Neural Cryptography: From Symmetric Encryption to Adversarial Steganography

Dylan Modesitt, Tim Henry, Jon Coden, and Rachel Lathe

***Abstract*— Neural Cryptography is an emergent field that aims to combine cryptography with Neural Networks for applications in cryptanalysis and encryption. In this paper, we (1) show Neural Networks are capable of performing symmetric encryption in an adversarial setting and improve on the known literature on this topic. We also (2) show that Neural Networks are capable of detecting known cryptographically insecure communication by having them play known cryptographic games based on Ciphertext Indistinguishability. Finally, we (3) present further research in Neural Steganography in the context of developing neural end-to-end stegonographic (image-in-image, text-in-image, video-in-video) algorithms in the presence of adversarial networks attempting to censor.**

I. Symmetric Encryption

Symmetric encryption is a form of encryption in which the sender and receiver use the same key to encrypt a plaintext and decrypt the corresponding cyphertext. Traditionally, symmetric encryption algorithms have used either block or stream ciphers. However, it has been demonstrated that in a system of neural networks, with end-to-end adversarial training, they can learn how to perform forms of 'encryption' and 'decryption' without the use of a specific cryptographic algorithm [1].

A. *Prior Work*

The work done by Adabi and Andersen can be summarized as follows. They create three neural agents: Alice, Bob, and Eve. Alice receives as input a plaintext in the form of a fixed length bitstring, P, as well as a private key K. In practice, these are the same length though hypothetically this is not required. Bob receives the output of Alice, C, as well as the private key and is expected to produce the original message P. Eve is another network that receives Alice's output C, but does not receive the private key. The architecture of these individual networks consist each of a single feed forward layer with no bias and σ activation, being followed by four 1-dimensional convolutions with $tanh$ activations, presumably with the intention to diffuse. The networks are then trained in an adversarial fashion with the following constructions for loss (where d is the $L1$ norm):

$$L_E(\Theta_A, \Theta_E, P, K) := d(P, E(\Theta_E, A(\Theta_A, P, K)))$$

$$L_B(\Theta_A, \Theta_B, P, K) := d(P, E(\Theta_B, A(\Theta_A, P, K), K))$$

$$L_{AB}(\Theta_A, \Theta_B, P, K) := L_B(\cdots) - L_E(\cdots)$$

where the training goal is to choose the optimal Θ_A, Θ_B such that L_{AB} is minimized.

This construction was trained in rounds trading off freezing Bob/Eve and unfreezing Eve/Bob, and it showed to be relatively sturdy to decryption from Eve while Bob was able to learn.

The researchers showed that the networks did not learn XOR for doing a sort of one-time pad, but rather some other hard-to-invert function. Evidence of this is found in that "a single-bit flip in the key typically induces significant changes in three to six of the 16 elements in the ciphertext, and smaller changes in other elements". As we will describe, calling these operations 'encryption' is quite misleading.

1) Replication: In replicating the paper's findings, we found somewhat poorer results than described, as well as design decision that lend themselves to obfuscation rather than encryption. Both implementations by others [2], as well as our own Keras implementation, performed worse than the original paper and converged less frequently than the stated $1/2$ or $1/3$. We had to make minor deviations from the original training schedule, such as slightly pre-fitting the Alice-Bob network before

*This work was done as the final project to 6.857: Computer and Network Security at the Massachusetts Institute of Technology

adversarial iterations, to get convergence at all. We also had to iterate more frequently between networks, as too long between switching led to much more chaotic results.

Furthermore, the original paper considers the scheme to be ‘secure’ against the adversary if newly initialized Eve(s) can not converge within 5 epochs. However, we believe that this condition is quite insufficient, as in our experience, simply giving Eve more time often resulted in her recovering a significant number of bits.

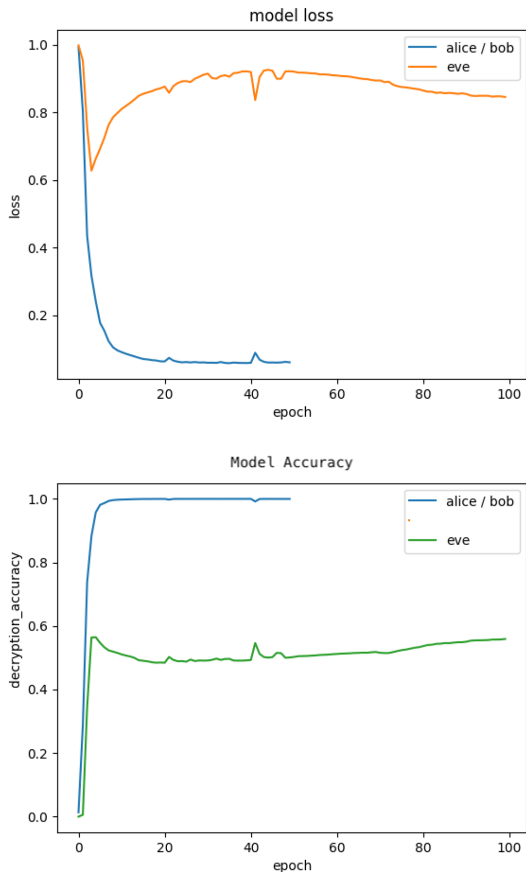

Fig. 1. MSE Loss and Decryption Accuracy in a Successful Convergence

Above are two examples of training where Eve does not gain any accuracy advantage after post-fitting, and one where she learns to decrypt nearly 70% of the plaintext. Both experiments ran with an additional 50 epochs for Eve to post-fit. The prior happened roughly only 1 in 6 trails, despite normal vs xavier initialization.

