Encrypting and Decrypting Images using Adversarial Neural Networks in ECB Mode

P. Rivas-Perea[1], Prabuddha Banerjee
[1]Department of Computer Science, Marist College, Poughkeepsie, New York, USA

Abstract— *In contrast to ordinary data encryptions, images posses unique features which include high correlation among its pixels, bulky data capacity, and higher redundancy rate of its bits; thus, as per the traditional encryption methods such as the Electronic Codebook(ECB) mode is much more susceptible to codebook attack. In this paper, we propose to use neural networks that can learn to use secret keys to protect information from other neural networks which could possibly be an AI attacker. Thus, we have a system that consists of neural networks named Alice and Bob as per the convention, and we aim to limit what a third neural network named Eve learns from eavesdropping on the communication between Alice and Bob. We didn't use any specific cryptographic algorithms to these neural networks; instead, we train end-to-end adversarial neural network. Specifically, we focus on developing a binary-string block cipher in ECB mode, and we specify the model in terms of adversarial networks. Using Python, we encrypted the Image data to analyze and train the Neural networks according to the algorithm's performance and thereby encrypting and decrypting the image. We provided the implementation by Symmetric Encryption , and some proof-of-concept code that will allow a user to understand the encryption scheme learned by neural networks through adversarial training.*

Keywords: Adversarial Neural Network; encryption; Symmetric 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 (e.g., (Goodfellow et al., 2014a)) and solving multiagent problems (e.g., (Foerster et al., 2016a;b; Sukhbaatar et al., 2016)). 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 frequent are they successful in it. For instance, an encryption algorithm is said to be secure if no attacker can extract information about plaintexts f rom ciphertexts. Modern cryptography provides rigorous versions of such definitions (Goldwasser & Micali, 1984).

For a given problem Neural networks have ability to selectively explore the solution space. This feature finds a natural niche of application in the field of cryptanalysis. Neural networks provides new methods of attacking a ciphering algorithm based on the fact that neural networks can reproduce any function, this is considered as very powerful computational tool as for any cryptographic algorithm function its inverse-function will always be present.

To design any neural network, adversaries play very important role. They arise, in particular, in work on adversarial examples (Szegedy et al., 2013; Goodfellow et al., 2014b) and on generative adversarial networks (GANs) (Goodfellow et al., 2014a). In the latter context, the adversaries are the neural networks that the sample data was drawn from a data distribution or generated by any model or machine. Furthermore, as per the Cryptography definition, practical approaches to train Generative adversarial networks does not consider all the adversaries in class, rather adversaries in small size which are improved upon by training. We built our work based upon these ideas.

In general Neural networks are not significant at cryptography as even the simplest of neural network cannot perform XOR operation, which is commonly used in cryptographic algorithms. But, after training Neural Networks can be taught to protect the secrecy of their data from other neural networks: they can discover and learn different forms of encryption and decryption, without specifically educating them the algorithms and its purpose.

We performed here an addressed to learn symmetric encryption (which is exclusively shared-key encryption, in this same keys are used for encryption and decryption purposes) and our results related to it.

2. Methodology

In the basic setup there are 3 neural networks of which the first one is Alice who is embedded into encryption algorithm, so she will encrypt the image. Then the next neural network Bob is processing decryption algorithm. Both Alice and Bob are performing symmetric encryption, so they have a common key for encrypting their communication. Lastly Eve's neural network

will be eavesdropping and try to attack or hack their communi-
cation.
But here Eve doesn't have the key of their communicating image.
Here we are encrypting the image in two main stages. In the first stage we take the image and convert into strings of byte arrays. The byte arrays are flattened and are made in contiguous array. Thereafter, the hex string is converted into binary numbers in short batch sizes. This done so that the losses during conversion are minimized.
The encryption here is governed by:

$$C_i = E_k(P_i) \ldots\ldots\ldots (1)$$

Where E_k is the encryption key, P_i is the plain text and C_i is the Cipher text of the current block.
During encryption the key which is of 16 bit size binary array in the following form of:

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

The binary key, used to encrypt in Plaintext in batch sizes as the images have very large size and stored in arrays. The encrypted image `enc_img` is then passed to the `bin2img` This function send the They are then further sent to Alice-bob and Eve's network for training.

