, these binary files are converted to grayscale images for visualization purposes. Next, we experiment with a number of noise reduction techniques, including contrast limited adaptive histogram equalization (CLAHE), non-local means denoising filter, bilateral filter, denoise total variation (TV), Chambolle filter, and Wiener filter. A discrete wavelet transformer (DWT) technique is used to compress and reduce the size of grayscale malware images without losing many important details. The output images from the preceding step are then fed into machine learning classifiers. The proposed methodology is shown in Figure 1.

Figure 1. Proposed methodology.

3.1. Data Collection

Data play a crucial role in the research process. Therefore, it is essential to collect such information from a trustworthy source. To understand what a dataset is, it is necessary to examine its components. It can be in various formats, including text, audio, and video. In the area of malware detection and classification, four large datasets are often used to find and categorize malware. These datasets are BIG2015 [48], MalImg [49], Malicia [50], and Malevis [51]. However, none of these datasets contains the PE files to examine the source file. Therefore, we use a memory forensics-based malware dataset [52] that contains a total of 4294 malware and benign dump files. These dump files are obtained through the creation of a virtual Windows 10 environment. The system’s memory image is extracted using Prodcump and saved as a dump file. Table 1 shows a breakdown of the proposed dataset used in this paper.
Table 1. Memory-based malware dataset.

| Category | Classes          | Quantity |
|----------|------------------|----------|
| Adware   | Adposhel         | 457      |
| Worm     | Allaple.A        | 437      |
| Adware   | Amonetize        | 436      |
| Worm     | AutoRun-PU       | 196      |
| Adware   | BrowseFox        | 190      |
| Trojan   | Dinwood/rfn      | 127      |
| Adware   | InstallCore.C    | 467      |
| Adware   | MultiPlug        | 488      |
| Trojan   | Vilsel           | 389      |
| Virus    | VBA              | 499      |
| Benign   |                  | 608      |

- **Adware**

  Adware is software that is supported by advertising and displays advertisements on devices such as computers and mobile phones. Adware can also be referred to as madware when it is installed on a mobile device [53]. Adware targets both consumers and businesses. However, it primarily targets individuals, typically by luring them in with the promise of new content (such as games, movies, or deals).

- **Worm**

  A worm is a type of malicious software that can make copies of itself and spread throughout a network by taking advantage of holes in the system’s security. Attachments sent via email, text messages, file-sharing programs, social networking sites, network shares, removable drives, and software vulnerabilities are all potential vectors for its propagation.

- **Trojan**

  Trojans are deceitful programs that give the appearance of carrying out one function, but, in reality, carry out another malicious function. They may pose as free software, videos, or music, or even as advertisements that look completely legitimate.

- **Virus**

  A computer program can replicate itself and spread to another computer without the user’s knowledge or permission. If a computer is infected with a virus, the information stored on it could be erased or corrupted.

3.2. **Binary to Visualization**

A file without any text is referred to as a binary file. It stores information as bytes, which are typically interpreted differently than characters in text files. Its primary function is to simplify data storage. Typically, the headers of these files contain instructions on how to access the data it contains. These can store any type of data required by a computer system. The memory dump files obtained in the preceding step are referred to as binary files. In this step, these binary files are turned into color images with different sizes depending on how big they are. We use a python script to do the conversion process from binary into color images [54]. The transformation of the binary to visualization process takes two important steps: (i) binary to 8-bits conversion and (ii) 8-bits to image presentation.

The term “8-bit” refers to a method of storing graphic information in the memory of a computer or an image file, with each pixel represented by 8 bits (1 byte). These images are capable of being rendered in color or grayscale. Grayscale images only have a single channel, whereas color images have three: R (red), G (green), and B (blue). The pixel range of these images is between 0 and 255 and it is calculated through $2^n - 1$, where $n$ is the bit number. These images are the combination of pixels that build a digital image. A pixel is one of the many tiny dots or squares that combine to form an image on a computer screen. A greater number of pixels makes an image appear more realistic or precise. These pixel values specify how dark or light the images are. Each pixel value in an image represents a
The term “8-bit” refers to a method of storing graphic information in the memory of a computer or an image file, with each pixel represented by 8 bits (1 byte). These images are capable of being rendered in color or grayscale. Grayscale images only have a single channel, whereas color images have three: R (red), G (green), and B (blue). The pixel range of these images is between 0 and 255 and it is calculated through $2^n - 1$, where $n$ is the bit number. These images are the combination of pixels that build a digital image. A pixel is one of the many tiny dots or squares that combine to form an image on a computer screen. A greater number of pixels makes an image appear more realistic or precise. These pixel values specify how dark or light the images are. Each pixel value in an image represents a significant aspect of the image. The graphical process of the binary to visualization process is depicted in Figure 2.

![Figure 2](image_url)

**Figure 2.** The transformation process from binary to grayscale image.

As previously stated, the size of malicious image files is dependent on the size of the binary file. The visualization process produces images with a dimension range of 1024 to 1580 pixels. Initially, we transform the binary files into color images with dimensionality depending on the size of the individual file in order not to lose any information in different color channels (RGB). The information in Table 2 illustrates how to determine the dimensions of each image. Algorithm 1 illustrates how the malware binary files are initially transformed into color images in ‘PNG’ format.

Due to its high dimensionality, it necessitates an enormous amount of storage and processing power. We propose a fixed size of $224 \times 224$ pixels for all images to reduce memory consumption and computational expense using a computer vision library in Python. Figure 3 depicts the grayscale images of various classes of malware.

| S. No | File Range                | Dimensions |
|-------|---------------------------|------------|
| 1     | File_size < 10,240        | 32         |
| 2     | 10,240 <= File_size <= 10,240 * 3 | 64         |
| 3     | 10,240 * 3 <= File_size <= 10,240 * 6 | 128        |
| 4     | 10,240 * 6 <= File_size <= 10,240 * 10 | 256        |
| 5     | 10,240 * 10 <= File_size <= 10,240 * 20 | 384        |
| 6     | 10,240 * 20 <= File_size <= 10,240 * 50 | 512        |
| 7     | 10,240 * 50 <= File_size <= 10,240 * 100 | 786        |
| 8     | Else                      | 1024       |
Algorithm 1: Transformation algorithm for dumping (binary) files to RGB images

Input: A path to a dataset of memory dump files
Output: Transformation of RGB images with various dimensionality

Start
Set B a list of memory dump files
Set width None
Set height None
Set data an empty list []
For i in length(B) where i integer numbers range from 0 to n.
If file_size of B[i] < 10,240
  width = 32
If 10,240 <= file_size of B[i] <= 10,240 * 3
  width = 64
If 10,240 * 3 <= file_size of B[i] <= 10,240 * 6
  width = 128
If 10,240 * 6 <= file_size of B[i] <= 10,240 * 10
  width = 256
If 10,240 * 10 <= file_size of B[i] <= 10,240 * 20
  width = 384
If 10,240 * 20 <= file_size of B[i] <= 10,240 * 50
  width = 512
If 10,240 * 50 <= file_size of B[i] <= 10,240 * 100
  width = 786
Else
  width = 1024
End If
height = (file_size/width)/+1
image = Width × height
image.save[‘PNG’]
data.append[image]
End For
End

Figure 3. Grayscale images of various malware classes after the transformation process.

3.3. Denoising

The presence of artifacts in an image that do not originate from the content of the original source itself is referred to as noise in the image [20,55]. In a broad sense, noise can be defined as a statistical variation of a measurement that is produced as a result of a random process. In digital images, noise emerges as an artifact in the image and appears as a grainy structure covering the image. This can be seen when comparing two images side by side. Noise is an unwelcome or disturbing artifact that, in most cases, lowers the subjective quality of an image. Noise can manifest itself in a number of distinct ways and can take on a variety of appearances within an image. It is a challenging task to remove noise from an image because the noise is linked to the high-frequency content, also known as the details. As a consequence of this, the objective is to find a solution that minimizes the amount of background noise, while minimizing the amount of information that is lost.

The presence of noise in an image can have either an additive or multiplicative impact on the image’s overall quality, depending on the nature of the noise. In the Additive Noise Model, presented in Equation (1), a corrupted noisy signal is produced by adding an additive noise signal to the original signal. On the other hand, the Multiplicative Noise Model, presented in Equation (2), multiplies the authentic signal by the noise signal:

\[ I(x, y) = o(x, y) + n(x, y) \]  
\[ I(x, y) = o(x, y) \times n(x, y) \]

where \( o(x,y) \) is the intensity of the original image, and \( n(x,y) \) is the noise that is added or multiplied to produce the corrupted signal \( I(x,y) \) at the \( (x,y) \) pixel position.

Noise can be introduced into a digital image by a variety of means, including during the process of acquiring the image, transmitting it, and encoding it. The number of pixels in an image that is damaged can be used to quantify the amount of noise in the image. There is a very broad spectrum of different kinds of noise. We concentrate primarily on...
3.3. Denoising

The presence of artifacts in an image that do not originate from the content of the original source itself is referred to as noise in the image [20,55]. In a broad sense, noise can be defined as a statistical variation of a measurement that is produced as a result of a random process. In digital images, noise emerges as an artifact in the image and appears as a grainy structure covering the image. This can be seen when comparing two images side by side. Noise is an unwelcome or disturbing artifact that, in most cases, lowers the subjective quality of an image. Noise can manifest itself in a number of distinct ways and can take on a variety of appearances within an image. It is a challenging task to remove noise from an image because the noise is linked to the high-frequency content, also known as the details. As a consequence of this, the objective is to find a solution that minimizes the amount of background noise, while minimizing the amount of information that is lost.

The presence of noise in an image can have either an additive or multiplicative impact on the image’s overall quality, depending on the nature of the noise. In the Additive Noise Model, presented in Equation (1), a corrupted noisy signal is produced by adding an additive noise signal to the original signal. On the other hand, the Multiplicative Noise Model, presented in Equation (2), multiplies the authentic signal by the noise signal:

\[
I(x, y) = o(x, y) + n(x, y)
\]  
\[
I(x, y) = o(x, y) \times n(x, y)
\]

where \(o(x,y)\) is the intensity of the original image, and \(n(x,y)\) is the noise that is added or multiplied to produce the corrupted signal \(I(x,y)\) at the \((x,y)\) pixel position.

