

Sticking Information in Plain Sight: Encoding and Detecting Hidden Stickers in the Real World

Supplementary Material

Christina Shatford[✉] and Szymon Rusinkiewicz[✉]

Princeton University, Princeton NJ 08544, USA
{shatford, smr}@princeton.edu

1 Introduction

This supplementary material provides additional visual results of our method, extending what was presented in Section 4. It also provides additional information on the network architecture of our encoder and decoder.

2 Simulation Examples

Fig. 1 presents examples of stickers being added to scenes in simulation.

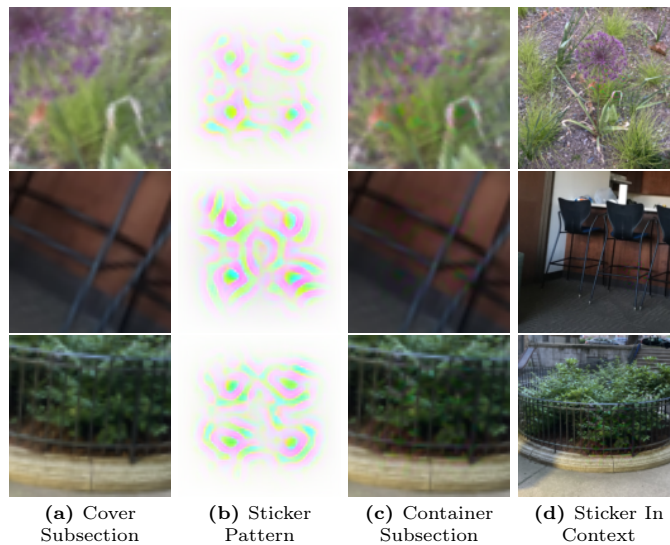


Fig.1: Examples of our method in simulation. (a) Cover subsection used to generate a sticker. (b) Sticker pattern created given cover subsection and random 100-bit secret. (c) Cover subsection with sticker applied to it. (d) Full container image, including the container subsection. All stickers in (d) are able to be localized and decoded with 95% accuracy or better.

3 Real-World Dataset

Our real-world dataset¹ evaluates stickers placed onto a variety of surfaces, under varying illumination and imaging conditions. As an overview, the material types present are: plastic, mesh, plush fabric, taut fabric, loose fabric, paper, porcelain, weathered wood, finished wood, cardboard, glass, wallpaper, painted walling, marble, horse hair plaster, linoleum, and manufactured countertops. The most prevalent surface colors are: green, teal, gray, red, tan, white, brown, yellow, pink, purple, blue, striped blue and white, and varied patterns of 3 or more colors. Fig. 2 presents samples of the different conditions under which the dataset was constructed, while Fig. 3 displays a larger sample of 56 out of the 84 images in our real-world dataset. Note that every real-world container image has misalignment errors. Each sticker was placed over the expected cover subsection with varying levels of accuracy. A benefit of our method is how robust it is to placement error; in practice, the user does not have to carefully match the placement of the sticker to the exact cover section it was optimized for.

