# Hiding Images Within Images

Shumeet Baluja, *Member, IEEE*

**Abstract**—We present a system to hide a full color image inside another of the same size with minimal quality loss to either image. Deep neural networks are simultaneously trained to create the hiding and revealing processes and are designed to specifically work as a pair. The system is trained on images drawn randomly from the ImageNet database, and works well on natural images from a wide variety of sources. Beyond demonstrating the successful application of deep learning to hiding images, we examine how the result is achieved and apply numerous transformations to analyze if image quality in the host and hidden image can be maintained. These transformation range from simple image manipulations to sophisticated machine learning-based adversaries. Two extensions to the basic system are presented that mitigate the possibility of discovering the content of the hidden image. With these extensions, not only can the hidden information be kept secure, but the system can be used to hide even more than a single image. Applications for this technology include image authentication, digital watermarks, finding exact regions of image manipulation, and storing meta-information about image rendering and content.

**Index Terms**—Information Hiding, Image Verification, Image Trust

## 1 Introduction

Information hiding is most commonly associated with well publicized nefarious endeavors, such as secretly planning and coordinating criminal activities through hidden messages in images posted on public sites [1]–[3]. Beyond the multitude of misuses, however, hiding information can be used for practical positive applications as well. For example, hidden images used as watermarks embed authorship and copyright information without visually distorting the image [4]. Other meta-information, such as motion vectors, bounding boxes, extended-color and depth information for each pixel can be hidden without noticeably changing the appearance of the image. Perhaps even more timely, we provide a way to handle the rapidly growing problem of fake and partially altered images on popular social media and news sites. By hiding invisible markers throughout an image, even subtle alterations can be easily detected by analyzing the reconstruction of the markers – all without compromising the visual integrity of the viewed image [5].

The challenge of good information hiding arises because embedding a message can alter the appearance and underlying statistics of the carrier (the *host image*). The amount of alteration depends on two factors: first, the amount of information that is to be hidden. The most common form of information hiding has been hiding a relatively small number of bits - for example for text messages [6]–[8]. The longer the message, the more the potential distortion. Second, the amount of visible alteration depends on the carrier image itself. Hiding information in the noisy, high-frequency, regions of an image yields less humanly detectable perturbations than hiding in the flat regions. Work on estimating hiding capacity can be found in [9].

The most common hiding techniques manipulate the least significant bits (LSB) of images - whether done uniformly or adaptively [10], [11]. Though often not visually observable, statistical analysis of the images can reveal whether the resultant files have been altered. Other methods of hiding attempt to preserve the host images' statistics by creating and matching models either explicitly [12] or through deep learning [13].


Fig. 1. Samples from the full-image hiding system. From the left: host image, hidden image, the container image (the container holds/hides the hidden image within it while looking like the host), and the recovered hidden image – this is extracted from only the container. The last two columns are the errors for the container vs. host and reconstructed vs. hidden images (enhanced $5\times$). The bottom two rows show examples of larger reconstruction errors, likely due to the highly saturated colors that were not well represented in the training set.

Though similar conceptually to *steganography* [1], [14]–[19], five key differences set this work apart:

- The hidden information need not be encoded perfectly; small errors in the hidden image are acceptable. It is possible to explicitly balance the reconstruction quality of the visible and hidden images, see Figure 1.

• *S.Baluja is with Google-AI, Google, Inc. Email: shumeet@google.com*

*Manuscript submitted June, 2018*

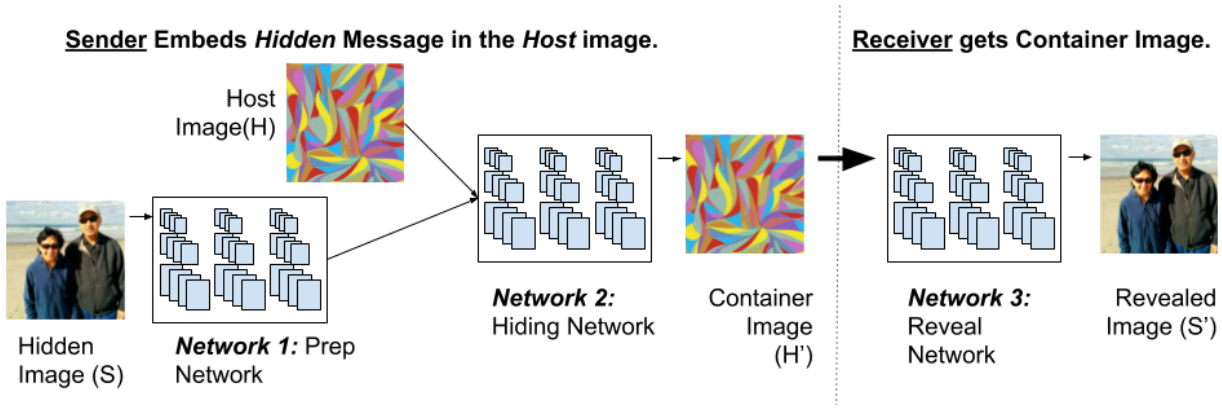


Fig. 2. The three components of the full system. Left: Hidden-Image preparation. Center: Hiding the image in the host image. Right: Uncovering the hidden image with the Reveal-Network; this is trained simultaneously, but is used by the receiver.

- The visible information need not be transmitted perfectly. In steganographic systems, the hidden information is "fragile" [8], such that small modifications to the visible image may yield large errors in the hidden message. In our system, local changes to the visible image yield local changes to the hidden image.
- The orders of magnitude of the amount of information to be hidden – there is a 1:1 ratio of hidden to host information. In terms of bits per pixel, we are attempting to encode 24bpp.
- We *implicitly model* the distribution of the statistics of natural images rather than creating explicit models. This is achieved through using a deep neural network trained with a large set of host and hidden images.
- Given the amount of information that we hide, we do not attempt to conceal the existence of a hidden message. We do, however, present methods to obfuscate the content of the hidden message.

These distinctions between our work and steganography are important. Due to the amount of information hidden, we cannot assume that the fact that there is hidden information in the image can remain undetected. Nonetheless, we will often refer to tools and techniques often used with steganography to help understand and analyze our approach.

Aside from steganography, good examples of information hiding in images have been explored in [20]–[22]. Notably, in these studies, the authors have encoded extra information about the visible image within the transmitted image, for example color information in a gray-scale version of the same image. By extracting the hidden information, the receiver is able to convincingly reproduce the image's colors. In our experiments, we explore a generalization of this; we will hide an entirely independent image into another.

Despite recent impressive results achieved by incorporating deep neural networks into finding the existence of hidden messages (*steganalysis*) [13], [23]–[25], there have been relatively fewer attempts to incorporate neural networks into the hiding process itself [26]–[30]. Of these studies, some have used deep neural networks (DNNs) to select which LSBs to replace in an image with the binary representation of a text message. Others have used DNNs to determine which bits to extract from the container images. Recently, [31], proposed a technique that uses a novel learning approach with shallow neural networks to create a system to hide short messages in images. Based on DCT coefficients, their method initializes networks trained to embed messages in images that are quite tolerant to common transformations, such as rotation, inversion, and color transformations. Perhaps the most advanced use of neural networks was done concurrently to the work presented here [32]–[35]. These systems created DNNs trained with adversaries for hiding messages; Generative-Adversarial-Networks (GANs) [36] were used to provides an error signal based upon the discoverability of the hidden message. These studies encoded relatively small messages, more typical of standard steganographic studies, and showed good resistance to existence discovery.

In this work, the neural network determines where to place the hidden information, as well as how to compress and represent it. As will be shown, the hidden image is dispersed throughout the bits in surrounding pixels and across all the color channels. A decoder network, that has been simultaneously trained with the encoder, is used to reveal the hidden image. The networks are trained only once and are independent of the host and hidden images. They also only work with each other, as they are trained as a pair.

In the next section, we will describe how the system of neural networks is simultaneously trained to hide and recover images. The basic system was originally presented in [37]. An analysis of where the information is stored, and a presentation of various methods for discovering the hidden information is presented in Section 3. The presence of hidden information can be discovered by training learning-based detectors. To address this, in Section 4, we present two methods to obscure the hidden information even when the original, unaltered, cover image is publicly available. Beyond making the hidden information more difficult to identify, we also demonstrate how to embed more than a single image.