Noise can be introduced into a digital image by a variety of means, including during the process of acquiring the image, transmitting it, and encoding it. The number of pixels in an image that is damaged can be used to quantify the amount of noise in the image. There is a very broad spectrum of different kinds of noise. We concentrate primarily on the most significant types, which are gaussian noise, salt-and-pepper noise, poisson noise, and speckle noise.

- Gaussian Noise

Gaussian noise is a well-established type of statistical noise with a probability density function (PDF) that is equivalent to the normal distribution [56]. The presence of Gaussian noise is characterized by a signal-wide distribution that is constant. A noisy image has pixels that are composed of the sum of their original pixel values, as well as a random value based on Gaussian noise. A bell shape can be seen when looking at the probability distribution function for a Gaussian distribution. The most normal deployment of additive white Gaussian noise can be found in a variety of different applications for Gaussian noise. The Gaussian distribution function of Gaussian noise, also known as the probability distribution function, is depicted in Figure 4, along with a pixel representation of Gaussian noise. Equation (3) is used to compute the Gaussian noisy image.
\[ P_G(z) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(z - \mu)^2}{2\sigma^2}}. \]  

where \( z \) represents a gray level, \( \mu \) the mean value, and \( \sigma \) the standard deviation.

Figure 4. Gaussian noisy image.

- **Salt-and-Pepper Noise**

  This type of noise is exclusively present in grayscale images (black and white images). As the name implies, salt (white) in pepper (black) has white spots in the dark regions, while pepper (black) in salt (white) has black spots in the white regions [57]. In other words, an image with salt-and-pepper noise contains a small number of dark pixels in bright regions and a small number of light pixels in dark regions. Salt-and-pepper noise is known as impulse noise as well. It can be brought on by a variety of factors, including dead pixels, analog-to-digital conversion errors, and bit transmission errors, etc. The application of a median filter, morphological filter, or contra harmonic mean filter can effectively eliminate this type of noise, as well as other types of noise. Salt-and-pepper noise is visible whenever there is a sudden change in brightness or contrast in an image. Figure 5 illustrates how the malware image is transformed into a noisy image after adding the salt and pepper.

Figure 5. Salt-and-Pepper noisy image.

- **Speckle Noise**

  Speckle noise is a multiplicative type of noise, in contrast to Gaussian noise and salt-and-pepper noise. This causes a reduction in image quality during diagnostic examinations by giving images the appearance of a backscattered wave, which is caused by a large number of microscopic reflections that are dispersed as they flow through internal organs [58]. The observer has a more difficult time differentiating the fine details in the images as a result of this. This particular kind of noise can be seen in the images produced by a wide variety of systems, such as synthetic aperture radar (SAR) and ultrasound imaging, among a great number of others. The speckle noise is presented in Equation (4),

\[ \text{Ob}(x, y) = \text{Or}(x, y)\text{MI}(x, y) + \text{Ad}(x, y) \]  

where \( \text{Ob} \), \( \text{Or} \), \( \text{MI} \), and \( \text{Ad} \) represent the observed image, original image, multiplicative component, and an additive component of the speckle noise, respectively. Figure 6 depicts
the observed image following the application of speckle noise. It is not easy for a human to differentiate between original malware and a speckle noise image. However, if it is converted into matrix format, you may discover that the pixel values of both images have significantly diverged from one another.

**Figure 6.** Speckle noisy image.

- Poisson Noise

The statistical properties of electromagnetic waves, including X-rays, visible light, and gamma rays, are to be held responsible for the appearance of this noise. Sources of X-rays and gamma rays release a certain number of photons in a given amount of time. In medical imaging systems that use X-rays and gamma rays, these rays are injected into the patient’s body directly from their point of origin. These sources exhibit unpredictable fluctuations in the number of photons. The image that is gathered as a result has randomness in both space and time. This noise is also referred to as quantum noise (photon noise), as well as shot noise. The Poisson noise distribution can be represented in Equation (5),

\[
P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}
\]

where, \( x = 0, 1, 2, 3, 4 \ldots \), \( \lambda \) is the number of occurrences in the interval, and \( e \) is the Euler’s constant.

Figure 7 depicts the malware image after the Poisson Probability Distribution is deployed.

**Figure 7.** Poisson noisy image.

Denoising an image means removing noise from an already noisy image to restore it to its original state. Denoising an image, however, can make it difficult to differentiate between noise, edges, and textures, because these are high-frequency components. As a result, the denoised images may invariably lose some details. In general, one of the most important challenges is to extract meaningful information from images that are contaminated with noise during the process of noise removal to produce images of high quality. However, it is still a difficult and potentially open-ended task. The primary reason for this is that, from a mathematical point of view, image denoising is an inverse problem, and its solution is not unique. This means that there is more than one way to solve the problem; therefore, it is required to use various filter techniques to minimize the noise from images. The term “noise filtering” refers to a series of processes that are carried out to eliminate the noise that is associated with the image data that are obtained from construction and infrastructure sites.
As we do not know what type of noise is present in the malware images, it is necessary to employ an efficient and effective denoising technique to compensate for the data corruption that has occurred. The use of spatial filters such as means and medians is employed to get rid of image noise. Despite this, spatial filters have the undesired effect of blurring the edges of images, in addition to smoothing the data to reduce noise. In our paper, we propose the various denoising filter techniques listed below:

- **Contrast Limited Adaptive Histogram Equalization (CLAHE)**
  To improve image contrast, we use contrast limited adaptive histogram equalization (AHE). Ordinary histogram equalization computes a global equalization, whereas adaptive histogram equalization computes multiple histograms, each corresponding to a distinct section of the image, and uses them to redistribute the image’s lightness values. It is appropriate for enhancing local contrast and defining edges in each image region. CLAHE, a variant of adaptive histogram equalization, limits AHE’s tendency to overamplify noise in relatively homogeneous image regions. It is capable of performing histogram equalization in small patches or small tiles, while maintaining a high level of accuracy and contrast limiting. This small region is referred to as tiles. The adjacent tiles are then merged using bilinear interpolation to eliminate the artificial borders. This algorithm can be utilized to enhance image contrast.

- **Bilateral Filter**
  When smoothing with a Gaussian distribution, we take the weighted average of the surrounding pixel values. The weights are calculated in inverse proportion to the distance from the geographic center of the neighborhood. The bilateral filter adds a tonal weight, in addition to these spatial weights. Pixel values that are relatively similar to the pixel value in the center are weighted more heavily than more dissimilar pixels. Due to the tonal weighting, the bilateral filter can preserve edges, which are defined as significant tonal value differences, while smoothing out relatively flat regions. The bilateral filter image can be defined in Equation (6) and Equation (7),

\[ I_{SBF}(y) = \frac{1}{K^q} \sum_{y_i \in \Omega} I(y_i) h_{\sigma_r}(\|I(y_i) - I(y)\|) h_{\sigma_s}(\|y_i - y\|) \]  

and,

\[ K^q = \sum_{y_i \in \Omega} h_{\sigma_r}(\|I(y_i) - I(y)\|) h_{\sigma_s}(\|y_i - y\|) \]  

where \(I_{SBF}\) represents the denoised image, \(I\) is the actual image, \(Y\) represents the coordinates of pixels that are required to be filtered, \(\Omega\) represents window that are centered pixels, \(h_{\sigma_r}\) represents the Gaussian range kernel, and \(h_{\sigma_s}\) represents the space kernel.

- **Wiener Filter**
  When an image is blurred by a known lowpass filter, it is possible to recover the image using inverse or generalized inverse filtering, which is a deconvolution restoration technique. In other words, it is possible to recover an image after it is blurred by a filter. If we take the approach of reducing one type of degradation at a time, we can develop a restoration algorithm for every type of degradation, and then easily combine these algorithms. By executing a tradeoff, the Wiener filtering method achieves the optimal balance between inverse filtering and noise smoothing. It eliminates the additive noise, while simultaneously inverting the blurring. Regarding the mean square error, Wiener filtering achieves the best results possible. In other words, it aids in reducing the overall mean square error and smoothing out the noise during the inverse filtering process. The Wiener filtering technique produces a linear approximation of the original image. It can be presented in Equation (8)
$$W(f_1, f_2) = \frac{H^*(f_1, f_2)S_{xx}(f_1, f_2)}{|H(f_1, f_2)|^2S_{xx}(f_1, f_2) + S_{\eta\eta}(f_1, f_2)}$$

where $f_1, f_2$ are the pixel values at the x-axis and y-axis, $S_{xx}(f_1, f_2)$ is the power spectrum of the original image, $S_{\eta\eta}(f_1, f_2)$ is the power spectrum of the additive noise, and $H(f_1, f_2)$ is the blurring filter. The Wiener filter has two distinct components, an inverse filtering component and a noise smoothing component. Not only does it perform deconvolution by inverse filtering (high-pass filtering), but it also eliminates noise with a compression operation (low-pass filtering).

- **Gaussian Filter**

  An image is processed with a low-pass filter known as a Gaussian filter to reduce noise (high-frequency components) and blur specific areas of the image. To achieve the desired result, the filter is implemented as an Odd-sized Symmetric Kernel (DIP version of a Matrix) that is applied to each pixel of the Region of Interest (RoI). The kernel is insensitive to abrupt color changes (edges) because the central pixels contribute more to the final value than those on the periphery. The Gaussian filter is a Gaussian function approximation. To apply a Gaussian filter to an image, we must first specify the size of the kernel/matrix used for denoising the image. The sizes are typically even numbers, allowing the central pixel to compute the overall results. In addition, kernels are symmetric and have the same number of rows and columns. The Gaussian function computes the values within the kernel, as shown in Equation (9),

$$G(x, y) = \frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where $x$ and $y$ are two image coordinate values, $\pi$ the constant PI value 3.13, and $\sigma$ represents the standard deviation.

- **Total variation**

  Total variation (TV) denoising is a noise removal process that is used in signal processing, more specifically, in image processing. It is also known as total variation regularization and total variation filtering (filter). It is predicated on the idea that signals that contain an excessive amount of information, some of which may be dubious, have a high total variation; more specifically, the integral of the absolute image gradient is high. According to this principle, removing unwanted detail, while maintaining important details such as edges, can be accomplished by lowering the total variation of the signal, provided that the reduced signal is an accurate representation of the original signal. This method of removing noise has advantages over more straightforward techniques such as linear smoothing and median filtering, both of which reduce noise but also smooth away edges to a greater or lesser degree. Total variation denoising, on the other hand, is an exceptionally efficient edge-preserving filter. This means that it can simultaneously preserve edges, while simultaneously smoothing away noise in flat regions, and it can do this even at low signal-to-noise ratios. In this paper, we use the Chambolle technique of TV in the images to reduce the number of unwanted artifacts from malware images.

