StegaStamp: Invisible Hyperlinks in Physical Photographs Supplement

1. StegaStamp Examples

See Figure 1 for additional examples of encoded images and their residuals.

2. Supplemental Videos

https://youtu.be/E8OqgNDBGO0

This video provides an overview of StegaStamps with example use cases and a condensed demonstration of in-the-wild results.

https://youtu.be/jpbRhOH3D9Y

This video is a compilation of multiple in-the-wild captures. The first set of clips visualizes the output bounding polygons along with the percentage of bits recovered correctly out of 100. We filter the output to only show detections where the bit accuracy is greater than 70 percent. We note that the messages are regularly recovered with greater than 90% accuracy when they are properly detected. The second set of clips demonstrates the used of BCH error correction [2] to robustly detect and correct recovered codes. The transmitted data consists of 56 message bits and 40 error correcting bits. When the accuracy is greater than 95% (fewer than 5 corrupted bits), the original 56-bit message can be recovered exactly. If too many bits are corrupted, the error correcting fails and we filter out the proposal. The video represents successfully decoded StegaStamps with green polygons. The decoded code is printed above the polygon. Note that for most real world applications, it is only necessary to recover the code in a single video frame to count it as successfully scanned.

3. Comparison Details

We compare our method to Baluja [1], HiDDeN [5], and LFM [3]. Baluja was designed to hide images within images, which differs from our task of hiding a bitstring within an image. To account for this, we convert our 100 bit message into a 10×10 grid of ones and zeros that is upscaled to the resolution of the cover image. During decoding we round the model output to 0 and 1 and take the mode within each upscaled block. As the original model was trained to hide natural images, we retrain the model from scratch to hide our bitstring grids.

		Mean Acc. ↑	bits/MP ↑
	Baluja [1]	0.51	0.5
	HiDDeN [5]	0.65	125
	LFM [3] (printed)	0.61	287
	LFM [3] (screen)	0.93	1109
Ours	None	0.49	0.1
Ours	Pixelwise	0.51	0.2
Ours	Spatial	0.89	318
Ours	All	0.99	571

Table 1: Quantitative comparison of other methods and our ablations. We show numbers in terms of fraction of bits correctly recovered (mean accuracy) as well as bits-per-megapixel (bits/MP). Higher is better for both metrics. The bits/MP metric normalizes the message length and image sizes between different methods. All methods except "LFM [3] (screen)" (cellphone camera/cellphone screen) are reported on the cellphone camera/consumer printer pipeline. We report LFM's results in this additional case because it was explicitly designed for screen/camera transmission.

HiDDeN was trained to hide 30 bit messages in 128×128 pixel images. We observed a significant drop in accuracy when we trained a model to hide 100 bit messages in 400 pixel images, therefore we report accuracy results on the 30 bit in 128^2 image version.

LFM [3] was trained to encode 1024 bit messages as 4×4 pixel blocks in a 256×256 pixel image. To encode our 100 bit message, we allocated 9 blocks for each message bit (we therefore only use a 244×244 pixel subset of the image). We average and round the 9 block predictions to recover the message bit.

Each compared method encodes a different length message into a different size image. However, if we treat the mean bit recovery accuracy (first column in Table 1) as the crossover probability p in a binary symmetric channel, we can use information theory to calculate the channel capacity

