Neural-Based Adversarial Encryption of Images in ECB Mode with 16-bit Blocks[⋆]

Pablo Rivas[0000−0002−8690−0987] and Prabuddha Banerjee

Computer Science, Marist College, Poughkeepsie NY 12601, USA
{pablo.rivas,prabuddha.banerjee1}@marist.edu

Abstract. Digital images possess rich features that are highly correlated among neighboring pixels and highly redundant bits. Thus, the study of encryption algorithms over images is a subject of study to determine encryption quality. In this paper, we study the applicability of neural networks in learning to use secret keys to protect information (e.g. images) from other neural networks. We implement a well-known adversarial neural network architecture that is designed to learn to establish its own encryption method to protect itself from an attacker that is also a neural network. Our preliminary results suggest that this type of neural architecture is capable of securing communications in the midst of a neural-based attacker using only 16-bit blocks.

Keywords: adversarial neural networks · encryption · symmetric key encryption.

1 Introduction

Recently, there has been interest in encryption and decryption of communications using Adversarial Neural Networks [1]. This is due to Adversarial Neural Networks which can be trained to protect the communication using Artificial Intelligence. As neural networks can be used for advanced functional specifications which can be categorized into complex task and those are beyond simple functional specifications. These objectives include, for example, generating realistic images [5] and solving multiagent problems [3, 9]. Advancing these lines of work, we showed that neural networks can learn to protect their communications in order to satisfy a policy specified in terms of an adversary.

Cryptography concerns broadly with algorithms and protocols that will give certainty that the secrecy and integrity of our information is maintained. Cryptography is the science of protection our data and communication.

A secured mechanism is where it achieves its goal against all attackers. So these form Turing machines or a cryptographic mechanism are formed as programs. Hence attackers may be described in those terms as well as the complexity of time and how frequently are they successful in decoding messages. For instance, an encryption algorithm is said to be secure if no attacker can extract

⋆ Supported by the VPAA's Office at Marist College.

information about plaintexts from ciphertexts. Modern cryptography provides rigorous versions of such definitions [4]. For a given problem Neural networks have the ability to selectively explore a solution space. This feature finds a natural niche of application in the field of cryptanalysis.

Neural networks provides new methods of attacking an encryption algorithm based on the fact that neural networks can reproduce any function; this is considered as a very powerful computational tool since any cryptographic algorithm must have an inverse function.

In the design of the neural network we study in this paper, adversaries play a very important role. They were originally proposed using what is known as 'adversarial examples' that were found using a neural architecture named a *generative adversarial network* (GAN) [5]. In this context, the adversaries are neural networks that sample data from a distribution whose parameters are learned from the data. Furthermore, as per the Cryptography definition, practical approaches to training Generative adversarial networks do not consider all possible adversaries, rather adversaries relatively small in size which are improved upon by training. We built our work based upon these ideas.

Historically, neural networks have received criticism in the area of cryptography since earlier models were not able to model an XOR operation, which is commonly used in cryptography. However, nowadays neural networks can easily model XOR operations and be taught to protect the secrecy of data from other neural networks; such models can discover and learn different forms of encryption and decryption, without specifically creating an existing algorithms or even knowing the type of data they will encrypt.

In this paper we will discuss the implementation of such architecture in the context of image encryption and will comment on the results obtained. This paper is organized as follows: Section 2 discusses the background pertaining to cryptography and adversarial learning along with the methodology used. Section 3 discusses our experiments and the results obtained. Finally, Section 4 draws conclusions of our paper.

2 Background and Methodology

2.1 Ciphertext-only attacks

In the basic architecture proposed by [1], there are 3 neural networks of which the first one is Alice who is part of the cryptographic learning model, so is tasked with encrypting binary strings (e.g. of an image) provided a key. Bob, another friendly neural network, is tasked with learning to decrypt Alice's message. Both Alice and Bob are performing symmetric-key encryption, so they have a common key for encrypting their communication; this key needs to be provided by a human prior to the learning prior to the learning process. Lastly Eve's neural network will be eavesdropping and try to attack or hack their communication by observing the ciphertext. This is known as a ciphertext-only attack [8]. While this is a neural network-based architecture, the overall encryption scheme is well known in textbooks as the one shown in Fig. 1.