- **Non-Local Means Denoising**

  A non-local means of denoising is another technique used to remove noise from images. When smoothing an image, “local mean” filters take the mean value of a group of pixels surrounding a target pixel and use that to determine the smoothness of the image. Non-local means filtering, on the other hand, takes the mean of all pixels in the image and weights it according to how similar each pixel is to the target pixel. In comparison to algorithms based on the local mean, this yields an image that retains much more of its original clarity after filtering, while suffering far less loss of information. As a consequence of this, the method can properly preserve textures that, if processed by another denoising technique, would become blurry. Equation (10) presents a mathematical representation of the non-local means denoising technique where $p$ and $q$ are two points in the image.
\[ u(p) = \frac{1}{C(p)} \int_{\Omega} v(q) f(p, q) \, dq \]  

(10)

\( u(p) \) = indicate a filter value at the pixel point \( p \)

\( v(q) \) = an unfiltered value at point \( q \)

\( f(p, q) \) = represents a weighted function and \( C(p) \) calculates the normalization factor

\[ C(p) = \int_{\Omega} f(p, q) \, dq \]  

(11)

The objective of the weighting function is to establish the degree to which the image located at point \( p \) is connected to the image located at point \( q \). Equation (11) presents a mathematical form to calculate the weighted function where \( h \) and \( B(p) \) are filter parameters and local mean values, respectively.

\[ F(p, q) = e^{-\frac{||B(q) - B(p)||^2}{h^2}} \]  

(12)

Algorithm 2 illustrates how the malware images can be transformed into an image that has the least noise in it.

### Algorithm 2: Transformation algorithm for the grayscale image to remove noise

- **Input:** A path to a dataset image
- **Output:** A set of denoised images

1. **Start**
2. Set data an empty list [ ]
3. For each pixel of a single image
   - Calculate Normalization function (11)
   - Evaluate weighted function Equation (12)
   - Calculate non-local means Equation (10)
   - data.append[denoised image]
4. **End For**
5. **End For**
6. **End**

3.4. **Image Compression/Contraction**

Image compression is a process that is applied to a graphics file to reduce the amount of space that the file takes up in bytes, while maintaining an acceptable level of image quality. If the file size is decreased, then a given amount of space on the hard disk or in the memory can hold a greater number of images. Before beginning the processing of larger images or videos, it is necessary to first compress any images that need to be transferred. An encoder is what accomplishes the process of compressing images and then outputs the image in its compressed form. Mathematical transformations are an extremely important part of the process of data compression. There are various widely used techniques, such as fractal, wavelets, chroma sub-sampling, transform, and run-length encoding. Similarly, there are linear-based filters such as PCA, Factor Analysis (FA), Linear Discriminant Analysis (LDA), and Truncated Singular Value Decomposition (SVD). These dimensionality techniques are either sensitive to the scale of the feature, extracting the feature based on assumption, not robust against a non-linear problem, or have a low interpretation of feature issues, and many more. However, in this paper, we propose to use a wavelet approach.

Wavelet transforms are mathematical tools that can be used to analyze data in which features vary across a variety of scales. In the case of signals, features can take the form of slowly varying trends, transients, or frequencies that change over time. Edges and textures are examples of features that can be found in images. The limitations of the Fourier transform were the primary motivation behind the development of wavelet transforms. Equation (13) present the wavelet transform,
\[ F(a, b) = \int_{-\infty}^{\infty} f(x) \psi^*_a(bx) \, dx \] (13)

where * is the complex conjugate symbol and \( \psi \) represents a function.

Wavelet transforms can be classified into two broad classes: the continuous wavelet transform (CWT) and the discrete wavelet transform (DWT). A formal tool that is used in mathematics to provide an overcomplete representation of a signal is called the CWT. This is achieved by granting the wavelets’ translation and scale parameters the freedom to continuously vary in value. In contrast, the DWT is an implementation of the wavelet transform that uses a discrete set of wavelet scales and translations, while adhering to some predetermined rules. This transform decomposes the signal into a set of mutually orthogonal wavelets, which is the primary distinction between DWT and CWT. In this paper, we propose to use DWT to compress the malware images without losing too many artifacts for machine learning to distinguish between malicious and benign images.

The DWT is most useful when applied to non-stationary signals because of its dynamic nature. By applying this transformation, one can obtain a high temporal resolution for high-frequency components, while maintaining a good frequency resolution for low-frequency components. This method starts with a mother wavelet, which can be a Haar, a Morlet, or a Daubechies, amongst other possible choices. After that, the signal is transformed into scaled and shifted iterations of the mother wavelet. Two filters, high-pass and low-pass filters, are applied to a signal. After that, the image is decomposed into high-frequency (details) and low-frequency components (approximation). At every level, we get four sub-signals. The approximation displays the overall trend of pixel values, as well as the horizontal, vertical, and diagonal details. Figure 8 shows how the malware image is compressed to level three during the DWTC procedure. Figure 9 depicts the transformation process of a single level.

![Discrete Wavelet Transform process of level three.](image)

**Figure 8.** Discrete Wavelet Transform process of level three.
3.5. Machine Learning

To automate the process of malware detection and classification, various researchers use machine learning and deep learning. Deep learning algorithms provide many advantages over machine learning classifiers. However, it requires a huge number of data features and high computational costs to train and test it. In this paper, we propose to use machine learning classifiers. The images that are previously compressed during the DWT process are converted into feature vectors without a feature selection process to feed them to machine learning classifiers. Various classifiers are widely used in the research, such as SVM with RBF and Linear kernel, XGBoost, DT, RF, bagging classifier, logistic regression, AdaBoost classifier, gradient boosting classifier, and histogram-based gradient boosting classifier.

3.6. Evaluation Metrics

Evaluation is the process of using a variety of evaluation metrics to gain an understanding of the performance of a machine learning model, as well as the strengths and weaknesses of the model. This can be done to gain a better understanding of how machine learning works. Evaluation of a model is required for determining its usefulness of a model during the preliminary stages of research. Evaluation of models also has a role to play in the monitoring of models. In this research, we make use of a wide variety of performance metrics, including,

- Peak Signal-to-Noise Ratio (PSNR)

The PSNR block determines the peak signal-to-noise ratio between two images and expresses the result in decibels. When comparing the quality of the original image to that of a compressed version, this ratio is the metric of choice. When the PSNR is higher, the quality of the compressed image, as well as the reconstructed image, is improved. Equation (14) is used to calculate the PSNR:

\[
PSNR = 20 \log_{10} \left( \frac{MAX_f}{\sqrt{MSE}} \right) \tag{14}
\]

Mean Square Error (MSE)

The MSE value denotes the average difference in all of the image’s pixels across the board. A greater value for MSE indicates a greater disparity between the original image and the image that has been processed. A mathematical presentation of MSE is found in Equation (15):

\[
MSE = \frac{1}{mn} \sum_{0}^{m-1} \sum_{0}^{n-1} || f(i,j) - g(i,j) ||^2 \tag{15}
\]

Structural Similarity Index (SSIM)

When comparing images, the Mean Squared Error, also known as MSE, which is simple to calculate, may not be a much better sign of how similar the images are perceived.
to be to one another. The Structural Similarity Index, also known as SSIM, is an attempt to make up for this deficiency. It does so by taking into account texture and awarding a higher score to images that may give the impression of being similar to one another. It is defined in Equation (16):

$$SSIM(x, y) = \frac{(2\mu_x \mu_y + c_1) (2\sigma_{xy} + c_2)}{\left(\mu_x^2 + \mu_y^2 + c_1\right) \left(\sigma_x^2 + \sigma_y^2 + c_2\right)}$$  (16)

where

- $\mu_x = \text{the average of } x$
- $\mu_y = \text{the average of } y$
- $\sigma_x^2 = \text{the variance of } x$
- $\sigma_y^2 = \text{the variance of } y$
- $\sigma_{xy} = \text{the variance of } x \text{ and } y$
- $c_1 = \text{represents } (k_1L)^2$
- $c_2 = \text{represents } (k_2L)^2$

- **Universal Image Quality Index (UQI)**

The Universal Image Quality Index is first presented (UQI). Loss of correlation, luminance distortion, and contrast distortion are the three factors that are considered when determining the quality of an image by UQI. It is represented in Equation (17):

$$UQI = \frac{4\sigma_{xy}}{\sigma_x^2 + \sigma_y^2} \left(\frac{\text{mean}^2 + \text{mean}^2}{\text{variance}^2 + \text{variance}^2}\right)$$  (17)

- **Accuracy**

Accuracy can be defined as the proportion of total data points that correspond to accurate predictions. Formally, it is defined as the number of true positives and true negatives divided by the number of true positives, true negatives, false positives, and false negatives. It is represented in Equation (18):

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative}}$$  (18)

- **Precision**

This refers to the quantification of the number of predictions for the positive class that belong to the positive class. It is represented in Equation (19):

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$  (19)

- **Recall**

This determines how many positive class predictions are made using all of the positive examples in the dataset. It is represented in Equation (20):

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$  (20)

- **F1-score**

This is an elegant way to summarize the predictive performance of a model by combining two metrics, precision and recall, that would otherwise compete with one another. It is represented in Equation (21):
\[
F1 - \text{score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]

- **Computational cost**

  The computational cost of a specific method or approach in machine learning or deep learning is simply a measurement of the number of resources that it uses. Knowing the computational cost is important so that we can determine how much time or computing power we require to train the model.

- **Memory Utilization**

  Memory utilization is the average utilization that is derived from the percentage of available memory that is being used at any given moment. This percentage can be used as a starting point for calculating memory utilization.