¹ <https://github.com/cshatford/SIIPS-RWDataset.git>

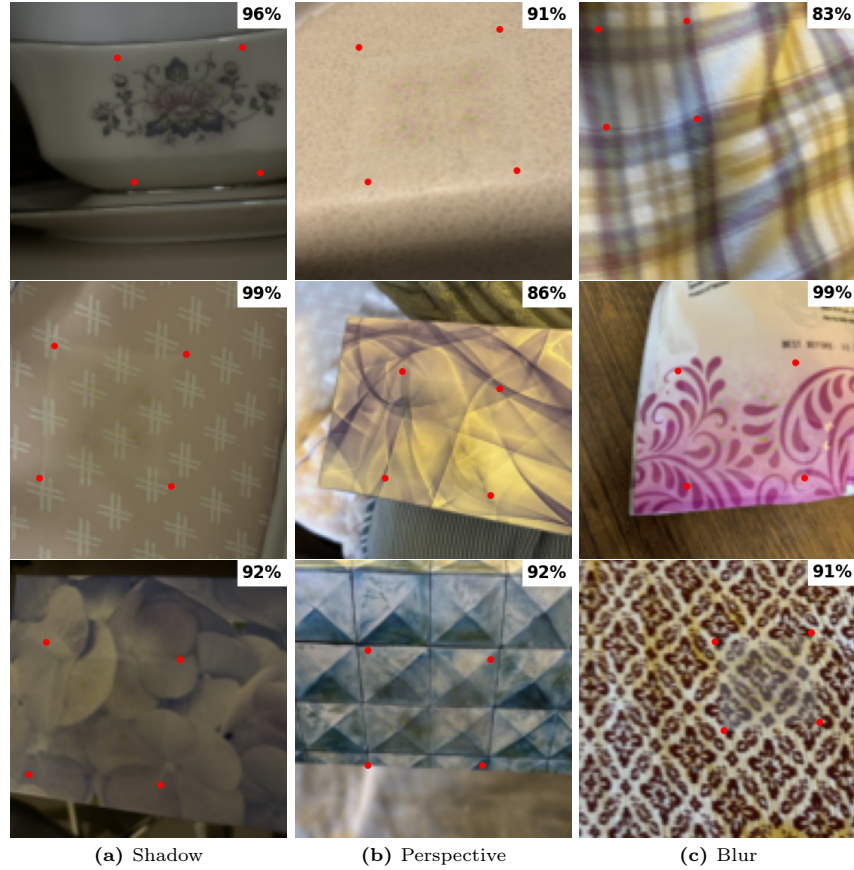


Fig. 2: Varying illumination and imaging conditions. Our real-world test set contains images that test our pipeline under a variety of imaging conditions. (a) Examples of images taken where lighting conditions introduce shadows. (b) Example of images taken from different perspectives. (c) Examples of blurred images.

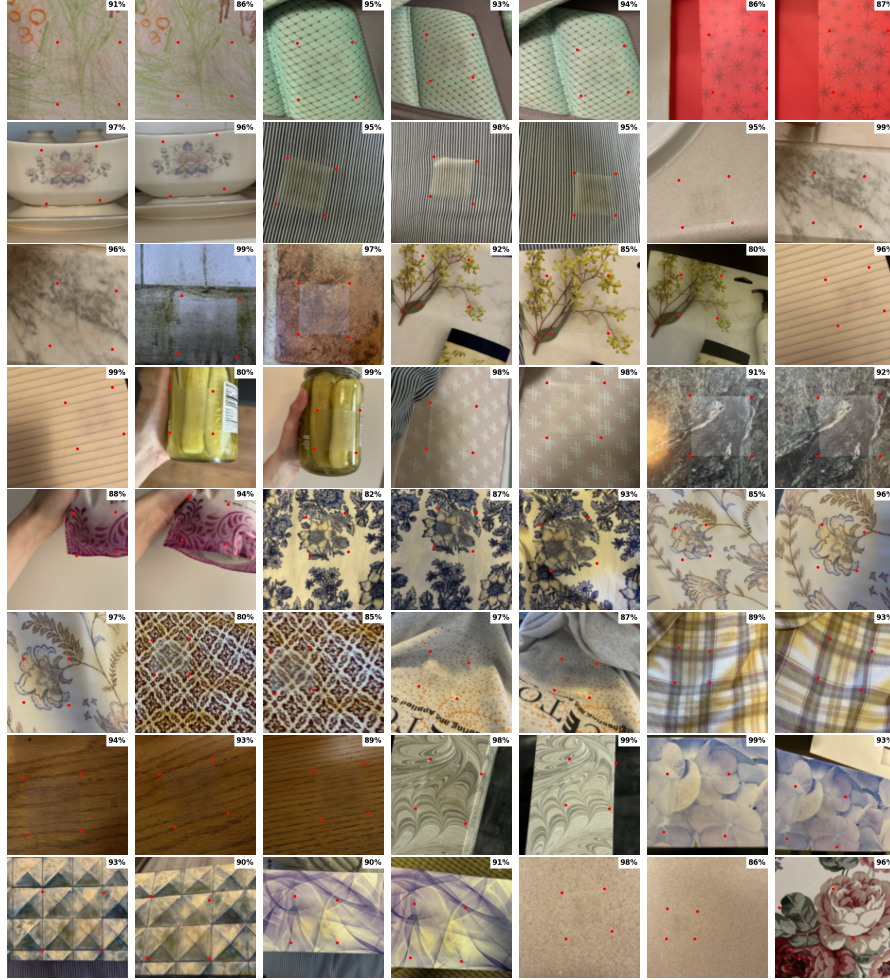


Fig. 3: Representative examples from our real-world dataset. The predicted corner locations of each sticker are shown in red, and the resulting decoding accuracy of sending the predicted location through our decoder is shown in the upper right corner of each image.