## 2 Learning to Hide Full-Size Images

Though information hiding is often conflated with cryptography, in our approach, the closest analogue is image compression through auto-encoding networks [38]–[46]. The trained system, shown in Figure 2, learns to compress and

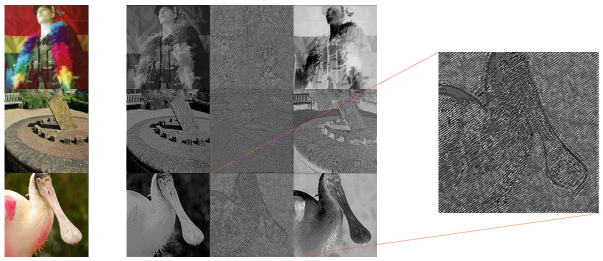

Fig. 3. Transformations made by the preparation network (3 examples shown). Left: Original Color Images. Middle: three channels of information extracted by the preparation network that are input into the hiding network (other channels not shown). Right: zoom of the edge-detectors. In the most easily recognizable example, the 2nd channel activates for high frequency regions, e.g. textures and edges (shown enlarged (right)).

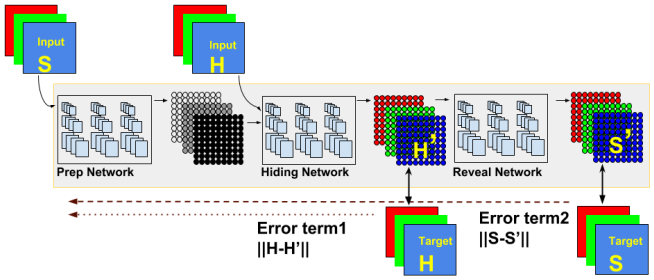


Fig. 4. The three component networks are trained as a single network, thereby pairing the encoding and decoding portions to work exclusively with each other. The first error term 1 affects the Preparation and Hiding component networks. Error term 2 affects all 3 components. $S$ is the hidden image, $H$ is the host image.

place the hidden image into the least noticeable portions of the host image.

The full system is a series of three networks that are trained as a single, large, network (Figure 2). The first component, the Preparation-Network, prepares the image to be hidden. The main function of this network is to transform the RGB-pixels of the hidden image into features that can be used by the Hiding-Network [1]. These features are not *a priori* specified; they are learned. An analysis of the transformations done by the preparation network revealed ones that are commonly useful for compressing images, such as edges and orthogonal components [47]. See Figure 3.

The actual embedding of the hidden image into the host image is done by the Hiding-Network. The Hiding-Network receives the output of the Preparation-Network and the host image as input. The input is formatted as an $N \times N$ pixel field (for our studies, $N = 200$), with depth concatenated RGB channels of the host image and the transformed channels of the hidden image. The output of this network is the *Container* image ($N \times N$, RGB pixels). The container image should appear as similar to the host as possible, while also containing enough information to recreate the hidden image.

The third component, the Reveal-Network, is responsible for extracting the hidden image from the container. Though this network is used only by the receiver, all three components are trained as a single network. This pairs the sender and receiver sides of the communication to work exclusively with each other.

The basis of this approach borrows heavily from auto-encoding networks [38] in which a network internally creates a compressed representation of the image [39]. In this system, instead of simply encoding a single image through a bottleneck for compression, two images are combined such that the intermediate representation (the container image) appears similar to the host image. The system is trained by reducing the following error: ($H$ and $S$ are the host and hidden images respectively, $\beta$ is their weighting):

$$\mathcal{L}(H, H', S, S') = ||\mathbf{H} - \mathbf{H}'|| + \beta||\mathbf{S} - \mathbf{S}'|| \qquad (1)$$

The form of error propagation is vital (see Figure 4). Notice that the error $||\mathbf{H} - \mathbf{H}'||$ does not change the weights of the Reveal-Network. The Reveal-Network does not need to be able to reconstruct the host image; it needs to only recover the hidden image from the container. In contrast, all of the networks are trained with the signal from $\beta||\mathbf{S} - \mathbf{S}'||$ since each portion of the network is responsible for holding and forwarding information about the hidden image. By propagating this error signal to both the Preparation and Hiding networks, the representations formed early in the system encode information about the hidden image.

30 neural architectures and training hyper-parameter were used to find the best network for the task. The final network employed is given in Table 1. Similar architectures revealed approximately the same performance with visually indiscernible differences. All images, both input and output, are $200 \times 200$ pixels, with 3-channel depth (RGB). Networks were trained with ADAM [48] and weights were initialized with variance scaling (0.1). Batch-Normalization was not used. All experiments were implemented using TensorFlow [49] with default parameters, except as specified above.

Results from the best of the 30 networks are shown in Figure 1. Unless reported otherwise, $\beta = 0.75$ (the other two settings of $\beta$ tried, 1.0 and 1.25, revealed nearly visually identical reconstruction performance. The network trained with 0.75 gave a small quantitative benefit in the networks we trained). Quantitatively, the average color error, per RGB channel, for the host image was 2.4 (out of a possible 256) and 3.4 (/256) on the hidden. This was measured on two test sets: those from the ImageNet challenge ('validation' images) and the second composed of 200 cell-phone and DSLR photos. None of the test images were used in training.

## 3 Where is the Hidden Image Encoded?

The primary focus of this paper is to concretely demonstrate that it is possible to encode a large amount of information in an image with limited visually noticeable artifacts. However, no explicit attempt has been made to actively hide the existence of that information from machine detection. Though we cannot expect to completely hide the fact that 1/2 of the information is part of a hidden message, measures can be taken to make it more difficult to discover. First, we must determine where the information of the hidden image resides.

1. The function of the preparation network and the hiding network can be combined into a single network. However, training with this division resulted in a faster training and easier analysis.

TABLE 1
Network Architectures

| Network | Input | Hidden (size, stride, depth, activation) | Output | Notes |
|---|---|---|---|---|
| Preparation | 200×200×3 → (Image to Hide) | → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (2 × 2, 1 × 1, 30, relu) → 2D-Conv (2 × 2, 1 × 1, 7, tanh) → | →200×200×7 (Transformed Hide Image) | A variety of output dimensions were tried (5-12). Most yielded similar performance to the 7 channel version shown here. |
| Hiding | 200×200×3 + 200×200×7 → (Host Image & Transformed Hide Image) | → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (2 × 2, 1 × 1, 30, relu) → 2D-Conv (2 × 2, 1 × 1, 3, tanh) → | →200×200×3 (Container Image) | The output should appear similar to the original host image. |
| Reveal | 200×200×3 → (Container Image) | → 2D-Conv (4 × 4, 1 × 1, 100, relu) → 2D-Conv (4 × 4, 1 × 1, 100, relu) → 2D-Conv (4 × 4, 1 × 1, 50, relu) → 2D-Conv (2 × 2, 1 × 1, 3, tanh) → | →200×200×3 (Reconstructed Hidden Image) | This network resides with the receiver. This decoded output should appear similar to the hidden image. |

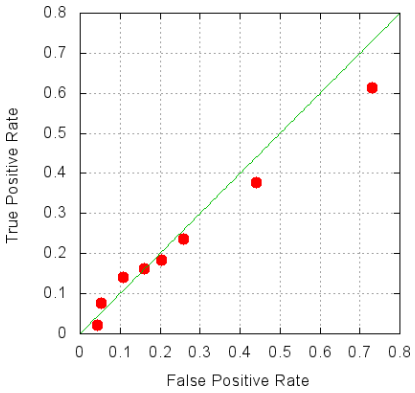


Fig. 5. ROC curves: True Positive Rate vs. False Positive Rate for **StegExpose** when detecting images embedded via DNNs.