4. **Experiment, Result, and Findings**

  In Table 3, the specification of the hardware that is utilized in the experiment is outlined. For the experiment, a programming interface based on Python is used. We make use of a number of Python libraries, including NumPy, Pandas, Matplotlib, cv2, PyWavelets, Scikitplot, Sklearn, Scipy, and Skimage.

  | Parameters          | Values                      |
  |---------------------|-----------------------------|
  | Operating System    | Windows 7 Professional—64 bit |
  | RAM                 | 32                          |
  | Processor           | 2.40 GHz                    |
  | Hard Drive          | 512 SSD                     |

  The experiments are carried out with great care and attention to detail by fine-tuning the various parameters. Converting the binary image of the malware into a grayscale image with dimensions of 224 by 224 is accomplished with the help of a Python script. It takes two days to complete the process of transforming binary data into an image. The initial size of the images was inconsistent, and this is because multiple malicious processes were running in the system before the memory acquisition step. However, in this paper, we keep the dimensions constant to reduce the amount of memory and computational cost. We believe that these images may contain different types of noise during the time of transformation or encoding. However, it is unknown what type of noise these images may contain. To denoise the images, we first added noise, such as Gaussian, salt-and-pepper, poisson, and speckle, to the original malware image to determine the filter techniques that can reduce or eliminate unwanted artifacts. Various filter techniques, such as CLAHE, bilateral filter, Wiener filter, Gaussian filter, non-local means, and total variation filter, are used. Each filter technique is extensively utilized with various values with respective parameters.

  During the CLAHE denoising technique, it is observed that increasing the ‘tileGridSize’ parameter’s values resulted in a decrease in the performance evaluation, including MSE, PSNR, and UQI. The tileGridSize parameter divides the image into equal-sized rectangular tiles. The findings of the detailed experiments are presented in Table 4, which also reveals that the best performance was achieved by setting the ‘tileGridSize’ parameter to (1,1) and the ‘clipLimit’ parameter to 0.001, 0.01, respectively.

  The bilateral filtering or bilateral smoothing technique introduces another Gaussian filter that takes into account the varying intensities to keep the edges intact. A bilateral filter denoising technique has a couple of important parameters, such as ‘diameter’, ‘sigmacolor’, and ‘sigmaspace’, to remove the noise from the image. We have tried a variety of different values for these parameters to determine the difference in effectiveness between the noisy and denoised images. Detailed experimentation of various parameter values is presented in Table 5. Here the dimension parameter is proportional to sigmaspace. The values 65, 5,
and 50 for dimension, sigmacolor, and sigmaspace, respectively, produce the best results in terms of MSE, PSNR, and UQI.

Table 4. Implementation of the CLAHE technique with diverse parameter values.

| Noise Type     | CLAHE Parameters     | Evaluation Metrics |
|----------------|----------------------|--------------------|
|                | tileGridSize | clipLimit | MSE     | PSNR    | SSIM         | UQI     |
| Gaussian       | (1,1)        |           | 0.001   | 0.7203  | 49.55, 0.9999 | 0.9999  |
|                |             |           | 0.01    | 0.7203  | 49.55, 0.9999 | 0.9994  |
|                |             |           | 0.05    | 0.7251  | 49.52, 0.9999 | 0.9994  |
|                |             |           | 0.1     | 0.9854  | 48.19, 0.9996 | 0.9993  |
|                |             |           | 0.2     | 2.4     | 42.8, 0.9991  | 0.9991  |
|                |             |           | 0.5     | 59.93   | 30.35, 0.9981 | 0.9937  |
|                | (2,2)        |           | 0.001   | 0.732   | 49.48, 0.9996 | 0.9994  |
|                |             |           | 0.01    | 0.732   | 49.48, 0.9999 | 0.9993  |
|                |             |           | 0.05    | 0.7673  | 49.28, 0.9999 | 0.9993  |
|                |             |           | 0.1     | 1.0815  | 48.21, 0.9999 | 0.9993  |
|                |             |           | 0.2     | 3.9     | 42.21, 0.9992 | 0.9989  |
|                |             |           | 0.5     | 56.26   | 30.62, 0.9946 | 0.9935  |
| Salt-and-Pepper | (1,1)        |           | 0.001   | 0.7063  | 49.64, 0.9999 | 0.9994  |
|                |             |           | 0.01    | 0.7063  | 49.64, 0.9999 | 0.9994  |
|                |             |           | 0.05    | 0.8246  | 48.96, 0.9999 | 0.9993  |
|                |             |           | 0.1     | 0.973   | 48.24, 0.9992 | 0.9993  |
|                |             |           | 0.2     | 5.08    | 41.07, 0.9998 | 0.9919  |
|                |             |           | 0.5     | 48.52   | 31.27, 0.9995 | 0.9886  |
|                | (2,2)        |           | 0.001   | 0.8209  | 48.98, 0.9992 | 0.9993  |
|                |             |           | 0.01    | 0.8209  | 48.98, 0.9999 | 0.9923  |
|                |             |           | 0.05    | 0.9708  | 48.25, 0.9992 | 0.9923  |
|                |             |           | 0.1     | 2.6     | 43.96, 0.9991 | 0.9922  |
|                |             |           | 0.2     | 5.95    | 40.38, 0.9988 | 0.9918  |
|                |             |           | 0.5     | 47.87   | 31.32, 0.9995 | 0.9884  |
| Poisson        | (1,1)        |           | 0.001   | 0.7347  | 49.46, 0.9951 | 0.9959  |
|                |             |           | 0.01    | 0.7347  | 49.46, 0.9951 | 0.9559  |
|                |             |           | 0.05    | 0.7673  | 49.16, 0.9951 | 0.9559  |
|                |             |           | 0.1     | 0.9971  | 48.14, 0.9951 | 0.9559  |
|                |             |           | 0.2     | 5.44    | 40.77, 0.9947 | 0.9554  |
|                |             |           | 0.5     | 70.63   | 29.64, 0.9893 | 0.9495  |
|                | (2,2)        |           | 0.001   | 0.7755  | 49.23, 0.9951 | 0.9959  |
|                |             |           | 0.01    | 0.7755  | 49.23, 0.9951 | 0.9559  |
|                |             |           | 0.05    | 0.8004  | 48.63, 0.9951 | 0.9559  |
|                |             |           | 0.1     | 0.9954  | 48.15, 0.9951 | 0.9559  |
|                |             |           | 0.2     | 6.63    | 39.91, 0.9946 | 0.9552  |
|                |             |           | 0.5     | 65.25   | 29.98, 0.9893 | 0.9497  |
Table 4. Cont.

| Noise Type | CLAHE Parameters | Evaluation Metrics |
|------------|------------------|--------------------|
|            | tileGridSize clipLimit MSE PSNR SSIM UQI |
| Speckle     |                  |                    |
| (1,1)      | 0.001 0.737 49.45 0.9951, 0.9999 0.956     |
|            | 0.01 0.737 49.45 0.9951, 0.9999 0.956     |
|            | 0.05 0.7849 49.18 0.9951, 0.9999 0.956   |
|            | 0.1 0.9956 48.14 0.9951, 0.9999 0.9559   |
|            | 0.2 4.64 41.46 0.9948, 0.9999 0.9555     |
|            | 5 69.83 29.68 0.9894, 0.9977 0.9497     |
| (2,2)      | 0.001 0.7608 49.31 0.9951, 0.9999 0.956     |
|            | 0.01 0.7608 49.31 0.9951, 0.9999 0.956     |
|            | 0.05 0.84 48.88 0.9951, 0.9999 0.956     |
|            | 0.1 0.9947 48.15 0.9951, 0.9999 0.9559     |
|            | 0.2 6.58 39.94 0.9946, 0.9998 0.9553     |
|            | 5.66 29.91 0.9892, 0.9975 0.9496     |

Table 5. Implementation of the bilateral filter with diverse parameter values.

| Noise Type | Bilateral Filter Parameters | Evaluation Metrics |
|------------|----------------------------|--------------------|
|            | Dimension SigmaColor SigmaSpace MSE PSNR UQI |
| Gaussian   | 5 35 10 167.54 25.88 0.9674     |
|            | 50 168.02 25.87 0.9676     |
|            | 150 169.54 25.83 0.9686     |
|            | 10 370.48 22.42 0.9489     |
|            | 50 372.8 22.41 0.9492     |
|            | 150 373.33 22.4 0.9489     |
|            | 10 199.57 25.12 0.9603     |
|            | 50 202.14 25.07 0.9595     |
|            | 150 203.41 25.04 0.9597     |
|            | 10 480.9 21.31 0.9331     |
|            | 50 484.81 21.27 0.9316     |
|            | 150 487.43 21.25 0.9313     |
|            | 10 0.7149 49.58 0.9996     |
|            | 15 0.9597 49.93 0.9997     |
|            | 150 0.6559 49.96 0.9997     |
|            | 10 15.19 36.31 0.9942     |
|            | 50 15.35 36.26 0.9942     |
|            | 150 15.28 36.28 0.9942     |
|            | 10 0.7075 49.63 0.9996     |
|            | 50 0.62 50.2 0.9997     |
|            | 150 0.6314 50.09 0.9996     |
|            | 10 15.31 36.28 0.9938     |
|            | 50 14.96 36.38 0.9944     |
|            | 150 15.05 36.35 0.9943     |
Table 5. Cont.

| Noise Type          | Bilateral Filter Parameters | Evaluation Metrics |
|---------------------|-----------------------------|--------------------|
|                     | Dimension | SigmaColor | SigmaSpace | MSE  | PSNR | UQI   |
| Salt-and-Pepper 5   | 35        | 10         | 141.64     | 26.6 | 0.9872 |
|                     | 50        | 142.28     | 26.59      | 0.9884 |
|                     | 150       | 142.19     | 26.6       | 0.9867 |
|                     | 50        | 150        | 324.8      | 23.01 | 0.979 |
|                     | 100       | 326.47     | 22.99      | 0.9784 |
|                     | 10        | 325.15     | 23         | 0.9785 |
|                     | 50        | 326.47     | 22.99      | 0.9784 |
|                     | 150       | 326.47     | 22.99      | 0.9784 |
|                     | 50        | 186.26     | 25.42      | 0.9747 |
|                     | 50        | 187.44     | 25.4       | 0.974  |
|                     | 150       | 189.34     | 25.35      | 0.9683 |
|                     | 25        | 475.98     | 21.35      | 0.9543 |
|                     | 50        | 482.74     | 21.29      | 0.9501 |
|                     | 150       | 481.88     | 21.3       | 0.9526 |
|                     | 55        | 10         | 0.8669     | 48.75 | 0.9997 |
|                     | 10        | 0.8613     | 48.77      | 0.9996 |
|                     | 150       | 0.8903     | 48.63      | 0.9995 |
|                     | 15        | 13.91      | 36.69      | 0.9907 |
|                     | 50        | 13.86      | 36.71      | 0.9919 |
|                     | 150       | 13.92      | 36.69      | 0.9874 |
|                     | 5         | 0.8757     | 48.7       | 0.9996 |
|                     | 10        | 0.8722     | 48.72      | 0.9997 |
|                     | 150       | 0.8817     | 48.67      | 0.9997 |
|                     | 15        | 13.65      | 36.77      | 0.991  |
|                     | 50        | 13.56      | 36.8       | 0.9901 |
|                     | 150       | 13.51      | 36.82      | 0.9888 |
|                     | 65        | 10         | 147.79     | 26.43  | 0.967  |
|                     | 35        | 148.47     | 26.41      | 0.967  |
|                     | 50        | 148.5      | 26.41      | 0.967  |
|                     | 150       | 325.15     | 23         | 0.9511 |
|                     | 50        | 326.27     | 22.99      | 0.951  |
|                     | 150       | 326.32     | 22.99      | 0.951  |
| Poisson 5           | 10        | 193.73     | 26.25      | 0.9224 |
|                     | 50        | 167.22     | 25.18      | 0.9209 |
|                     | 150       | 197.36     | 25.17      | 0.9203 |
|                     | 35        | 10         | 464.03     | 21.46  | 0.8926 |
|                     | 50        | 471.63     | 21.39      | 0.8907 |
|                     | 150       | 471.92     | 21.39      | 0.8906 |
### Table 5. Cont.