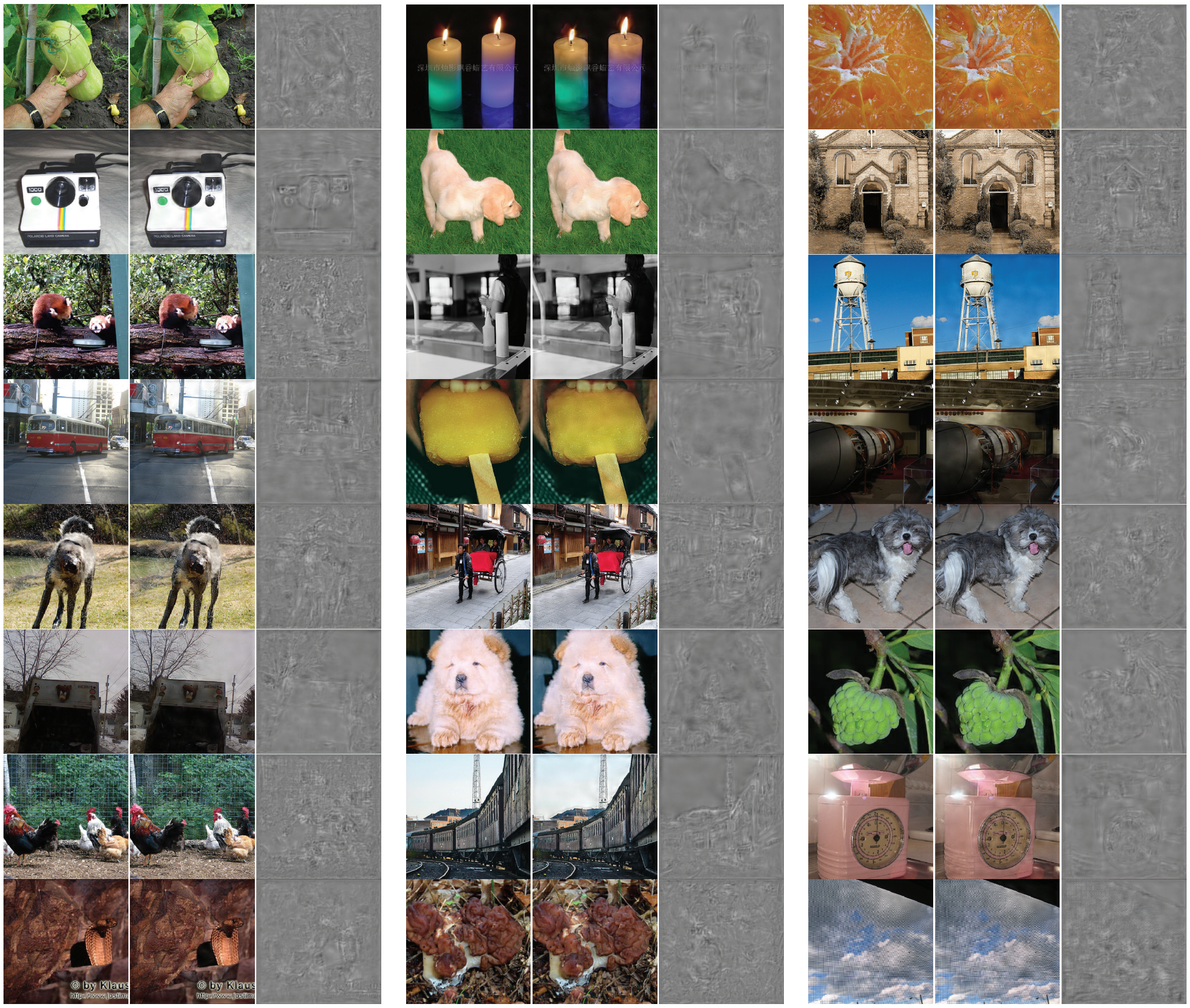

Figure 1: Additional examples of encoded images and their residuals.

(with unit "bits"):

$$C(p) = 1 - (-p \log_2 p - (1-p) \log_2 1 - p) \qquad (1)$$

If we divide $C(p)$ by the number of pixels N_{pix} in the original image, we get the expected number of bits-per-pixel transmitted by that method. Multiplying $\frac{C(p)}{N_{pix}}$ by 10^6 yields our bits-per-megapixel metric in the second column of Table 1.

4. Architecture Details

Network architectures for our encoder (Table 3) and decoder (Table 4). Our detector uses the BiSeNet [4] architecture.

	PSNR ↑	SSIM ↑	LPIPS ↓
Baluja [1]	24.61	0.926	0.256
HiDDeN [5] (native)	31.07	0.940	0.070
HiDDeN [5]	24.55	0.775	0.202
LFM [3]	20.89	0.910	0.315
Ours	27.25	0.927	0.194

Table 2: Quantitative comparison of encoded image quality, indicating how well hidden the message is. For HiDDeN [5] we show both the metrics for the original lower resolution (native 128×128) and upsampling to our compared resolution of 400×400 with bicubic interpolation. At full resolution, our method produces an encoded image most similar to the original in all metrics.

Layer	k	s	chns	in	out	input
inputs			6			image + secret
conv1	3	1	6/32	1	1	inputs
conv2	3	2	32/32	1	2	conv1
conv3	3	2	32/64	2	4	conv2
conv4	3	2	64/128	4	8	conv3
conv5	3	2	128/256	8	16	conv4
up6	2	1	256/128	16	8	upsample(conv5)
conv6	3	1	256/128	8	8	conv4 + up6
up7	2	1	128/64	8	4	upsample(conv6)
conv7	3	1	128/64	4	4	conv3 + up7
up8	2	1	64/32	4	2	upsample(conv7)
conv8	3	1	64/32	2	2	conv2 + up8
up9	2	1	32/32	2	1	upsample(conv8)
conv9	3	1	70/32	1	1	conv1 + up9 + inputs
conv10	3	1	32/32	1	1	conv9
residual	1	1	32/3	1	1	conv10

Table 3: Our encoder network architecture. **k** is the kernel size, **s** the stride, **chns** the number of input and output channels for each layer, **in** and **out** are the accumulated stride for the input and output of each layer, **input** denotes the input of each layer with + meaning concatenation and "upsample" performing $2\times$ nearest neighbor upsampling. A ReLU is applied after each layer except the last.

Layer	k	s	chns	in	out	input
conv1	3	2	3/32	1	2	image
conv2	3	2	32/64	2	4	conv1
conv3	3	2	64/128	4	8	conv2
fc0			320000			flatten(conv3)
fc1			320000/128			fc0
fc2			128/6			fc1
image_warped			3/3			transf(image, fc2)
conv1	3	2	3/32	1	2	image_warped
conv2	3	1	32/32	2	2	conv1
conv3	3	2	32/64	2	4	conv2
conv4	3	1	64/64	4	4	conv3
conv5	3	2	64/64	4	8	conv4
conv6	3	2	64/128	8	16	conv5
conv7	3	2	128/128	16	32	conv6
fc0			20000			flatten(conv7)
fc1			20000/512			fc0
secret			512/100			fc1

Table 4: Our decoder network architecture. We indicate convolutional layers with the prefix "conv" and fully connected layers with the prefix "fc." The first half of the network outputs an affine warp that is applied using a differentiable spatial transformer layer ("transf"). The warped result is fed into the second part of the network. A ReLU is applied after each layer except the last layer before the spatial transformer.

5. Code

The code and pretrained networks can be found at https://github.com/tancik/StegaStamp.

References

[1] Shumeet Baluja. Hiding images in plain sight: Deep steganography. In *NeurIPS*, 2017. 1, 2

[2] Raj Chandra Bose and Dwijendra K Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 1960. 1

[3] Eric Wengrowski and Kristin Dana. Light field messaging with deep photographic steganography. In *CVPR*, 2019. 1, 2

[4] Changqian Yu, Jingbo Wang, Chao Peng, Changxin Gao, Gang Yu, and Nong Sang. Bisenet: Bilateral segmentation network for real-time semantic segmentation. In *ECCV*, 2018. 2

[5] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *ECCV*, 2018. 1, 2