3.1 Occlusion

Fig. 4 presents additional images from our real-world dataset of stickers with occlusion added to them in the form of either real-world or digital objects. See Fig. 5 for the three instances in the dataset where either a sticker was unable to be detected, or the localization was poor.

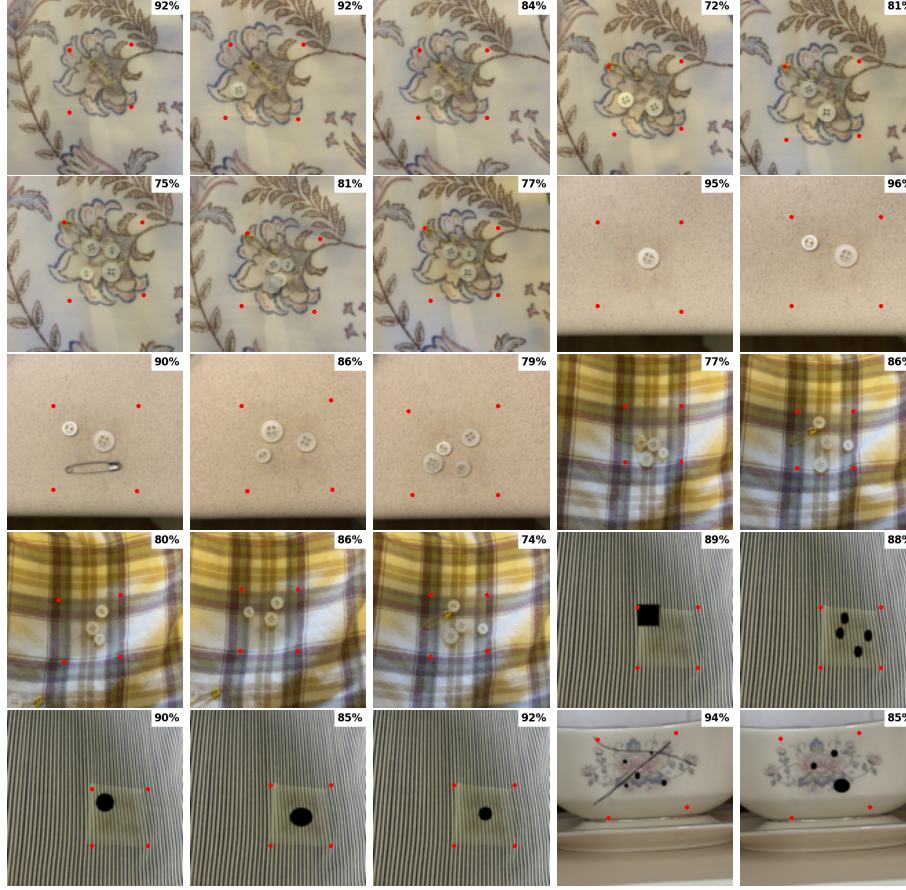


Fig. 4: Occlusion. Examples of localized and decoded stickers when occlusion is present. Despite the fact that our method was not explicitly trained to handle occlusion, it nevertheless detects and decodes the real-world sticker in all of these cases. See Fig. 5 for failure cases.

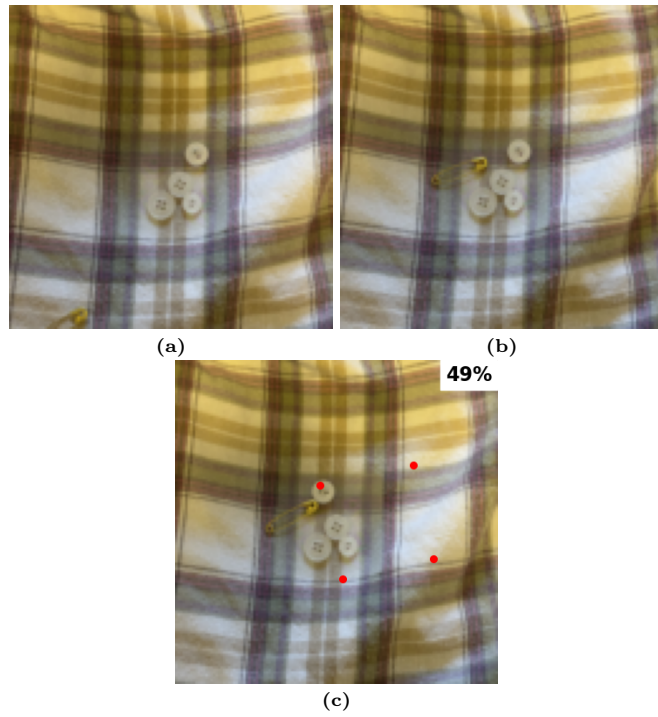


Fig. 5: Failures with occlusion. (a)-(b) Two instances where the occluded sticker was not detected in the container image. (c) An instance where the occluded sticker was poorly localized and decoded.