| Noise Type | Bilateral Filter Parameters | Evaluation Metrics |
|------------|-----------------------------|-------------------|
|            | Dimension | SigmaColor | SigmaSpace | MSE    | PSNR    | UQI    |
| 55         | 10        | 0.7635     | 49.3       | 0.999  |
|            | 15        | 0.755      | 49.35      | 0.999  |
|            | 150       | 0.7562     | 49.34      | 0.999  |
|            | 10        | 14.82      | 36.42      | 0.9522 |
|            | 50        | 15.21      | 36.3       | 0.9519 |
|            | 150       | 15.23      | 36.3       | 0.9519 |
|            | 10        | 0.7771     | 49.22      | 0.9988 |
|            | 50        | 0.742      | 49.42      | 0.9988 |
|            | 150       | 0.7439     | 49.41      | 0.9988 |
|            | 10        | 14.87      | 36.4       | 0.9523 |
|            | 50        | 14.96      | 36.37      | 0.952  |
|            | 150       | 14.98      | 36.37      | 0.952  |
| 65         | 10        | 147.73     | 26.43      | 0.9667 |
|            | 50        | 148.39     | 26.41      | 0.9667 |
|            | 150       | 148.41     | 26.41      | 0.9667 |
|            | 10        | 325.33     | 23         | 0.9512 |
|            | 50        | 326.48     | 22.99      | 0.9511 |
|            | 150       | 326.53     | 22.99      | 0.9511 |
| 25         | 10        | 196.71     | 25.19      | 0.9226 |
|            | 50        | 200.32     | 25.11      | 0.9213 |
|            | 150       | 200.42     | 25.11      | 0.9212 |
|            | 10        | 471.37     | 21.39      | 0.8928 |
|            | 50        | 479.16     | 21.32      | 0.8909 |
|            | 150       | 479.43     | 21.32      | 0.8909 |
| Speckle    | 10        | 0.7767     | 49.22      | 0.9987 |
| 55         | 15        | 0.7722     | 49.25      | 0.9987 |
|            | 150       | 0.7756     | 49.23      | 0.9987 |
|            | 10        | 15.05      | 36.35      | 0.9522 |
|            | 50        | 15.38      | 36.25      | 0.9519 |
|            | 150       | 15.4       | 36.25      | 0.9519 |
|            | 10        | 0.7925     | 49.14      | 0.9989 |
|            | 50        | 0.7527     | 49.36      | 0.9989 |
|            | 150       | 0.7523     | 49.36      | 0.9989 |
|            | 10        | 14.95      | 36.38      | 0.9519 |
|            | 50        | 14.19      | 36.37      | 0.9519 |
|            | 150       | 14.19      | 36.37      | 0.9519 |

Total variation with a Chambolle filter utilizes two crucial parameters, ‘noise’ and ‘weight’, to denoise the image. However, we found that the parameter with the name ‘eps’ has almost no impact on the deployment process. As a result, we constant its value to 0.002. The best possible outcome is accomplished through the utilization of this technique.
by setting the ‘weight’ value to 0.1. Table 6 presents comprehensive experimentation by making use of a variety of different values for its parameter.

Table 6. Implementation of the total variation—Chambolle with diverse parameter values.

| Noise Type      | TV Chambolle Parameters | Evaluation Metrics |
|-----------------|-------------------------|--------------------|
|                 | Weight | eps | MSE  | PSNR | UQI  |
| Gaussian        | 0.1    | 0.0002 | 0.0174 | 65.7 | 0.8986 |
|                 | 0.2    | 0.0002 | 0.0299 | 63.37 | 0.8409 |
|                 | 0.3    | 0.0002 | 0.03 | 63.35 | 0.84 |
|                 | 0.4    | 0.0002 | 0.034 | 62.8 | 0.8193 |
|                 | 0.5    | 0.0002 | 0.0358 | 62.8 | 0.8105 |
|                 | 0.6    | 0.0002 | 0.037 | 62.44 | 0.805 |
|                 | 0.7    | 0.0002 | 0.0377 | 62.35 | 0.8081 |
|                 | 0.8    | 0.0002 | 0.0382 | 62.3 | 0.7989 |
|                 | 0.9    | 0.0002 | 0.0385 | 62.27 | 0.7971 |
| Salt-and-Pepper | 0.1    | 0.0002 | 0.017 | 65.82 | 0.9282 |
|                 | 0.2    | 0.0002 | 0.0337 | 62.84 | 0.833 |
|                 | 0.3    | 0.0002 | 0.038 | 62.33 | 0.7879 |
|                 | 0.4    | 0.0002 | 0.0406 | 62.04 | 0.7692 |
|                 | 0.5    | 0.0002 | 0.0429 | 61.79 | 0.756 |
|                 | 0.6    | 0.0002 | 0.0443 | 61.66 | 0.7488 |
|                 | 0.7    | 0.0002 | 0.0452 | 61.57 | 0.7441 |
|                 | 0.8    | 0.0002 | 0.0457 | 61.52 | 0.7415 |
|                 | 0.9    | 0.0002 | 0.0461 | 61.48 | 0.7395 |
| Speckle         | 0.1    | 0.0002 | 0.0149 | 66.37 | 0.8784 |
|                 | 0.2    | 0.0002 | 0.0254 | 64.07 | 0.8132 |
|                 | 0.3    | 0.0002 | 0.0256 | 64.04 | 0.814 |
|                 | 0.4    | 0.0002 | 0.0294 | 63.44 | 0.787 |
|                 | 0.5    | 0.0002 | 0.0312 | 63.18 | 0.7751 |
|                 | 0.6    | 0.0002 | 0.0326 | 62.99 | 0.766 |
|                 | 0.7    | 0.0002 | 0.0333 | 62.89 | 0.7609 |
|                 | 0.8    | 0.0002 | 0.0338 | 62.84 | 0.7579 |
|                 | 0.9    | 0.0002 | 0.034 | 62.8 | 0.7562 |
| Poisson         | 0.1    | 0.0002 | 0.0149 | 66.39 | 0.8777 |
|                 | 0.2    | 0.0002 | 0.0217 | 64.75 | 0.8394 |
|                 | 0.3    | 0.0002 | 0.0252 | 64.1 | 0.815 |
|                 | 0.4    | 0.0002 | 0.029 | 63.49 | 0.7888 |
|                 | 0.5    | 0.0002 | 0.0308 | 63.23 | 0.7761 |
|                 | 0.6    | 0.0002 | 0.0322 | 63.04 | 0.767 |
|                 | 0.7    | 0.0002 | 0.0329 | 62.94 | 0.7619 |
|                 | 0.8    | 0.0002 | 0.0334 | 62.88 | 0.7589 |
|                 | 0.9    | 0.0002 | 0.0336 | 62.85 | 0.7572 |
The Wiener filter denoising technique is utilized in this experiment. It possesses two essential parameters, including ‘mysize’ and ‘noise’. Nonetheless, during deployment, we observed that the ‘mysize’ parameter has a significant impact on image denoising, whereas the ‘noise’ parameter makes little difference. Therefore, its values are used as a constant. The optimal result obtained with the ‘mysize’ parameter was the value (3,3). Table 7 displays the results obtained by varying its parameter values.

Table 7. Implementation of the Wiener filter with diverse parameter values.

| Noise Type       | Wiener Filter Parameters | Evaluation Metrics |
|------------------|--------------------------|--------------------|
|                  | Mysize | MSE   | PSNR   | UQI   |
| Gaussian         | (5,5)  | 0.0224| 64.54  | 0.8669|
|                  | (7,7)  | 0.0260| 63.96  | 0.8411|
|                  | (9,9)  | 0.0278| 63.68  | 0.8316|
|                  | (11,11)| 0.0291| 63.48  | 0.8248|
|                  | (3,3)  | 0.0171| 65.78  | 0.902 |
|                  | (13,13)| 0.0301| 63.34  | 0.8192|
| Salt-and-Pepper  | (5,5)  | 0.0279| 63.66  | 0.841 |
|                  | (7,7)  | 0.0323| 63.02  | 0.8026|
|                  | (9,9)  | 0.0348| 62.7   | 0.7835|
|                  | (11,11)| 0.0364| 62.51  | 0.7735|
|                  | (3,3)  | 0.0187| 65.4   | 0.9147|
|                  | (13,13)| 0.0376| 62.37  | 0.765 |
| Speckle          | (5,5)  | 0.0187| 65.39  | 0.8532|
|                  | (7,7)  | 0.0222| 64.69  | 0.8107|
|                  | (9,9)  | 0.0237| 64.38  | 0.7943|
|                  | (11,11)| 0.025  | 64.15  | 0.7879|
|                  | (3,3)  | 0.0136| 66.79  | 0.8879|
|                  | (13,13)| 0.0259| 63.98  | 0.7804|
| Poisson          | (5,5)  | 0.0186| 65.42  | 8513  |
|                  | (7,7)  | 0.0218| 64.72  | 0.8118|
|                  | (9,9)  | 0.0234| 64.42  | 0.7984|
|                  | (11,11)| 0.0247| 64.19  | 0.7884|
|                  | (3,3)  | 0.0135| 66.81  | 0.8872|
|                  | (13,13)| 0.0256| 64.03  | 0.7811|