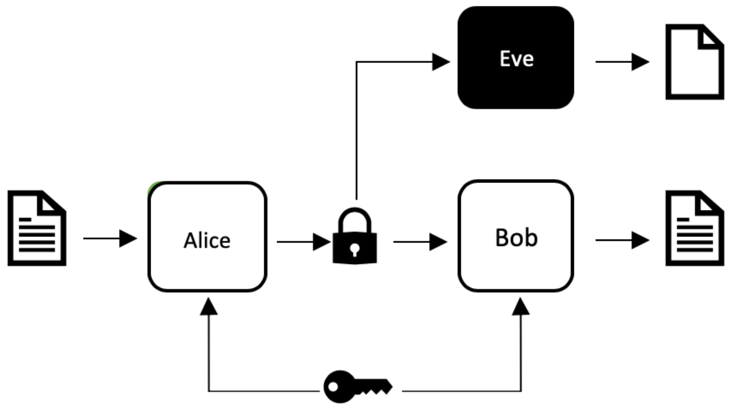

Fig. 1: A traditional encryption scheme with a twist: Alice, Bob, and Eve are neural networks.

2.2 Block cipher mode: ECB

All block ciphers require specific chunks of data (usually binary) of specific sizes that will be used as input or output. Traditionally, these blocks of data are encrypted and decrypted; however, what happens to those blocks internally on the cipher is unique to each cipher. As long as we have information that can be broken down into binary strings of information (e.g. images) we can use any block cipher.

In this paper we are concerned with encrypting images. We used the Python programming language in which we take the image and convert into strings of byte arrays and then the byte arrays are flattened and converted into binary strings of the size of the desired block. In this preliminary research we limited our study to 16-bits of information per block. The choice of this block size is due to the large training time for larger blocks.

There are several ways or *modes* in which block ciphers operate. Each mode provides additional security. The only mode that does not provide additional security is known as Electronic Code-Book (ECB) mode. It is important to remark that in order to study the quality of a cipher we must study it by itself with no additional help from other advanced (and much better) modes, so ECB is used here for functionality and to allows us an objective measure of quality of the cipher.

The encryption task in ECB mode here is given by:

$$C_i = E_k\left(P_i\right), \quad i = 1, 2, 3, \ldots \tag{1}$$

where the function $E_k(\cdot)$ is the encryption algorithm operating with secret key k, P_i is the i-th block of plain text, and C_i is the corresponding i-th block of cipher text.

During encryption the key, which is of 16 bits in size, is represented as an array of the following form:

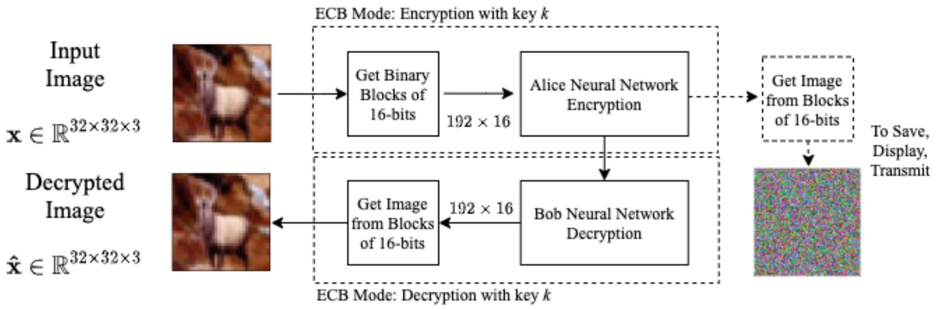
ECB Mode: Encryption with key k
Input Image
$\mathbf{x} \in \mathbb{R}^{32\times32\times3}$
Get Binary Blocks of 16-bits
192 × 16
Alice Neural Network Encryption
Get Image from Blocks of 16-bits
To Save, Display, Transmit
Decrypted Image
$\hat{\mathbf{x}} \in \mathbb{R}^{32\times32\times3}$
Get Image from Blocks of 16-bits
192 × 16
Bob Neural Network Decryption
ECB Mode: Decryption with key k

Fig. 2: ECB mode of operation on an image based on key k. The encrypted image, *ciphertext*, can be saved, displayed, transmitted, or further analyzed as needed.

```
key = [0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]
```

Decryption works in similar way but in reverse w.r.t. encryption, i.e., first the cyphertext blocks are passed through the decryption nerual network (Bob) and the original plaintext blocks are recovered. The decryption is here governed by:

$$P_i = D_k\left(C_i\right), \quad i = 1, 2, 3, \ldots \tag{2}$$

where $D_k(\cdot)$ is the decryption algorithm operating with key k, and P_i and C_i are as in Eq. (1).