At the end of the day, we do think these results are interesting but do have some fundamental

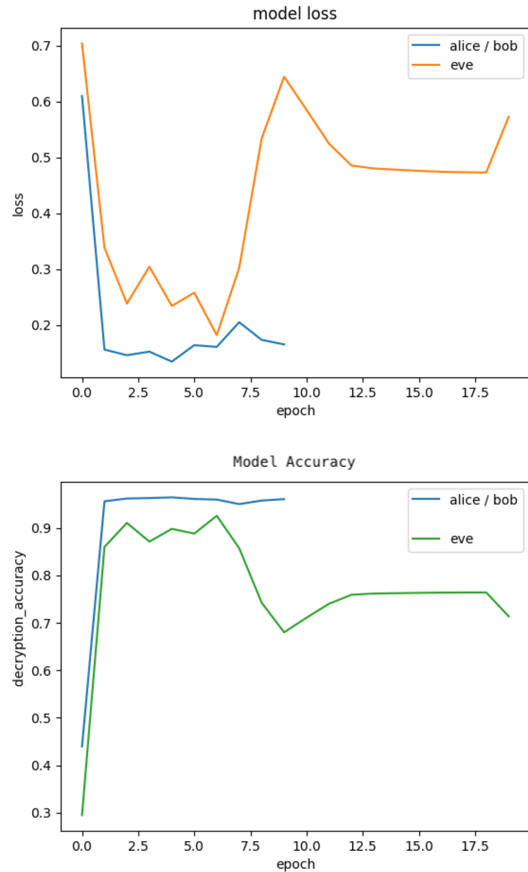

Fig. 2. MSE Loss and Decryption Accuracy in an Un-Successful Convergence

flaws. Firstly, the outputs of the networks are continuous floating point numbers. Thus, C, P_b, and P_e can technically be any floats within the range of the final activation. This turns out to be a huge problem, as we often observed that C, even during a convergence where Eve does not decrypt, simply performs a sort of mapping to arbitrary values between $-1, 1$, effectivly increasing the ‘base’ of the encoded image. The specifics of the encryption and decryption operations are hard to discern, but it surely seems that Alice and Bob learn more of an obfuscation than a method of encryption despite the encryption being ‘key dependent’.

Furthermore, we believe that there are several quite unfair assumptions made here that make the task easier for Alice and Bob than Eve beyond the absence of the key. The condition that Eve needs to decrypt the ciphertext is absurdly stringent and does not lend itself well to any reasonable definition for being secure, but rather encourages

obfuscation.

B. Ciphertext Indistinguishability for Eve

Instead of having Eve produce the plaintext, P, without the key K, from the ciphertext C, we think a better construction would be to ask Eve given a plaintext, P and a possible ciphertext C', does $C' = Alice(P)$? This lends itself to a more natural GAN description where Eve acts as a discriminator. The networks can continue to be trained adversarially where Eve's loss is now binary cross-entropy on the truth label:

$$L_E := -\sum_i (y_i' \log(y_i) + (1 - y_i') \log(1 - y_i))$$

This architecture, visually, looks as follows:

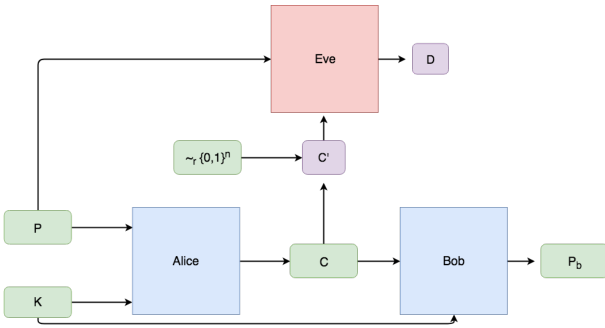

Fig. 3. True GAN setup where Eve judges Ciphertext Indistinguishability

Other formulations, such as given two plaintexts, P_1 and P_2, and two ciphertexts, C_1, and C_2, is $Alice(P_1) = C_1 \wedge Alice(P_2) = C_2$ or is $Alice(P_1) = C_2 \wedge Alice(P_2) = C_1$, could be the subject of further exploration; however, we did not want to over-complicate training without a noticeable logical benefit to the reconstruction of the game rules.

With the simpler formulation, Alice and Bob are trained with the added loss term, beyond the MSE of the decryption on the plaintext, of the binary crossentropy into the discriminator (frozen) of labels that suggest the ciphertext is not the encryption of the plaintext, in standard GAN fashion. Alternatively, we considered the loss of

$$\gamma * 0.5 - \frac{sum_i |y_i|}{n}$$

to encourage uniformly distributed predictions from the discriminator. This lightens the incentive for Alice and Bob to get higher than 50% false negatives, which can only last temporarily. We found that the latter loss term helped in earlier convergence, but could cause stability issues at converged values where the discriminator chooses to predict values all of a single class. Thus, modifying the gamma factor during training helped improve this stability.

We additionally found that using this criteria for Eve resulted in a 0% success rate out of 100 trials with the network architectures described above. Eve always obtains nearly 100% accuracy with the above metric and with all other network architecture and training procedures remaining the same.

Furthermore, we attempted to weaken Eve slightly by adding Gaussian Noise to the false Ciphertexts as to make them look more continuous, given the convergence pattern we described. However, any remotely reasonable level of noise ($[0 - 0.5]$) std did not hinder Eve almost at all. Thus, this seemed to be a complete failure.