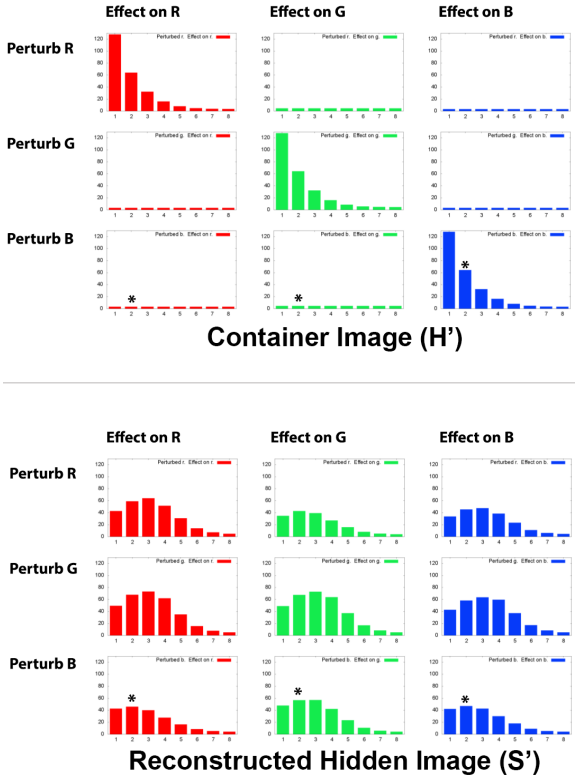


Fig. 6. Sensitivity to Bits in the Container Image. (Left:) Control case, the container image. Changing a bit in the container image obviously has no effect on other bits. (Right): However, that same bit flip in the container image has effects across all color bands in the recovered hidden image. For example, if we want to measure the effect of flipping the 2nd bit in the Blue channel, we would look at the marked entries (marked with *).

Is the network simply hiding the information about the hidden image in the least significant bits of the host image? Tools exist to find hidden information in the LSBs; a publicly available steganalysis toolkit, **StegExpose**, was used to test the reliance on LSBs for our hidden images [50]–[52]. Per the description of the tool: *"StegExpose rating algorithm is derived from an intelligent and thoroughly tested combination of pre-existing pixel based steganalysis methods including Sample Pairs by Dumitrescu (2003), RS Analysis by Fridrich (2001), Chi Square Attack by Westfeld (2000) and Primary Sets by Dumitrescu (2002)"* [50]. In addition to the default settings (threshold=0.2), the threshold was varied throughout a large range. The ROC curve is shown in Figure 5. Note the little variation beyond random guessing (green line).

StegExpose would have been able to find the information if it were simply placed in the LSB bits. We turn to a second method to find where the information is stored. The images used in the study are composed, at each pixel, of 24 bits $(8 \times (R, G, B))$. If we flip the first bit of the R channel of all the pixels in the *container image* $(H')$, we can measure its effects on the reconstructions on the container image itself and also, by propagating the modified image through reveal network, on the reconstruction of the hidden image. See Figure 6. The top half of the figure (container image) should be considered the "control"; the upper left bar-chart shows that the effects of changing a bit in the Red channel of the container only has an effect on the red-channel in the container, and that the magnitude is proportional to the significance of the bit. This is exactly as expected and holds true for the Green and Blue channels as well (the diagonal).

Next, lets examine the effects that changing a bit in the container image has on the reconstructed hidden image. See

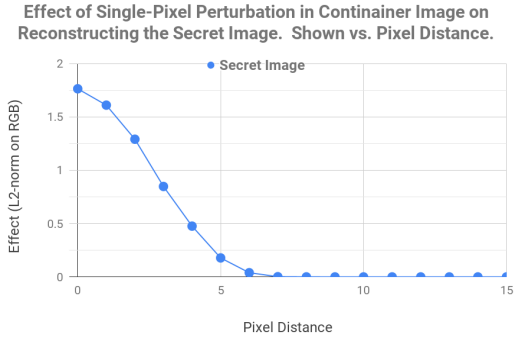


Fig. 7. How far does the effect of changing a pixel in the container image reach when decoding the hidden image? This figure shows the average effect that perturbing a pixel in the container image has on the hidden image reconstruction, as a function of distance from the perturbed pixel. After approximately a distance of 7 pixels, the effect is negligible.

Figure 6 (bottom half, "Reconstructed Hidden Image"). Here, we see that a bit flip in any bit position in any color channel of the **container image**, has an effect across *all color channels* in the hidden image's reconstruction. Moreover, the error does not follow the standard significance ordering of the bit positions. In other words, the information for the hidden image is spread across the color channels — the reason it was not detected by simply looking at the LSB.

In addition to distributing the hidden image information across the color-bits, the information is also spread in the spatial dimension. To measure this, we employ a similar procedure as before. As the baseline, we first place the hidden image, $S$, in a host image, $H$, which creates the container image $C$. $C$ is passed through the Reveal network, revealing $S'$. Then, to examine the spatial distribution, we modify a single pixel in $C$, giving $C_m$. $C_m$ is passed through the Reveal network, yielding $S'_m$. For each pixel, we measure $||S' - S'_m||_2$. This is reported in Figure 7 as a function of pixel-distance from the perturbed pixel.

Two observations should be noted. First, the representation for the hidden image is distributed – both in surrounding pixels and in color bits. On average, the encoding for each pixel of the hidden image is distributed in pixels that are up to a distance of 7 away from the corresponding pixel in the container image. Second, the amount of spatial distribution is directly related to the neural network architecture and the size of the convolutions. Had we wanted to encourage a wider distribution, more or larger convolutions could have been employed, as well as Atrous convolution layers [53].

Moving beyond LSB analysis, multiple studies have shown that deep networks are competitive and often better than the more established methods of steganalysis that use hand selected image features [23], [24]. We trained detection networks as binary classifiers using unperturbed ImageNet images as negative samples and our container images as positive examples. After an empirical examination of input representations and architectures, we discovered that the detection networks performed well representing an image with $24 \times (N \times N)$ stacked binary input layers. Each of the 24 layers represented a bit in the pixel representation of the input image. The networks were trained in an analogous manner to others in this study.

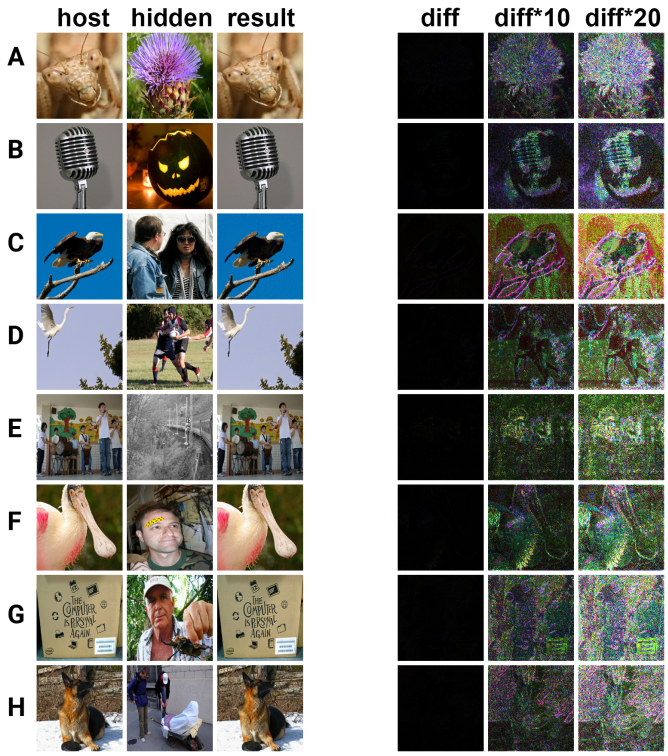


Fig. 8. The hidden image is embedded within the host image; the result, the container image, is shown in the column 'result'. In the hypothetical case of an adversary gaining access to the original, unaltered, host image, the adversary can magnify the difference between the original host and the container, see column 'diff', 'diff*10', 'diff*20'. When magnified 10x and 20x, traces of the *hidden* image are visible, particularly rows A,B,C,G.

The detection networks obtained between 90-95% combined classification rates. For comparison, the same networks were retrained to find Least-Significant-Bit substitution (where the $L$ least significant bits of each color channel in the host image are replaced with the $L$ most significant bits from the hidden image). The networks were trained and tested with $L = 1, 2, 3, 4$; all achieved over 99% combined rates. An additional set of tests revealed that even when a single bit was randomly placed in one of the 4 bit positions, the detection rate remained above 99%. Given these detection rates on even a single bit position, the high detection rates with a full image are not surprising.

The rates of discoverability makes this technique unsuitable for applications requiring the undetectability of hidden information; however, its use for data-hiding, data augmentation, and image-content alteration detection remain intact. We now turn our attention to the discoverability of the *content* of the hidden image, rather than just the existence of a hidden image.