The Decryption works in similar way so its just reverse of encryption i.e., in two stages where in the cipher text is first converted into hexadecimal numbers and after that its converted to array of bytes. These arrays of bytes are thereafter reshaped in python and formed back to an image. The decryption is here governed by:

$$P_i = D_k(C_i) \ldots\ldots (2)$$

Where Dk is the Decryption key which will be used for decrypting the image.

At the Neural Architecture training of Alice two input are passed to encrypt the message, of which the first is the message to be encrypted and the next is the secret key. This is will be done so as to make the output in 2-dimmension in mini-batches instances along the axis which is 1. Then these are reshaped for the plotting purpose.
The training of Bob is identical to that of Alice, except now the input for the function will be a encrypted text as for the previous one was a plain binary text.

Now, for the Eve's architecture the setup has two Dense layer. A Dense Layer is output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as
the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if bias is True). The idea behind this is to give the Eve's network a better chance to decrypt the message since the access to secret key is only limited to Alice and Bob, Eve only has access to cipher text.

Note that, with this formulation, C, P_{Bob}, and P_{Eve} may consist of arbitrary floating-point numbers even if P and K consist of 0s and 1s. In practice, our implementations constrain these values to the range (−1, 1), but permit the intermediate values.
Eve's goal is simple: to reconstruct P accurately (in other words, to minimize the error between P and P_{Eve}). Alice and Bob want to communicate clearly (to minimize the error between P and P_{Bob}), but also to hide their communication from Eve.

3. Experimental Setup and Results

Using the Python language we implemented the EBC based encryption protocol of an image; we successfully executed the encryption process by taking an input image path, image. For the setup we have taken 3 model parameters for encryption of m_bit for message encryption. This is of size 16 bits. Further, the parameters used are k_bit that is the key for encryption and lastly the c_bit for the cipher text block. Here all are of size 16 bits.
We use the inputs and train the neural networks as well as the adversarial network for encrypting the message and then decrypting it. This mode of encryption involves Symmetric Encryption. Here the adversary will be a "passive attacker", it will not initiate sessions, inject messages or modify messages in transit in this mode.

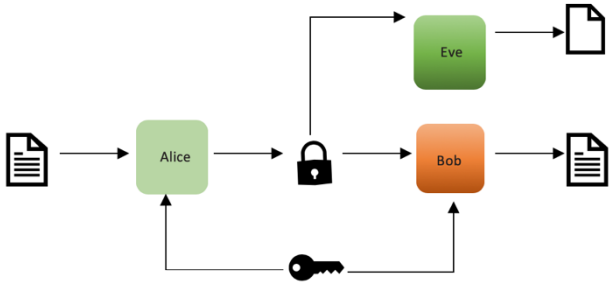

Figure 1: Alice, Bob, and Eve, with a symmetric cryptosystem

In the Alice's algorithm the training is done by giving an input of image's binary bit. So we do that by giving it value in m_bit with a additional value of key to it. The overall process of Alice and Bob are same so we check the losses of alice and bob as combined loss called abeloss.

As per [1] the Alice network has concatenated 16-bit(N) input into a 2N vector input, their values are between -1 and 1. The values are somewhat like:

```
c2      1.000000
```

```
c3      0.155269
c4      0.037282
c10     0.023214
c5      0.020128
c7      0.018525
c0      0.018297
c8      0.012671
c13     0.011509
c12     0.001859
c11    -0.006156
c14    -0.027707
c1     -0.035589
c9     -0.060452
c6     -0.078208
c15    -0.260225
Name: c2, dtype: float64
```

The values 16X4 Fully connected layer are then processed in four 1D-convulational layers. As per the definition a convolutional layers is defined in terms of their window size, input depth, and output depth. Each layer has a "stride"—the amount by which the window is shifted at each step. Here after each layer for processing of the image we used a sigmoid nonlinear unit after each layer for the processing of the image except the final layer. After the final layer, where the output is reduced to 16 elements, a tanh nonlinear unit is used to restrict the values of the output between [-1,1] the tanh range .
To train the network we use mini-batches of size 256, this does not restricts their learning over time as we require Alice and Bob to have a strong successful communication and achieve a solution which is strong and not encrypt able by Eve. The training is alternated between Alice-bob and Eve, so once Alice-bob gets trained for a batch Eve gets trained for two minibatches. This was done according[1] to computational edge to Eve's training as we even give Eve a chance to decrypt the message with equal opportunity. We wanted Alice and Bob to defeat the best version of Eve rather than a fixed Eve. So the training we did were in 3 phases, firstly when Alice-Bob and Eve all neural networks are trained simultaneously, and the result of their final training was:

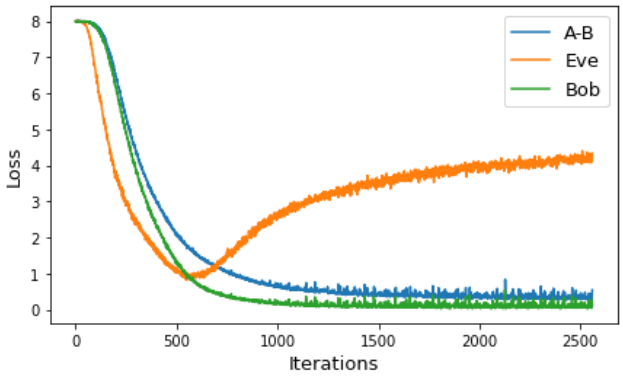

Fig1: Alice – Bob and Eve are trained simultaneously.

Where Bob and Eves accuracy were as shown:

```
Bob % correct:  92.94 %
Eve % correct:  0.61 %
```

As per [1] a per-example loss function for Eve is defined as:

$$L_E(\theta_A,\theta_E,P,K) = d(P,E(\theta_E,A(\theta_A,P,K)))\ldots\ldots(3)$$

Where θ_A, θ_E are all the Parameters of Alice and Eve respectively and P is Plaintext and K is the key.
Therefor the loss of Eve is defined by the L distance between the Alice's input and Eve's output. The per-example loss of the network is the average number of incorrect bits guessed by the neural network. An entire average is taken for the entire minibatches processed by Eve with K.mean().The minimum value of the loss is 0 (Eve guesses all the bits correctly), while the maximum is 16 (Eve is wrong about all the bits, in which case we flip the prediction and recover the message).

But for the loss of Alice-Bob is calculated differently. So first Bob finds the solution of the encrypted message successfully, also Alice learns the encryption which Eve cannot decrypt successfully. When Eve starts to guess a few bytes correctly then the square term ensures growth of error. Then Alice changes the encryption scheme to avoid the attackers rapidly. Overall the Alice-Bob loss will be:

```
abeloss = bobloss + K.square(m_bits/2 -
eveloss)/( (m_bits//2)**2 )
```

So when Eve has a large error and imposes less penalty we prefer to use quadratic formula to emphasis the situation. Adopting this formula as happens occasionally that Eve guess few bytes correctly but which may be random effective guess. By improved training adoption of these formulation allows us to have a meaningful example of loss function and instead of looking at larger batch statistics and improved the strength of training.

In the Next phase, the training of Alice and Bob training was frozen and only Eve was trained:

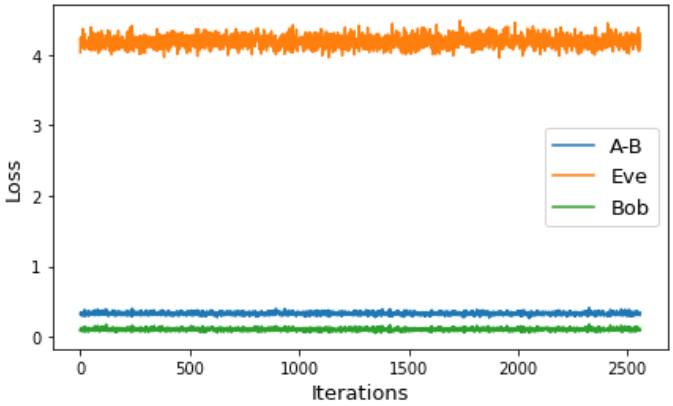

Fig2: Alice – Bob training are freezed and Eve is trained.

Here the accuracy of Bob and Eve are:

```
Bob % correct:  92.68 %
Eve % correct:  0.63 %
```

In the last phase when all were trained The resulting performance
of Bob and Eve was evaluated on a sample of 10,000 ciphertexts.
In this case the accuracy came out to be :

```
Bob % correct:  99.89 %
Eve % correct:  0.12 %
```

Further the graph which got plotted in Python according to their performance is as depicted by fig:

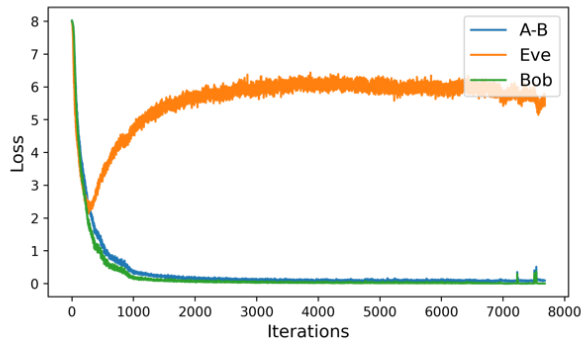

Fig3 : Alice – Bob and Eve are trained simultaneously.