C. Encouraging Element-Wise Operations

In order to encourage element-wise operations, rather than relying on a series of fully connected layers and convolutions to learn some operation, we construct a network architecture based on the following: given two inputs of the same length, learn only a function that is element-wise of those inputs producing a same-length output. Specifically, inputs i and j have enumerated entries $i_{1...n} \wedge j_{1...n}$. Each i_k, j_k are two inputs to a small feed forward network of its own construction, with variable numbers and sizes of hidden layers and variable activations. This feed forward network could enforced be the same for all elements, or it could be an entirely separate kernel. Our custom element-wise layer in Keras that can handle arbitrary network depths, sizes, activations, and kernel configurations. The following figure diagrams this network architecture.

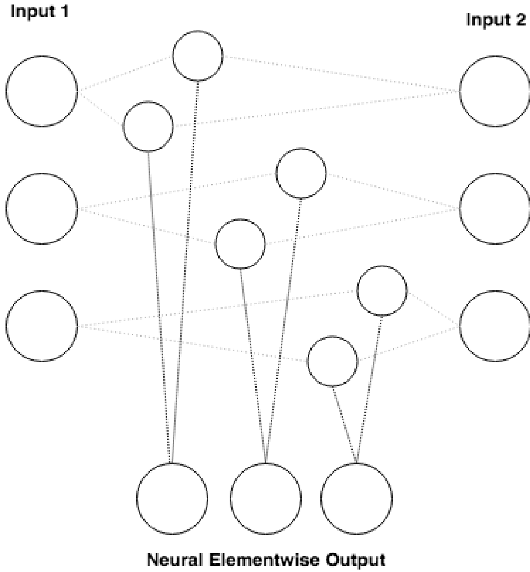

Fig. 4. Element-Wise Architecture Diagram with two 3-Element Inputs

With this network design, we hope to be capable of learning bit-wise functions, specifically xor and $nxor$. To verify, as a proof of concept, we verified that this network architecture is capable of learning sequence to sequence xor with nearly perfect accuracy after a small number of steps.

We use this new layer construction to completely change the network architecture of Alice, Bob, and Eve. Alice and Bob both get two bitwise network layers, one with n hidden layers and one with 1 to produce a valid size result. The activation on the first layer should be something non vertically-symmetrical, as to not require those functions where $[0, 1]$ and $[1, 0]$ can not map to the same thing. Otherwise, without biase, it is impossible to have a function that does the contrary like xor. $ReLu$ showed to be a reasonable candidate. The final activation need be something that can produce valid bits. In our experimentation, we found that only two options alleviated the problem of mid-range floating point mapping. We could either add an additional loss term to Alice's output that punished deviation from one of the bit values, 1 or -1. This loss term was

$$\lambda(1 - \frac{\sum_i |y_i|}{n})$$

where λ could help weight this term. This felt like holding the hand of the activation, and also complicates the loss. Alternatively, we used the following 'hard tanh' function.

$$htanh(x) = 2 * max(0, min(1, x_i * 0.2 + 0.5)) - 1$$

based of the known 'hard sigmoid' function. We found this activation was an excellent choice to remove any ambiguity over floating point results. Furthermore, when placed on both Alice and Bob, floating point results for ciphertext or recovered plaintext were practically eliminated. Eve was given a larger network, where she had an additional intermediate bitwise layer and additional dense layers after the bitwise later, all with significantly more units. We think this raw parameter advantage helps aleviate weak eve concerns.

D. Results and Analysis

This problem is now essentially reduced to attempting to get the networks of Alice and Bob to learn xor or nxor implicitly, in an adversarial setting, when it is the best choice to secure communication. We found that this does indeed happen, but with several caveats.

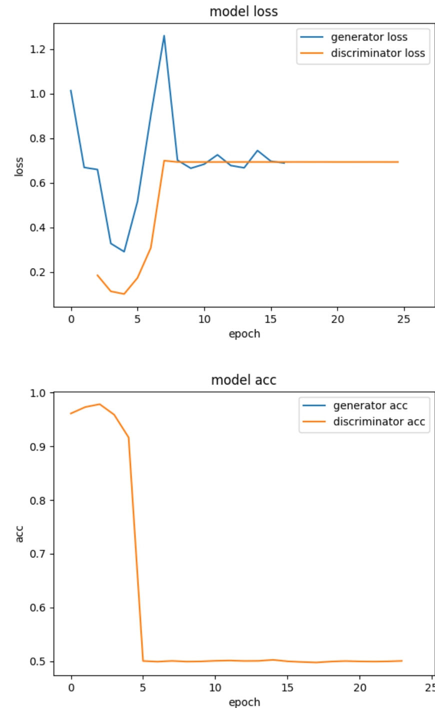

Fig. 5. Sucessfull element-wise hard bit output conversion to the one time pad. The above shows overlaid loss (with false label bcc on the generator), and the below shoes the accuracy of the discriminator. Results were confirmed to be $nxor$

Oftentimes, instead of learning the highly non-linear xor and nxor, the networks choose to fall into a shortcut. Sometimes, it would ignore adversarial loss entirely, and learn the identify function and be unable to escape this local minimum. This was rather rare though. More often, the network would decide to learn a function like $nand$ or or, which can produce 75% reproduction accuracy, but lead to less discriminator accuracy near 60%.

We found that $xor/nxor$ was learned implicitly roughly only $1/12$ times. These odds can be raised by forcing $\Theta_a = \Theta_b$ and using the same weights, independent of index. However, this is beyond hand holding. Thus, initialization and optimization technique plays a huge role in what approach the network takes. We found no standard initialization that provided better results. Furthermore, more 'sophisticated' optimizers, like Adam, unsurprisingly yielded the best results.