### 3.1 Hidden Content Discovery

In our first examination, let us assume that the sender of the message has (unwisely) chosen to use a publicly available image as the host image. Under this assumption, we must assume that the adversary is also able to find other copies of the publicly available image (that were not transformed to contain the hidden image). By dramatically magnifying the pixel-wise differences between the container and the

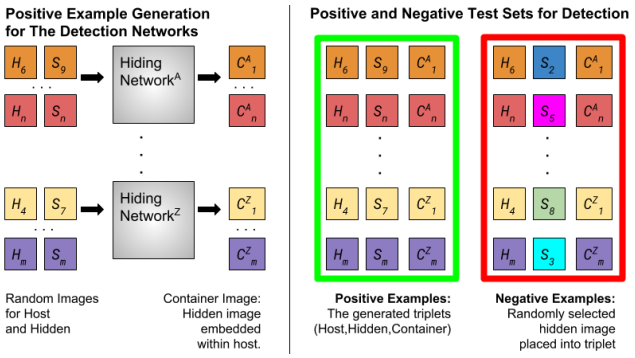


Fig. 9. Creating training examples for the discrimination network by generating triplets {cover,hidden,container} from the discarded networks.

unaltered, found, host image, portions of the hidden image are revealed. See Figure 8 for examples. As the hidden image is not cryptographically secured, visually recognizable features of the hidden image are visible.

The hidden image is most readily seen in host images that contain large uniform patches of color (Figure 8 -A,B,C & G). What if the sender had been careful to only use higher-entropy images? Would the hidden information still be discernible? To answer this from the perspective of an attacker, we assume that the adversary does not have access to the exact network that the sender employs. However, the attackers can train their own deep-image-hiding networks successfully. With that ability, can they create an adversary for *our* network? Recall that the representations developed by the adversary's hiding-network will be different than ours (as mentioned earlier, each network's encoder and decoder are paired to work exclusively only together). Instead of attempting to mimic our system, the adversary can instead turn to gathering large-scale statistics about the hiding process by training many, randomly initialized, variants of the hiding system. We simulate this as follows.

Our goal is to create a *Discriminiation Network* that can determine whether an image (the container) contains another image (the hidden). Recall that for obtaining a system of encoding-decoding networks that worked well, we trained 30 different networks with varying architectures and backpropagation hyper-parameters. From these 30, we had selected the best and discarded the rest.[2]

We return to these previously-discarded networks and gather samples from *each* of the discarded, but fully trained, networks in the form of triplets: {cover-image, hidden-image, container-image}. Each of these triplets was added to the "positive" group. A "negative" set of examples was created in which, for each of these triplets, the hidden image was replaced with a randomly selected image. The task for these new Discrimination Networks is then to correctly identify that the hidden images in the negative set are *not* embedded in the container images and that the hidden images in the positive set *are* embedded within the containers. This process is shown in Figure 9. Why do we do this? If it is possible to accurately train this discrimination network, then it confirms that there is enough discoverable information in the container image for an external attacker to gain information about the hidden image.

Given the set of positive and negative examples, we trained multiple deep *Discrimination Networks* to distinguish whether a specific hidden image was embedded in the container. In these experiments, the discrimination network was also given as input the original, unperturbed, host image. Recall that the training samples were gathered from many, separately trained, networks. Thus, the detector solved this task for multiple hiding networks, not just a single one (in Figure 9, this is denoted by $HidingNetwork^A$ — $HidingNetwork^Z$). Nearly perfect accuracy was achieved for all of the networks that contributed samples to the training. Moreover, 95-99% accuracy was obtained on triplets generated by **our** hiding network — a network that it was not trained on.

This implies that if the hidden image is known, the evidence to verify its presence is reliably extractable from the container without access to the specific decoding network. This attack assumes that the adversary has access to the unaltered host image. Let us now make the attack even broader: what happens when the unaltered host is not available? We attempted to retrain the Discrimination Networks without being given the unaltered host. Surprisingly, the detection accuracy remains high, 94.5%.

The high accuracy rates without the original host images means that there is a discoverable transformation of the container image that yields information about the embedded image. As our final attack, we verify this through the simplest attack: We transform *only* the container image through a series of blur operations, yielding $container_{blur}$. We then compute the difference between this and the container image (in each R,G,B, channel) and intensify the results, yielding $Revealed$, where $Revealed = min(255, |container - container_{blur}| \times V)$ and $5 \leq V \leq 20$. As can be seen in Figure 10, this simple transformation also yields smaller, but nonetheless present, hints to the contents of the embedded image — without access to the original host or the original hidden image.

Visually, it is sometimes difficult to discern the objects in the hidden image when simply removing the blurred container image. Nonetheless, even this simple analysis reveals components of the hidden image, most clearly in those that are embedded in the lower-entropy regions of the host image. Earlier we had stated that merely knowing the existence of a hidden image did not necessarily reduce the effectiveness of this technique for watermarking or image-content-alteration detection. However, if the actual content of the hidden image is known, this can pose a greater problem as it makes removing the hidden image or replicating portions of the hidden image easier after an image modification has been made. Next we explore methods to make the hidden information less discernible.

## 4 Obfuscating the Hidden Image

In the previous section, we demonstrated that it was possible to discover remnants of the hidden image in the container image. This was done with and without access

2. There was a significant variation in performance of the discarded networks, ranging from error rates $1.02 - 3\times$ of the best network. Each network encodes the hidden image in a potentially unique way as there were no hints given specifying in which bit positions or in which pixels and pixel-neighbors to encode the hidden image.

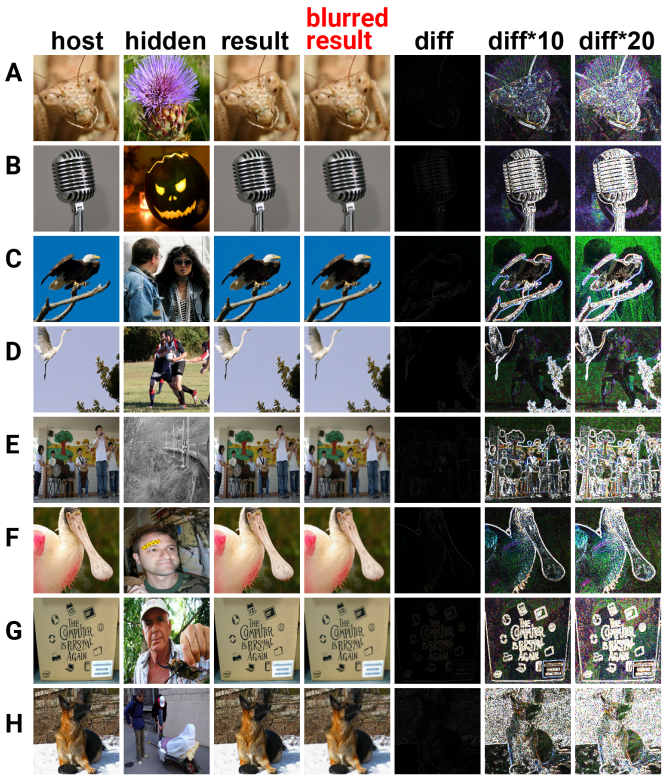


Fig. 10. What can we see without the unmodified host image? If we blur the 'result' image and subtract that from the 'result' image, we see remnants of the hidden image (rows A,B,C in particular). Color-constant regions in the host image are most susceptible to this. This is a potential vulnerability even without access to the original host image.

to the unaltered host image, and with simple and more sophisticated DNN based attacks. In this section, we present two method to obfuscate the hidden image. Desiderata for the obfuscation techniques are (1) Even if an unperturbed host image is obtained, the content of the hidden image is difficult to ascertain. (2) The obfuscation technique must have the property that errors remain *local*, e.g. errors are not additive/multiplicative and do not propagate. Because of the large amount of hidden information, neither the host nor the hidden image may be encoded perfectly. Errors in the reconstruction should impact as few pixels in the recovered hidden image as possible. Ideally, a single error only affects a single pixel.

