

# IMAGE STEGANOGRAPHY USING DEEP LEARNING TECHNIQUES

by

**Anthony Rene Guzman**

**A Thesis**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Master of Science in Computer Science**



Department of Engineering, Technology, and Computer Science

Fort Wayne, Indiana

May 2022

**THE PURDUE UNIVERSITY GRADUATE SCHOOL**  
**STATEMENT OF COMMITTEE APPROVAL**

**Dr. Venkata Inukollu, Chair**

School of Engineering, Technology, and Computer Science

**Dr. Amal Khalifa**

School of Engineering, Technology, and Computer Science

**Dr. Mohammadreza Hajiarbabi**

School of Engineering, Technology, and Computer Science

**Dr. David Liu**

School of Engineering, Technology, and Computer Science

**Approved by:**

Dr. Venkata Inukollu

*Dedicated to my father*

## **ACKNOWLEDGMENTS**

I would like to acknowledge the College of Engineering, Technology, and Computer Science at Purdue University Fort Wayne for funding the purchase of the Google Colab Pro subscription used in this research. By funding the use of Google Colab Pro, results were obtained more efficiently than previous attempts with the network training process. I would also like to acknowledge Dr. Amal Khalifa for all her help throughout the entire research process. Without her guidance and assistance throughout the entire research process, this work would not have been possible. Lastly, I would like to acknowledge the anonymous reviewers for their many insightful comments and suggestions.

## TABLE OF CONTENTS

LIST OF TABLES .....	6
LIST OF FIGURES .....	7
ABSTRACT .....	8
1. INTRODUCTION .....	9
1.1 Steganography .....	9
1.2 Image Steganography .....	10
1.3 Convolutional Neural Networks .....	13
2. LITERATURE REVIEW .....	22
2.1 Spatial Domain Techniques .....	22
2.2 Transform Domain Techniques .....	24
2.3 Neural Network Techniques .....	28
2.4 Summary .....	29
3. METHODOLOGY .....	32
3.1 The Original Method .....	32
3.2 The Enhanced Network .....	35
4. RESULTS .....	37
4.1 Model Training & Testing .....	37
4.2 Experiment Setup .....	37
4.3 Results of the Original Network .....	41
4.4 Results of the Enhanced Network .....	46
5. CONCLUSIONS .....	53
REFERENCES .....	55

## LIST OF TABLES

Table 2.1. Review of various image steganography methods. ....	30
Table 4.1. List of labels and images used between experiments. ....	39
Table 4.2. Side-by-side comparison of steganographic and extracted images generated by the original network trained on the ImageNet dataset and the original network trained on the Imagenette and Linnaeus 5 dataset. ....	42
Table 4.3. The steganographic and extracted images generated by the original method. ....	43
Table 4.4. Side-by-side comparison of the steganographic and extracted images generated by the original and proposed methodologies. ....	48
Table 4.5. The steganographic and extracted images generated by the enhanced methodology..	49
Table 4.6. A comparison of performance metrics between the original and enhanced networks.	52

## LIST OF FIGURES

Figure 1.1. The basic architecture of the digital image steganography process. ....	11
Figure 1.2. Structure of a basic neural network. ....	13
Figure 1.3. High-level architecture of a perceptron. ....	15
Figure 1.4. Illustration of a CNN with several convolutional layers to extract the features of an input image. ....	16
Figure 1.5. Effects of a small and large learning rate on the gradient descent method. ....	17
Figure 1.6. The convergence of the gradient descent method on a local minimum, where a global minimum exists that provides a more optimal solution. ....	18
Figure 2.1. Illustration of encrypting and embedding text into a cover image's pixels. ....	23
Figure 2.2. Example of a two-level discrete wavelet transform. ....	26
Figure 2.3. Labeled wavelet decomposition performed at three levels. ....	27
Figure 3.1. Structure of the overall network. The structure of the preparatory network is identical between the hiding and reveal networks, with the exception of the hiding and reveal networks taking an additional step of performing a two-dimension convolution on the concatenated tensors. ....	33

## ABSTRACT

Digital image steganography is the process of embedding information within a cover image in a secure, imperceptible, and recoverable way. The three main methods of digital image steganography are spatial, transform, and neural network methods. Spatial methods modify the pixel values of an image to embed information, while transform methods embed hidden information within the frequency of the image. Neural network-based methods use neural networks to perform the hiding process, which is the focus of the proposed methodology.

This research explores the use of deep convolutional neural networks (CNNs) in digital image steganography. This work extends an existing implementation that used a two-dimensional CNN to perform the preparation, hiding, and extraction phases of the steganography process. The methodology proposed in this research, however, introduced changes into the structure of the CNN and used a gain function based on several image similarity metrics to maximize the imperceptibility between a cover and steganographic image.

The performance of the proposed method was measured using some frequently utilized image metrics such as structured similarity index measurement (SSIM), mean square error (MSE), and peak signal to noise ratio (PSNR). The results showed that the steganographic images produced by the proposed methodology are imperceptible to the human eye, while still providing good recoverability. Comparing the results of the proposed methodology to the results of the original methodology revealed that our proposed network greatly improved over the base methodology in terms of SSIM and compares well to existing steganography methods.



# 1. INTRODUCTION

## 1.1 Steganography

Steganography is the process of hiding a message by placing it within another, innocuous medium. The literal meaning of the word, derived from the Greek language, means “covered writing” [1]. The creator of the word, Trithemius, used the Greek words “steganos” and “graphia”, which mean “covered” and “writing”, respectively. The earliest known application of steganography was the use of invisible ink. One notable example of steganography is found in a story by Greek historian Herodotus. In this story, a Milesian tyrant ordered a slave be sent to the city of Miletus with a secret message tattooed onto his scalp. To obscure the tattooed message from, the slave allowed for time to pass such that his hair grew back, fully obscuring the tattoo. Once he arrived at Miletus, he shaved his head to reveal the tattooed message to the city advisor, Aristagoras. Upon discovery of the message, Aristagoras started a revolution against the Persian king [2]. In this scenario, the slave was the carrier of the message, the tattooed scalp was the hidden information, the tyrant was the sender, and Aristagoras was the receiver.

Another ancient example of steganography is the story of Demaratus, also written by Herodotus. Demaratus, who would go on to eventually become the king of Sparta, alerted the Persian king Xerxes of an impending invasion of Greece by using an early form of steganography [3]. Demaratus scraped the wax off the writing surface of a wooden writing tablet then embedded his message into the underlying wood. He then re-coated the writing tablet with a fresh coat of wax, giving the appearance of an unused writing tablet. In this case, the seemingly unused wooden writing tablet was the carrier of the message, the underlying scratches composed the hidden information, Demaratus was the sender, and Xerxes was the receiver.

More recently, there are many examples of steganography and information embedding in everyday life. When a \$100 (USD) bill is held to a light, one can faintly see the visage of Benjamin Franklin. \$100 bills also contain microprinting, which is nearly invisible to the naked eye. Using a magnifying glass on a \$100 bill reveals tiny text printed on the currency. Of more relevance to this research, however, is the steganography of digital information. Today, many methods exist for embedding hidden information inside images, audio files, hypertext transport protocol (HTTP) headers, and even plain text files [4].

Images can hide information in the visually uninteresting parts, reducing the likelihood that someone would be able to tell that hidden information exists. However, the ability to hide information extends to other types of files as well. Audio files can also embed information in several ways, such as by manipulating the least significant bits (LSB) or using transform techniques (e.g., Discrete Wavelet Transform (DWT)) [5]. Nearly any type of digital data can be embedded into an audio file, from text files to other audio files. To this end, any type of digital data can be embedded into a cover digital medium. This research, however, focuses on the process of embedding a digital image within another.

## **1.2 Image Steganography**

Images provide an ideal medium for embedding hidden information for several reasons. The first reason is that images contain many different characteristics, such as bit depth, colors, edges, corners, dimensions, and metadata [6]. Each of these characteristics provide the ability to hide a payload into an image with ease. In addition to these characteristics, the metadata of certain image formats can also be manipulated to embed information. Graphics Interchange Format (GIF) image files contain a palette of the colors used within the image. By permutating the image's color map, the image remains the same perceptually but contains some hidden information [4]. The second reason is that images are resistant to changes in pixel values, with some exceptions. Many steganography methods modify the least significant bits of the pixels to embed the bits of a hidden image into a cover image. Since most images are split into three, 8-bit wide color channels, red, green, and blue, this provides plenty of space in which information can be embedded.

The basic architecture of image steganography is shown in Figure 1.1. A sender uses two components, the message and payload, to construct the steganographic message. The construction of the steganographic message is done by an encoding process. For the sake of clarity, several types of encoding processes will be described later in this study. Once the encoding process is complete, the steganographic image is then transmitted to the receiving party. The transmission of the steganographic image can be done over secured or unsecured channels, since entities who gain access to the steganographic image will not benefit from it unless they are explicitly aware that it contains a hidden payload. Once the steganographic image is received, the receiving party decodes the message using a decoding process. In the majority of steganographic methodologies, the

decoding process is the same as the encoding process but performed in the opposite order. Once the decoding process is complete, the receiver will be able to obtain the hidden image.

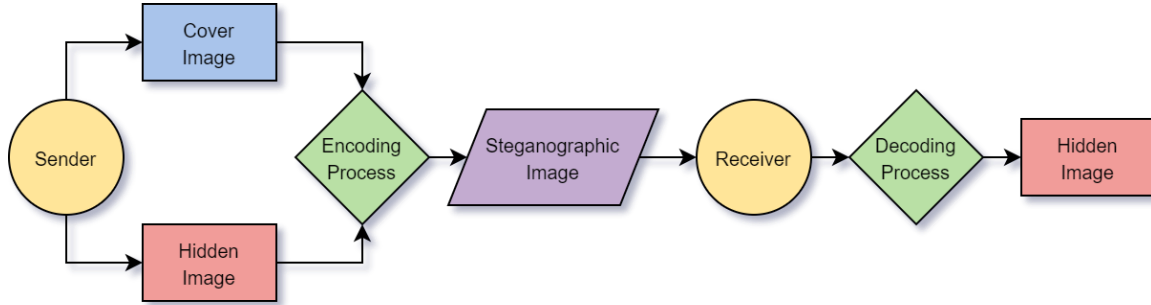


Figure 1.1. The basic architecture of the digital image steganography process.

A variety of standardized measurements are used to evaluate image steganography techniques in terms of the hiding capacity, retrieval similarity, hiding effectiveness, and resistance to attacks. These measurements can be used to compare the performance of steganography techniques. Since many techniques vary greatly between methodologies, implementations, and image datasets used, fluctuations of image metrics are to be expected. To mitigate these fluctuations, this research relied on the use of standardized image sets such as ImageNet [7] and Linnaeus 5 [8].

Below lists several common metrics that are used to evaluate image steganographic techniques:

- Bits per pixel (BPP): a measurement used to determine the hiding capacity of a steganography technique. BPP values have a range of  $[0, 24]$  for regular RGB images. A greater BPP means that the technique can hide more information inside the pixels of the cover image. BPP is measured by the following algorithm, where  $E$  represents the embedding capacity of the technique and  $H$  and  $W$  represent the height and width of the cover image, respectively [9]:

$$BPP = \frac{E}{H*W} \quad (1.1)$$

- Mean of Square Error (MSE): a measurement used to determine the average square error between the cover and steganography images. A lower MSE value represents a lower error measure between a cover and steganographic image. MSE is measured by the algorithm provided in Equation 1.2, where  $M$  represents the number of image rows,  $N$  represents the number of image columns,  $J(i, j)$  represents the cover image dimensions, and  $J'(i, j)$  represents the steganographic image dimensions:

$$MSE = \frac{\sum_{i=1}^m \sum_{j=1}^n (J(i, j) - J'(i, j))^2}{M * N} \quad (1.2)$$

- Peak Signal to Noise Ratio (PSNR): a measurement used to determine the difference between the cover and steganographic images. A higher PSNR value represents less distortion and greater similarity to the original image. PSNR is measured by the algorithm provided in Equation 1.3, where  $n$  represents the maximum pixel value and  $MSE$  represents the mean of square error:

$$PSNR = 10(\log_{10}(\frac{n^2}{MSE})) \quad (1.3)$$

- Structured Similarity Index Measure (SSIM): a measurement used to determine the similarity between two images [10]. A higher SSIM value represents a greater similarity between the cover and steganographic images. SSIM is measured by the algorithm provided in Equation 1.4, where  $\mu_x$  represents the mean of  $x$ ,  $\mu_y$  represents the mean of  $y$ ,  $\sigma_{xy}$  represents the covariance of  $x$  and  $y$ ,  $\sigma_x^2$  represents the mean variance of  $x$ ,  $\sigma_y^2$  represents the mean variance of  $y$ , and  $c_1$  and  $c_2$  represent two variables to stabilize division, in the event that the denominator is weak [11]:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \quad (1.4)$$

- Normalized Cross Correlation (NCC): a measurement used to determine the degree of similarity between the original and extracted secret image. A higher NCC value

represents a greater similarity between the original and extracted secret images. NCC is measured by the algorithm provided in Equation 1.5, where the  $O_{pq}$  represents the original image and  $C_{pq}$  represents the extracted secret image:

$$NCC = \sum_{p=1}^x \sum_{q=1}^y \frac{(O_{pq} * C_{pq})}{(O_{pq})^2} \quad (1.5)$$

### 1.3 Convolutional Neural Networks

Deep learning is a type of approach to artificial intelligence that uses neural networks to achieve a task or goal [12]. Neural networks are loosely modeled after the human brain in the sense that a neural network consists of many interconnected nodes that communicate between each other. To this end, they are aptly named ‘neural networks.’ The implementation and architecture of neural networks can vary greatly, but the foundation is largely the same. Neural networks are comprised of several layers, with the first layer being the input layer and the last layer being the output layer [13]. In between the input layer and output layer are the hidden layers, where computation of the neural network is performed. Figure 1.2 shows the structure of a basic neural network.