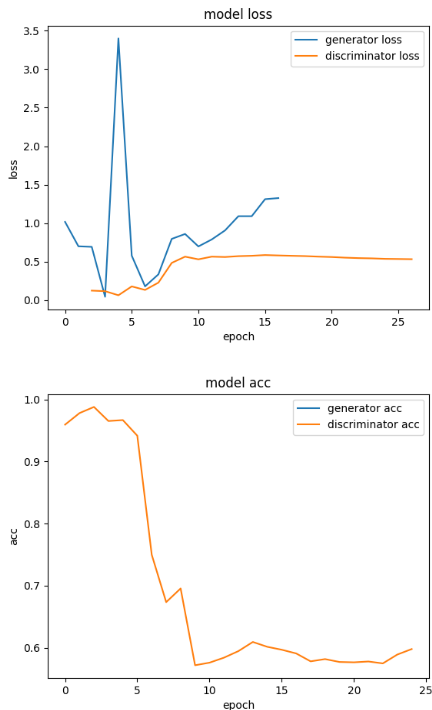

Fig. 6. Local optimum learning of $nand$ for securing communications

Finally, we did experiment with using BNNs [4]. However, we had little success. Perhaps further research into this formulation could be interesting. Also, it may be the case that more 'curiosity' based optimization methods with Reinforcement Learning could provide more success here; however, we feel that the already drastically instable setting of this 3 Agent GAN combined with additional in-stable RL methods like Q-learning and Policy Gradients would be a true recipe for disaster.

However, an additional take-away helped us look at a new approach to 'secure' communication. Alice and Bob tended to perform obfuscation rather than encryption. So instead of forcing the problem of having networks re-discover the one-time pad, we wanted to embrace the strengths of Deep Learning and perhaps provide more useful applications within the realm of secure communication. For this reason, we decided to explore Neural Steganography, an also new (but perhaps more useful), topic in Deep Learning, and did so with much success.

II. Introduction to Steganography

Steganography is the practice of hiding data within data, oftentimes in plain sight where the human eye cannot detect that any secret might be concealed. This is different than most cryptographic practices where encryption is the focal point of protecting the secret; however, steganography is powerful practice that can be used to fool more passive observers. If an observer doesn't think that any secret is present they won't take any actions to alter or block the secret message from reaching the intended observer. In this sense, there are two parts in steganography, first there is the secret which is message that is hidden and the cover, the data that is supposed to hide the secret from an observer. This means that steganography can be used to hide secrets in a myriad of data types if there are techniques to obfuscate the data.

The first mention of steganography can be dated back to 1499 Johannes Trithemius' book *Steganographia* which appeared to be a book about using spells and magic to communicate across long distances, but in reality was a book dedicated to early cryptographic and steganographic pratices. It only took scholars over a hundred years to actually figure out the hidden meaning behind Trithemius' work, and since then steganography has evolved to allow people to hide secrets in all types of data from text in images to video-in-video. In this part of the paper we will be focusing

on hiding text in images, images in images, and video-in-video using adversarial neural networks to devise a hiding scheme that can avoid a passive observer. However, in order to validate that our neural network can create a sophisticated scheme that is capable of avoiding detection from most observers, we first need to construct an adversary that has learned how to detect the most common steganographic practices of hiding data in images.

III. Goals

We present several neural applications to Steganography, with the ultimate goal of having Neural Networks perform stenographic algorithms on different data both with the standard steganographic goals (hidden secret looks like the cover and the reconstructed secret is accurate), as well as being undetectable to an adversary looking for steganography. We think this serves as a good architectural candidate for censorship-resistant steganography among other applications. However, before presenting such a construction, we attempt to create an adversary that can detect (perform a steganalysis) on the most common deterministic Steganography algorithm.

IV. Least Significant Bit

Least Significant Bit (LSB) is the most common technique employed by practitioners intent on hiding binary data within binary data. The LSB scheme is quite simple to employ. We will explain this in the context of hiding text in an image. For every pixel in an image there is an associated (r,g,b) tuple, which defines the color of the pixel. Each of these entries are byte and range in decimal 0 to 255. In order to encode some text you first convert the text value to binary, i.e. a $\rightarrow$ 0110 0001. Now, to encode the information match the parity of the R, G, or B value to the parity associated with 0 or 1 in the binary string representation of the character, i.e. the least significant bit of the color. In order to best demonstrate this, we will encode the character "a" in a sample 4×4 image.

We start with our binary string representation of "a", 0110 0001, and a matrix of (R,G,B) triples each representing a pixel in an image.

$$\begin{bmatrix} (13, 32, 128) & (96, 47, 26) \\ (211, 5, 44) & (69, 17, 200) \end{bmatrix}$$

We start now with the highest order bit of "a" and the R value of the pixel in the top left corner. The highest order bit in "a" is 0 and the R value of the pixel in the top left corner is 13, therefore we want to change the parity of this value to match that of the bit, so we change the R value to 14. The (R,G,B) values of each picture look like this after this initial update step.

$$\begin{bmatrix} (\mathbf{14}, 32, 128) & (96, 47, 26) \\ (211, 5, 44) & (69, 17, 200) \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} (14, \mathbf{33}, 128) & (96, 47, 26) \\ (211, 5, 44) & (69, 17, 200) \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} (\mathbf{14,33,129}) & (\mathbf{96,48,26}) \\ (\mathbf{212,5}, 44) & (69, 17, 200) \end{bmatrix}$$

The values in bold in our final image represent the bits that we've used to encode the character "a". Notice, some of the bits haven't changed from the original image because they already matched the parity of the relevant bit.

Note that in LSB, If someone were to look at the (R,G,B) values for each pixel in an image without a secret and then convert the parity of those values to bits, the resulting bitstring would be fairly random. If the encoded secret is random, it would be exceedingly difficult for a passive observer to detect if there is an underlying secret or not; however, if an observer can detect any sort of pattern in what is the binary string, then the observer should be able to detect if there is some information being encoded. Therefore, in order to see if a neural network could actually create a useful steganalysis adversary we wanted to arm our adversary with the ability to detect LSB.