The reconstruction process of an image in Python follows a similar process as prior to encryption, i.e. in two stages where in the recovered plaintext is first converted into hexadecimal numbers, then converted to array of bytes, and then these arrays of bytes are thereafter reshaped forming an image. This process is exemplified in Fig. 2.

2.3 Neural architecture

Fig. 3 depicts the neural architecture implemented in Python with Keras on TensorFlow; the original architecture was presented in [1], and is based on Generative Adversarial Networks [5]. The neural architecture has two identical networks, Alice and Bob; however, Bob is a mirror (reverse) of Alice. The Alice network has a combination of one-dimensional convolutional layers and dense layers; there are a total of four convolutional layers, the first three have hyperbolic tangent activations (tanh), and the last convolutional layer has a sigmoid activation as the output. At the input layer, Alice has a dense layer with a tanh activation. Bob is the exact mirror of Alice, *i.e.* in opposite order.

The input to Alice has to be two vectors: a binary string denoting the plaintext $P \in \mathbb{B}^{16}$ and a binary string denoting the key $k \in \mathbb{B}^{16}$. In this implementation we used 16-bits for both. The output of Alice is a vector of codes known as the ciphertext $C \in \mathbb{R}^{16}$;

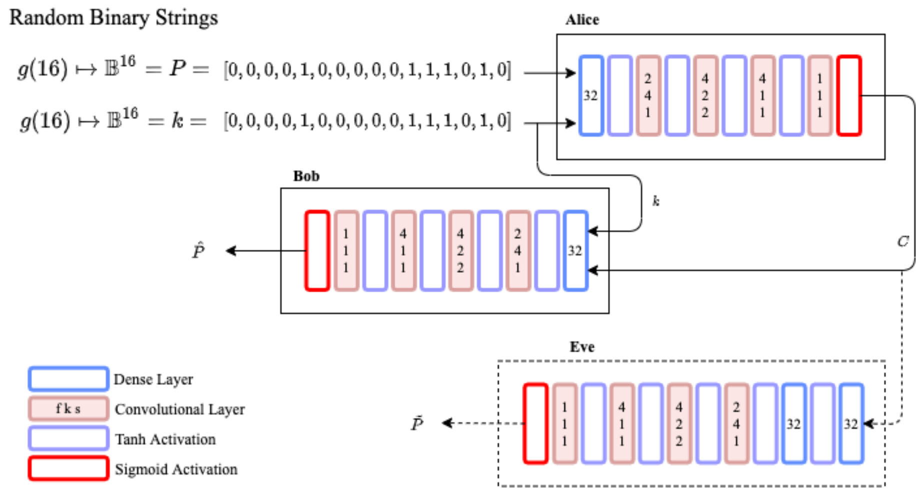

Fig. 3: Neural architecture of the cipher. During training, Alice and Bob are trained using randomly generated keys and plaintext while Eve attempts to recover the plaintext from the ciphertext and no key.

During training, we randomly generated pairs of (P, k) in mini-batches of size 256, which was determined experimentally to be a nice trade-off between training speed and performance. A standard random number generator function $g(16) \mapsto \mathbb{B}^{16}$ was used to produce binary strings of length 16. We also determined experimentally that 30 epochs were sufficient to train the architecture securely.

Before we discuss the adversarial nature of the training, we now address the design of Eve, the network designed to break the code. Eve's architecture is identical to Bob's, except that it has two stacked Dense layers with tanh activations, instead of only one, and it is also followed by the same convolutional layers as Bob. The idea behind this additional dense layer is to give the Eve's network a better chance to retrieve the plaintext from the ciphertext.