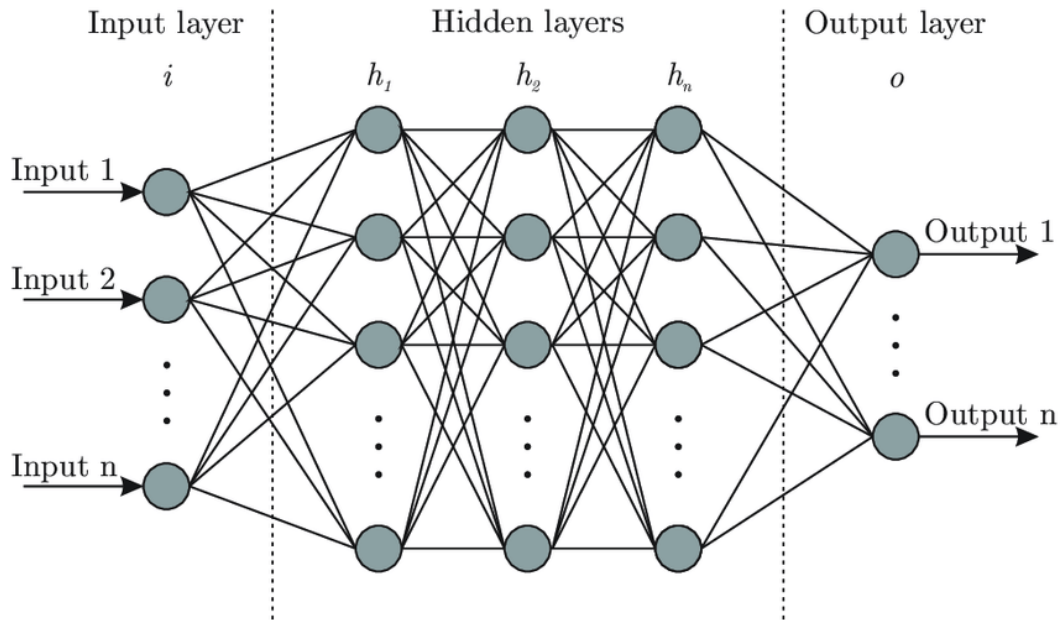


Figure 1.2. Structure of a basic neural network.<sup>1</sup>

<sup>1</sup> Image source: <https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed>

Depending on the application and domain, the amount of hidden layers of a network will vary. The number of layers in a neural network is referred to as its depth. Arguments exist pertaining to whether the input and output layers are included in a network's depth value. Generally, however, the depth value does not include the input layer because it is not considered an active layer of the network. Each hidden layer consists of many perceptrons, which are also referred to as nodes or neurons. These perceptrons are units that have one or more inputs and are used in conjunction with an activation function to create an output variable [14]. The number of perceptrons in a neural network is referred to as its size, and the number of perceptrons within a particular layer is referred to as the layer's width.

Perceptrons consist of several parts, to include the inputs, weights, bias, weighted sum, activation function, and output [15]. The inputs may refer to either the outputs processed by the input layer or the outputs of a hidden layer. The weights of a perceptron are mathematical values used to manipulate the inputs of the perceptron. A perceptron's weights are unique for each input. The bias of the perceptron is a numerical value that has its own weight and is used to directly manipulate the values of the perceptron. Bias is used to determine whether the perceptron will produce an output and can also be used to determine the degree to which a perceptron produces an output [16]. The perceptron's inputs and bias are manipulated by its set of weights to produce its weighted sum. The weighted sum is subsequently used by the activation function to determine whether the perceptron produces an output or not. An activation function can also determine the strength of the output signal. There exist many types of activation functions, each of which falling into the category of piecewise linear activation functions or locally quadratic activation functions [17]. Piecewise linear activation functions are comprised of a finite number of linear segments across an equal number of intervals and include functions such as Rectified Linear Unit (ReLU) and Exponential Linear Unit activation functions. Locally quadratic activation functions are defined as being smooth, non-linear, and possessive of a nonzero second derivative. Examples of locally quadratic activation functions include the sigmoid, tangent hyperbolic, and Swish activation functions [18]. Because each activation function is different and can greatly affect the performance of a network, there is not a one-size-fits-all solution for determining which activation function a network should use. Figure 1.3 shows the structure of a perceptron, with weighted sum

being comprised of the inputs, weights, and bias, and the output defined by result of the weighted sum being used in the activation function.

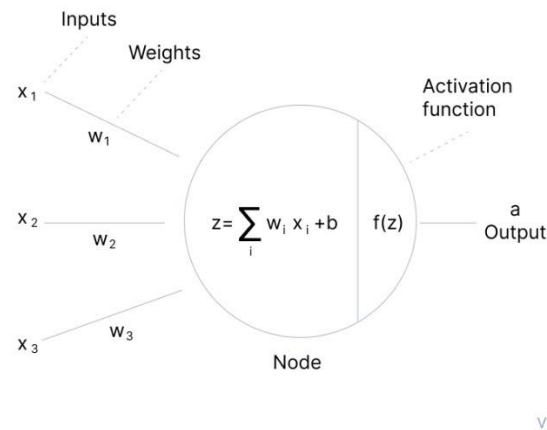


Figure 1.3. High-level architecture of a perceptron.<sup>2</sup>

Convolutional networks are special types of neural networks which are designed to work specifically with two-dimensional data, such as images and media [19]. An example illustration of a CNN can be seen in Figure 1.4. In this figure, an input image is processed by a convolutional layer. Convolutional layers are layers that form the building blocks of CNNs. Integral to the formation of convolutional layers are convolutions, which are operations that involve multiplying a set of weights by the input. In this case, the input is a digital image. When an image or other two-dimensional input is processed by a convolutional layer, important information about the input is discovered, such as edges and corners [20]. The distinction between important and unimportant information is decided by the activation function. Activation functions work by returning a value from a set of inputs. In the case of the ReLU activation function, the function returns  $\max(0, x)$  for any negative or positive input  $x$ . In the network structure illustrated in Figure 1.4, the ReLU activation function is used. That information is provided to the pooling layer, where the spatial size of the convolution is reduced while summarizing and maintaining the important details about the convolution. This, in turn, reduces the computational power needed to process the data because there is less information to process. This is achieved by one or more of several types of pooling functions, such as average and max pooling.

<sup>2</sup> Image source: <https://www.v7labs.com/blog/neural-networks-activation-functions>

Weights are coefficients that are applied to each input of a perceptron that can greatly affect its output. When a network is first run, the weights are randomized and tuned as the network continues to run. As the network continues to run, the weights are updated with every iteration and adjust per the loss function, also referred to as the cost function. The loss function is a calculation that is used to determine the losses between a predicted value and the actual value [21].

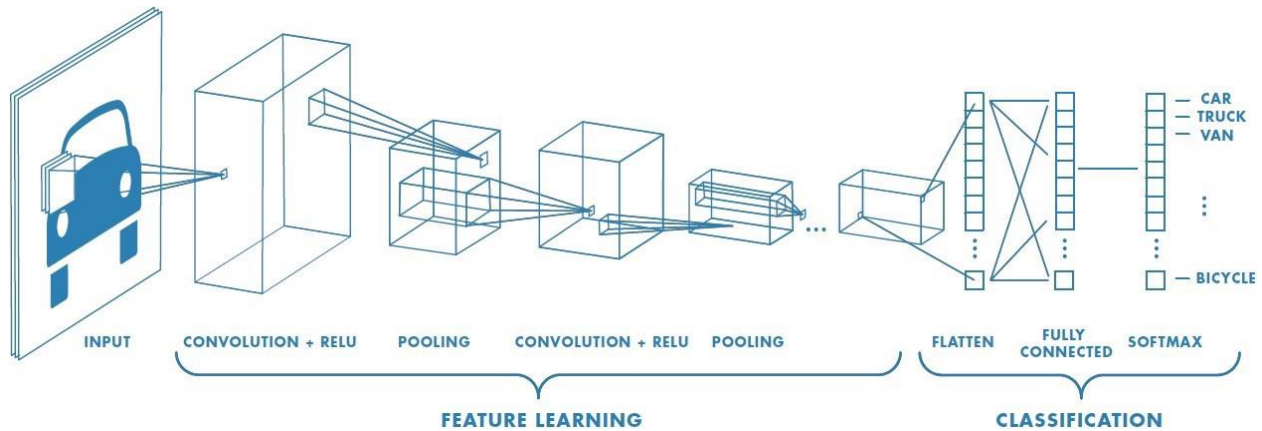


Figure 1.4. Illustration of a CNN with several convolutional layers to extract the features of an input image.<sup>3</sup>

One method of accelerating the training process involves feeding the input data of a network in batches, as opposed to individual samples. The batch size of a network refers to the number of samples passed into the neural network at a time. For this research, the batch size refers to the number of pairs of cover and hidden images processed by the network for each step of the training process.

Gradient descent is a method used to optimize the learning process of a neural network. In the context of neural networks, a gradient is a calculation that dictates how to adjust the network parameters such that the deviation of the output and error are reduced as much as possible [22]. When the network is initialized and the weights of the perceptrons are randomized, one of several types of gradient descent methods are used to minimize the loss function. Gradient descent works by picking a random spot on the loss function and computing its derivative. Once obtained, the tangent line at the random spot is calculated and can be used to determine the slope at the starting point. The slope is used in conjunction with the learning rate and the cost function to determine

<sup>3</sup> Image source: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>



how the weights and bias of the perceptrons are updated. The goal of gradient descent is to minimize the cost function; since the slope of the tangent line directly correlates with the error rate, the network will calculate the gradient descent and update the perceptrons after every iteration to make the slope of the line as close to zero as possible [23]. The learning rate describes the number of steps taken to reach the minimum of the cost function. Figure 1.5 shows the result of two different learning rates.

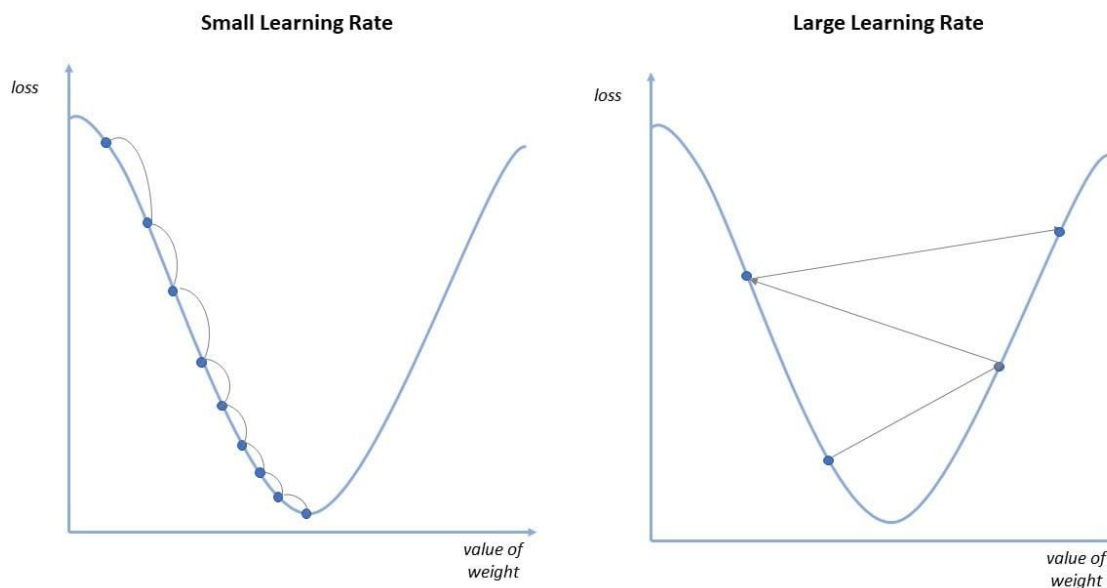


Figure 1.5. Effects of a small and large learning rate on the gradient descent method.<sup>4</sup>

There are three main types of gradient descent methods: batch, stochastic, and stochastic mini-batch gradient descent. Batch gradient descent calculates the error sum for all the data in an input batch and subsequently updates the perceptrons. Although batch gradient descent provides superior computational efficiency when compared to other methods, it is not ideal when used with large datasets, since all data needs to be stored in memory while processing the gradient. Batch gradient descent is also prone to converging on a local minimum of the loss function, instead of the global minimum. This is because the computed loss is based on the sum of losses across the entire batch of data. Figure 1.6 shows two types of loss functions that have a local minimum and global minimum.

