

Hiding in Plain Strokes: Handwriting and Applications to Steganography

James Hahn
University of Pittsburgh
Pittsburgh, PA 15260
jamhahn@yahoo.com

Adriana Kovashka
University of Pittsburgh
Pittsburgh, PA 15260
kovashka@cs.pitt.edu

1. Introduction

Steganography, the process of hiding secret information within normal-looking data (a cover), lies at the intersection of computer science and security. Recently, covers have come in the form of images [2]. Standard methods consider fully saturated (dense) images, which simplifies the encoding and decoding tasks. First, slight modifications of pixel intensities (needed to encode a message) are imperceptible when the image is dense, but easy to detect if an image is sparse (e.g. text on a plain white background). Further, if the message being encoded is dense too (e.g. also an image), slight decoding mistakes are not important, because a human viewer can still correctly perceive the meaning.

We consider a complementary and more challenging scenario: encoding a sparse message into a sparse cover. We consider a sequence of handwriting strokes as the cover, and embed a secret message by altering each coordinate in the cover, rather than superimposing a new signal (message).

The stages of our approach are as follows. First, the secret information is encrypted before combining it with the cover using neural cryptography. Second, the information is hidden in the stroke data of a person's handwriting on a white background. We utilize coordinate sequences of human handwriting, modified with slight offsets dependent on the secret information being encoded. The new generated coordinates are nearly identical to the original ones, preserving the primary structure of the handwriting. Our method thus explores a new avenue of steganography based on data *modification* rather than *embedding*.

2. Handwriting Generation

The first step in the process is generating handwriting, which we are going to use as our cover image. Handwriting is particularly suitable as a toy problem because it is replicable, but unique in style, and is an everyday form of communication. For example, one might wish to encode secret messages into their signature, which they include at the bottom of their emails. We utilize Graves' approach to handwriting generation using an LSTM [4]. The LSTM operates over sequences of pen-tip locations from the IAM

Online Handwriting Database: the x and y coordinates, and an end-of-sequence binary flag. The model also relies on a mixture of Gaussians: at each timestep, the LSTM outputs a value to parameterize the Gaussian distributions, which predict the location of the next coordinate in the sequence based on the previously generated coordinates.

3. Data Encryption

We will hide data in the generated handwriting sample images, which will eventually be covers. Before this step, the secret data is encrypted to add another level of security. Recently, machine learning and encryption have intersected to form a new field, neural cryptography. Researchers [1, 3] recently developed pipelines to generate a unique, symmetric key cipher. User Alice wants to send a message to Bob, so she encrypts secret data M with key K to produce M' , and Bob decrypts M' with the same key K . They add one adversary, Eve, who receives M' and attempts to recover M without access to the key. Alice and Bob attempt to minimize Bob's reconstruction error and make Eve's reconstruction error close to 50% (random guess), while Eve attempts to minimize her own reconstruction error. This approach lacks robustness to an arbitrary adversary.

We propose a new approach where Dave, a new adversary, receives the encrypted data M' and attempts to recover M without access to the key, similar to Eve. However, the main difference is Alice and Bob are unaware of Dave's existence. Therefore, even if they learn a robust algorithm against Eve, they may not be able to protect against an arbitrary adversary. With appropriate settings, we can achieve 0% reconstruction error for Bob, 50% reconstruction error for Eve (random guessing bits of data), and 50% reconstruction error for Dave. Thus, Alice and Bob successfully learn a robust, generalized encryption algorithm.

4. Steganography

Next, once the secret data is encrypted and coordinates are generated for a cover image of handwriting, these must be combined. An autoencoder network is trained to read in an (x, y) coordinate location and 2 bits of secret data. This outputs some new coordinate with a small displacement. For example, the pixel coordinate $(1, 2)$ combined

with the data (1, 1) (binary) might produce (1.23, 1.94). To produce discrete coordinates from continuous data, we add a block of grey pixels for floating point locations. This is in contrast to a specific, discrete pixel location, such as (1, 1), which has one black pixel at that location, surrounded by white pixels. Then, should one want to extract the encoded secret message, this displaced coordinate is passed into a de-steganography network to output the original encrypted data. In this work, as proof of concept, we use a fixed cover.