In order to find the losses of the Alice, bob and Eve, the neural networks are first connected as the fig:1

```
aliceout = alice([message, key])
bobout = bob( [aliceout, key] )
# bob sees ciphertext AND key
eveout = eve( aliceout )
# eve doesn't see the key
```

Finally, for the Encryption of images, the message Image is first converted into a string of bytes array. But before doing that it needs to be raveled or flattened for the purpose making it in 1D array in contiguous form. To convert the ciphertext to something human-readable we convert each coordinate of the 16-dim encoding into a 16-bit binary string, and concatenate them together to get a binary encoding. Therefore, each human-readable character goes from an 16-bit padded binary representation to a 16·16=256-bit ciphertext representation. For example, the plaintext “adi” has (one of many) binary representation

```
000000000000001100101000
```

In the decryption process we back track the encoding process starting with the binary string we get of key ,as well as for the encrypted message. For the experiment the incorrect key used by Eve for decrypting is kept same as that of Alice which she used for encrypting except just changing one bit of it:

```
the_key =
[0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1]


wrong_key = [0,1,0,1,0,1,1,1,0,1,0,1,0,1,0,1]
                      ^
```

Changed to 1 from 0

This is done to check if incase Eve tries to tamper the key or in case Bob gets an incorrect image how much the image decryption process gets affected by it. The encrypted image array as per enc-rypted by Alice is stored in `enco`. The image is encrypted as per the method in batch sizes of 256, so to store the encrypted value of entire image Alice is required to encrypt the remaining part and append to the parameter `enc_img`. Similarly, when Bob decrypts the cipher text with the help of correct key in batches, it stores the array of bytes in deco and repeats this process at the end once again and stores in `dec_img`.
Finally we ask Bob again to decrypt the image with incorrect key using the identical process but now with the incorrect or tamper-ed key and stored in the parameter called `wrong_img`.
Since the images are in binary form and if we try to put the decrypted image with correct key together we will get some arbitrary message as:

```
?pfx?bhj?t,z!!ma!nhr:exs!qlt,apl:id?.j
```

So, we Binarize the value using the Python’s Binarizer function and passing a threshold value to it. Then we transform the image to get the final image.
Hence, at the end we can say Bob is able to decrypt the message correctly with the help of correct key as Fig: but when he tries to decrypt the file with tampered key he gets the decrypted image as Fig5 second image.

Finally, at the end its worth mentioning that the Neural networks can be used to protect the communication after training. The resu-ltant graph obtained by just sending an image for communication and there by attacked by an adversarial network Eve shows here defeat of the attacking network to decrypt the communication even after getting trained of the decryption process. This shows that encryption of Images and then decryption of that by Neural networ-ks in Python can be successfully done for future.

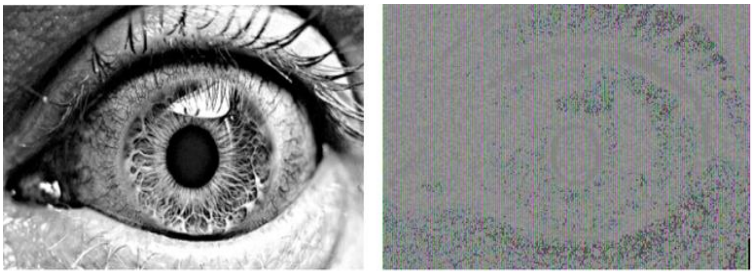

Fig4: First one is the Message i.e. an image of an eye and second is its encrypted message format.