<sup>4</sup> Image source: <https://www.ibm.com/cloud/learn/gradient-descent>

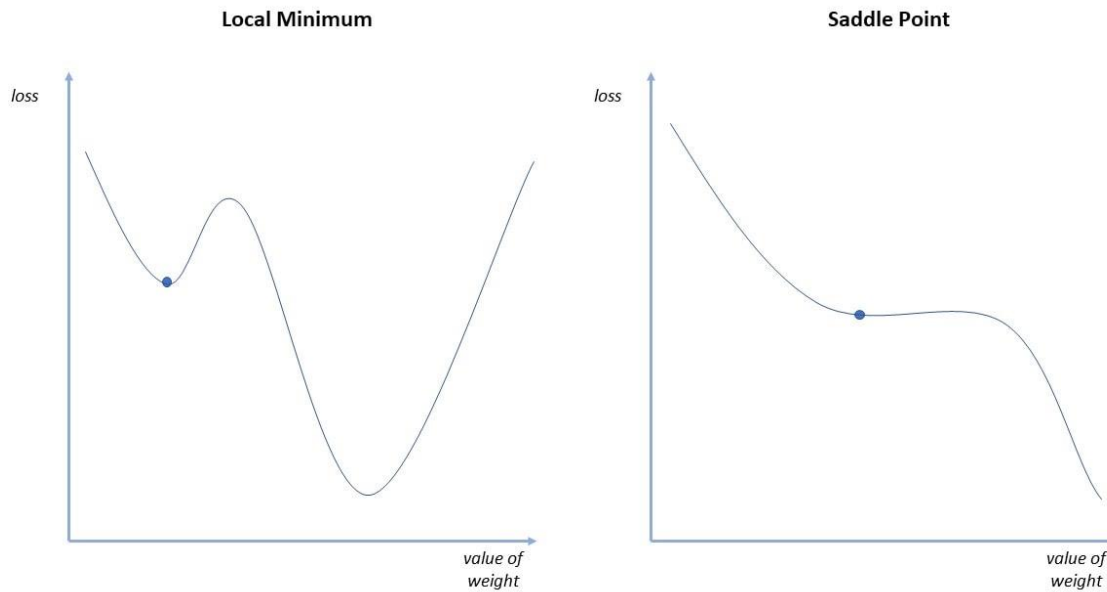


Figure 1.6. The convergence of the gradient descent method on a local minimum, where a global minimum exists that provides a more optimal solution.<sup>5</sup>

Stochastic gradient descent differs by calculating the gradient for each sample in the input batch. This method is advantageous with regards to storage, since one training sample is being stored in memory at a time. This, however, comes at the cost of computational speed. Since the gradient is computed for each training sample, this difference between data samples provides more noise (e.g., randomness) and is helpful in determining the global minimum.

Stochastic mini-batch gradient descent combines characteristics of batch and stochastic gradient descent methods by splitting the input batch into smaller batches and calculating the gradient for each batch. This, in turn, provides the efficiency of the batch gradient descent and the speed of the stochastic gradient descent. The size of the smaller batches can vary between implementations and may need to be adjusted based on overall batch size and application of the network [24].

Two popular extensions to the gradient descent method include AdaGrad and RMSProp. AdaGrad maintains a learning rate for each dimension of the network which allows the method to work best for sparse gradients [25]. AdaGrad works by altering the learning rate for each

---

<sup>5</sup> Image source: <https://www.ibm.com/cloud/learn/gradient-descent>

dimension of the network, since some perceptrons may obtain better results by having a different learning rate from others. This is done by obtaining the sum of the squared gradients for each perceptron, given a batch of data [26]. The sum is then used to determine the learning rate for the perceptron by dividing the initial learning rate by the sum of squared gradients. It is possible for a divide-by-zero operation to occur here, therefore a very small value can be added to the denominator to eliminate this possibility. Once the perceptron's learning rate has been updated, the weights can be updated by using the new learning rate. The process of updating the perceptron's learning rate continues with each mini-batch.

RMSProp is an extension to RProp, another version of the gradient descent method. Resilient Propagation (RProp) extends gradient descent by maintaining the signs of the last two computed gradients for a particular weight and updating it based on its update value [27]. The update value is initially set and is continuously updated based on the recently obtained gradients. If the most recently obtained gradient changes its sign (e.g., negative to positive or positive to negative), it can be surmised that the last update was too large, and the update value is decreased by a scalar. Conversely, if the gradient did not change signs, the update value is slightly increased to accelerate the training process. Once the update value is modified, the two most recently obtained gradients are used to determine how it updates the weight. If both gradients' signs are negative, the weight is added by its update value and if both signs are positive, the weight is decreased by its update value. If the signs are different, however, it can be determined that the step was too large and is reverted. RMSProp extends the gradient descent method by maintaining a moving average of the squared gradient obtained for each weight [28]. This moving average is based on the recent magnitudes of the squared gradients for the associated weight. The obtained gradients are then divided by this moving average. The motivation here is that the weights should be updated based on the size and frequency of the gradients.

The gradient descent method and other optimization algorithms are used to train the network by updating the weights and bias of the perceptrons after every step is taken in the training process. As the network continues to update its weights and biases and reach the minimum of the loss function, diminishing results will be returned from the training process. In fact, several issues can start to arise by not ending the training process. Overfitting is one of the major problems associated with training a network that describes the phenomenon where the network is overtrained on the training dataset. A network that is over-fitted to the training dataset performs well on the training

dataset but performs poorly on the testing dataset [29]. Overfitting can be observed when the network starts to learn and train on the noise of the training dataset, which in turn yields new, suboptimal training updates [30].

Several methods of mitigating overfitting exist, such as early stopping, network reduction, regularization, and dataset expansion. As the name suggests, early stopping involves stopping the training process earlier than anticipated. It is difficult to determine when to stop the training process early without any indicator that the network is being overfitted. Three elements of early stopping exist to rectify this: performance monitoring, stop triggers, and model choice. Performance monitoring involves analyzing the performance of the network training process as it occurs in real time. Certain metrics can be good indicators of when to stop training, such as incremental losses obtained when using a validation dataset or evaluating the network losses after every training epoch. Stop triggers can be implemented in the network code to automatically terminate the training process at the end of an epoch when the current epoch's losses were greater than those of the last epoch. Model choice involves saving the weights and biases of the network's perceptrons to a file after a certain amount of epochs have completed. By having frequent backups of the network at different points in the training process, the network whose weights and biases produce the smallest error can be selected for integration.

Network reduction can also be used to prevent overfitting by reducing the network complexity through pruning. Pruning is used to reduce the size and complexity of a network while maintaining or losing as little accuracy as possible [31]. Pruning involves many different methodologies, such as assigning scores to network parameters and removing the parameters that contribute the least towards the output. Pruning can be done before the training epoch has completed (pre-pruning) or after the training epoch has completed (post-pruning) [32]. Pre-pruning methods use certain criteria to determine when the network should discontinue adding conditions to a rule or adding rules to the description of the model. Post-pruning methods split the training dataset into a larger dataset and a smaller dataset, referred to as the growing set and pruning set, respectively [33]. With post-pruning, the first pruned epoch runs on the growing set and deletes rules and conditions from the network until an increase in error is detected when used with the pruning set. Pre-pruning methods are more computationally efficient than post-pruning methods, but post-pruning methods are much more accurate than pre-pruning methods.

Regularization describes the process of altering the network during the training process to reduce the generalization errors, which describe the network's ability to accurately respond to unencountered data [34]. Regularization is used to reduce the overfitting problem by reducing the size of the perceptrons' weights. Large network weights signal that the network is overly trained on the training dataset and will be sensitive to noise or new data [35]. By having smaller weights, the network is less likely to be overfitted to the training dataset. Popular regularization methods include L1 regularization, L2 regularization, dropout, additive noise, and weight constraint. L1 regularization involves adding a penalty term to the loss function equal to the sum of absolute values of all the perceptrons' weights. By adding this penalty term, the perceptron is more severely penalized for using larger weights than smaller weights and is therefore encouraged to use smaller weights. L2 regularization is identical to L1 regularization with the exception of the penalty term representing the sum of squared values of all the perceptrons' weights. Dropout is implemented on a per-layer basis in a network and is used to randomly drop perceptrons from a layer based on a parameterized probability coefficient. By doing this, the network becomes more generalized and less tailored to the training dataset [36]. Adding noise can also reduce the generalization error of the network by expanding the size of the training set, which is especially beneficial for smaller datasets [37]. By adding a small amount of noise to each training sample, it can be guaranteed that no samples are completely identical. Finally, weight constraints can be implemented to mitigate overfitting. Weight constraints are alternatives to weight penalties in the sense that the penalty term is the size of the weights. When the size of a weight exceeds a certain value, the weight is subsequently scaled down [38]. This difference between weight constraints and weight regularization methods is that weight constraints force the weights to be smaller, instead of 'recommending' the weights to be smaller.

## 2. LITERATURE REVIEW

This section is intended to provide a literature review of some of the numerous digital image steganography methods. There are three main categories that image steganography techniques fall under, which are spatial domain techniques, transform domain techniques, and neural networks [39]. It should be noted, however, that image steganography techniques may fall into several categories by using a combination of different techniques.

### 2.1 Spatial Domain Techniques

Many of the common spatial domain techniques involve altering the least significant bits (LSB) of the pixels of an image to embed information into a cover image. Often, these methods rely on altering between one to four of the LSB of an image's pixels, since changes within this range are usually imperceptible to the human eye [40]. Figure 2.1 demonstrates the basic architecture of spatial domain steganography techniques. Here, the sender is taking the optional step of encrypting the secret text 'Hello' with an encryption algorithm which outputs the cipher text, 'C4Za9'. Each character of the cipher text is split into individual American Standard Code for Information Interchange (ASCII) characters. Each ASCII character has its own decimal and binary value, with 'C' being represented as 67 in decimal notation and 01000011 in binary notation. With the entire binary sequence now known, these values can be embedded into the cover image by modifying the LSB of the individual pixels. Since color images are split into three, 8-bit wide color channels, this gives plenty of space to embed information while remaining undetectable to the human eye. Once the secret message has been embedded into the cover image, the steganographic image is then sent to the receiver. The receiver, in turn, will be able to extract the hidden message by performing the steps taken to embed the message in the opposite order. If the sender decides to encrypt the hidden message before embedding, the receiver will have to know the decryption key to perform the decryption process. Otherwise, the receiver will only receive an encrypted message and be unable to decipher the correct message.

Note that the encryption step is optional in the embedding process. However, by encrypting the secret message, the security of the steganographic system is increased. In addition, the detail of the embedding process greatly varies between different implementations. As a result, the results

are not guaranteed to be the same. In other words, if someone sends a steganographic image encoded with one algorithm, it is unlikely that the receiver will be able to decipher it without using the same algorithm.