As opposed to a GAN, the generator is not part of the architecture in this case, instead we have Alice and Eve competing against each other, where Eve is the adversary. Alice and Bob are not compeeting but they resemble an autoencoder architecture in which the input of Alice (i.e. P and k) is encoded into C and Bob is taksed with recovering P from C and k. While this task might be straight-forward in a traditional sense, the code that is produced by Alice is affected by the degree to which Eve is able to recover P from C alone. We call Bob's reconstruction $\hat{P}$ and Eve's reconstruction $\tilde{P}$. During an ideal training session we want $s(P, \hat{P}) \to 0$ and $s(P, \tilde{P}) \to \infty$, where $s(\cdot)$ is some similarity measure where zero means they are identical.

The specific losses used are the following:

$$L_{\text{Eve}} = \frac{1}{m}\sum_{i}^{m} |P_i - \tilde{P}_i| \tag{3}$$

$$L_{\text{Bob}} = \frac{1}{m}\sum_{i}^{m} |P_i - \hat{P}_i| \tag{4}$$

$$L_{\text{Alice-Bob}} = L_{\text{Bob}} + \frac{\left(\frac{|P|}{2} - L_{\text{Eve}}\right)^2}{\left\lfloor\frac{|P|}{2}\right\rfloor^2} \tag{5}$$

where the batch size m is 256 and $|P|$ denotes the size of the plaintext, which is 16 in this case.

Eve's goal is simple: to reconstruct P accurately by minimizing L_{Eve}. Bob simply wants to recontruct the plaintext perfectly using the secret key k by minimizing L_{Bob}. Alice and Bob want to communicate clearly by making Alice produce a code that Bob can recover using the key while preventing Eve from learning to recover the plaintext without a key.

3 Experiments and Results

During our experiments implementing the architectures shown in Fig. 3, we observed convergence of the network as early as in 3000 training steps (or around 12 epochs), as shown in Fig. 4. The total number of steps shown in the figure come from the selected batch size, which is 256, and the total number of epochs, which was 30; thus, the horizontal axis from 1 to 7680. The plot shown in Fig. 4 is produced by overlaying 30 different experiments. A typical single experiment looks like the one shown in Fig. 5.

From Fig. 4 or Fig. 5 we can see three different lines; each line corresponds to the loss functions as the training takes place over time, i.e. L_{Eve}, L_{Bob}, and $L_{\text{Alice-Bob}}$. As expected, both loses L_{Bob} and $L_{\text{Alice-Bob}}$ are minimized while L_{Eve} is maximized. This min-max game is part of the fundamental definition of adversarial learning [6].

During the training of the models, for each trainin step, both Alice and Bob are trained together so that Alice learns to produce a ciphertext based on a unique key, and Bob learns to retrieve the plaintext from ciphertext and key. Next, Eve is trained on the ciphertext preventing Alice from adapting its parameters. However, note that in training Eve, we are drawing twice as a many cyphertext samples as with Bob, hoping to have the best version of Eve in every iteration. The parameter updates are done with a stochastic gradient descent algorithm known as RMS Prop [2, 7], with standard parameters.

In order to measure the quality of the cipher we produce pairs of 10,000 binary strings and their keys, (P, k), and observe the percentage of plaintext bits recovered by Bob and Eve. We performed this experiment 30 times and produced the histogram counts shown in Fig. 6. We can see that most of the