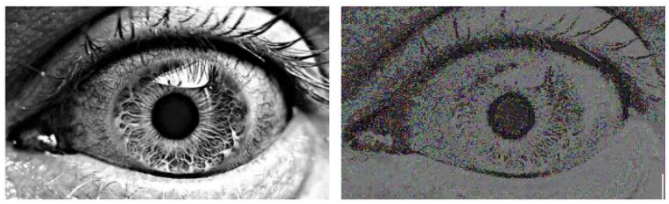

Fig5: First one is the decrypted message of the Fig4 with correct key and the second is the Decryption of same image with incorrect key

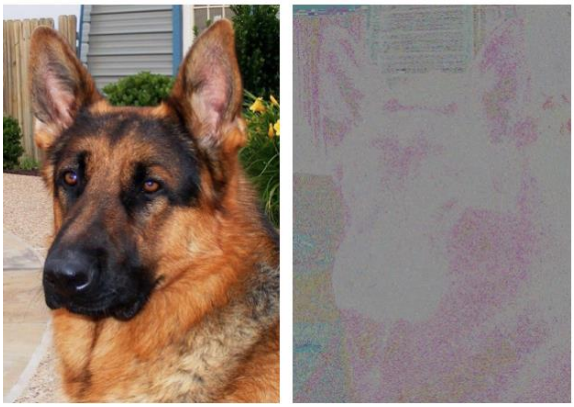

Fig6: First one is the Message i.e. an image of a dog and second is its encrypted message form

Fig5: First one is the decrypted message of the Fig6 with correct key and the second is the Decryption of same image with incorrect key

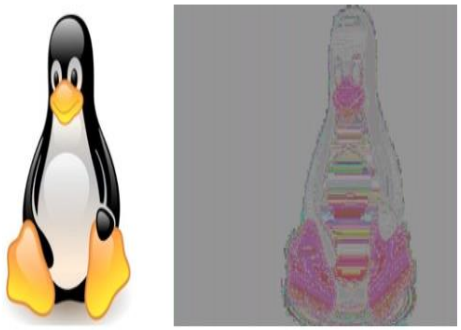

Fig7: First one is the Message i.e. an image of a Linux and second is its encrypted message form

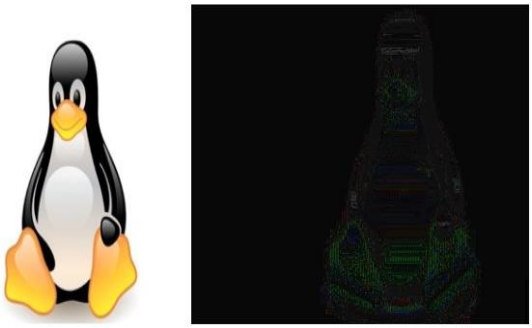

Fig8: First one is the decrypted message of the Fig7 with correct key and the second is the Decryption of same image with incorrect key

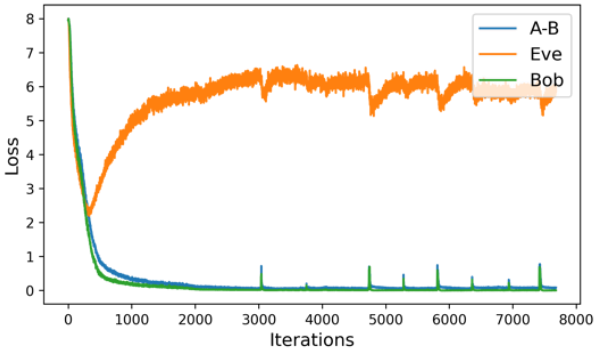

Fig9: This is image of Alice- Bob and Eve after they are trained and when they are finally given an image for encryption and decryption.

4. Conclusion

In this paper we analyzed a Neural Networks algorithm for encryption and decryption of images that exhibits qualities for using it in near future for protection of our communications, as well as it will be a key component of encryption methods for some applications. On the contrary Cryptography is vast than encryption. In the same way for the task of encryption and decryption message work may be considered in this feature. Again this could be used in other mode of plaintext or message for communication. To concl-

ude this method is useful in neural network for cryptographic protection and for the protection from attacker. It may be effective in making sense of metadata and in traffic analysis. The neural network would be perfect at cryptanalysis.

Reference

[1] Martín, Andersen, and D. G., ""Learning to Protect Communications with Adversarial Neural Cryptography," *arXiv.org*, 21-Oct-2016. [Online]. Available: https://arxiv.org/abs/1610.06918. [Accessed: 13-May-2020].

[2] "Neural cryptography," *Wikipedia*, 16-Apr-2020. [Online]. Available: https://en.wikipedia.org/wiki/Neural_cryptography. [Accessed: 13-May-2020].