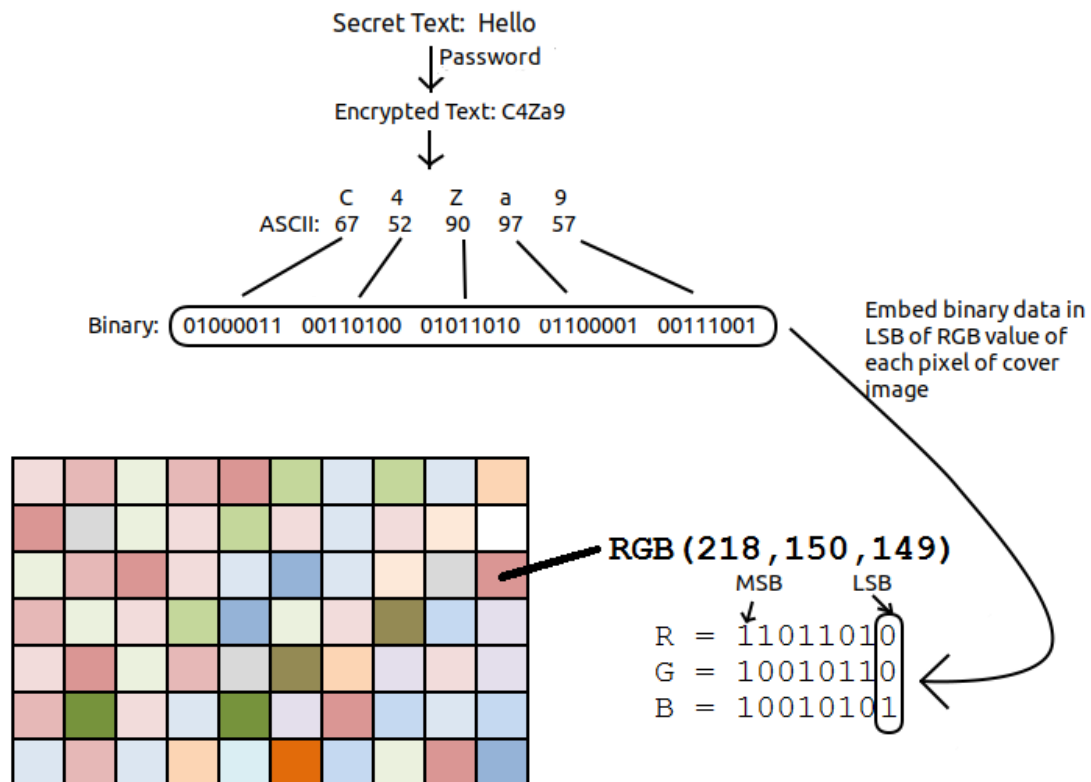


Figure 2.1. Illustration of encrypting and embedding text into a cover image's pixels.<sup>6</sup>

There are many types of spatial domain techniques that rely on altering the LSB of the pixels of an image, such as the LSB++ method introduced in [41]. The LSB++ method works by encrypting the payload image using a key, creating the encrypted message. A second key is used to lock the frequency difference of two adjacent image bins with a gray value equal to double the unit, creating the cover image. The embedding process is done by combining the cover image and encrypted message with a third key. LSB++ improves upon other LSB-based techniques by maintaining sensitive pixels in the cover image so that the level of distortion is reduced throughout the embedding process. This provides a level of security against histogram analysis.

<sup>6</sup> Image source: <https://medium.com/analytics-vidhya/shh-your-secret-is-safe-a-simple-guide-to-steganography-in-python-89116582277e>

Another example of a spatial domain steganography technique uses a codeword formed using secret data and its CRC-32 checksum [40]. The codeword is then compressed with Gzip and encrypted with AES, where the codeword is subsequently added to the encrypted header and embedded into the cover image. The embedded process is done by using the Fisher-Yates Shuffle algorithm to select the next pixel for embedding. To increase the security of the technique, the LSB of the three color channels (red, green, blue) of the embedded pixel are used to hide one byte. To extract the embedded image, the embedding steps are reversed, using the AES symmetric key to decrypt the payload.

In [42], the authors proposed a spatial domain technique using LSB substitution, pixel value differencing (PVD), and exploiting modification directions (EMD). The image is scanned in and partitioned into 2x2 non-overlapping pixel blocks. The average PVD is calculated for each pixel block. If the PVD is greater than fifteen, it is an edge; otherwise, it is considered a smooth region. Next, the edges are processed for embedding using LSB substitution and PVD, while the smooth regions are processed for embedding using LSB substitution and EMD. The PVD and EMD are used in the algorithm to determine the hiding capacity for each pixel block. The authors proposed two techniques where one uses pixel blocks of sizes 2x2 and the other uses pixel blocks of sizes 3x3. They discovered that using 2x2 pixel blocks produces better PSNR values, while 3x3 pixel blocks produces greater hiding capacity.

## **2.2 Transform Domain Techniques**

Transform domain techniques are steganography methods which use mathematical transformation function to embed a hidden image within a cover image. A transform is a function that is used to alter the structure of the data into waves, such as sine and cosine waves. One example of a transform is the Fourier transform. The Fourier transform is a transform function that is used to decompose a function into waves of different amplitudes and frequencies [43]. Another example of a transform function is the Discrete Wavelet Transform, which decomposes a function into a series of wavelets. Wavelets are wave-like oscillations which, unlike waves, are localized in time. Wavelets have two components, scale and location. Scale, otherwise known as dilation, describes how stretched or compressed a wavelet is. Location describes where the wavelet is positioned in time or space [44].



The scale and location of a wavelet are determined by two parameters,  $a$  and  $b$ .  $a$  is used to adjust the scale of the wavelet and  $b$  is used to adjust the location of the wavelet. The Discrete Wavelet Transform uses a finite number of wavelets to determine how much of a wavelet can be found within a given signal. In this case, the signals are images. The Haar two-dimensional DWT decomposes an image into four coefficients, which are approximation, horizontal, vertical, and diagonal coefficients. The approximation coefficients are composed of the low-low (LL) sub-band wavelet coefficients and correspond to a down-sampled version of the original image [45]. The horizontal coefficients are composed of the low-high (LH) sub-band wavelet coefficients. The vertical coefficients are composed of the high-low (HL) sub-band wavelet coefficients. Lastly, the diagonal coefficients are composed of the high-high (HH) sub-band wavelet coefficients. Figure 2.2 shows the output of a two-level DWT on an image. In the upper-left corner, the original image is scaled to half its original size and corresponds to the LL sub-band wavelet coefficients. The decomposed image to the right of the grayscale image shows the vertical details of the original image. The decomposed images beneath and diagonal to the grayscale image show the horizontal and diagonal details of the original image, respectively.

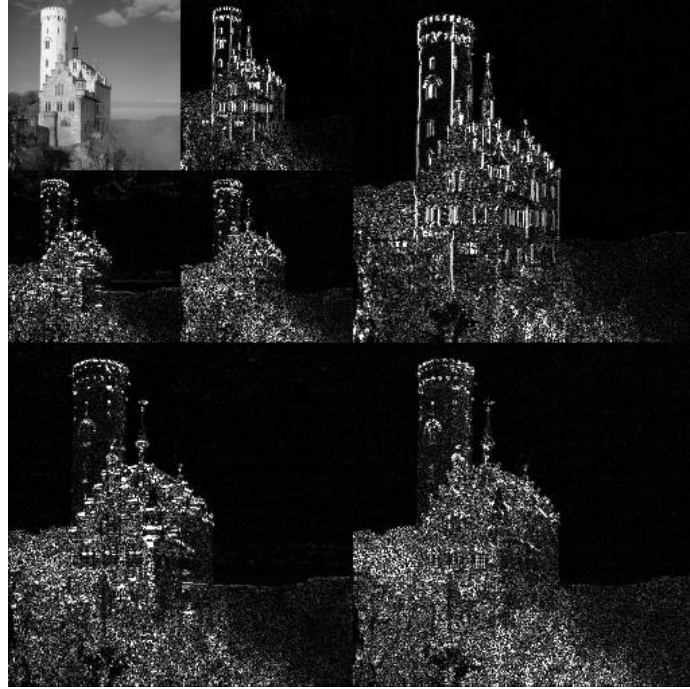
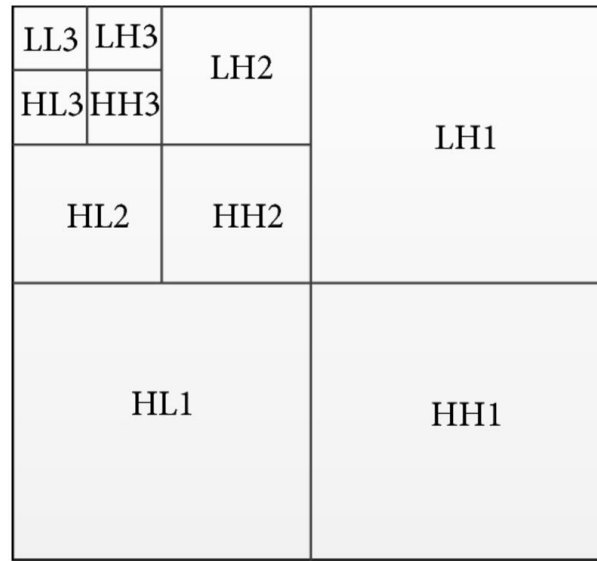


Figure 2.2. Example of a two-level discrete wavelet transform.<sup>7</sup>

One novel image steganography methodology used the Finite Ridgelet Transform (FRT), Discrete Wavelet Transform (DWT), and Arnold Scrambling in order to provide high imperceptibility and hiding capacity in comparison to other algorithms [46]. In this technique, FRT is used in tandem with DWT due to an increased embedding capacity the two algorithms provide when combined. The red, green, and blue color channels of the cover image are divided and their ridgelet coefficients are computed. Then, the low-low (LL) sub-band wavelet coefficients are gathered for each color channel. These LL subband wavelet coefficients are subsequently modified according to the bits of the scrambled secret color image and an insertion factor. Next, an inverse single-level DWT transform is performed on the modified LL sub-band wavelet coefficients with the original low-high (LH), high-low (HL), and high-high (HH) sub-band wavelet coefficients to get the modified ridgelet coefficients of each color channel. Figure 2.3 shows the decomposition of sub-band wavelet coefficients at three levels. Inverse FRT is performed on the modified ridgelet coefficients to get the altered color channels of the cover image. Last, the steganographic image is constructed using the modified color channels.

---

<sup>7</sup> Image source: [https://upload.wikimedia.org/wikipedia/commons/e/e0/Jpeg2000\\_2-level\\_wavelet\\_transform-lichtenstein.png](https://upload.wikimedia.org/wikipedia/commons/e/e0/Jpeg2000_2-level_wavelet_transform-lichtenstein.png)



1, 2, 3: decomposition levels  
H: high frequency bands  
L: low frequency bands

Figure 2.3. Labeled wavelet decomposition performed at three levels.<sup>8</sup>

Another technique relied on DWT in conjunction with all phase discrete cosine biorthogonal transform (APDCBT) and singular value decomposition (SVD). APDCBT is a transform that improves upon the discrete cosine transform (DCT) by offering greater performance in high-frequency and low-frequency aggregation. SVD is a transform that is used in numerical analysis and principal component analysis. First, the Haar DWT is applied to the cover image in order to obtain the high-frequency sub-bands (HH, HL, LH) and the low-frequency sub-band (LL). HL and LH are both used for inserting two identical watermarks. Here, the block-based APDCBT is used to obtain each sub-block by dividing the image into 8x8 blocks. Next, the direct current (DC) coefficients of each sub-block are used to create a new coefficient matrix. SVD is then applied to this coefficient matrix to obtain the singular value matrix. The first watermark is inserted into the coefficient matrix, where the inverse SVD is then applied to obtain a modified coefficient matrix. Inverse APDCBT is then applied to this matrix to obtain the HL sub-band. This same process is

---

<sup>8</sup> Image source: [66]

followed with the second watermark in order to obtain the LH sub-band. Finally, the inverse DWT is used to obtain the steganographic image.

## 2.3 Neural Network Techniques

More recently, there have been many research efforts to utilize deep learning in order to achieve image steganography. One such research that has implemented neural networks to perform image steganography was performed by [47]. Their technique was implemented using three CNNs, each of which consisting of five convolution layers. The first network is the preparatory network, whose primary function is to prepare the secret images to be embedded. This is done by scaling the hidden image to the same size as the cover image and transform the hidden image into useful features for embedding, such as edges and corners. The second network, known as the hiding network, takes the output of the preparatory network and a cover image as input and creates the steganographic image. This network has five convolution layers that have fifty layers each of 3x3, 4x4, and 5x5 patches. The final network, referred to as the reveal network, is the ‘decoder’ of the steganographic image. Its purpose is to remove the cover image to reveal the embedded image. The reveal network also calculates the error between the cover and steganographic images, which is subsequently used to adjust the weights of the preparatory and hiding networks. The error between the embedded and recovered images, however, is used to adjust the weight of all three networks. This technique has demonstrated its security against several kinds of attacks, even instances where an attacker has access to the original cover and steganographic images.