3.2 Wrinkles

Fig. 6 presents additional images from our real-world dataset of wrinkled stickers applied to surfaces. In two cases, shown in Fig. 7, the sticker was either poorly decoded, or not detected.

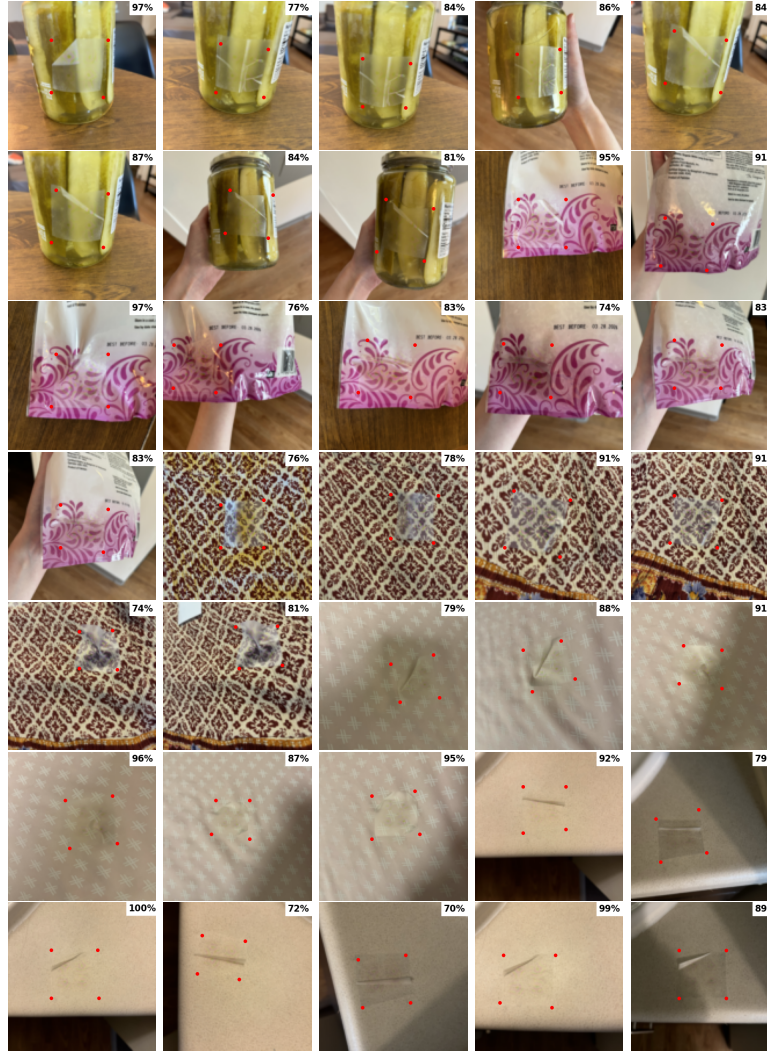


Fig. 6: Wrinkling. Examples of localized and decoded stickers when the stickers were wrinkled during application. Despite the fact that our method was not explicitly trained to handle wrinkled stickers, it nevertheless detects and decodes the real-world sticker in all of these cases. See Fig. 7 for failure cases.

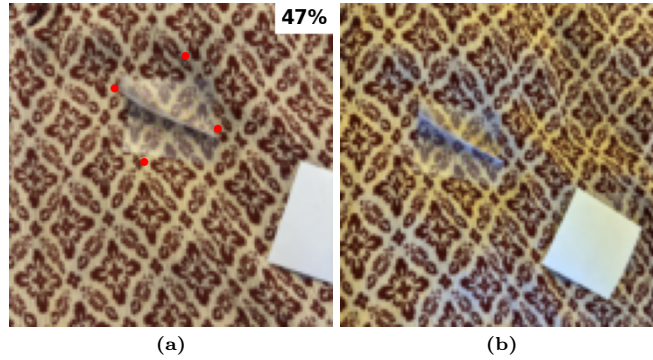


Fig. 7: Failures with wrinkling. (a) An instance where a wrinkled sticker was poorly decoded. (b) An instance where a wrinkled sticker was not detected in the container image.