A technique known as NL mean denoising is frequently employed to reduce the number of computations necessary. Patch size, h values, and patch distance are the three most important parameters to take into consideration. We have experimented extensively with the combination of a wide range of values to locate the result that provides the best PSNR, MSE, and UQI performance metrics. The degree of filtering that is applied is determined by the value of the parameter h, which takes the value 3, 2, or 1. Similarly, the values (11,9,5) and (3,5,7) are used for the parameter’s patch size and patch distance, respectively. We obtained the best possible result by setting the values for h, patch size, and patch distance to 1, 5, and 3, respectively. Table 8 displays the information in greater detail by making use of a combination of several different parameter values.
Table 8. Implementation of the non-local means with diverse parameter values.

| Noise Type  | Non Local Means Parameters | Evaluation Matrices |
|-------------|----------------------------|---------------------|
|             | Patch Size | h Values | Patch Distance | MSE   | PSNR  | UQI   |
| Gaussian    | 11         | 3        | 0.0351         | 62.67 | 0.8127 |
| Salt-and-Pepper | 9     | 3        | 0.0382         | 62.28 | 0.7957 |
| Salt-and-Pepper | 5     | 5        | 0.0394         | 62.17 | 0.789  |
| Salt-and-Pepper | 3     | 7        | 0.0322         | 63.04 | 0.8245 |
| Salt-and-Pepper | 2     | 5        | 0.0358         | 62.58 | 0.8045 |
| Salt-and-Pepper | 7     | 7        | 0.0373         | 62.4  | 0.7967 |
| Gaussian    | 9          | 3        | 0.0351         | 62.67 | 0.8123 |
| Salt-and-Pepper | 5     | 3        | 0.0382         | 62.28 | 0.7957 |
| Salt-and-Pepper | 7     | 7        | 0.0322         | 63.04 | 0.8245 |
| Salt-and-Pepper | 2     | 5        | 0.0358         | 62.58 | 0.8045 |
| Salt-and-Pepper | 7     | 7        | 0.0373         | 62.4  | 0.7967 |
| Gaussian    | 5          | 3        | 0.0351         | 62.67 | 0.8135 |
| Salt-and-Pepper | 9     | 3        | 0.0382         | 62.28 | 0.7957 |
| Salt-and-Pepper | 7     | 7        | 0.0322         | 63.04 | 0.8245 |
| Salt-and-Pepper | 2     | 5        | 0.0358         | 62.58 | 0.8045 |
| Salt-and-Pepper | 7     | 7        | 0.0373         | 62.4  | 0.7967 |
| Gaussian    | 3          | 5        | 0.0351         | 62.67 | 0.8135 |
| Salt-and-Pepper | 11    | 3        | 0.0382         | 62.28 | 0.7957 |
| Salt-and-Pepper | 5     | 7        | 0.0322         | 63.04 | 0.8245 |
| Salt-and-Pepper | 2     | 5        | 0.0358         | 62.58 | 0.8045 |
| Salt-and-Pepper | 7     | 7        | 0.0373         | 62.4  | 0.7967 |
The techniques that are used for denoising are evaluated alongside noise, such as Gaussian, salt-and-pepper, spackle, and poisson, to identify the most effective method and the values that should be assigned to their parameters. During the process of experimentation, we do not just use these denoising techniques singly; rather, we combine them to
get the best possible result. Despite this, the observation reveals the integration negative impact on performance in terms of MSE, PSNR, and UQI. Figure 10 presents a comparison of the various denoising techniques applied to a malware image that has been affected by Gaussian, salt-and-pepper, speckle, and poisson noise. It demonstrates how the non-local means denoising technique can effectively remove noise without discarding an excessive amount of information.

![Figure 10](image-url)

**Figure 10.** A comparison of denoising techniques in terms of PSN result for various images with noising.

At this point in the experimentation process, we have obtained an optimal technique for removing noise from malware images and are required to implement it on the original grayscale images dataset. After the implementation, the obtained denoised malware images dataset is fed into the wavelet transform to lower the memory utilization and computational cost. A DWT of wavelet transform techniques is utilized with level three for the compression, and, as a result, the dimensionality of the images is produced to the size of $28 \times 28$. Various families, such as Daubechies, Symlets, Biorthogonal, Coiflets, and Reverse Biorthogonal, are utilized during the experiment, and the results are evaluated using PSNR. PSNR values of 90.36 are produced by Daubechies, Biorthogonal, and Reverse Biorthogonal. Table 9 displays the various results obtained after deploying a variety of DWT families, along with the parameters that are specific to each family.
| Family        | Discrete Wavelet Transform Parameters | Evaluation Metric |
|---------------|---------------------------------------|-------------------|
|               | db1                                   | 90.36             |
|               | db2                                   | 95.17             |
|               | db3                                   | 49.55             |
|               | db4                                   | 56.35             |
|               | db5                                   | 54.1              |
|               | db6                                   | 50.02             |
|               | db7                                   | 49.45             |
|               | db8                                   | 50.22             |
|               | db9                                   | 50.83             |
|               | db10                                  | 53.01             |
| Daubechies    | sym2                                  | 51.14             |
|               | sym3                                  | 51.05             |
|               | sym4                                  | 51.07             |
|               | sym5                                  | 51.34             |
|               | sym6                                  | 51.14             |
|               | sym7                                  | 51.26             |
|               | sym8                                  | 51.17             |
|               | sym9                                  | 48.63             |
| Symlets       | bior1.1                               | 90.36             |
|               | bior1.3                               | 75.79             |
|               | bior1.5                               | 70.15             |
|               | bior2.2                               | 56.7              |
|               | bior2.4                               | 68.45             |
|               | bior2.6                               | 64.17             |
|               | bior2.8                               | 66.44             |
|               | bior3.1                               | 48.13             |
|               | bior3.3                               | 54.63             |
|               | bior3.5                               | 54.01             |
|               | bior3.7                               | 53.72             |
|               | bior3.9                               | 53.7              |
|               | bior4.4                               | 51.17             |
|               | bior5.5                               | 5.09              |
|               | bior6.8                               | 51.14             |
As a result, multiple families produced the same values of PSNR to evaluate the malware images. We have chosen to conduct our experiment using the Daubechies family’s db1 value. In the final step, the dataset of malware images is now being fed into the machine learning system to detect and classify malware families. We compared the accuracy, precision, recall, and f1-score of ten of the most popular machine learning classifiers. Initially, the malware image dataset is converted into a feature vector before being fed to classifiers. None of the machine learning classifiers’ parameters is tuned at the start of the process. The goal is to achieve the top five results, after which it is necessary to fine-tune its parameters to achieve further improvements. Figure 11 displays the experiment results using a variety of machine learning classifiers. The results show that the extra tree classifier performs well, with an accuracy of 97.19%, precision of 97.63%, recall of 96.11%, and f1-score of 96.76%. Machine learning techniques such as SVM with RBF kernel, histogram-based gradient boosting, k-nearest neighbor, and random forest are used to obtain the other closest results.

| Family       | Discrete Wavelet Transform Parameters | Evaluation Metric |
|--------------|---------------------------------------|-------------------|
| Coiflets     |                                        | PSNR              |
|              | coif1                                 | 57.7              |
|              | coif2                                 | 58.24             |
|              | coif3                                 | 56.31             |
|              | coif4                                 | 50.89             |
|              | coif5                                 | 52.54             |
|              | coif6                                 | 56.15             |
|              | coif7                                 | 57.62             |
|              | coif8                                 | 58.37             |
|              | coif9                                 | 51.51             |
|              | coif10                                | 54.15             |
|              | rbior1.1                              | 90.36             |
|              | rbior1.3                              | 71.75             |
|              | rbior1.5                              | 69.87             |
|              | rbior2.2                              | 52.85             |
|              | rbior2.4                              | 57.96             |
|              | rbior2.6                              | 57.23             |
|              | rbior2.8                              | 59.82             |
|              | rbior3.1                              | 49.38             |
|              | rbior3.3                              | 51.65             |
|              | rbior3.5                              | 51.96             |
|              | rbior3.7                              | 52.15             |
|              | rbior3.9                              | 52.37             |
|              | rbior4.4                              | 51.17             |
|              | rbior5.5                              | 51.09             |
|              | rbior6.8                              | 51.14             |

Table 9. Cont.
As a result, multiple families produced the same values of PSNR to evaluate the malware images. We have chosen to conduct our experiment using the Daubechies family’s db1 value. In the final step, the dataset of malware images is now being fed into the machine learning system to detect and classify malware families. We compared the accuracy, precision, recall, and f1-score of ten of the most popular machine learning classifiers. Initially, the malware image dataset is converted into a feature vector before being fed to classifiers. None of the machine learning classifiers’ parameters is tuned at the start of the process. The goal is to achieve the top five results, after which it is necessary to fine-tune its parameters to achieve further improvements. Figure 11 displays the experiment results using a variety of machine learning classifiers. The results show that the extra tree classifier performs well, with an accuracy of 97.19%, precision of 97.63%, recall of 96.11%, and f1-score of 96.76%. Machine learning techniques such as SVM with RBF kernel, histogram-based gradient boosting, k-nearest neighbor, and random forest are used to obtain the other closest results.

These top five classifiers have undergone hyperparameter tuning to improve overall performance. Table 10 provides an in-depth look at the hyperparameter settings for the top five machine learning classifiers, each of which is presented with its corresponding value. The SVC with kernel values of RBF, gamma 0.01, and C 100 shows the improvement in the results in terms of accuracy, precision, recall, and f1-score. Parameter C performs a crucial function by requiring the classifier to avoid misclassification and overfitting during training, whereas the gamma indicates the influence, with lower values indicating that the training data are distant, and higher values indicating that the training data are close.