Another example of deep neural networks in use to perform digital image steganography can be seen in [10], where the authors combined the use of the Discrete Cosine Transform (DCT), Image Elliptic Curve Cryptography (ECC), and the SegNet CNN in order to achieve a hiding capacity of approximately 23 bits per pixel (bpp). ECC is an asymmetric encryption algorithm used in various security applications, such as the Diffie-Hellman key exchange algorithm. This technique also uses a network structure similar to [47], where a pre-processing network, encoding network, and decoding network are used together to achieve steganography. The pre-processing network normalizes the secret image and extracts important features. This is achieved using DCT and ECC to obtain a secret image and encrypted image. The encoding network encodes the hidden and cover images such that they are the same size and embeds the hidden image into the cover image. The decoding network extracts the hidden image from the cover image. The decoding

network will output the steganographic image, which will need to be further decrypted by using inverse DCT and ECC.

Deep convolutional generative adversarial networks have also been used in the field of steganography, as seen in [48]. Generative adversarial networks are a type of neural network structure that contain two primary networks, a generative model and a discriminatory model [49]. The generative model is used to create steganographic images that look like samples from the dataset, while the discriminatory model is used to detect if the image has a hidden message. The benefit of this structure is that both networks use the results to improve the network's hiding capabilities; when the generative model's output is detected by the discriminatory model, its weights are adjusted to embed the data more effectively. Inversely, if the discriminatory model raises a false positive or negative, it uses the data from the other network to learn to discriminate more effectively. The method implemented in this research is to use noise, a hidden image, and a secret key to create the steganographic image, which is then passed to the receiving party. The receiving party must use the secret key to decrypt the message, while the discriminatory model also receives the steganographic message for training purposes. The discriminatory model's loss is calculated as the average cross-entropy, which is then used to update both models.

## **2.4 Summary**

Table 1 summarizes a variety of different steganographic techniques. The domain and methodology for each technique is summarized, and the average PSNR, NCC, SSIM, and MSE values for each technique are displayed. The blindness of each technique is also noted. A steganography method is considered blind if it has access to the steganographic image when extracting the hidden image from it. Conversely, if the technique does not know or assume anything about the given image when attempting the extraction process, it is not considered blind.

The preferred file formats for each technique are also described, as this may have a significant impact on how the images are processed by the network, due to differences in compression and how the data is stored. For example, image formats such as GIF and TIFF are better suited for image steganography due to their lossless nature, while JPEG and PNG formats may not be as suitable for this process because of their use of data compression techniques [50]. Finally, because some methods focus on certain characteristics of steganography such as imperceptibility or hiding capacity, certain values may be omitted from the table.

Table 2.1. Review of various image steganography methods.

Research	Domain	Method	Peak Signal to Noise Ratio (PSNR)	Normalized Cross-Correlation (NCC)	Structural Similarity Index Matrix (SSIM)	Mean Square Error (MSE)	Payload Size	Blind?	Preferred file format(s)	Robustness	Remarks
[40]	Spatial	Secret data is compressed using Gzip, encrypted using AES, and embedded based on Fisher-Yates Shuffle algorithm	40.0834 – 63.8619	1	Not provided	0.02 – 6.46	8 bpp	Yes	BMP, PNG, TIFF	Has the ability to detect alterations to the steganographic image using CRC-32 checksum.	Uses several algorithms in order to achieve a high PSNR, but it quickly drops as payload embedded data size increases. Highly resistant to histogram and Chi-square attacks
[10]	Frequency (Discrete Cosine Transform)	Secret image is decomposed using DCT, embedded into cover image using deep neural network, decrypted using the ECC key.	Average 40.5726	Not provided	0.9602	Not provided	~23.2 69 bpp	No	BMP, PNG	DCT is used to provide robustness by changing the structure of the secret image.	Uses CNNs, discrete cosine transform, and elliptic curve cryptography to transform and hide an image into a cover.
[51]	Spatial	Secret data is hidden using modified LSB substitution method and uses edge preserving modules to ensure minimal distortion.	Average 46	Not provided	Not provided	Avg 1.4	~1 – 3 bpp, depending on # of LSB	Yes	JPG	Robustness is not a component of this proposed method.	More LSB are used for edges than smooth areas to provide imperceptibility.
[46]	Frequency (Finite Ridgelet Transform, Discrete Wavelet Transform)	RGB channels images are scrambled using Arnold scrambling. FRT and DWT are used to obtain the ridgelet coefficients and wavelet subbands, respectively. Inverse FRT and DWT is applied to obtain steganographic image.	Average 58.9967	1	Not provided	Not provided	8 bpp	No	Not provided	Very susceptible to attacks such as JPEG compression, cropping, noise, and histogram analysis.	Uses FRT with DWT to greatly increase the payload capacity. Arnold scrambling is used to provide security to secret image before insertion.

Table 2.1 continued.

[52]	Spatial	Eight-directional Pixel Value Differencing (PVD) is used to embed the pixels of a secret image into a cover image. The eight directions correspond to each of the pixels surrounding a particular pixel in the cover image.	Variant 1: Average 39.55 Variant 2: Average 37.22	Not provided	Avg 0.9985	Not provided	Variant 1: 3 Variant 2: Avg 3.31	No	Not provided	Robustness is achieved through its resistance to RS and Pixel Difference Histogram (PDH) analysis.	Two variants of the method are provided. Variant 1 uses 3-bit substitution and has higher PSNR, while Variant 2 uses 4-bit substitution and has higher hiding capacity.
[53]	Spatial	Secret image is manipulated using Arnold's Cat Map, embedded into the edges of the cover image determined by Canny edge detection, and embedded using Least Significant Bit Matching Revisited [53].	38.6775 – 76.4144	Not provided	Not provided	Not provided	Not provided	Yes	TIFF	LSB Matching Revisited provides a level of robustness against asymmetric steganographic attacks.	This method requires secret images to be 10% of the size of the cover image for high PSNR and quality.

### 3. METHODOLOGY

#### 3.1 The Original Method

The steganographic channel implemented in this research was built on an unofficial implementation of the method proposed in [47]. The implementation was provided by [54] and is written in Python. The overall network structure is comprised of three individual networks, a preparatory, hiding, and reveal network. The preparatory network is used to take a secret image and process it for embedding, which occurs in the hiding network. The hiding network takes the processed secret image and a cover image as input and produces the steganographic image. The reveal network takes a steganographic image as input and reproduces the secret image. Each network is connected and is trained as a whole, instead of training each network separately. Figure 3.1 shows the structure of the preparatory, hiding, and reveal networks.

The primary method in which the network learns is through the use of a loss function and the Adam optimizer. The loss function for this network is provided by the following equation, where  $c$  represents the cover image,  $c'$  represents the steganographic image,  $s$  represents the secret image, and  $s'$  represents the recovered image:

$$L(c, c', s, s') = \|c + c'\| + \beta \|s - s'\| \quad (3.1)$$

The loss function is returned in three parts, the total, hiding, and extracted loss coefficients. The  $\|c + c'\|$  part of the loss function represents the error between the cover and steganographic image and is applied to preparatory and hiding network. The  $\beta \|s - s'\|$  part of the loss function represents the error between the hidden and extracted image and is applied to all three networks. It is worth mentioning that these were the original author's intentions; the implementation of that research only utilized the overall error coefficient.



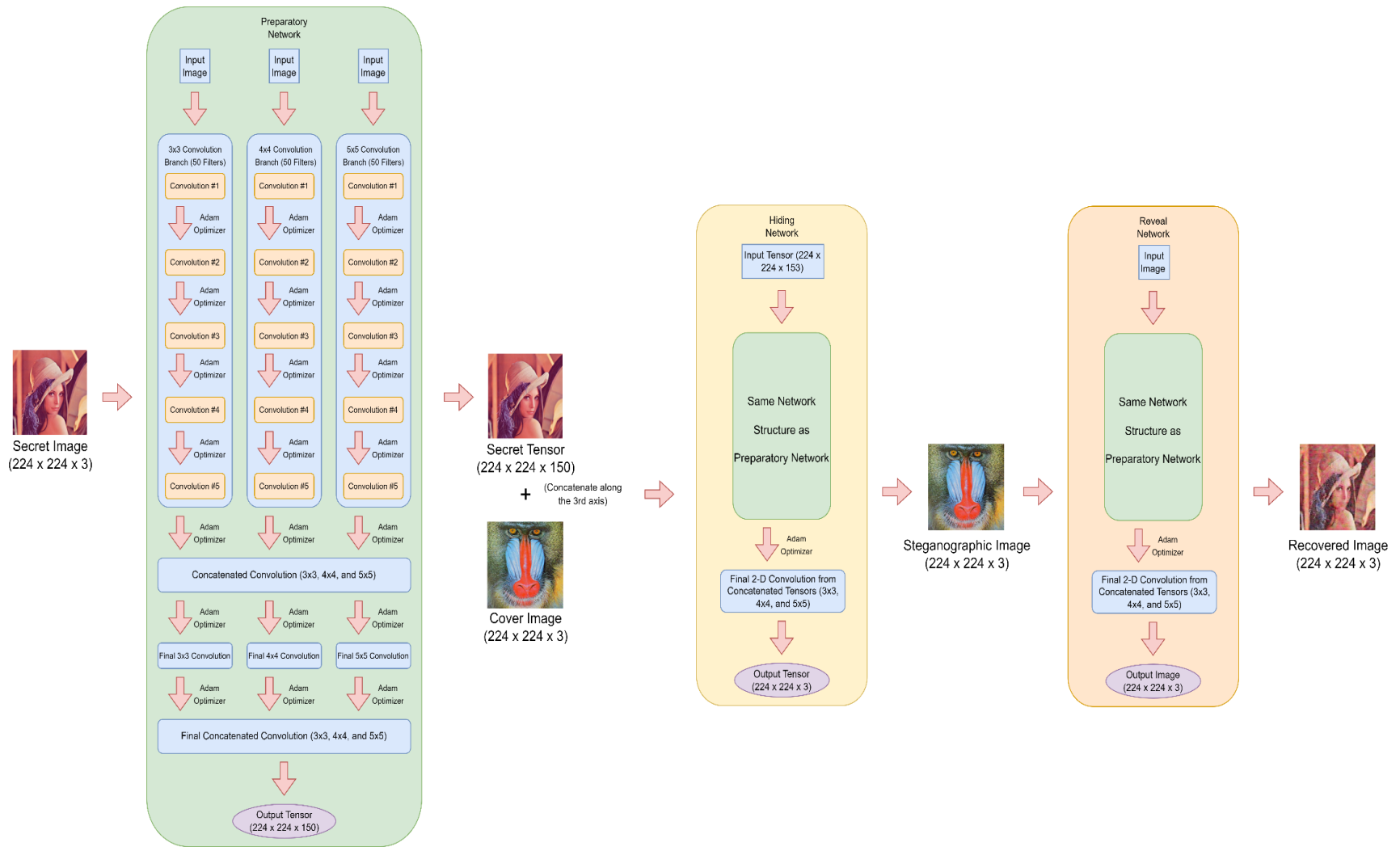


Figure 3.1. Structure of the overall network. The structure of the preparatory network is identical between the hiding and reveal networks, with the exception of the hiding and reveal networks taking an additional step of performing a two-dimension convolution on the concatenated tensors.

Adam is a stochastic gradient descent method that is used to train the three networks. The Adam optimizer works well with networks that have large input requirements and has been shown to work well with CNNs [55]. Adam works by combining the features of two extensions of the stochastic gradient descent method, Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) [56]. It borrows from AdaGrad the idea that each perceptron should have its own continuously updated learning rate, and from RMSProp the moving averages and the idea that the learning rates should be updated according to the average of the recent magnitudes obtained from the gradients of the perceptrons' weights. Adam differs from the moving average outlined in RMSProp by calculating an exponential moving average from the gradients and the square of the gradients, where hyperparameters  $\beta_1$  and  $\beta_2$  are used to adjust the decay rates of the exponential moving average.  $\beta_1$  is used to adjust the estimates of the gradients' first moments, otherwise known as mean, and  $\beta_2$  is used to adjust the estimates of the gradients' second moments, otherwise known as variance.

The following summarizes the main elements of the structure of the original network:

- Create a training graph, consisting of a preparatory network with 3x3, 4x4, and 5x5 convolution branches, hiding network (same structure), noise layer, and reveal network (same structure). The purpose of the training graph is to process a randomly selected cover and hidden image, produce a steganographic image, extract the hidden image, and update the network weights using the Adam optimizer.
  - o For each epoch, and for each step, randomly select a cover and hidden image from the datasets and provide as input to train the network.
    - On the hundredth step, test the network with a randomly selected cover and hidden image.
- Create a test graph with a preparatory network, hiding network, and reveal network. The test graph is used to test the embedding and extracting processes of the network and obtain the combined SSIM and PSNR values. Since it is not part of the training process, the Adam optimizer is not used to update the weights.
- Create a deployment graph with a preparatory network, hiding network, and reveal network. The deployment graph takes an input cover and hidden image and is used to produce a steganographic and extracted image.