4 Impact of Background Choice

In Fig. 8, we show examples of poor sticker-background pairs, that either arise from a background surface’s pattern or texture.

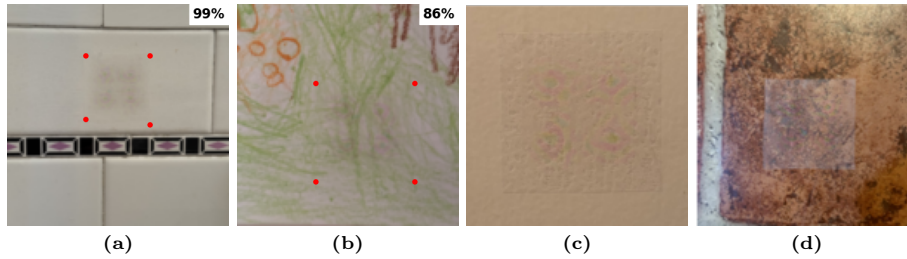


Fig. 8: Examples of poor sticker-background pairs, due to background pattern (top row) or background texture (bottom row). In (a), the pattern is more visible. In (b), color strokes in the background are similar to part of the pattern, resulting in a lower decode accuracy of 86%. In (c), the background surface is not completely flat, resulting in air bubbles under the sticker, making its appearance more noticeable. In (d), the finish of the background does not blend well with the sticker paper.

5 Multi-Sticker Detection

Our approach allows for multiple stickers to be found in an image. See Fig. 9 for examples of localization and detection when multiple stickers are included in an image.

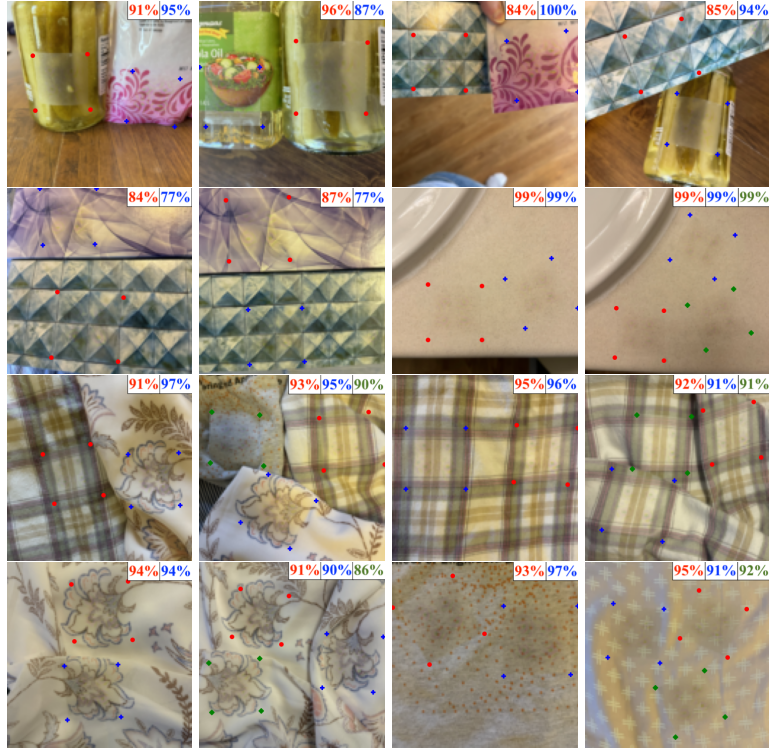


Fig. 9: Multi-sticker examples. The predicted corners are marked, with markers of the same color and shape predicted to belong to the same sticker. Our multi-sticker detection system was used to determine which corners belong to which stickers. The resulting decoding accuracies are in the upper right corner of each image.

6 Architecture

We use the same general architecture for our encoder and decoder as [4], with some small changes. Tab. 1 gives the architecture for the encoder: note that it is a UNet [2]. Tab. 2 presents the architecture for our decoder.

Table 1: Encoder architecture. All convolutional layers, noted as conv and up, have a stride of 1. When Maxpool is ‘Yes,’ MaxPooling is applied and the kernel size and stride are both 2. The orthog layer is a linear layer with an orthogonal parameterization. The input ‘inputs’ is created by reshaping and then resizes orthog, using nearest interpolation, to be the same dimensions as cover, then concatenating it with the cover image. The function upsample(layer) applies 2D nearest neighbor upsampling with a scale factor of 2 to layer. The operator + denotes concatenation.