Table 10. Hyper-tuning process of top five machine learning classifiers.

| Classifiers                  | Parameters Name | Parameters Values          | Optimal Values |
|-----------------------------|-----------------|----------------------------|----------------|
| SVC                         | C               | 1, 5, 10, 100, 1000        | 100            |
|                             | gamma           | 1, 0.1, 0.01, 0.001, 0.0001 | 0.01           |
|                             | kernel          | ‘RBF’, ‘Linear’            | ‘RBF’          |
| Extra Tree Classifier       | n_estimators    | 10, 50, 100, 200, 300      | 300            |
|                             | criterion       | ‘gini’, ‘entropy’, ‘log_loss’ | ‘Gini’        |
| Histogram-based Gradient Boosting | learning_rate | 0.1, 0.01, 0.001, 0.0001 | 0.1            |
|                             | loss            | ‘log_loss’, ‘auto’, ‘categorical_crossentropy’ | ‘categorical_crossentropy’ |
|                             | l2_regularization | 0, 1                          | 0              |
| K_Nearest Neighbor          | n_neighbors     | 1, 5, 10, 15, 20           | 1              |
|                             | weights         | ‘uniform’, ‘distance’       | ‘uniform’      |
|                             | algorithm       | ‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’ | ‘auto’        |
| Random Forest               | n_estimators    | 10, 50, 100, 155, 200      | 50             |
|                             | criterion       | ‘gini’, ‘entropy’, ‘log_loss’ | ‘gini’        |
After fine-tuning the parameters of the top five machine learning classifiers, we revealed that SVM with RBF kernel achieved the highest accuracy of 97.82%, precision of 97.66%, recall of 97.25%, and f1-score of 97.57%, while the extra tree classifier achieved the second highest accuracy of 97.28%, precision of 97.35%, recall of 96.12%, and f1-score of 96.73%. It is noted that hyper-tuning of SVM with RBF kernel improved its performance more than other classifiers. Figure 12 depicts the results obtained after hyperparameter tuning is performed.

It is necessary to examine the confusion matrix of these classifiers to determine the prediction and types of errors, such as type-I and type-II, to distinguish between various malware classes. The confusion matrix consists of two crucial dimensions: the actual label and the predicted label. Actual labels are the true values for the given observations, while predicted labels are the values predicted by the classifier. The confusion matrix of the top five high-performance classifiers is depicted in Figure 13. The ‘Dinwod’ malware class performed the worst among all classifiers, and this is due to its visual similarity to the benign “other” class. Similarly, the ‘Adposhel’ and ‘BrowseFox’ classes are classified with 100% accuracy by all classifiers. It is noted that these classes are dissimilar, making it simple for classifiers to differentiate between them. In terms of binary classification, the confusion matrix also reveals that KNN and SVM with an RBF kernel are confused between benign and malicious images, despite the fact that both classifiers accurately identified various other malware classes.

To calculate the precision, recall, and f1-score for each class of malware, we display the classification of various malware classes in tabular format for the SVM with RBF kernel. Figure 14 displays the information in greater detail regarding the classification report generated by SVM with RBF kernel for each class in terms of precision, recall, and f1-score. These metrics of performance, as well as accuracy, must be given equal consideration. According to the literature review, accuracy does not always provide the most effective result; as a result, other performance metrics should be considered. The results demonstrate that malware classes such as ‘Adposhel’, ‘BrowseFox’, and ‘VBA’ performed exceptionally well using the SVM with RBF kernel classifier of machine learning, whereas ‘Dinwod’ performed the worst among the remaining classes. There could be a number of reasons why this class of malware performs less well than others, including
(i) the lack of insufficient training data for the classifier, (ii) the loss of a large number of sensitive data during compression, and (iii) insufficient noise removed by denoising techniques. To determine the root cause, additional investigation is still necessary. In conclusion, our techniques improve not only accuracy, but also precision, recall, f1-score, and memory storage, while reducing the computational cost.

Figure 13. Confusion matrix of best five machine learning classifiers.
Figure 13. Confusion matrix of best five machine learning classifiers.

Figure 14. Classification report of SVM with RBF kernel.

To demonstrate the effectiveness of the proposed techniques in detecting and classifying malware classes when applied to an unbalanced dataset, it is observed that the performance of the proposed technique outperformed other relevant approaches based on visualization. We compared the outcome to the most recent research published in [12,13,36,38,59]. The experimentation shows that feature engineering is a critical step before the data is fed into machine learning or deep learning. Feature engineering techniques such as GIST descriptor combined with machine learning are proposed in [12]. Similarly, a HOG feature descriptor in conjunction with multi-layer perceptron is proposed in [38]. A combination of the positive features of both these feature descriptors is proposed in [13]. However, it is observed that it is computationally and memory-intensive. To circumvent the high computational cost and memory consumption, our proposed method effectively reduces the cost in terms of computing and memory and outperforms the competition in terms of accuracy, precision, recall, and f1-score. Our proposed technique begins by denoising the images to eliminate irrelevant texture features, and then it reduces the dimensions of these images to a size of $28 \times 28$ to maintain the essential texture features, while minimizing the amount of information that is lost. Table 11 compares the performance of the proposed technique to the most pertinent research.

Table 11. A comparison of achieved results with the most relevant research.

| S.No | Ref. | Accuracy | Precision | Recall | F1-Score |
|------|------|----------|-----------|--------|----------|
| 1    | [12] | $91.40\%$ | $91.50\%$ | $91.40\%$ | $91.50\%$ |
| 2    | [38] | $94.54\%$ | $94.60\%$ | $94.50\%$ | $94.50\%$ |
| 3    | [59] | $96.93\%$ | $97.00\%$ | $96.90\%$ | $96.90\%$ |
| 4    | [13] | $96.36\%$ | $96.40\%$ | $96.40\%$ | $96.40\%$ |
| 5    | [36] | $97.01\%$ | $97.36\%$ | $95.65\%$ | $96.36\%$ |
| 6    | Proposed | $97.82\%$ | $97.66\%$ | $97.25\%$ | $97.57\%$ |

5. Discussion

We used a denoising technique to remove noise from the converted malware images, which improved the performance of machine learning models in terms of accuracy, precision, recall, and f1-score. We considered common noise types, such as Gaussian, Salt-and-Pepper, Speckle, and Poisson, and applied various denoising filter techniques...
to determine the optimal values for each parameter. However, one of the unanticipated results demonstrates that a malware class called 'Dinwod' performed poorly, indicating that the denoising technique removed too many important artifacts from the images and made it difficult for the machine learning model to differentiate between various malware types. It is therefore necessary to conduct additional research into the cause of the loss of crucial malware artifacts belonging to specific classes.

In this paper, we used DWT compression techniques to reduce the dimensionality and remove the irrelevant or duplicate artifacts from malware images without sacrificing too many key artifacts. Using this technique yields results that are highly efficient in terms of computation and memory consumption. Nonetheless, throughout the experimental phase, we discovered that the performance of distinct malware classes changes with DWT levels 1, 2, and 3. The findings reveal that the procedure of dimensionality reduction and compression performed well for the majority of malware classes, but degraded the performance for a few forms of malware, necessitating more investigation. Lastly, in this paper, we do not consider a feature selection technique to further reduce the number of features. We are hopeful that, by utilizing a feature selection technique, the cost of computing and memory usage can be further reduced without sacrificing performance accuracy.

6. Conclusions

The detection and classification of malware play a crucial role in cyber security. It is necessary to comprehend the behavior of malware to detect it. The most prevalent analysis methods include static, dynamic, and hybrid. However, such an analysis requires domain-specific expertise to identify the crucial characteristics of a suspicious file, which is not always possible. In this paper, memory dump files from malware are converted into grayscale images to avoid necessitating domain expertise. We believed that during the dumping and visualization processes, image noise is also introduced. Numerous studies have proposed detecting and classifying malware using visualization. However, none of the researchers emphasizes the significance of denoising visual images. This paper proposes two important techniques: non-local means denoising for noise removal and discrete wavelet transform for image compression to reduce image dimensions. Finally, these images are fed into classifiers that utilize machine learning. SVM with RBF kernel achieved the best accuracy, precision, recall, and f1-score among the best classifiers. In the future, it will be necessary to focus on malware classes for which the detection rate is less than 90%. The denoising of grayscale malware images is still novel in the detection and classification of malware, and we are optimistic that more research is necessary. This study has not evaluated cloud computing technology; nonetheless, we are hopeful that using such an approach will also improve the level of cybersecurity in an online environment.

Author Contributions: S.S.H.S.: Conceptualization, data curation, formal analysis, visualization, validation, writing original draft, methodology, software, investigation. N.J.: Supervision, conceptualization, validation, methodology, investigation, writing review-editing. A.u.R.K.: supervision, conceptualization, validation, methodology, investigation, writing-review, project administration. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the Ministry of Higher Education Malaysia under the Transdisciplinary Research Grant Scheme (TRGS) of Grant No. ‘TRGS/1/2020/UNITEN/01/1/2’ and BOLDRefresh2022 Publication Fund by Universiti Tenaga Nasional.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.
References

1. Shafique, A.; Mehmoood, A.; Alawida, M.; Khan, A.N.; Khan, A.U.R. A Novel Machine Learning Technique for Selecting Suitable Image Encryption Algorithms for IoT Applications. *Wirel. Commun. Mob. Comput.* **2022**, *2022*, 5108331. [CrossRef]

2. Aslan, Ö.A.; Samet, R. A comprehensive review on malware detection approaches. *IEEE Access* **2020**, *8*, 6249–6271. [CrossRef]

3. Harter, G.T.; Rowe, N.C. Testing Detection of K-Ary Code Obfuscated by Metamorphic and Polymorphic Techniques. In *National Cyber Summit*; Springer: Cham, Switzerland, 2021; pp. 110–123.

4. Indusface. New Malware Report. Available online: https://www.indusface.com/blog/15-malware-statistics-to-take-seriously-in-2022/#_ednref1 (accessed on 10 June 2022).

5. AV-TEST. Malware Development. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 12 June 2022).

6. Abbas, S.; Faisal, M.; Rahman, H.U.; Khan, M.Z.; Merabti, M.J.A. Masquerading attacks detection in mobile ad hoc networks. *IEEE Access* **2018**, *6*, 55013–55025. [CrossRef]

7. SecureList. Mobile Malware Report. Available online: https://securelist.com/it-threat-evolution-in-q1-2022-mobile-statistics/106589/ (accessed on 10 June 2022).

8. Kaspersky. Malware Attack on PC. Available online: https://securelist.com/it-threat-evolution-in-q1-2022-non-mobile-statistics/s/106531/ (accessed on 15 June 2022).

9. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic malware analysis in the modern era—A state of the art survey. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–48. [CrossRef]

10. Shihwail, R.; Omar, K.; Ariffin, K.Z. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *Int. J. Adv. Sci. Eng. Inf. Technol.* **2018**, *8*, 1662–1671. [CrossRef]

11. Damodaran, A.; Troia, F.D.; Visaggio, C.A.; Austin, T.H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hacking Tech.* **2015**, *13*, 1–12. [CrossRef]

12. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, Pittsburgh, PA, USA, 28 October 2020; pp. 1–7.