### 3.2 The Enhanced Network

In this research, the network structure is largely similar to the original implementation. That is, the final network consisted of a training, testing, and deployment graph. The training graph consisted of a preparatory network, hiding network, and reveal network. The preparatory network consisted of five two-dimensional convolution layers, with each convolution layer having fifty filters each of 3x3, 4x4, and 5x5 patches. Each patch has a kernel size of five and has a ReLU activation function.

In the pursuit of better imperceptibility, hiding capacity, and robustness, many modifications to the original network were made:

1. In order to eliminate the reliance of training on the ImageNet dataset, the normalization and denormalization methods were removed from the network. This means that the preparatory network no longer normalized and denormalized image batches based on the mean and standard deviation from the ImageNet dataset. After examining the results of input and output images from the network, it was apparent that these values were not creating accurate output results and thus were removed. These images had a much lower contrast ratio and appeared to have a type of sepia filter applied.

2. The hyperparameters were also adjusted from the original network as well. The batch size was increased from 8 to 16 with the hopes that the increased size of the batch would produce more varied results in image details and, in turn, provide more variety in the training process. Both networks were trained for ten epochs. However, since the steps taken per epoch were directly correlated to the batch sizes, the original network's number of steps per epoch was 1,784. Since the batch size was doubled in the proposed network, the number of steps per epoch was reduced to 891. The learning rate was slightly increased from 0.0001 to 0.0002. The motivation of increasing the learning rate by a small amount was to speed up the training process, but not too much that the network converges on the wrong solution.  $\beta$ , also known as momentum, was decreased from 0.75 to 0.25. The momentum was decreased to reduce the speed of the training process, since it is possible that the large momentum value attributed to the artifacts in the resulting images.

3. In [47], the author adds a noise layer to the network such that the network is less likely to embed the information inside the LSB of the cover image's pixels and to increase the imperceptibility of the hiding results. The noise layer was removed to release this unnecessary

constraint. It is possible to further improve the hiding imperceptibility results by utilizing hiding methodologies within the LSB, in conjunction with other hiding methodologies the network decides to use.

4. The learning goal of the network was altered to focus on imperceptibility instead of minimizing image quality losses. That is, originally, the loss function consisted of two parts, one representing the MSE between the cover and steganographic images and the other representing the MSE between hidden and extracted images. The loss function was reworked to a “gain function”, where the losses were not calculated based on MSE coefficients but based on the combination of PSNR and SSIM coefficients. Because higher PSNR and SSIM coefficients positively correlate to greater image quality, the loss function was reworked to maximize this value. In the implementation, the loss function was changed to a “gain function” by minimizing the inverse of the PSNR and SSIM values since TensorFlow does not provide a built-in function for maximizing tensors. In fact, this modification had the most impact on the results - as demonstrated in the Results section using both the quantitative and visually observable results.

## 4. RESULTS

### 4.1 Model Training & Testing

While importing the network for our testing purposes, it was assumed that the ImageNet dataset was used to train the network and obtain these results, since the image pixels values were normalized using the mean and standard deviation values from ImageNet dataset. To provide the most equitable comparison between the original and proposed network designs and minimize differences in network training, similar datasets were obtained and used in the training and testing processes. To train and test the original network, the Linnaeus 5 [8] and Imagenette [57] datasets were used.

Linnaeus 5 is a dataset that contains 6,000 square images of five different types of objects. The images in the Linnaeus 5 dataset are saved in a JPG format, with each image having a height and width of 128 pixels. The Imagenette is derived from the original ImageNet dataset and consists of 9,468 images of ten different types of objects. The Imagenette dataset was used instead of the full ImageNet dataset due to limited storage space for training samples. The images provided by the Imagenette dataset were also saved in a JPG format but varied widely in height and width. Because the preparatory network takes care of resizing and formatting the images for use in the network, however, the varying image sizes from this dataset was not an issue.

The input cover and hidden images were provided to both the original and proposed network in the JPG format. When the images were supplied to the preparatory and hiding networks, they were first resized to 224 pixels square with the goal of hiding one full-size color image inside another color image of equal size.

### 4.2 Experiment Setup

For this experiment, the network proposed in [47] was built based on the Python implementation provided in [58]. The code was written and revised using the first major version of the Python TensorFlow library. Additional libraries were used in this experiment to include scikit-image for gathering the image metrics [59] and Matplotlib for displaying the images inside the Jupyter Notebook [60]. Google Colaboratory, also referred to as Colab, was used as the code editor and execution environment for training and testing the network. Colab works by hosting the

interpretation and execution of Jupyter notebooks on their cloud service, where the Jupyter notebooks are Python-based computational documents. To accelerate the training process, the Pro version of Colab was used, which enabled the use of an Nvidia Tesla T4 or Nvidia Tesla P100 GPU for the training process. Additionally, using Colab Pro enabled the use of a high-RAM runtime environment, which also reduced the time spent training the network.

To ensure that a wide variety of images were chosen for testing the networks, images that contain lots of color and image details were collected from various sources. Table 4.1 shows the ten image pairs that will be used throughout the results section to compare between implementations. Some images in the table, such as Baboon and Lena, are popular choices for testing neural networks that perform some type of image processing. Other images are samples from the datasets the network was trained on, such as the Karaoke and French Horn images. There are also a few original images, such as Graffiti and Lotus.

Table 4.1. List of labels and images used between experiments.

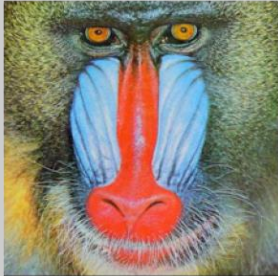









Label	Cover Image	Label	Hidden Image
Baboon		Graffiti	
Berries		Karaoke	
Chainsaws		Lena	
Church		Lotus	
Dog		Parachute	

Table 4.1 continued.

<b>Fish</b>		<b>Parrot</b>	
<b>French Horn</b>		<b>Pens</b>	
<b>Garbage Truck</b>		<b>Peppers</b>	
<b>Gas Pump</b>		<b>Stained Glass</b>	
<b>Golf Balls</b>		<b>Thistle</b>	



### 4.3 Results of the Original Network

This section focuses on the results of the original, unmodified code from which this work was derived. As seen in Table 4.2, the resulting steganographic image from the original network shows faint traces of the secret image. The SSIM and PSNR values between the cover and steganographic images from the original implementation were 0.93041 and 28.69137 dB, respectively. Values in these ranges tend to be on the lower end of the results from other image steganography methods. These faint traces are highlighted when compared side-by-side to the steganographic images generated by the original network trained on our datasets.

When trained on the Imagenette and Linnaeus 5 datasets, the original network clearly showed traces of the hidden image within the steganographic image. As shown in Table 4.3, the resulting steganographic and revealed images show the artifacts of the hiding process much more clearly than the author's sample output images, even with many more training steps taken. Important image details such as edges and corners are visually perceptible in the steganographic image produced by the original network. A close examination of the steganographic image reveals a discolored version of the cover image. However, despite the vast increase in training iterations, the images in our implementation of the original network did not use the normalization and denormalization processes. This may explain why the hiding artifacts may be more apparent.

With regards to metrics obtained from training the original network with the datasets obtained in this research, there are great improvements to be desired. The SSIM and PSNR values of the sample steganographic image generated by the original network trained on the ImageNet dataset were 0.93041 and 28.69137 dB, respectively. Expectedly, the SSIM and PSNR values of the same steganographic image generated by the original network trained on the Imagenette and Linnaeus 5 datasets were lower, resulting in 0.89914 and 27.69137 dB, respectively. Although the SSIM and PSNR values were similar between the two implementations, the steganographic process of the original network trained on the Imagenette and Linnaeus 5 datasets is not visually imperceptible and can easily be spotted by the naked eye. Moreover, the visually important details of the hidden image like edges and corners can be detected in the steganographic image without tools or adjustments, which does not provide a good method for image steganography. Comparing the results from the code provided by [58] with the original network trained on the proposed network's dataset highlights the details of the hidden image embedded within the steganographic image.

Table 4.2. Side-by-side comparison of steganographic and extracted images generated by the original network trained on the ImageNet dataset and the original network trained on the Imagenette and Linnaeus 5 dataset.









Network Version	Cover Image	Steganographic Image	Extracted Image	Hidden Image
Original Network (Original Dataset)				
Image Metrics	SSIM: 0.93041 PSNR: 28.69137		SSIM: 0.73294 MSE: 0.00749	
Original Network (Proposed Network's Dataset)				
Image Metrics	SSIM: 0.89914 PSNR: 27.61369		SSIM: 0.84422 MSE: 0.00296	

Table 4.3. The steganographic and extracted images generated by the original method.






Original Network		
Experiment	Steganographic Image	Recovered Image
<b>Cover: Baboon, Hidden: Graffiti</b>		
<b>Image Metrics</b>	SSIM: 0.93687 PSNR: 27.96642	SSIM: 0.87001 MSE: 0.00297
<b>Cover: Berries, Hidden: Karaoke</b>		
<b>Image Metrics</b>	SSIM: 0.93457 PSNR: 25.62975	SSIM: 0.85424 MSE: 0.0026
<b>Cover: Chainsaws, Hidden: Lena</b>		
<b>Image Metrics</b>	SSIM: 0.91134 PSNR: 29.35	SSIM: 0.91303 MSE: 0.00162
<b>Cover: Church, Hidden: Lotus</b>		
<b>Image Metrics</b>	SSIM: 0.93737 PSNR: 29.59204	SSIM: 0.86413 MSE: 0.00222

Table 4.3 continued.

<b>Cover: Dog, Hidden: Parachute</b>		
<b>Image Metrics</b>	SSIM: 0.90129 PSNR: 25.7312	SSIM: 0.93006 MSE: 0.00138
<b>Cover: Fish, Hidden: Parrot</b>		
<b>Image Metrics</b>	SSIM: 0.90204 PSNR: 25.77007	SSIM: 0.92166 MSE: 0.0013
<b>Cover: French Horn, Hidden: Pens</b>		
<b>Image Metrics</b>	SSIM: 0.87437 PSNR: 27.37058	SSIM: 0.77593 MSE: 0.0045
<b>Cover: Garbage Truck, Hidden: Peppers</b>		
<b>Image Metrics</b>	SSIM: 0.87806 PSNR: 27.89983	SSIM: 0.89549 MSE: 0.0019



Table 4.3 continued.

<b>Cover: Gas Pump, Hidden: Stained Glass</b>		
<b>Image Metrics</b>	SSIM: 0.81223 PSNR: 25.3442	SSIM: 0.83418 MSE: 0.00576
<b>Cover: Golf Balls, Hidden: Thistle</b>		
<b>Image Metrics</b>	SSIM: 0.90461 PSNR: 25.99714	SSIM: 0.87267 MSE: 0.0023

#### 4.4 Results of the Enhanced Network

This section focuses on the results of the proposed network with the various changes made. After making the changes to the network structure and parameters, the results of the embedding and extracting process were much more favorable in terms of imperceptibility and recoverability. As seen in Table 4.4, the resulting steganographic and extracted images generated by the proposed network resulted in greater PSNR and SSIM values than the values generated by the original network. This enhancement in imperceptibility was supported by the high SSIM and PSNR values generated between the two, with average values of 0.99711 and 43.04027 dB, respectively. As seen in Table 4.5, the results of the network were significantly improved when compared to the results from the original network. Compared to the results of the original network, the proposed network was much more effective at hiding the secret image inside the cover image. The SSIM and PSNR values of the sample steganographic image generated by the proposed network were 0.99498 and 41.41283 dB, respectively. These values are much greater than the SSIM and PSNR values of the sample steganographic image generated by the original network, which were 0.93041 and 28.69137 dB, respectively. When compared side-by-side, the steganographic images generated by the proposed network were virtually indistinguishable to the naked eye, while the steganographic images generated by the original network clearly exposed details of the hidden image.

The recoverability of the proposed network was also greater than that of the original network. Comparing the extracted images from the original and proposed networks revealed that the original network's extraction process lost some of the sharpness and minute details of the image. Additionally, extracted images contained a distorted, wave-like pattern. In Table 4.4, the sample extracted image generated by the original network resulted in SSIM and MSE values of 0.84422 and 0.00296, respectively. The proposed network improved slightly on the similarity between the hidden and extracted image with a SSIM value of 0.87402. However, the error between the hidden and extracted image resulted in a slightly greater MSE value of 0.003. Visually, the extraction process of the proposed network maintains the sharpness and the smaller details of the image. However, the contrast of the extracted images was noticeably reduced, and the extracted images were slightly grainy. From Table 4.5, the average SSIM and MSE values of the proposed network were 0.89496 and 0.00294, respectively. The average SSIM and MSE values of the original

network were 0.87314 and 0.00266, respectively. The increased image sharpness of the proposed network's extracted images is supported by the slightly increased SSIM value. The slightly higher MSE values may explain the graininess of the extracted images generated by the proposed network.