V. Constructing an Adversary to Detect Least Significant Bit

Neural networks excel at learning to detect statistical patterns, as they serve as a good universal function approximator, which is why it makes sense that creating an adversary to detect text-in-image LSB encoding should be possible. Obviously, the adversary will not be able to detect all types of LSB encoding as things like encrypted

text will appear random; however, the adversary should be able to detect valid ascii characters encoded as plaintext into an image. The adversary can do this because the binary representation of ASCII characters is not random, nor are valid English words random. For example, lowercase letters begins with 011 and uppercase letters begins with 010, so our neural network should be able to detect if there is some pattern if those three binary values are consistently showing up in order.

Next we need to define what kind of game we want our adversary to play. We want the adversary to detect if there is secret text encoded within the image. The adversary does not need to detect and reveal what the secret is, but rather have confidence that the pixel values are not random. Therefore, our game will be seeing if the adversary can detect if there is a secret encoded within the image with better than 50% accuracy. We chose this game because when LSB is put into practice, it's to avoid the detection of any secret being transmitted from one party to another. Therefore, if an observer detects the possible presence of a secret, it is often just as bad as the secret itself being revealed.

A. *Data and Network Construction*

The first step in defining how we were able to construct this network is to talk about the data set that we used. We used approximately 50,000 32×32 images from ImageNet and drew from a dictionary of 10,000 words which were then randomly chosen to be embedded into images using Least Significant Bit.

The adversary is then trying to determine if there is a hidden message in a given image. We find that CNNs make little sense to use here. The input for the CNN will be scaled down, making the individual pixel values (and, importantly, their parity) very obfuscated. Thus, steganographic encryptions that rely on subtle pixel value changes will be difficult or impossible to capture. These assumptions hold in our experimentation as we were unable to have a CNN converge to detect LSB steganography. Rather, we want to have the network learn some mapping to particular pixel values [0-255] and what those values mean (parity, for example).

The first step in figuring out this mapping is to use a flatten layer, which reduces a 32×32 image to a list of 3072 (R,G,B) values.

$$\begin{bmatrix} (13, 32, 128) & (96, 47, 26) \\ (211, 5, 44) & (69, 17, 200) \end{bmatrix}$$
$$\downarrow$$
$$[\ 13,\ 32,\ 128,\ 96,\ 47,\ 26,\ 211,\ 5,\ 44,\ 69,\ 17,\ 200]$$

Now that we've flattened our image, we can more easily find a mapping for each pixel value. We use an embedding layer, usually used to translate word encodings to a vector describing their meaning, but in this case the embedding layer will be mapping pixel values to their associated parity. The embedding layer has an input dimension of 256 because the (R,G,B) range from 0 to 255. This kind of 'pixel embedding' trick helps us detect steganographic encryptions that produce pixel-based patterns in the image. LSB and many other stenographic algorithms fall under this umbrella.

B. *Results and Analysis*

We used standard supervised learning techniques to train the LSB adversary. We measured our loss in terms of binary cross entropy and trained our LSB adversary for 50 epochs with a batch size of 32 in each epoch to ensure that our adversary adequately converged to being able to detect LSB. The following graphs indicate the loss over these iterations.

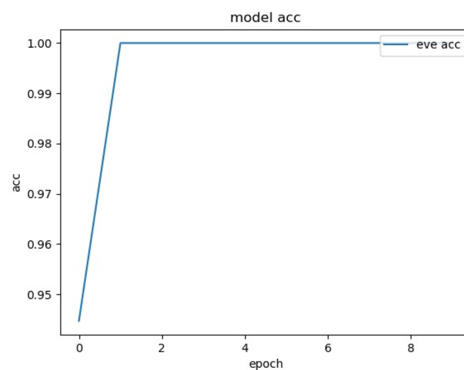

Fig. 7. Training Loss of Least Significant Bit Adversary

As we can see in our results, the adversary was able to perform quite strongly and learn to detect if there was an LSB encoding within a image quite quickly. This demonstrates that this network will make for a worthy adversary for

our Encoding Network in order to elicit a strong encoding scheme.

VI. Teaching Neural Networks to Hide Data in Data

Now that we have a sufficiently strong adversary, we want to see if we can design a network that will be able to hide a secret in plainsight in the presence of this adversary. We explore neural approaches for text in image, image in image, and video in video. Furthermore, we use roughly the same network for all pairings of data.

We again used about 50,000 32×32 images from ImageNet for image secretes and covers, about 50 downsized 32×32 30 frames per second videos for the video secrets and covers, and roughly 10,000 valid English words. For training and testing we randomly sampled from these datasets.

We will refer to the LSB adversary that we created in the previous section as our Censor, which we will train adversarially against our Encoding Network. The Encoding Network is the network that will be learning how to hide secrets in the covers.