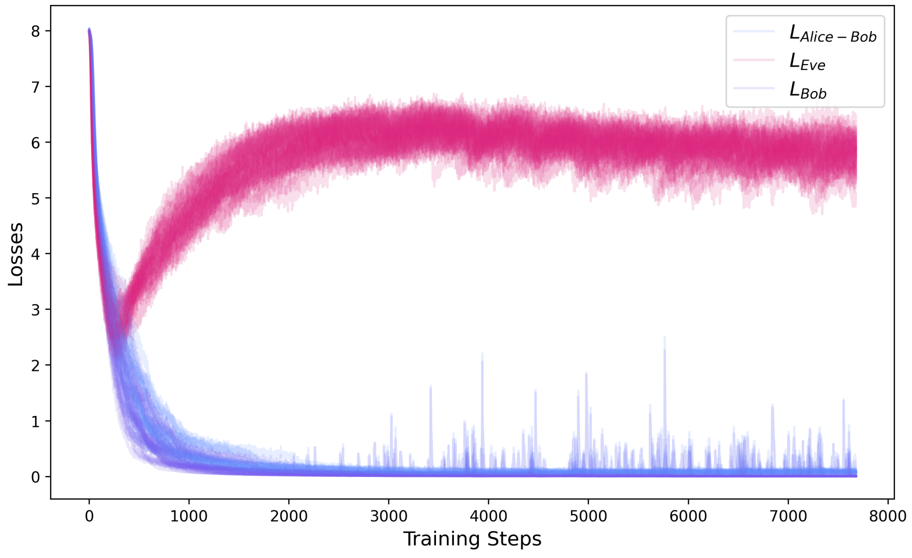

Fig. 4: Loss functions, L_{Eve}, L_{Bob}, and $L_{\text{Alice-Bob}}$, are minimized over time as training steps are performed. Results of 30 experiments are shown.

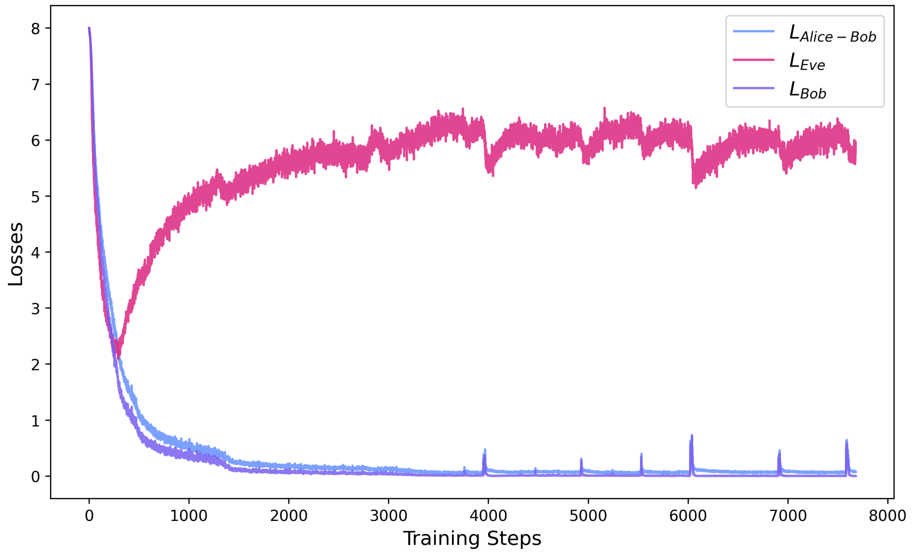

Fig. 5: Loss functions, L_{Eve}, L_{Bob}, and $L_{\text{Alice-Bob}}$, are shown for a single training experiment.

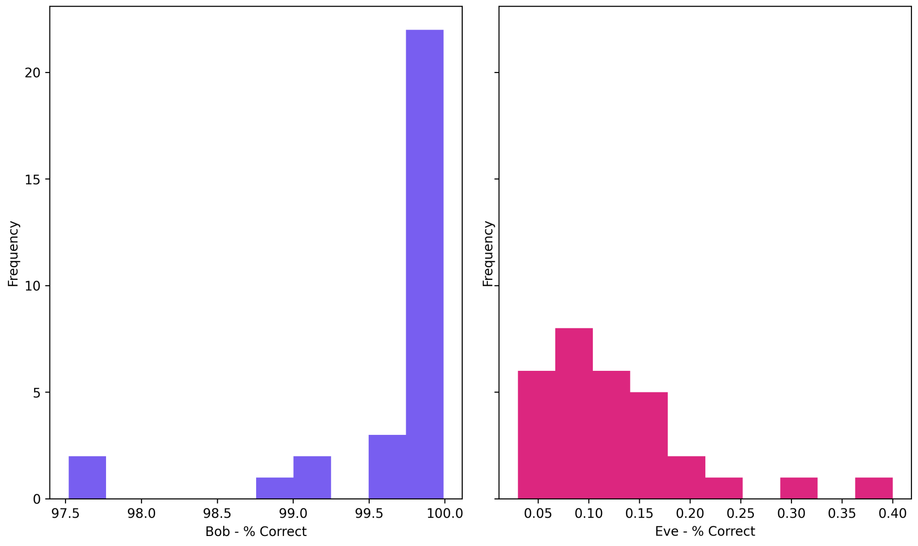

Fig. 6: Histogram that counts the frequency of both Bob and Eve correctly recovering all the bits in the original plaintext. As desired, Eve has a low percentage, while Bob has a very high percentage. These frequency counts are based on 30 independent experiments.

time Bob can reconstruct the plaintext bits beyond a 99.5% of accuracy. On the other hand, Eve can usually recover no more than 0.15% of the bits in the plaintext. This is clearly not a perfect hack of the information encoded. We can interpret this results a having the best Eve network not being able to crack the cipher, and Alice and Bob being able to figure out a way to encode information to a very high degree of secrecy.