Two loss functions are used to train this network: cover loss and decoding loss. The cover loss is the L_1 distance from the new perturbed coordinate to the original coordinate; we want this distance to be small so that the resulting image looks like a plausible handwriting image. The decoding loss is the L_1 of decoded and original data; we want small values so if we Alice sends an encoded message to Bob (unbeknownst to Eve or Dave), he can recover it. In our results, the decoding loss was 0.0029, indicating an ability to successfully retrieve the original data. For the cover loss, the sum of x and y displacements was 0.1700 on average.

5. Pipeline and Integration

After discussing handwriting generation, data encryption, and steganographic coordinate displacement, we describe how to integrate these individual processes into a full steganography pipeline. During encoding, some secret data M is read into a system. This data is converted to binary and split into chunks of 128 bits since that is the input required for the pre-trained neural cryptography network. These chunks are combined to form the encrypted data M' . This encrypted data is split into mini-chunks of 2 bits since that is the amount of data combined with coordinates in the steganography network. These 2-bit chunks are sequentially encoded with the original generated coordinates for handwriting, C , to produce new, slightly displaced coordinates C' . Then, C' are all printed onto a canvas, such as a PNG image file, to produce the final handwriting sample, which is the container image.

In order to decode, the container image is read into memory. An 80×80 window, anchored at the top-left of the image, is extracted from the image. This window is passed into a convolutional neural network with structure similar to LeNet. These image features and an initial coordinate state are passed into an LSTM to produce the initial coordinate in the sequence of the handwriting. Then, this predicted coordinate and a horizontally shifted window are once again passed into the network to produce the next coordinate. This process is repeated until all coordinates in the sequence are predicted where the end of the sequence is marked by two sequential coordinates with an end-of-sequence flag of 1. Then, these extracted coordinates (C') are sequentially passed into the decoder part of the steganography auto-encoder, to extract C and M' . Next, M' is passed into the decryption network to produce M .

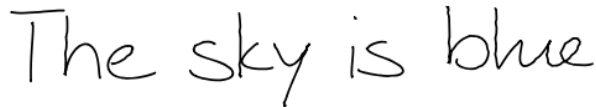


Figure 1. The steganographic container image with the secret message “Meet me on the east side at 11:03pm.”

Figure 1 displays results of the encoding. Although hard to notice, the container images contain small perturbations of the coordinates. Thus, smaller scales will lead to a passerby not suspecting any encoding. The secret message is successfully recovered.

6. Discussion and Future Work

Our newly proposed steganography pipeline improves on previous machine learning steganography papers in several aspects. First, the handwriting cover image is sparse (vast whitespace), but there is no risk of secret message/image artifacts in the form of background noise. Next, the cover images can be any arbitrary size and contents, which contrasts with the pre-trained neural networks of image input size $N \times N$ in other steganography papers. The above image with cover text “The sky is blue” is arbitrary, where any form of text can be generated. Finally, the encoded data lies in the modification of the image’s appearance, rather than being embedded into the pixel intensity values. As such, the image *appearance* is modified, rather than the *contents*, providing a new avenue of steganography and further robustness on the human visual inspection task.

We hope this toy problem can be generalized and abstracted to other dataset domains and tasks. For example, cover images can be sparse sketches whose coordinates are modified, or on regular images, textures or mid-level semantics (attributes) can be modified to embed different forms and amounts of data. In the future, we will exploit generative adversarial networks operating over coordinates or semantic properties. We will also develop an end-to-end approach that uses a key to both encrypt and steganographize a message, for added robustness to adversaries.

References

- [1] M. Abadi and D. G. Andersen. Learning to protect communications with adversarial neural cryptography. *CoRR*, abs/1610.06918, 2016.
- [2] S. Baluja. Hiding images in plain sight: Deep steganography. In *NeurIPS*, 2017.
- [3] M. Coutinho, R. de Oliveira Albuquerque, F. Borges, L. J. García-Villalba, and T.-H. Kim. Learning perfectly secure cryptography to protect communications with adversarial neural cryptography. In *Sensors*, 2018.
- [4] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.