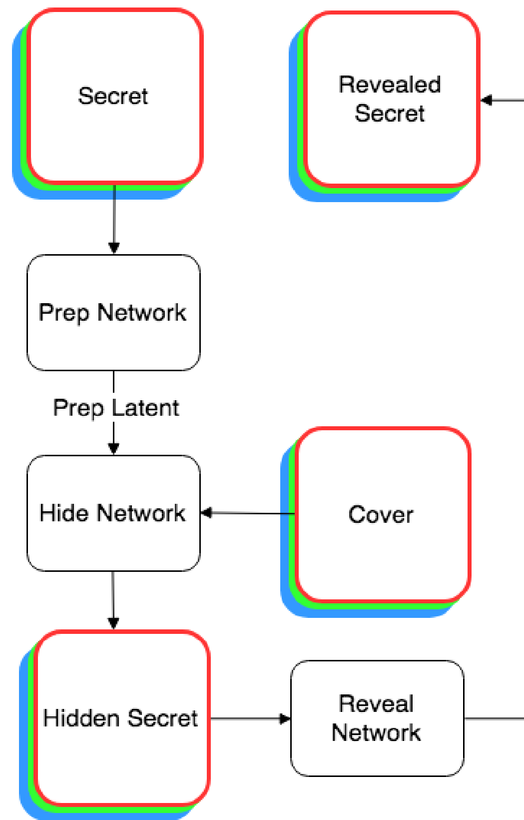

Fig. 8. High-Level Representation of the Encoding Network

A. *Encoding Network Construction*

From a high level our Encoding Network, an adversarial neural network, is comprised of three networks working together against the Censor. The three networks are the Prep Network, the Hide Network, and the Reveal Network. The architecture and design of these three is inspired by the previous work by Baluja (2017) [3]. The Prep Network prepares the Secret to be hidden in the Cover, while the Hide Network then hides the Secret within the Cover. Finally, the Reveal Network exists to make sure that the Hidden Secret can be reconstructed after being hidden. This last part is extremely important as a steganographic model would be useless if the receiver could not read the message. The following graphic depicts how the three networks interact with one another.

While each of the networks has a different function, they all have a similar structure. Each network takes in an input which is an image along with identifying information in the form (height, width, channels). Channels represents the number of values associated with a given pixel. When hiding text in an image, we can represent the text in its binary form with each pixel only having one value representing it (0 or 1) instead of three values for standard color images. With images and videos we set the channel value to 3 because for each pixel there is an associated (R,G,B) triple.

In each network, if it's necessary, we concatenate the output of the previous network with the input of the next network. Each network is constructed from three modules, the Input Module, Output Module, and the Convolutional Module. This construction can be seen in the figure below.

Let's first look at the Convolutional Module. This block is made up of three sets of four convolutional layers One set of convolutional layers has a 3×3 window size, the second set has a window size of 4×4, and the last set of convolutional layers has a window size of 5×5. Each convolutional layer has a stride length matching that of its window size. All three sets of convolutional layer sequences operate independently until they are concatenated. This means that the different sequences of layers are able to capture information

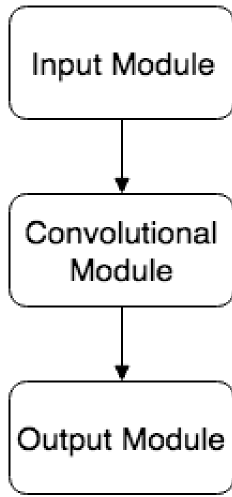

Fig. 9. Prep, Hide, and Reveal Network Module Representation

about the image at different scales. The 3×3 convolutional layers will capture small details, while the the 4×4 layers will capture the relationship and interaction of those small regions, and finally the 5×5 layers can capture the big picture of the image. We demonstrate this architecture in the following figure. The Input Module function in two

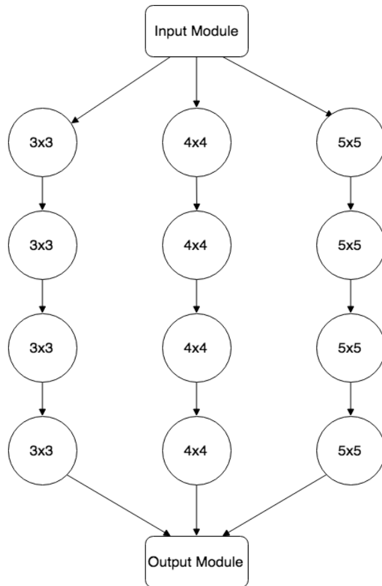

Fig. 10. Convolutional Module Representation

ways. In one instance, like with the Prep Network, the Input Module is simply one layer that takes in the input and then passes it to the Convolutional Module. However, in the Hide Network the Input Module concatenates the two inputs, the prep latent and the cover. The Output Module is actually made up of three sets of layers. First, the Output Module concatenates the outputs from the three sets of convolutional layers. Then it passes this output through one round of 3×3, 4×4, and 5×5 convolutional layers and then finally concatenates the result. This construction can be seen in the figure below.

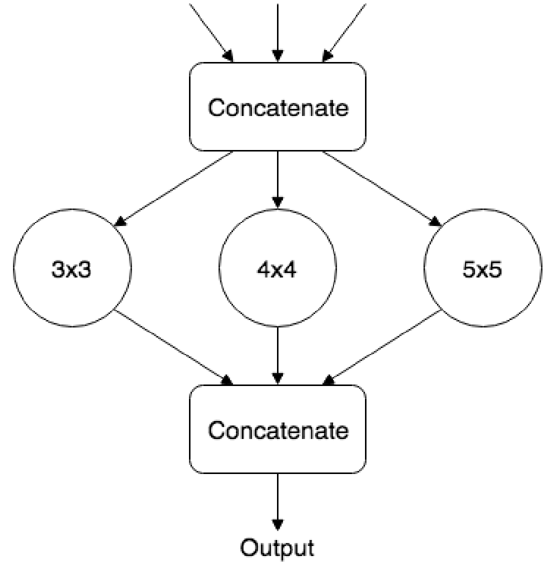

Fig. 11. Output Module Representation

Now we can return to our overall construction. The Prep Network creates the Prep Latent, which is the what the network learns to be the best mapping from the Secret to the relevant pixels needed to encode the Secret in the Cover. Next, we concatenate the prepared Secret with the Cover, and pass that through the Hide Network in order to finally create the Hidden Secret. The power of neural networks is most evident here because the Encoding Network learns the best mapping to hide the Secret within the Cover, regardless of how complicated the scheme would be to devise manually.