Comparing the quantitative and visual results between the original and proposed methodology, it can be surmised that the change in network structure and loss function resulted in greatly improved steganography results. Table 4.6 shows the comparison of image metrics obtained from the steganographic and extracted images generated by the original and enhanced networks. For all test images, the proposed network greatly outperformed the original network in terms of the imperceptibility and similarity between the cover and steganographic images. Additionally, the proposed network slightly improved regarding the similarity between the hidden and extracted images. However, since the goal was to improve the imperceptibility of the steganography process, this resulted in extracted images with slightly higher MSE values in most cases. As supported by the image metrics highlighted in Table 4.6, the changes introduced by the proposed methodology resulted in a highly imperceptible image steganography method and produced highly desirable steganographic images without sacrificing much in terms the extracted image's details.

Table 4.4. Side-by-side comparison of the steganographic and extracted images generated by the original and proposed methodologies.






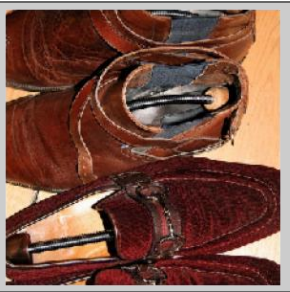


Network Version	Cover Image	Steganographic Image	Extracted Image	Hidden Image
Original Network (ImageNet dataset)				
Image Metrics	SSIM: 0.93041 PSNR: 28.69137		SSIM: 0.73294 MSE: 0.00749	
Proposed Network				
Image Metrics	SSIM: 0.99498 PSNR: 41.41283		SSIM: 0.87402 MSE: 0.003	



Table 4.5. The steganographic and extracted images generated by the enhanced methodology.

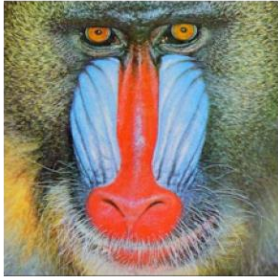







<b>Proposed Network</b>		
<b>Experiment</b>	<b>Steganographic Image</b>	<b>Recovered Image</b>
<b>Cover: Baboon, Hidden: Graffiti</b>		
<b>Image Metrics</b>	SSIM: 0.99748 PSNR: 42.85748	SSIM: 0.88449 MSE: 0.00262
<b>Cover: Berries, Hidden: Karaoke</b>		
<b>Image Metrics</b>	SSIM: 0.99848 PSNR: 44.33152	SSIM: 0.93213 MSE: 0.00131
<b>Cover: Chainsaws, Hidden: Lena</b>		
<b>Image Metrics</b>	SSIM: 0.99652 PSNR: 42.4834	SSIM: 0.91373 MSE: 0.00167
<b>Cover: Church, Hidden: Lotus</b>		
<b>Image Metrics</b>	SSIM: 0.99789 PSNR: 44.17786	SSIM: 0.90252 MSE: 0.00204

Table 4.5 continued.









<b>Cover: Dog, Hidden: Parachute</b>		
<b>Image Metrics</b>	SSIM: 0.99853 PSNR: 44.72544	SSIM: 0.91306 MSE: 0.00213
<b>Cover: Fish, Hidden: Parrot</b>		
<b>Image Metrics</b>	SSIM: 0.99773 PSNR: 44.487	SSIM: 0.92739 MSE: 0.00158
<b>Cover: French Horn, Hidden: Pens</b>		
<b>Image Metrics</b>	SSIM: 0.99739 PSNR: 42.45224	SSIM: 0.83423 MSE: 0.00498
<b>Cover: Garbage Truck, Hidden: Peppers</b>		
<b>Image Metrics</b>	SSIM: 0.99713 PSNR: 43.06884	SSIM: 0.89151 MSE: 0.00306

Table 4.5 continued.

<b>Cover: Gas Pump, Hidden: Stained Glass</b>		
<b>Image Metrics</b>	SSIM: 0.99499 PSNR: 41.65722	SSIM: 0.81930 MSE: 0.00767
<b>Cover: Golf Balls, Hidden: Thistle</b>		
<b>Image Metrics</b>	SSIM: 0.99494 PSNR: 40.16172	SSIM: 0.92120 MSE: 0.00231

Table 4.6. A comparison of performance metrics between the original and enhanced networks.

<b>Experiment</b>	<b>Original Network</b>		<b>Proposed Network</b>	
	Steganographic Image	Recovered Image	Steganographic Image	Recovered Image
<b>Cover: Baboon, Hidden: Graffiti</b>	SSIM: 0.93687 PSNR: 27.96642	SSIM: 0.87001 MSE: 0.00297	SSIM: 0.99748 PSNR: 42.85748	SSIM: 0.88449 MSE: 0.00262
<b>Cover: Berries, Hidden: Karaoke</b>	SSIM: 0.93457 PSNR: 25.62975	SSIM: 0.85424 MSE: 0.0026	SSIM: 0.99848 PSNR: 44.33152	SSIM: 0.93213 MSE: 0.00131
<b>Cover: Chainsaws, Hidden: Lena</b>	SSIM: 0.91134 PSNR: 29.35	SSIM: 0.91303 MSE: 0.00162	SSIM: 0.99652 PSNR: 42.4834	SSIM: 0.91373 MSE: 0.00167
<b>Cover: Church, Hidden: Lotus</b>	SSIM: 0.93737 PSNR: 29.59204	SSIM: 0.86413 MSE: 0.00222	SSIM: 0.99789 PSNR: 44.17786	SSIM: 0.90252 MSE: 0.00204
<b>Cover: Dog, Hidden: Parachute</b>	SSIM: 0.90129 PSNR: 25.7312	SSIM: 0.93006 MSE: 0.00138	SSIM: 0.99853 PSNR: 44.72544	SSIM: 0.91306 MSE: 0.00213
<b>Cover: Fish, Hidden: Parrot</b>	SSIM: 0.90204 PSNR: 25.77007	SSIM: 0.92166 MSE: 0.0013	SSIM: 0.99773 PSNR: 44.487	SSIM: 0.92739 MSE: 0.00158
<b>Cover: French Horn, Hidden: Pens</b>	SSIM: 0.87437 PSNR: 27.37058	SSIM: 0.77593 MSE: 0.0045	SSIM: 0.99739 PSNR: 42.45224	SSIM: 0.83423 MSE: 0.00498
<b>Cover: Garbage Truck, Hidden: Peppers</b>	SSIM: 0.87806 PSNR: 27.89983	SSIM: 0.89549 MSE: 0.0019	SSIM: 0.99713 PSNR: 43.06884	SSIM: 0.89151 MSE: 0.00306
<b>Cover: Gas Pump, Hidden: Stained Glass</b>	SSIM: 0.81223 PSNR: 25.3442	SSIM: 0.83418 MSE: 0.00576	SSIM: 0.99499 PSNR: 41.65722	SSIM: 0.81930 MSE: 0.00767
<b>Cover: Golf Balls, Hidden: Thistle</b>	SSIM: 0.90461 PSNR: 25.99714	SSIM: 0.87267 MSE: 0.0023	SSIM: 0.99494 PSNR: 40.16172	SSIM: 0.92120 MSE: 0.00231

## 5. CONCLUSIONS

In this research, the fields of digital image steganography and convolutional neural networks were explored to develop a novel method of embedding and extracting images. A close look was taken at many existing image steganography methods, which utilized techniques from the spatial, frequency, and neural network domains. To produce novel findings, this research started with a methodology proposed by [47] and implemented by [58], where significant changes to the network structure and parameters were made in order to obtain results that greatly improved on the original implementation. The network changes responsible for these improved results were removal of the image normalization methods, careful tuning of the network hyperparameters, omission of the noise layer, and the rework of the loss function to prioritize the similarity of the cover and hidden images over the reduction of the error between the input and generated images. These changes resulted in a network that imperceptibly embedded a color image within another color image of the same size and capacity.

There are several ways in which this work can be improved upon. The Python code for the proposed network was written using the first version of TensorFlow. Created by Google Brain, TensorFlow 1.0 was released by in 2017 and its successive major revision, TensorFlow 2.0, was released in 2019 [61]. In the second version, many fundamental changes were made to the library, and the results obtained by converting the existing code to TensorFlow 2.0 may be more desirable than the results obtained by continuing to use original version.

Improvements may also be made to the network by using a larger dataset with a greater variety of images. As mentioned in the Introduction, introducing noise to the training process can help mitigate the opportunity for the network to overfit to the training dataset. The Imagenette and Linnaeus 5 datasets both contain images with set labels for the purposes of image classification (e.g., dogs, flowers). To reduce the opportunity for the network to learn the details of images that classify these types of images and rely on them when performing the embedding and extracting processes, greater variety in the image datasets would be more ideal for the purposes of reducing overfitting. To this end, subsequent research on this methodology would also benefit from splitting the datasets into explicit training and testing datasets. Although the results obtained from the proposed methodology were greatly improved from the original, using images from the testing dataset foreign to the network would likely produce much better results.

Another avenue for future work would involve performing a steganalysis on the steganographic images produced by the proposed methodology. By analyzing a steganographic image, it is possible to detect the presence of a hidden image. If steganalysis is performed on the steganographic images and the presence of hidden information is detected, the proposed methodology can be altered to evade the detection. Conversely, if steganalysis is performed and it is determined that the cover image does not contain hidden information, then the proposed methodology would not require modifications to evade detection.

Lastly, the image metrics produced by this network would benefit from a proper and impartial comparison of other existing image steganography techniques. Because each implementation uses its own image sets for producing metric results, some of which are controlled by copyrights and intellectual property laws, it is not always feasible to obtain a completely fair comparison between methodologies. Additionally, frequently used images such as the Lena and Baboon images may not always be identical between uses due to factors such as compression and differences in image formats. For standardized, fair-use images that are used in existing methodologies, however, an avenue for future work would be to use these images to provide for a more fair and equitable comparison of the image metrics.

As witnessed by the significant improvement in results between the original and proposed methodologies, the importance of the structure and hyperparameters of the network should be emphasized. Small changes such as alterations in the training datasets, loss functions, and hyperparameters can have a profound influence on the training process and network results.

## REFERENCES

- [1] T. Jamil, "Steganography: the art of hiding information in plain sight," *IEEE Potentials*, vol. 18, no. 1, pp. 10-12, 1999.
- [2] I. Cox, M. Miller, J. Bloom, J. Fridrich and T. Kalker, *Digital Watermarking and Steganography*, San Francisco: Elsevier Science & Technology, 2007.
- [3] Herodotus, R. B. Strassler and A. L. Purvis, *The Landmark Herodotus : the Histories*, New York City: Pantheon Books, 2007.
- [4] D. Artz, "Digital Steganography: Hiding Data within Data," *IEEE Internet Computing*, vol. 5, no. 3, pp. 75-80, 2001.
- [5] F. Djebbar, B. Ayad, K. A. Meraim and H. Hamam, "Comparative study of digital audio steganography techniques," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2012, no. 1, pp. 1-16, 2012.
- [6] "What is bit depth?," Florida Center for Instructional Technology, 2018. [Online]. Available: <https://etc.usf.edu/techease/win/images/what-is-bit-depth/>. [Accessed 3 April 2022].
- [7] L. Fei-Fei, J. Deng, O. Russakovsky, A. Berg and K. Li, *ImageNet*, Stanford: Stanford University, 2020.
- [8] G. Chaladze and L. Kalatozishvili, "Linnaeus 5 Dataset for Machine Learning," 2017.
- [9] A. A. Zakaria, M. Hussain, A. W. A. Wahab, M. Y. I. Idris, N. A. Abdullah and K.-H. Jung, "High-Capacity Image Steganography with Minimum Modified Bits Based on Data Mapping and LSB Substitution," *Applied Sciences*, vol. VIII, no. 11, pp. 2199-2218, 2018.
- [10] X. Duan, D. Guo, N. Liu, B. Li, M. Gou and C. Qin, "A New High Capacity Image Steganography Method Combined With Image Elliptic Curve Cryptography and Deep Neural Network," *IEEE Access*, pp. 25777-25788, 2020.
- [11] Z. Wang, E. P. Simoncelli and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers*, Pacific Grove, 2003.