13. Bozkir, A.S.; Tahillioglu, E.; Aydos, M.; Kara, I. Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. *Comput. Secur.* **2021**, *103*, 102166. [CrossRef]

14. Willems, C.; Holz, T.; Freiling, F. Toward automated dynamic analysis using cwsandbox. *Secur. Priv.* **2007**, *5*, 32–39. [CrossRef]

15. Shijo, P.; Salim, A.J.P.C.S. Integrated static and dynamic analysis for malware detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [CrossRef]

16. Shihwail, R.; Omar, K.; Zainol Ariffin, K.A.; Al Afghani, S. Malware detection approach based on artifacts in memory image and dynamic analysis. *Appl. Sci.* **2019**, *9*, 3680. [CrossRef]

17. Russo, F. A method for estimation and filtering of Gaussian noise in images. *IEEE Trans. Instrum. Meas.* **2003**, *52*, 1148–1154. [CrossRef]

18. Pimpalkhute, V.A.; Page, R.; Kothari, A.; Bhurchandi, K.M.; Kamble, V.M. Digital image noise estimation using DWT coefficients. *IEEE Trans. Image Process.* **2021**, *30*, 1962–1972. [CrossRef] [PubMed]

19. Kollem, S.; Reddy, K.R.L.; Rao, D.S. A review of image denoising and segmentation methods based on medical images. *Int. J. Mach. Learn. Comput.* **2019**, *9*, 288–295. [CrossRef]

20. Ahmad, K.; Khan, J.; Iqbal, M.S.U.D. A comparative study of different denoising techniques in digital image processing. In Proceedings of the 2019 8th International Conference on Modeling Simulation and Applied Optimization (ICMSAO), Manama, Bahrain, 15–17 April 2019; pp. 1–6.

21. Hashemi, H.; Hamzeh, A. Visual malware detection using local malicious pattern. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 1–14. [CrossRef]

22. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 3–7 September 2018; pp. 533–537.

23. Salehi, Z.; Sami, A.; Ghiasi, M. Using feature generation from API calls for malware detection. *Comput. Fraud Secur.* **2014**, *2014*, 9–18.

24. Veeramani, R.; Rai, N. Windows api based malware detection and framework analysis. In Proceedings of the International Conference on Networks and Cyber Security, Alexandria, VA, USA, 14–16 December 2012.

25. Christodorescu, M.; Jha, S.; Kinder, J.; Katzenbeisser, S.; Veith, H. Software transformations to improve malware detection. *J. Comput. Virol.* **2007**, *3*, 253–265.

26. Ofyama, Y. Trends of anti-analysis operations of malwares observed in API call logs. *J. Comput. Virol. Hacking Tech.* **2018**, *14*, 69–85. [CrossRef]

27. Mehmoood, A.; Khan, A.N.; Elhadef, M. HeuCrip: A malware detection approach for internet of battlefield things. *Clust. Comput.* **2022**, 1–16.

28. Cheng, Y.; Fan, W.; Huang, W.; An, J. A shellcode detection method based on full native api sequence and support vector machine. In *Proceedings of the IOP Conference Series: Materials Science and Engineering, Birmingham, UK*, 13–15 October 2017; IOP Publishing: Bristol, UK; p. 012124.
29. Bayer, U.; Kirda, E.; Kruegel, C. Improving the efficiency of dynamic malware analysis. In Proceedings of the 2010 ACM Symposium on Applied Computing, Sierre, Switzerland, 22–26 March 2010; pp. 1871–1878.

30. Udayakumar, N.; Anandaselvi, S.; Subbulakshmi, T. Dynamic malware analysis using machine learning algorithm. In Proceedings of the 2017 International Conference on Intelligent Sustainable Systems (ICISS), Palladam, India, 7–8 December 2017; pp. 795–800.

31. Zhang, Z.; Qi, P.; Wang, W. Dynamic malware analysis with feature engineering and feature learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; pp. 1210–1217.

32. Jindal, C.; Sillis, C.; Aghakhani, H.; Long, K.; Kruegel, C.; Vigna, G. NeurLux: Dynamic malware analysis without feature engineering. In Proceedings of the 35th Annual Computer Security Applications Conference, San Juan, PR, USA, 9–13 December 2019; pp. 444–455.

33. Ijaz, M.; Durad, M.H.; Ismail, M. Static and dynamic malware analysis using machine learning. In Proceedings of the 2019 16th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 8–12 January 2019; pp. 687–691.

34. Raghiraman, C.; Suress, S.; Shivshankar, S.; Chaparani, R. Static and dynamic malware analysis using machine learning. In Proceedings of the First International Conference on Sustainable Technologies for Computational Intelligence, Islamabad, Pakistan, 8–12 January 2019; Springer: Cham, Switzerland; pp. 793–806.

35. Zhang, S.; Zhao, C.; Huang, B. Simultaneous static and dynamic analysis for fine-scale identification of process operation statuses. IEEE Trans. Ind. Inform. 2019, 15, 5320–5329.

36. Shah, S.S.H.; Ahmad, A.R.; Jamil, N.; Khan, A.U.R. Memory Forensics-Based Malware Detection Using Computer Vision and Machine Learning. Electronics 2022, 11, 2579.

37. Sihwail, R.; Omar, K.; Ariffin, K.A.Z. An effective memory analysis for malware detection and classification. Comput. Mater. Contin. 2021, 67, 2301–2320. [CrossRef]

38. Dat, Y.; Li, H.; Qian, Y.; Lu, X. A malware classification method based on memory dump grayscale image. Digit. Investig. 2018, 27, 30–37. [CrossRef]

39. Mosli, R.; Li, R.; Yuan, B.; Pan, Y. Automated malware detection using artifacts in forensic memory images. In Proceedings of the 2016 IEEE Symposium on Technologies for Homeland Security (HST), Waltham, MA, USA, 10–12 May 2016; pp. 1–6.

40. Rathnayaka, C.; Jamdagni, A. An efficient approach for advanced malware analysis using memory forensic technique. In Proceedings of the 2017 IEEE Trustcom /BigDataSE/ICCESS, Sydney, Australia, 1–4 August 2017; pp. 1145–1150.

41. Teller, T.; Hayon, A. Enhancing Automated Malware Analysis Machines with Memory Analysis; Black Hat: Las Vegas, NV, USA, 2014.

42. Nissim, N.; Lahav, O.; Cohen, A.; Elovici, Y.; Rokach, L. Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud. Comput. Secur. 2019, 87, 101590. [CrossRef]

43. Tien, C.-W.; Liao, J.-W.; Chang, S.-C.; Kuo, S.-Y. Memory forensics using virtual machine introspection for malware analysis. In Proceedings of the 2017 IEEE Conference on Dependable and Secure Computing, Taipei, Taiwan, 7–10 August 2017; pp. 518–519.

44. Choi, S.; Jang, S.; Kim, Y.; Kim, J. Malware detection using malware image and deep learning. In Proceedings of the 2017 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Korea, 18–20 October 2017; pp. 1193–1195.

45. Davies, S.R.; Macfarlane, R.; Buchanan, W.J. Evaluation of live forensic techniques in ransomware attack mitigation. Forensic Sci. Int. Digit. Investig. 2020, 33, 300979. [CrossRef]

46. Kumara, M.A.; Jaidhar, C.D. Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. Digit. Investig. 2017, 23, 99–123. [CrossRef]

47. Sali, V.R.; Khanuja, H. Ram forensics: The analysis and extraction of malicious processes from memory image using gui based memory forensic toolkit. In Proceedings of the 2018 Fourth International Conference on Computing Communication Control and Automation (ICCCUBEA), Pune, India, 16–18 August 2018; pp. 1–6.

48. Tekerek, A.; Yapici, M.M. A novel malware classification and augmentation model based on convolutional neural network. Comput. Secur. 2022, 112, 102515. [CrossRef]

49. Kalash, M.; Rochan, M.; Mohammed, N.; Bruce, N.D.; Wang, Y.; Iqbal, F. Malware classification with deep convolutional neural networks. In Proceedings of the 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), Paris, France, 26–28 February 2018; pp. 1–5.

50. Hemalatha, J.; Roseline, S.A.; Geetha, S.; Kadry, S.; Damaševičius, R. An efficient densenet-based deep learning model for malware detection. Entropy 2021, 23, 344. [CrossRef]

51. Aslan, Ö.; Yilmaz, A.A. A new malware classification framework based on deep learning algorithms. IEEE Access 2021, 9, 87936–87951. [CrossRef]

52. Dumpware10. Memory Based Malware Dataset. Available online: https://web.cs.hacettepe.edu.tr/~selman/dumpware10/ (accessed on 20 March 2022).

53. Norton. Adware. Available online: https://us.norton.com/internetsecurity-emerging-threats-what-is-grayware-adware-and-malware.html (accessed on 15 June 2022).

54. Github. bin2png Version. Available online: https://github.com/ESultanik/bin2png (accessed on 10 November 2021).

55. Garnett, R.; Huegerich, T.; Chui, C.; He, W. A universal noise removal algorithm with an impulse detector. IEEE Trans. Image Process. 2005, 14, 1747–1754. [CrossRef]
56. Kumain, S.C.; Singh, M.; Singh, N.; Kumar, K. An efficient Gaussian noise reduction technique for noisy images using optimized filter approach. In Proceedings of the 2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC), Jalandhar, India, 15–17 December 2018; pp. 243–248.

57. Azzeh, J.; Zahran, B.; Alqadi, Z. Salt and pepper noise: Effects and removal. JOIV Int. J. Inform. Vis. 2018, 2, 252–256. [CrossRef]

58. Duarte-Salazar, C.A.; Castro-Ospina, A.E.; Becerra, M.A.; Delgado-Trejos, E. Speckle noise reduction in ultrasound images for improving the metrological evaluation of biomedical applications: An overview. IEEE Access 2020, 8, 15983–15999. [CrossRef]

59. Rezende, E.; Ruppert, G.; Carvalho, T.; Theophilo, A.; Ramos, F.; Geus, P.D. Malicious software classification using VGG16 deep neural network’s bottleneck features. In Information Technology-New Generations; Springer: Cham, Switzerland, 2018; pp. 51–59.