The final step is to pass the Hidden Secret into the Reveal Network to make sure that the Secret can actually be recreated by the receiver. If we train this entire network adversarially against the Censor, we can create a sophisticated network capable of avoiding detection from a strong adversary.

In order to train this network we first define the loss function:

$$L(C, HS, S, RS) = \alpha(C - HS)^2 + \beta(S - RS)^2$$

where C is the Cover, HS is the Hidden Secret, S is the Secret, and RS is the Revealed Secret. $\beta = 0.75$ in practice. Then we train the networks as follows.

- 2 epochs where $\alpha = 0$
- 10 epochs with respect to $\alpha = 1$
- 5 epochs of transfer-learning to the Censor

The final result is a network architecture that can successfully hide text in images, images in images, and video in video. However, what is even more astounding is that we are in fact including 2 bits in every Hidden Secret image or video for every 1 bit that exists in the Cover.

B. *Hiding Images in Images*

Encoding images in images is the main goal of our network architecture. The main difference in how this is executed is that we have to do less preprocessing of the Secret unlike when we hide text in images. Furthermore, each pixel in the secret image is represented with three (R, G, B) values, so we need to make sure our network knows to use three channels for the image. As we can see in the graph below, our network performed strongly at hiding the Secret well and accurately decoding the Hidden Secret.

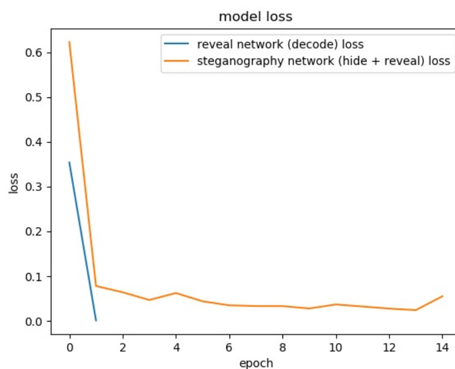

Fig. 12. Encoding Network Loss for Hiding Images in Images

Now we look at how our Censor performed in detecting if there was any encoding in the image. Our Censor was the LSB adversary that we constructed before; however, even though it was adept at identifying LSB encoding for text in image, the Encoding Network proved to be too difficult to crack. We see in the graph below that our loss hovered around 0.7, a poor value for such a loss metric.

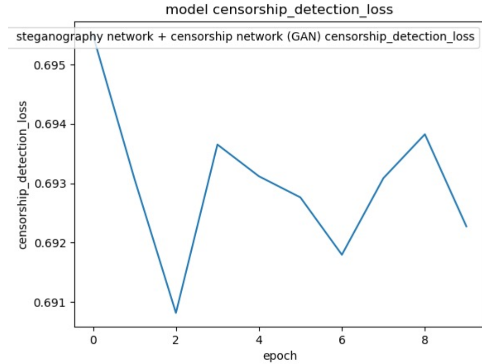

Fig. 13. Censor Loss for Hiding Images in Images

The poor performance of the Censor in this case can be attributed to two reasons. The first being that our Encoding Network created some random looking mapping of pixels that enabled it to evade any sort of pattern recognition that the Censor was able to detect. The second possibility is that while our Censor is good at detecting text in images, there is inherently more randomness in how images are encoded than text.

C. *Hiding Video in Video*

As we have a good image in image network, hiding video in video is a task easily done by just taking every pair of video frames for the Cover and Secret and using the Encoding Network to produce the Hidden secret and latter the reveal. The figure below shows subsequent frames with this method.

Furthermore, we have developed a web application so users can do this with their own content, and we plan to make it publicly available in the near future.

D. *Hiding Text in Images*

Finally, we wanted to test how well our network could hide text in images. We formatted the text as an image (a matrix) with one channel where each 'pixel' value is binary value corresponding to the binary representation of a character. While the Censor was able to easily detect text in image encoding based on LSB, our Encoding Network

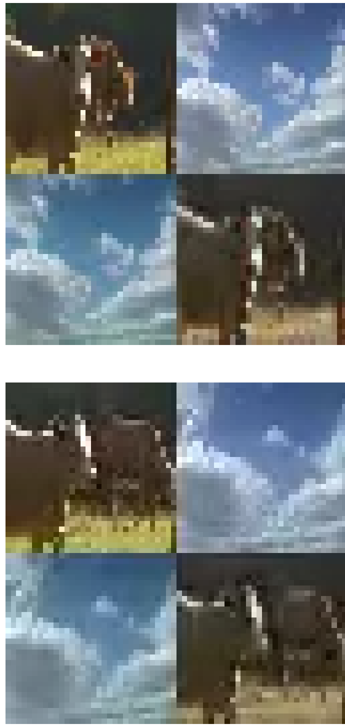

Fig. 14. Two frames of a secret video (top left) and cover video (top right), as well as their hidden secret (bottom left) and revealed secret (bottom right) counterparts.

was able to create a novel encoding scheme that was able to accurately hide and reveal the secret text. We can see these results in the figures below.

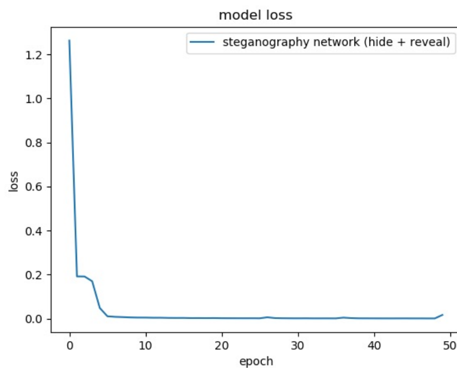

Fig. 15. Encoding Network Loss for Hiding Text in Images

VII. Further Research and Potential Applications

The possibilities for further research and applications of this topic are immense in certain aspects and useless in others. Steganography and security practices in general are far older than computing and machine learning, yet we see the massive potential that the union between these two subjects could bring.