A simple method for obfuscating the hidden image is to permute the pixels before they are passed to the hiding network [54]. The receiver of the image passes the container image through the decode-network, which reveals a scrambled hidden image that is then "un-permuted." This procedure has the important property that the error in each pixel's reconstruction is strictly local (affects no other pixel) and that the permuted image is unrecognizable even when viewed without any reconstruction error.

It is tempting to use the networks trained in the previous section with the permutation and un-permutation steps "wrapped" around the hiding system. However, this does not work: images are not "stored" as pixels in neural networks [38], [55]. A neural network concisely represents the information in an image by exploiting spatial coherency.

TABLE 2
Average error in R,G,B Channels (out of 256)

| | | host | Hidden |
|---|---|---|---|
| 1 | Permuted with a Single Permutation | 5.1 | 7.5 |
| 2 | Permuted With Multiple Permutations | 6.2 | 7.5 |
| 3 | Permuted w/Nets from Sect. 2 (no retrain) | 4.6 | 14.4 |
| 4 | Baseline: No Permutation (from Section 2) | 2.4 | 3.4 |

For example, encoding natural images that exhibit spatial structure is far more successful than encoding noise. Unfortunately, like noise, the permuted images lack spatial structure, thereby requiring the network to memorize pixel intensities. (The permuted images do, however, maintain the intensity distributions of natural images.) A system trained with only natural images is not able to perform well on pixel-permuted images. Rather, we need to repeat the entire training process; the system is retrained to hide images without the spatial coherence of natural images. As before, 30 networks were retrained with the same architectures used previously, supplemented with the permutation steps.

The results from the best network, evaluated on an independent test set, are shown in Figure 11. The intermediate permuted images are also shown. Note the lack of spatial structure in the permutation while the color distribution of the underlying hidden is discernible (column 3, labeled "Shuffled-1"). Most importantly, even when the errors are magnified in the reconstruction of the host image, there is no recognizable trace of the hidden image (Figure 11, Column 7). Table 2 gives the average error, per channel, of this system (Row #1) compared to one that does not use permutations (Row #4). Row #3 shows the results of using permutations with the networks trained previously (without permutations).

Training the networks to encode permuted images proceeds significantly slower than training the baseline hiding networks described in Section 2. Using a single, modest, GPU, training for permutation-hiding networks took over 2 weeks. The baseline networks reached peak performance in approximately 5 days of continuous training. Encoding permuted images is a significantly harder problem.

Finally, it is important to discuss whether the underlying permutation can be discovered? With enough permuted samples, such as encoding each frame in a video with this method, the statistics of natural images and spatial coherency constraints may yield clues on the real position of the pixels [56]. However, even this can be mitigated if we allow the permutation mappings to themselves be algorithmically chosen from a large number of known, or derivable, orderings, e.g. [57]. Experiments simulating this using randomly selected permutations are shown in Row #2 of Table 2.

### 4.1 Hiding Multiple Images

Currently, the system successfully embeds a full color $N \times N$ image in another of the same size (Figure 1) - the hidden image can either be permuted or un-permuted. In comparison to encoding non-pixel-permuted images, for a neural network to encode a permuted image, it must create a much more detailed representation of the pixels that does

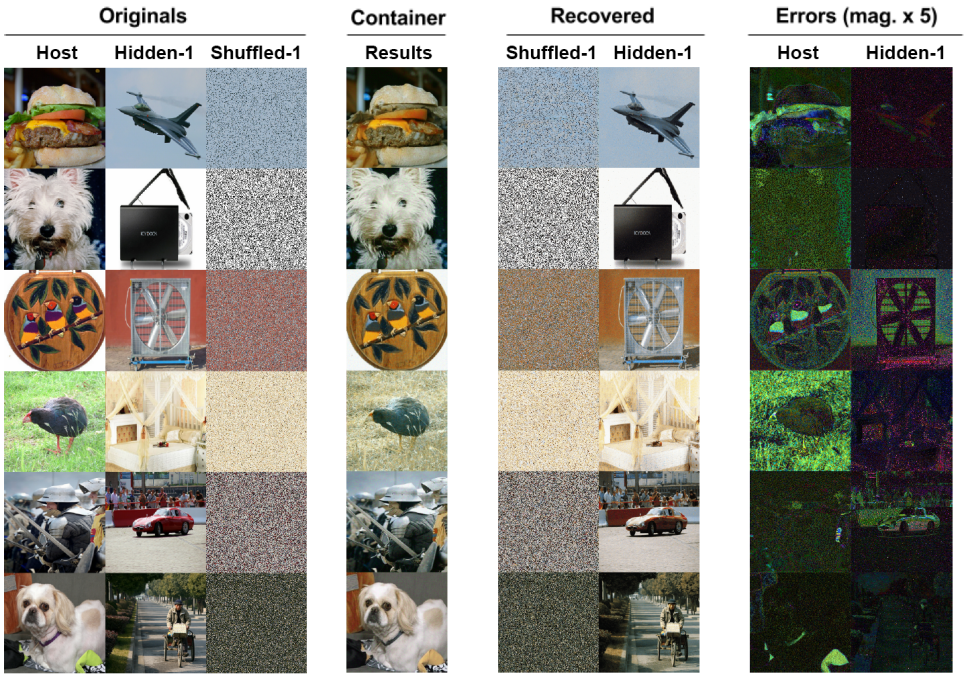


Fig. 11. Results with encoding and decoding permuted images. In the sender's column: from the left: host image, hidden image, the hidden image after pixels permutations and the result image. In the right half, the receiver's column: the decoded permuted-hidden image, and the un-permuted image. The residual error images are given for both the host and the recovered images in the last two columns. Most importantly, in comparison to Figures 8 & 10, note that there is no evidence of the hidden image in the errors for the host (second to last column).

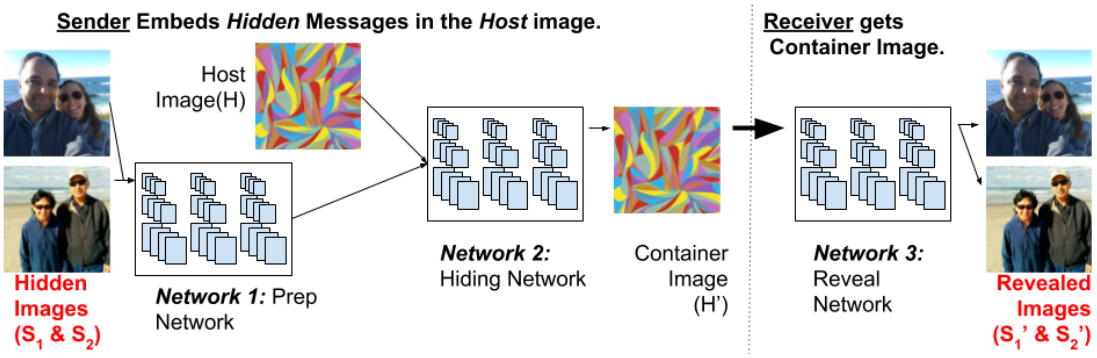


Fig. 12. System to hide two full $N \times N$ images within a host image of the same size. Compare this to Figure 2.

not rely on the spatial structure present in natural images. We present a second method of obfuscation: hiding multiple images. Instead of using the network's extra capacity for representing the permuted pixels more accurately, can the capacity be used to encode more than a single image? If we can hide multiple images, it is likely that any single image will be harder to reconstruct even when encoded without any perturbations. Beyond secrecy, the ability to hide more than a single image also enables us to hide multiple, independent, pixel-wise sources of supplementary information, such as depth, color or motion.

Concretely, we attempt to hide two full size $N \times N$ images — we are hiding $2\times$ the amount of information that is in the host, see Figure 12. The training proceeds in a manner similar to Section 2. For our experiments, the $L_2$ loss on each image was weighted equally. As before, this system requires retraining all three components. Sample results are given in Figure 13 and the quantitative error comparisons from the test set are provided in Table 3. As expected, the error increases but the reconstructions remain, visually, good.

Recall that in Section 3, we trained *Discrimination Networks* to determine whether a specific hidden image was embedded in a container image. We tested those same networks with this two-image hiding system. In the single image hiding task, the Detection networks had an accuracy of $\approx 94\%$. When encoding multiple images with the networks described in this section, the accuracy drops to $\approx 73\%$. As hoped, the process of hiding more information further obfuscates the hidden image.