Beyond the experiments where we have shown that the losses are minimized and the bit strings are recovered correctly to high accuracy and that the best versions of Eve cannot perform successful cipertext-only attacks, now we want to proceed with our final point which is its applicability in the encryption of images which contain highly-correlated information in neighbouring pixels. For this approach we take images and encrypt them with Alice, then we present and observe the encoded version looking for visual clues that might reveal the content of the plaintext (image), and then we use Bob to try to recover the plaintext using the wrong key, to make sure that Bob does not reveal details about the plaintext, and then we recover using the correct key to make sure the image was correctly recovered; the results are shown in Fig. 7.

From Fig. 7 we can see that for the cases in which images are properly encoded (rows a, b, c, and e) the corresponding image when the wrong key is used is also good quality. Here we refer to good quality and properly encoded as a way to say the images resemble random noise although some artifacts related to spatial

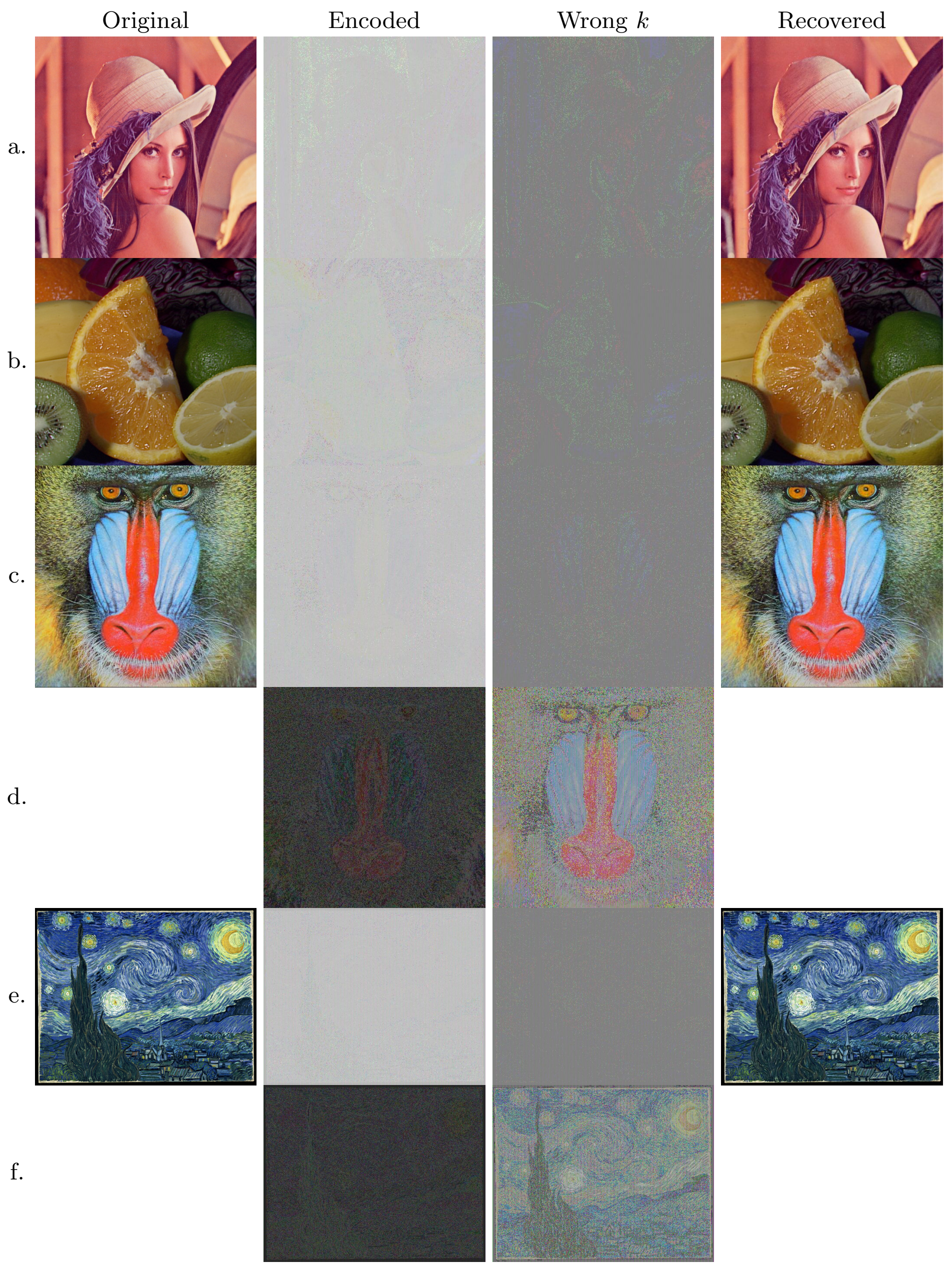

Fig. 7: Test of neural encryption of images. Rows a, b, c, and e are with high-quality ciphers, while rows d and f are with low-quality ciphers.

properties of the original might be visible yet not sufficient to reveal textures, colors, or general shapes. The figure also shows that the recovered image using the correct key leads to perfect reconstruction. However, while the results shown are common, we also take some of the worst models to show that a cipher can reveal information about the original image both in its encoded form, and in while trying to recover using the wrong key. Such results are shown in rows d and f. Note however, that in both cases, with a poor cipher or a good cipher, the wrong key that was used, was actually a key which differed in only one bit; e.g., if the key is originally:

$$[0,1,0,1,0,1,\underline{\mathit{0}},1,0,1,0,1,0,1,0,1],$$

then, the wrong key used was:

$$[0,1,0,1,0,1,\underline{\mathit{1}},1,0,1,0,1,0,1,0,1],$$

thus, providing with a more accurate representation a typical scenario in which a cipher should produce a completely different encoding if only one bit in the key is changed.

4 Conclusions