Our initial motivation was inspired by our TA and project advisor Jonathan Frankle who talked to us about how in countries with totalitarian regimes it can be difficult for citizens to discuss dissenting opinions online or to organize demonstrations or protests due to the ever watching government. Steganography can be utilized by these citizens to evade the watchful eye of the government without raising suspicion; however, with the continued improvements in computing, this becomes more difficult. Using a network like the one we've proposed here it could be possible to train our network against an adversary that mimics the censor in question, like we did with our LSB adversary, in order to ensure that people would be able to freely discuss their opinions. This application is contingent on being able to replicate a strong enough adversary because if our adversary is too weak the resulting encoding will be easily detected by a stronger adversary.

However, we think that the question of whether neural networks can perform encryption among themselves is trickier than it might seem. In order to gain a satisfactory outcome, much augmentation and complication needs to be introduced, taking away from the elegance of the original idea 'Neural Networks learning to encrypt'. Our approach narrowed the problem to be essentially, how hard is it to learn XOR in this loss space. Beyond the point of doing it to do it, we see little future application of this technology; however, we would be happy to see ourselves proved wrong.

APPENDIX

A. Open Source Software

You can visit all the code we made for our project https://github.com/DylanModesitt/neural-cryptography. Everything was written in python 3.6 with 3.7 __future__. We used Keras with a TF backend. Further setup instructions are described in the ReadMe. We hope you like it!

B. DES ECB Cryptanalysis

Additionally, we attempted to expose the weakness of DES ECB mode with a single key by utilizing neural networks. As we learned in 6.857, DES ECB is not CCA secure (an adversary can always pad an input message with an additional block and send it to the encryption oracle, and just compare all but the last block of the result with the ciphertext to win the CCA game).

Thus, we attempted to set up a form of an indistinguishability game, and create a neural network adversary to try to learn an advantage. A major constraint in creating a CCA-like game within a neural network structure is that DES ECB has permutations and s-boxes that are not differentiable which prevents us from giving our adversary encryption oracle or decryption oracle access.

Instead, we created a CPA-like game that presented our adversary with one plaintext and one ciphertext, and challenged it to determine whether or not the ciphertext was the encryption of the plaintext or just random bits. Our implementation was first tested by using DES ECB with one round (which effectively does not alter half of the plaintext) and observing that our adversary quickly converged to 100% accuracy. Ultimately, however, we were not able to find a network architecture that gave our adversary an advantage in this game, even with 2-round DES ECB encryption. This is a testament to the ”randomness” of DES.

C. Steganography Keras Model Diagram

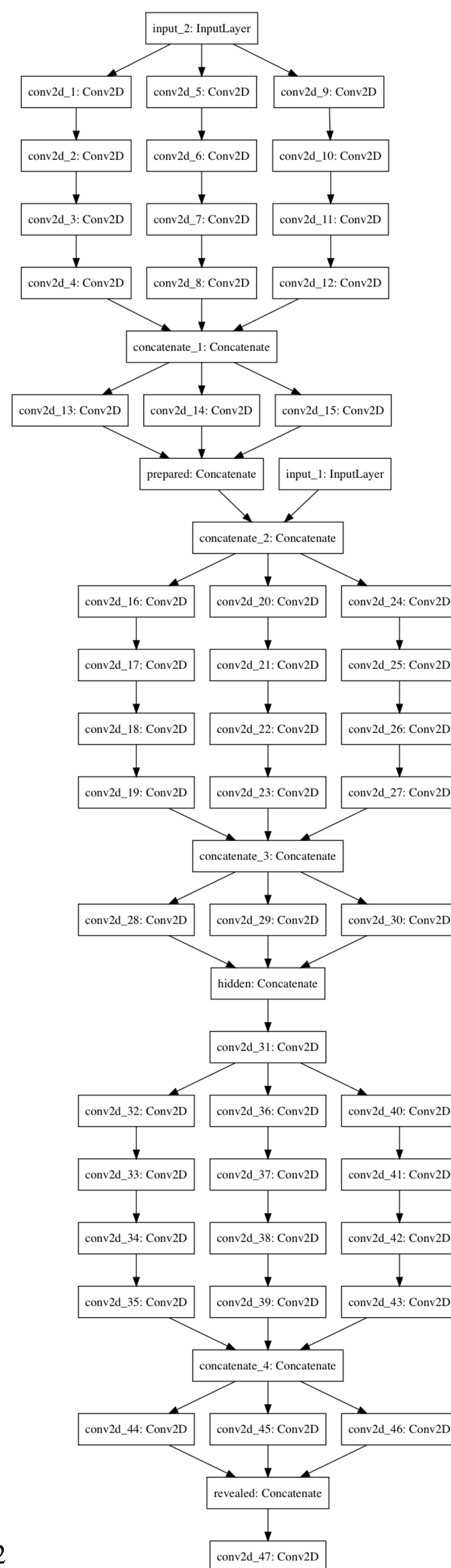

Fig. 16. Detailed Representation of the Full Encoding Network

ACKNOWLEDGMENT

REFERENCES

[1] M. Abadi and D. G. Andersen. Learning to Protect Communications with Adversarial Neural Cryptography. *ArXiv e-prints*, October 2016.

[2] ankeshanand. Adversarial Neural Cryptography in TensorFlow, 2015.

[3] Shumeet Baluja. Hiding images in plain sight: Deep steganography. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2069–2079. Curran Associates, Inc., 2017.

[4] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.