Finally, for completeness, we applied the perturbation and the multiple-image hiding in the same system. Two images are each permuted and hidden within a single host image. The reconstruction errors increase substantially over using only a single hidden image and also over the non-

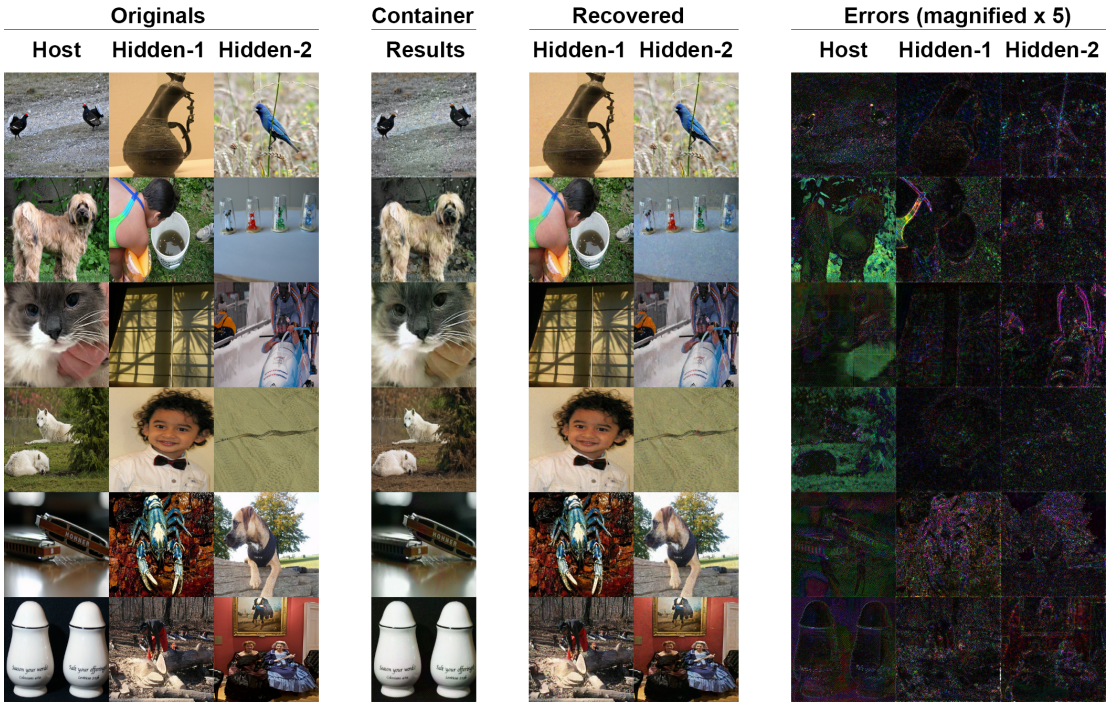


Fig. 13. 6 Results showing typical reconstructions when hiding 2 images within a single host image. Note that the colors appear far less saturated and there is modest noise in the flat regions of the host image.

TABLE 3
Average error in R,G,B Channels (out of 256) – When Hiding Two Images

| | host | $Hidden_1$ | $Hidden_2$ |
|---|---|---|---|
| Hiding 2 Images | 6.6 | 5.0 | 6.0 |
| Hiding 2 Images Single-Permut. | 9.2 | 10.2 | 11.4 |
| Hiding 2 Images Multiple-Permut. | 9.5 | 10.7 | 12.0 |
| Baseline: Single Image (Section 2 ) | 2.4 | 3.4 | n/a |

permuted two-hidden image systems. The increase in error was seen despite the much longer training times allowed for these networks: 30 days on a single GPU. Table 3 provides the quantitative results. Examples of the hidden and reconstructed images are shown in Figure 14. Despite the increase in noise and reconstruction errors, the content of the images, including the edges and gradients, remains largely intact and easily recognizable.

### 4.2 Effects of Image Degradation on Retrieval

Throughout the presentation of the experimental results, we have asserted that the container image appeared very similar to the host image, and the reconstructed images were easily recognizable as the hidden images. We presented the images for visual inspection as well as quantitatively measured the reconstruction errors of both the host image vs. the container image, and the reconstruction of the hidden image from the container image. As a final test, to quantitatively determine the recognizability of the resulting images, we measure how well this approach performs in conjunction with an image retrieval system [58]–[61]. Based on a query image, these systems are designed to retrieve similar appearing images from a large repository of images. The image-similarity system in [60], based on Min-Hash [62] and LSH [63], was used here without any modification.

The retrieval task was as follows: we used the container image, the reconstructed hidden image, and the second reconstructed image (for systems which hid 2 images) as the query images. The goal was to retrieve the **original** host, hidden-1 and hidden-2 images as the most similar match, respectively. The full database of images in the system included the ImageNet images, randomly selected images from photography sites on the web, and publicly available images from the Corel Database [64]. In total, there were approximately 2,650,000 images in the system. We measured the accuracy @1: *e.g.* is the correct image retrieved as the single top match (out of the 2,650,000) images. 1000 look-ups were conducted per experiment. As shown in Table 4, the scores were perfect for the majority of the experiments. For the single non-perfect experiment, only 1 out of 1000 retrievals did not work. This occurred on the most aggressive of the hiding procedures described in this paper: 2 images were hidden in a single network that were first obfuscated with non-stationary permutations. Finally, Table 5 shows the Peak Signal to Noise Ratio (PSNR) and the Structural Similarity Index (SSIM) [65], a perceptual metric to quantify image quality degradation between the original and reconstructed images.

## 5 Discussion and Future Work

We created a system composed of three deep neural networks to hide images within images. Embedding a full color image in another image of the same size necessitates devoting a large percentage of bits to the hidden image. However, through the compression performed by the deep-neural

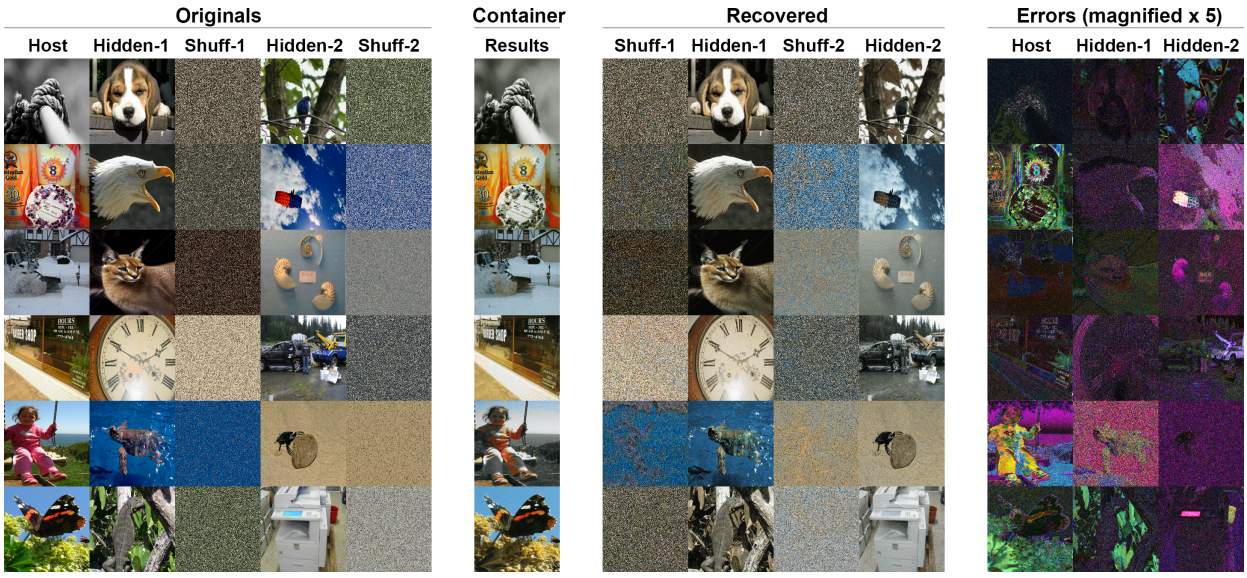


Fig. 14. Typical reconstructions when hiding 2 images within a single host image and perturbing the inputs to remove spatial coherency. The results show significant degradation both in the amount of noise and lack of saturation. The content of the images, including edges and gradients, remain easily recognizable and intact.