- [12] L. Hardesty, "Explained: Neural Networks," Massachusetts Institute of Technology, 14 April 2017. [Online]. Available: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>. [Accessed 19 September 2021].
- [13] G. Ognjanovski, "Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun," Medium, 14 January 2019. [Online]. Available: <https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a>. [Accessed 19 September 2021].
- [14] J. Brownlee, "How to Configure the Number of Layers and Nodes in a Neural Network," Machine Learning Mastery, 27 July 2018. [Online]. Available: <https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/>. [Accessed 19 September 2021].
- [15] A. Bhardwaj, "What is a Perceptron? – Basics of Neural Networks," Towards Data Science, 11 October 2020. [Online]. Available: <https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590>. [Accessed 2 April 2022].
- [16] deeplizard, "Bias In An Artificial Neural Network Explained | How Bias Impacts Training," DEEPLIZARD, 17 April 2018. [Online]. Available: <https://deeplizard.com/learn/video/HetFihsXSys>. [Accessed 2 April 2022].
- [17] A. D. Rasamoelina, F. Adjailia and P. Sincak, "A Review of Activation Function for Artificial Neural Network," in *International Symposium on Applied Machine Intelligence and Informatics*, Herlany, 2020.
- [18] I. Ohn and Y. Kim, "Smooth Function Approximation by Deep Neural Networks with General Activation Functions," in *Entropy*, Basel, 2019.
- [19] J. Brownlee, "How Do Convolutional Layers Work in Deep Learning Neural Networks?," Machine Learning Mastery, 17 April 2019. [Online]. Available: <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>. [Accessed 19 September 2021].
- [20] S. Saha, "Towards Data Science," Medium, 15 December 2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. [Accessed 26 February 2022].



- [21] P. S. Sinha, "Types of Loss Functions in Machine Learning," OpenGenus IQ, [Online]. Available: <https://iq.opengenus.org/types-of-loss-function/>. [Accessed 2 April 2022].
- [22] J. Durán, "Everything You Need to Know about Gradient Descent Applied to Neural Networks," Medium, 19 September 2019. [Online]. Available: <https://medium.com/yottabytes/everything-you-need-to-know-about-gradient-descent-applied-to-neural-networks-d70f85e0cc14>. [Accessed 26 March 2022].
- [23] I. C. Education, "What is Gradient Descent?," IBM, 27 October 2020. [Online]. Available: <https://www.ibm.com/cloud/learn/gradient-descent>. [Accessed 27 March 2022].
- [24] N. Donges, "Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms," Built In, 23 July 2021. [Online]. Available: <https://builtin.com/data-science/gradient-descent>. [Accessed 27 March 2022].
- [25] R. Gylberth, "An Introduction to AdaGrad," Medium, 2 May 2018. [Online]. Available: <https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827>. [Accessed 27 March 2022].
- [26] J. Brownlee, "Gradient Descent With AdaGrad From Scratch," Machine Learning Mastery, 11 June 2021. [Online]. Available: <https://machinelearningmastery.com/gradient-descent-with-adagrad-from-scratch/>. [Accessed 27 March 2022].
- [27] M. Riedmiller and H. Bruan, "A direct adaptive method for faster backpropagation learning: the RPROP algorithm," in *IEEE International Conference on Neural Networks*, San Francisco, 1993.
- [28] G. Hinton, N. Srivastava and K. Swersky, "rmsprop: Divide the gradient by a running average of its recent magnitude," *Neural Networks for Machine Learning*, 2012.
- [29] X. Ying, "An Overview of Overfitting and its Solutions," *Journal of Physics: Conference Series*, vol. 1168, no. 2, p. 022022, 2019.
- [30] J. Brownlee, "A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks," Machine Learning Mastery, 7 December 2018. [Online]. Available: <https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/>. [Accessed 2 April 2022].

- [31] D. Blalock, J. G. J. Ortiz, J. Frankle and J. Gutttag, "What Is The State of Neural Network Pruning?," in *3rd MLSys Conference*, Austin, 2020.
- [32] J. Hoare, "Machine Learning: Pruning Decision Trees," Displayr, 4 July 2017. [Online]. Available: <https://www.displayr.com/machine-learning-pruning-decision-trees/>. [Accessed 3 April 2022].
- [33] J. Furnkranz, "A Comparison of Pruning Methods for Relational Concept Learning," in *AAAI Workshop on Knowledge Discovery in Databases*, Vienna, 1994.
- [34] S. R. Shinde, "Improving Artificial Neural Network with Regularization and Optimization," Towards AI Inc., 29 September 2020. [Online]. Available: <https://towardsai.net/p/machine-learning/improving-artificial-neural-network-with-regularization-and-optimization>. [Accessed 3 April 2022].
- [35] J. Brownlee, "Use Weight Regularization to Reduce Overfitting of Deep Learning Models," Machine Learning Mastery, 19 November 2018. [Online]. Available: <https://machinelearningmastery.com/weight-regularization-to-reduce-overfitting-of-deep-learning-models/>. [Accessed 3 April 2022].
- [36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014.
- [37] J. Brownlee, "Train Neural Networks With Noise to Reduce Overfitting," Machine Learning Mastery, 12 December 2018. [Online]. Available: <https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/>. [Accessed 3 April 2022].
- [38] J. Brownlee, "A Gentle Introduction to Weight Constraints in Deep Learning," Machine Learning Mastery, 23 November 2018. [Online]. Available: <https://machinelearningmastery.com/introduction-to-weight-constraints-to-reduce-generalization-error-in-deep-learning/>. [Accessed 3 April 2022].
- [39] N. M. Surse and P. Vinayakray-Jani, "A Comparative Study on Recent Image Steganography Techniques Based on DWT," *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 1308-1314, 2017.

- [40] M. C. Kasapbasi and W. Elmasry, "New LSB-based colour image steganography method to enhance the efficiency in payload capacity, security and integrity check," *Sāadhanā*, pp. 43-68, 2018.
- [41] K. Ghazanfari, S. Ghaemmaghamsi and S. R. Khosravi, "LSB++: An Improvement to LSB+ Steganography," in *IEEE Region 10 International Conference*, Bali, 2011.
- [42] A. Pradhan, K. R. Sekhar and G. Swain, "Digital Image Steganography Using LSB Substitution, PVD, and EMD," *Hindawi*, vol. 2018, no. 1, pp. 1-11, 2018.
- [43] S. Talebi, "Time Series, Signals, & the Fourier Transform," Towards Data Science, 16 November 2020. [Online]. Available: <https://towardsdatascience.com/time-series-signals-the-fourier-transform-f68e8a97c1c2>. [Accessed 3 October 2021].
- [44] S. Talebi, "The Wavelet Transform. An Introduction and Example," Towards Data Science, 20 December 2020. [Online]. Available: <https://towardsdatascience.com/the-wavelet-transform-e9cfa85d7b34>. [Accessed 3 October 2021].
- [45] M. C. Nechyba, "Introduction to the Discrete Wavelet Transform (DWT)," 15 February 2004. [Online]. Available: [https://mil.ufl.edu/nechyba/www/eel6562/course\\_materials/t5.wavelets/intro\\_dwt.pdf](https://mil.ufl.edu/nechyba/www/eel6562/course_materials/t5.wavelets/intro_dwt.pdf). [Accessed 10 October 2021].
- [46] R. Thanki and S. Borra, "A color image steganography in hybrid FRT-DWT domain," *Journal of Information Security and Applications*, pp. 92-102, 2018.
- [47] S. Baluja, "Hiding Images in Plain Sight: Deep Steganography," in *31st Conference on Neural Information Processing Systems*, Long Beach, 2017.
- [48] D. Volkhonskiy, I. Nazarov and B. Evgeny, "Steganographic Generative Adversarial Networks," *International Society for Optics and Photonics*, p. 114333M, 2020.
- [49] M.-Y. Liu and O. Tuzel, "Coupled Generative Adversarial Networks," in *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, Barcelona, 2016.
- [50] R. H. Wiggins, H. C. Davidson, H. R. Harnsberger, J. R. Lauman and P. A. Goede, "Image File Formats: Past, Present, and Future," *RadioGraphics*, vol. 21, no. 3, pp. 789-798, 2001.

- [51] H. Dadgostar and F. Afsari, "Image steganography based on interval-valued intuitionistic fuzzy edge detection and modified LSB," *Journal of Information Security and Applications*, pp. 94-104, 2016.
- [52] G. Swain, "Digital Image Steganography Using Eight-Directional PVD," *Advances in Multimedia*, p. 13, 2018.
- [53] A. Delmi, S. Suryadi and Y. Satria, "Digital image steganography by using edge adaptive based chaos cryptography," *Journal of Physics Conference Series*, p. 012041, 2020.
- [54] H. Gupta, "buZZrobot," Medium, 11 February 2018. [Online]. Available: <https://buzzrobot.com/hiding-images-using-ai-deep-steganography-b7726bd58b06>. [Accessed 10 October 2021].
- [55] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference for Learning Representations*, San Diego, 2014.
- [56] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," Machine Learning Mastery, 3 July 2017. [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>. [Accessed 27 March 2022].
- [57] J. Howard, "Imagenette".
- [58] H. Gupta, "Deep-Steganography," 2018.
- [59] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu and scikit-image contributors, "scikit-image: Image processing in Python," 2014.
- [60] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007.
- [61] S. Gupta, "Tensorflow 1.0 vs. Tensorflow 2.0: What's the Difference?," Springboard, 27 October 2021. [Online]. Available: <https://www.springboard.com/blog/data-science/tensorflow-1-0-vs-tensorflow-2-0/>. [Accessed 3 April 2022].
- [62] X. Duan, N. Liu, M. Gou, W. Wang and C. Qin, "SteganoCNN: Image Steganography with Generalization Ability Based on Convolutional Neural Network," *Entropy*, pp. 22-36, 8 October 2020.

- [63] J. Mielikainen, "LSB Matching Revisited," *IEEE Signal Processing Letters*, pp. 285-287, 2006.
- [64] A. K. Sahu and M. Sahu, "Digital image steganography and steganalysis: A journey of the past three decades," *Open Computer Science*, pp. 296-342, 2020.
- [65] A. Khalifa, S. H. Hamad and A. A. Elhadad, "A Blind High-Capacity Wavelet-Based Steganography Technique for Hiding Images into other Images," *Advances in Electrical and Computer Engineering*, pp. 35-42, 2014.
- [66] X. Zhou, H. Zhang and C. Wang, "A Robust Image Watermarking Technique Based on DWT, APDCBT, and SVD," *Symmetry*, pp. 77-91, 2018.
- [67] R. Jain and J. Boaddh, "Advances in Digital Image Steganography," in *1st International Conference on Innovation and Challenges in Cyber Security*, Greater Noida, 2016.
- [68] Q. A. Al-Haija, C. D. McCurry and S. Zein-Sabatto, "An efficient deep learning-based detection and classification system for cyber-attacks in IoT communication networks," *Electronics*, pp. 21-52, 2020.
- [69] M. F. Tolba, M. A.-S. Ghonemy, I. A.-H. Taha and A. S. Khalifa, "High Capacity Image Steganography using Wavelet-Based Fusion," in *Proceedings of the 9th IEEE Symposium on Computers and Communications*, Alexandria, 2004.
- [70] H.-J. Shiu, B.-S. Lin, C.-H. Huang, P.-Y. Chiang and C.-L. Lei, "Preserving privacy of online digital physiological signals using blind and reversible steganography," *Computer Methods and Programs in Biomedicine*, pp. 159-170, 2017.
- [71] A. R. Calderbank, I. Daubechies, W. Sweldens and B.-L. Yeo, "Wavelet Transforms That Map Integers to Integers," *APPLIED AND COMPUTATIONAL HARMONIC ANALYSIS*, pp. 332-369, 1998.
- [72] S. Adhikari, "Shh! Your secret is safe — A simple guide to Steganography in Python," Medium, 25 May 2020. [Online]. Available: <https://medium.com/analytics-vidhya/shh-your-secret-is-safe-a-simple-guide-to-steganography-in-python-89116582277e>. [Accessed 2 October 2021].

- [73] L. Shukla, "Designing Your Neural Networks. A Step by Step Walkthrough," Towards Data Science, 23 September 2019. [Online]. Available: <https://towardsdatascience.com/designing-your-neural-networks-a5e4617027ed>. [Accessed 10 October 2021].
- [74] A. Damato, "JPEG2000 2-level Wavelet Transform-Lichtenstein," 17 May 2007. [Online]. Available: [https://upload.wikimedia.org/wikipedia/commons/e/e0/Jpeg2000\\_2-level\\_wavelet\\_transform-lichtenstein.png](https://upload.wikimedia.org/wikipedia/commons/e/e0/Jpeg2000_2-level_wavelet_transform-lichtenstein.png). [Accessed 10 October 2021].
- [75] S. Mazaheri, P. S. Sulaiman, R. Wirza, M. Z. Dimon, F. Khalid and R. M. Tayebi, "Hybrid Pixel-Based Method for Cardiac Ultrasound Fusion Based on Integration of PCA and DWT," *Computational and Mathematical Methods in Medicine*, vol. 2015, no. 1, pp. 1-16, 2015.