In this paper we studied the applicability of adversarial neural networks in the encryption and decryption of images. The cipher we implemented [1], exhibits high accuracy, fast convergence, and potential for usage in the protection of data communications and other similar applications. Our experiments suggest that, usually, the architecture produces good quality results on images, indicating low visual correlation in neighboring pixels, which validates the high-accuracy of the models produced experimentally each time. Furthermore, the proposed experiments validate current studies that show that adversarial training of neural networks has a great potential in the world of cryptography. The idea that one neural network can develop its own way of communications while there is a smart attacker that is listening and learning at the same time is an extraordinary achievement in the field of deep learning.

While the experiments shown in this paper were limited to only 16-bits for a key and plaintext, further research will be focused in using larger keys and larger blocks to study the impact on the quality of the cipher and its attacker; however, preliminary research shows that increasing the key and block size increases the training time exponentially. Therefore, further research must include a more efficient implementation.

Acknowledgements

This work was supported in part by the New York State Cloud Computing and Analytics Center, and by the VPAA's office at Marist College.

References

1. Abadi, M., Andersen, D.G.: Learning to protect communications with adversarial neural cryptography. arXiv preprint arXiv:1610.06918 (2016)
2. Dauphin, Y., De Vries, H., Bengio, Y.: Equilibrated adaptive learning rates for non-convex optimization. In: Advances in neural information processing systems. pp. 1504–1512 (2015)
3. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to communicate to solve riddles with deep distributed recurrent q-networks. arXiv preprint arXiv:1602.02672 (2016)
4. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of computer and system sciences **28**(2), 270–299 (1984)
5. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
6. Huang, R., Xu, B., Schuurmans, D., Szepesvári, C.: Learning with a strong adversary. arXiv preprint arXiv:1511.03034 (2015)
7. Mukkamala, M.C., Hein, M.: Variants of rmsprop and adagrad with logarithmic regret bounds. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 2545–2553. JMLR. org (2017)
8. Stanoyevitch, A.: Introduction to Cryptography with mathematical foundations and computer implementations. CRC Press (2010)
9. Sukhbaatar, S., Fergus, R., et al.: Learning multiagent communication with backpropagation. In: Advances in neural information processing systems. pp. 2244–2252 (2016)