TABLE 4
Retrieval results for all reported experiments from a database of 2,650,000 images. Accuracy @1 reported. 1000 trials per experiment.

| | Container→ host | Reconstructed Hidden1→Hidden1 | Reconstructed Hidden2→Hidden2 |
|---|---|---|---|
| Hiding 1 Image (no permutation) | 100% | 100% | n/a |
| Hiding 1 Image using Single-Permutation | 100% | 100% | n/a |
| Hiding 1 Image using Multiple-Permutations | 100% | 100% | n/a |
| Hiding 2 Images (no permutation) | 100% | 100% | 100% |
| Hiding 2 Images using Single-Permutation | 100% | 100% | 100% |
| Hiding 2 Image using Multiple-Permutations | 99.9% | 100% | 100% |

TABLE 5
PSNR and SSIM Scores. 1000 images evaluated for each measurement.

| | Container vs. Host (PSNR,SSIM) | Reconstructed Hidden1 vs. Hidden1 (PSNR,SSIM) | Reconstructed Hidden2 vs. Hidden2 (PSNR,SSIM) |
|---|---|---|---|
| Hiding 1 Image (no permutation) | 41.2 , 0.98 | 37.6 , 0.97 | n/a |
| Hiding 1 Image using Single-Permutation | 38.2 , 0.97 | 37.9 , 0.97 | n/a |
| Hiding 1 Image using Multiple-Permutations | 37.6 , 0.97 | 37.3 , 0.96 | n/a |
| Hiding 2 Images (no permutation) | 36.7 , 0.96 | 35.9 , 0.96 | 35.6 , 0.95 |
| Hiding 2 Images using Single-Permutation | 32.8 , 0.92 | 33.2 , 0.92 | 32.0 , 0.91 |
| Hiding 2 Image using Multiple-Permutations | 33.6 , 0.92 | 34.1 , 0.92 | 33.1 , 0.92 |

networks, the decoded results of the hidden image, as well as the appearance of the host image that contains the hidden image, closely approximate the original hidden and host images, respectively. In the majority of examined cases, no visually discernible artifacts are seen.

Nonetheless, when hiding this much information (1:1 ratio of hidden to cover information), the detection of hidden information is likely unavoidable. The more problematic issue was, as demonstrated in Section 3, even without access to the unaltered host image, traces of the *content* of the hidden image could be discerned. Moreover, this could be accomplished through a variety of techniques, ranging from simple image transformations of the container image to more sophisticated machine learning based adversaries that use the statistics gathered from many similar (though not necessarily the same) deep neural network embedding systems. For the purposes of watermarking and image-alteration detection and authentication, knowing the content of the hidden information makes it easier to remove and/or reproduce it when alterations are made.

To address this shortcoming, we extended the hiding system in two directions. First, we demonstrated the ability for the system to encode scrambled images (with both stationary and non-stationary permutations). Because the neural networks no longer can exploit spatial coherency, this necessitated retraining all of the components of the hiding system. Encoding the hidden image without taking advantage of the spatial coherency of natural images required the network to devote more representational capacity to the hidden image. Therefore, the reconstruction errors for

both the hidden and host image increased. Qualitatively, the reconstructed host and hidden remained good. Importantly, without the permutation key, no visible traces of the hidden image could be ascertained.

Second, we extended the system to hide multiple (two) images. This required the neural network to hide twice as much information as the host media itself. As before, this required the network to devote more of its capacity to representing information from the hidden images than to the host image. The system showed a graceful degradation of performance with the increased difficulty of the task, and the host and both hidden images were easily identifiable with similar error rates to when hiding a permuted image. As a final test, both techniques were combined and multiple permuted images were hidden with non-stationary permutations. Though the errors increased dramatically over encoding a single non-permuted image, the resulting images remained readily identifiable, albeit with less fidelity to the colors, in particular saturation levels. To quantify the quality of images, image retrieval experiments were conducted with an externally built system. In all but one of the retrieval experiments, the encoded and reconstructed images remained intact enough to perform perfectly. In the single non-perfect experiment, a single mistake was made (out of 1000 look-ups).

In this work, all of the container images were losslessly encoded. For the current system, that is a requirement as, intuitively, the best place to encode the hidden is in the bits that would otherwise be freed in compression. Future work should include training the neural networks directly in the frequency domain; this may allow the use of these techniques with image compression standards, such as JPEG [66].

Future studies should also explore full cryptographic schemes for the hidden image. Though the images after scrambling appear noisy, structure still exists with respect to the underlying distribution of pixel colors. However, full encryption schemes will likely obfuscate the color distributions as well, thereby making the procedure nearly equivalent to representing and encoding uniform noise. The more random the hidden image, the more representational capacity and training time will be required.

In this study, the detection networks were trained after the complete encoding system was created. However, using pre-trained and/or simultaneously trained detection networks in an adversarial learning framework, for example by extending the work of [32], [33] to significantly larger hidden messages, may better conceal the hidden image. The adversary can be used to provide a supplemental error signal based upon the discoverability of the hidden message that is to be minimized in addition to the reconstruction errors.

Finally, in terms of applications, in this paper, we have concentrated on hiding information to defeat adversaries with malicious intent, such as stealing or altering images. However, embedding images within images has other areas of application as well. For example, when storing motion thumbnails for videos, we have extended our networks to hide motion vectors instead of hidden images. This results in the static content (the host image) being viewable on all standard platforms and web-browsers. However, when the same image is displayed by users who have installed the neural network-decoder, they see the extra content - the hidden motion.

## Acknowledgments

The image shown in Figure 15 is hidden in the author photograph at the end of this paper.

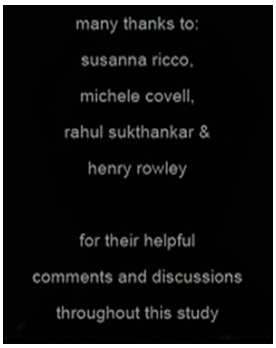


Fig. 15. This image will be revealed if the author's photograph (at the end of this article) is run through the decoder network. The text reads: "many thanks to: susanna ricco, michele covell, rahul sukthankar & henry rowley for their helpful comments and discussions throughout this study".

## References

[1] J. Fridrich and M. Goljan, "Practical steganalysis of digital images: State of the art," in *Electronic Imaging 2002*. International Society for Optics and Photonics, 2002, pp. 1–13.

[2] D. McCullagh, "Bin laden: steganography master?" *Wired News*, vol. 7, 2001.

[3] G. Goth, "Steganalysis gets past the hype," *IEEE Distributed Systems Online*, vol. 6, no. 4, p. 2, 2005.

[4] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker, *Digital watermarking and steganography*. Morgan Kaufmann, 2007.

[5] A. K. Jain and U. Uludag, "Hiding biometric data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 25, no. 11, pp. 1494–1498, 2003.

[6] W. F. Friedman, *An Introduction to Methods for the Solution of Ciphers*. Riverbank Laboratories, 1918.

[7] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, "Digital image steganography: Survey and analysis of current methods," *Signal processing*, vol. 90, no. 3, pp. 727–752, 2010.

[8] N. Provos and P. Honeyman, "Hide and seek: An introduction to steganography," *IEEE security & privacy*, vol. 99, no. 3, pp. 32–44, 2003.

[9] F. Yaghmaee and M. Jamzad, "Estimating watermarking capacity in gray scale images based on image complexity," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, no. 1, p. 851920, 2010.

[10] J. Fridrich, M. Goljan, and R. Du, "Detecting lsb steganography in color, and gray-scale images," *IEEE multimedia*, vol. 8, no. 4, pp. 22–28, 2001.

[11] A. A. Tamimi, A. M. Abdalla, and O. Al-Allaf, "Hiding an image inside another image using variable-rate steganography," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 4, no. 10, 2013.

[12] T. Pevnỳ, T. Filler, and P. Bas, "Using high-dimensional image models to perform highly undetectable steganography," in *International Workshop on Information Hiding*. Springer, 2010, pp. 161–177.

[13] M. Yedroudj, F. Comby, and M. Chaumont, "Yedrouj-net: An efficient cnn for spatial steganalysis," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'2018*, 2018.

[14] H. Ozer, I. Avcibas, B. Sankur, and N. D. Memon, "Steganalysis of audio based on audio quality metrics," in *Electronic Imaging 2003*. International Society for Optics and Photonics, 2003, pp. 55–66.

[15] G. C. Kessler and C. Hosmer, "An overview of steganography," *Advances in Computers*, vol. 83, no. 1, pp. 51–107, 2011.

[16] G. C. Kessler, "An overview of steganography for the computer forensics examiner," *Forensic Science Communication*, vol. 6, no. 3, 2014.

[17] ——, "An overview of steganography for the computer forensics examiner (web)," 2015. [Online]. Available: http://www.garykessler.net/library/fsc_stego.html

[18] J. Parikka, "Hidden in plain sight: The stagnographic image," https://unthinking.photography/themes/fauxtography/hidden-in-plain-sight-the-steganographic-image, 2017.

[19] F. Jessica, J. Kodovskỳ, V. Holub, and M. Goljan, "Breaking hugo–the process discovery," in *International Workshop on Information Hiding*. Springer, 2011, pp. 85–101.

[20] M. Chaumont and W. Puech, "Protecting the color information by hiding it," in *Recent Advances in Signal Processing*. InTech, 2009.

[21] K. Hayat, W. Puech, G. Gesquière, and M. Chaumont, "Wavelet-based data hiding of dem in the context of real-time 3d visualization," in *Visualization and Data Analysis 2007*. International Society for Optics and Photonics, 2007.

[22] M. Chaumont, W. Puech, and C. Lahanier, "Securing color information of an image by concealing the color palette," *Journal of Systems and Software*, vol. 86, no. 3, pp. 809–825, 2013.

[23] Y. Qian, J. Dong, W. Wang, and T. Tan, "Deep learning for steganalysis via convolutional neural networks," in *SPIE/IS&T Electronic Imaging*. International Society for Optics and Photonics, 2015, pp. 94 090J–94 090J.

[24] L. Pibre, J. Pasquet, D. Ienco, and M. Chaumont, "Deep learning is a good steganalysis tool when embedding key is reused for different images, even if there is a cover sourcemismatch," *Electronic Imaging*, vol. 2016, no. 8, pp. 1–11, 2016.

[25] R. Zhang, F. Zhu, J. Liu, and G. Liu, "Efficient feature learning and multi-size image steganalysis based on cnn," *arXiv preprint arXiv:1807.11428*, 2018.

[26] S. Husien and H. Badi, "Artificial neural network for steganography," *Neural Comp. and Applications*, vol. 26, no. 1, pp. 111–116, 2015.

[27] I. Khan, B. Verma, V. K. Chaudhari, and I. Khan, "Neural network based steganography algorithm for still images," in *Emerging Trends in Robotics and Comm. Tech.* IEEE, 2010, pp. 46–51.

[28] V. Kavitha and K. Easwarakumar, "Neural based steganography," *PRICAI 2004: Trends in Artificial Intelligence*, pp. 429–435, 2004.

[29] A. S. Brandao and D. C. Jorge, "Artificial neural networks applied to image steganography," *IEEE Latin America Transactions*, vol. 14, no. 3, pp. 1361–1366, 2016.

[30] R. Jarušek, E. Volna, and M. Kotyrba, "Neural network approach to image steganography techniques," in *Mendel 2015*. Springer, 2015, pp. 317–327.

[31] R. Jarusek, E. Volna, and M. Kotyrba, "Robust steganographic method based on unconventional approach of neural networks," *Applied Soft Computing*, vol. 67, pp. 505–518, 2018.

[32] J. Hayes and G. Danezis, "Generating steganographic images via adversarial training," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1951–1960.

[33] D. Volkhonskiy, I. Nazarov, B. Borisenko, and E. Burnaev, "Steganographic generative adversarial networks," *CoRR*, vol. abs/1703.05502, 2017. [Online]. Available: http://arxiv.org/abs/1703.05502

[34] H. Shi, J. Dong, W. Wang, Y. Qian, and X. Zhang, "Ssgan: Secure steganography based on generative adversarial networks," in *Pacific Rim Conference on Multimedia*. Springer, 2017, pp. 534–544.

[35] D. Hu, L. Wang, W. Jiang, S. Zheng, and B. Li, "A novel image steganography method via deep convolutional generative adversarial networks," *IEEE Access*, vol. 6, pp. 38 303–38 314, 2018.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Adv. in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[37] S. Baluja, "Hiding images in plain sight: Deep steganography," in *Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., 2017, pp. 2066–2076.

[38] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[39] P. Baldi and K. Hornik, "Neural networks and principal component analysis," *Neural networks*, vol. 2, no. 1, pp. 53–58, 1989.

[40] G. W. Cottrell and P. Munro, "Principal components analysis of images via back propagation," in *Visual Communications and Image Processing'88: Third in a Series*, vol. 1001. International Society for Optics and Photonics, 1988, pp. 1070–1078.

[41] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[42] J. Jiang, "Image compression with neural networks–a survey," *Signal Proc.: Image Communication*, vol. 14, no. 9, pp. 737–760, 1999.

[43] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," *arXiv preprint arXiv:1802.01436*, 2018.

[44] O. Rippel and L. Bourdev, "Real-time adaptive image compression," in *Int. Conference on Machine Learning (ICML), 2017*. IEEE, 2017.

[45] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," in *International Conference on Learning Representations*, 2017.

[46] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *Int'l. Conf. on Learning Representations (ICLR2017)*, April 2017. [Online]. Available: https://arxiv.org/abs/1611.01704

[47] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *JMLR*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[48] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[49] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[50] B. Boehm, "Stegexpose - A tool for detecting LSB steganography," *CoRR*, vol. abs/1410.6656, 2014. [Online]. Available: http://arxiv.org/abs/1410.6656

[51] "Stegexpose - github," https://github.com/b3dk7/StegExpose.

[52] darknet.org.uk, "Stegexpose – steganalysis tool for detecting steganography in images," https://www.darknet.org.uk/2014/09/stegexpose-steganalysis-tool-detecting-steganography-images/, 2014.

[53] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[54] D. Van De Ville, W. Philips, R. Van de Walle, and I. Lemahieu, "Image scrambling without bandwidth expansion," *IEEE Trans. on Circuits and Sys. for Video Technology*, vol. 14, no. 6, pp. 892–897, 2004.

[55] A. B. L. Larsen, S. K. Sønderby, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," *arXiv:1512.09300*, 2015.

[56] S. Li, C. Li, K.-T. Lo, and G. Chen, "Cryptanalysis of an image scrambling scheme without bandwidth expansion," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 3, pp. 338–349, 2008.

[57] J. Fridrich, M. Goljan, and D. Soukal, "Searching for the stego-key," in *Proceedings of SPIE*, vol. 5306, 2004, pp. 70–82.

[58] A. Qamra, Y. Meng, and E. Y. Chang, "Enhanced perceptual distance functions and indexing for image replica recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 3, pp. 379–391, 2005.

[59] G. Wang, D. Hoiem, and D. Forsyth, "Learning image similarity from flickr groups using fast kernel machines," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2177–2188, 2012.

[60] S. Baluja and M. Covell, "Waveprint: Efficient wavelet-based audio fingerprinting," *Pattern recognition*, vol. 41, no. 11, pp. 3467–3480, 2008.

[61] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2916–2929, 2013.

[62] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, "Finding interesting associations without support pruning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 64–78, 2001.

[63] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.

[64] D. Tao, X. Tang, X. Li, and X. Wu, "Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 7, pp. 1088–1099, 2006.

[65] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[66] J. Fridrich, T. Pevnỳ, and J. Kodovskỳ, "Statistically undetectable jpeg steganography: dead ends challenges, and opportunities," in *Proceedings of the 9th workshop on Multimedia & security*. ACM, 2007, pp. 3–14.

**Shumeet Baluja** is with Google, Inc., currently working on machine learning and computer vision. Previously, Shumeet was Chief Scientist at Lycos Inc., CTO of Jamdat Mobile, and SVP of R&D at eCompanies. Shumeet has published in numerous fields including computer vision and facial image processing, advertisement optimization, autonomous driving, machine learning, and high-dimensional optimization. He received his Ph.D. in computer science from Carnegie Mellon in 1996.