Layer	Kernel Size	Channel In	Channel Out	Padding	Activation	MaxPool	Input
orthog		100	300			No	secret
conv1	3	6	32	‘same’	ReLU	No	inputs
conv2	3	32	32	‘same’	ReLU	Yes	conv1
conv3	3	32	64	‘same’	ReLU	Yes	conv2
conv4	3	64	128	‘same’	ReLU	Yes	conv3
conv5	3	128	256	‘same’	ReLU	Yes	conv4
up6	2	256	128	‘same’	ReLU	No	upsample(conv5)
conv6	3	256	128	‘same’	ReLU	No	conv4 + up6
up7	2	128	64	‘same’	ReLU	No	upsample(conv6)
conv7	3	128	64	‘same’	ReLU	No	conv3 + up7
up8	2	64	32	‘same’	ReLU	No	upsample(conv7)
conv8	3	64	32	‘same’	ReLU	No	conv2 + up8
up9	2	32	32	‘same’	ReLU	No	upsample(conv8)
conv9	3	70	32	‘same’	ReLU	No	conv1 + up9 + inputs
rgb	1	32	3	0	Sigmoid	No	conv9
alpha	1	32	1	0	Sigmoid	No	conv9

Table 2: Decoder architecture. All convolutional layers have a stride of 1. Note that since we use BCE with logits loss, the decoder does not apply a Sigmoid activation after Linear2. However, a Sigmoid activation should be applied to the output of the decoder to get the decoded secret.

Layer	Kernel Size	Channel In	Channel Out	Padding	Activation	MaxPool	Input
conv1	3	3	32	‘same’	ReLU	Yes	image
conv2	3	32	32	‘same’	ReLU	No	conv1
conv3	3	32	64	‘same’	ReLU	Yes	conv2
conv4	3	64	64	‘same’	ReLU	No	conv3
conv5	3	64	64	‘same’	ReLU	Yes	conv4
conv6	3	64	128	‘same’	ReLU	Yes	conv5
conv7	3	128	128	‘same’	ReLU	Yes	conv6
linear1		512	1280		ReLU	No	flatten(conv7)
linear2		1280	100			No	linear1

6.1 Encoder

Given a $3 \times 128 \times 128$ background image and a location to place a sticker in that image, we take the image subsection that corresponds to where the sticker should be placed. The subsection is resized to $3 \times 64 \times 64$ and passed to our encoder along with a message. We call the image subsection the *cover* subsection.

The first layer of our encoder is a fixed linear layer with an orthogonal parameterization to which the 100-bit secret is passed. The output is unflattened to be $3 \times 10 \times 10$, interpolated to be the same size as the cover subsection, and concatenated with the cover subsection. That tensor is then passed through the UNet. The output of the encoder is ultimately a $3 \times 64 \times 64$ RGB tensor, rgb_s and a $1 \times 64 \times 64$ alpha tensor, $alpha_s$. The RGB and alpha tensors are resized to be the same shape as the original cover subsection. Next, the RGB tensor is blended with the original cover subsection, using the corresponding alpha tensor values and the following model for absorption of the different colors of light through the sticker:

$$(c_s, m_s, y_s) = 1 - (r_s, g_s, b_s) \quad (1)$$

$$r_{final} = r_c * (1 - (alpha_s * c_s)) \quad (2)$$

$$g_{final} = g_c * (1 - (alpha_s * m_s)) \quad (3)$$

$$b_{final} = b_c * (1 - (alpha_s * y_s)), \quad (4)$$

where r_c, g_c, b_c denote the RGB values of the cover subsection prior to being blended with the sticker, and $r_{final}, g_{final}, b_{final}$ are the simulated color values of the cover subsection blended with the pattern returned by the encoder. We then paste the newly encoded container subsection into the background image at the assigned location. We have verified that this formula predicts the observed color of a printed sticker placed over a real-world background with reasonable accuracy.

Note that the encoder network is designed to work for all cover images and all messages, but a single instance of running the network customizes the pattern for a particular cover image and message.

To simulate human placement error, we introduce jitter between the location of the image subsection that the sticker is generated for, and the location that the sticker is blended with. We do this by performing a random affine transformation on the background image before taking the cover subsection. We randomly shift the background image by $\pm 0.05 \cdot 128$ pixels in each of the horizontal and vertical directions. We also randomly rotate the background image by ± 5 degrees. Then we crop around the original parameterized location of the sticker and resize it appropriately before passing it to the encoder, as described above. Finally, we blend the generated sticker with the un-jittered cover subsection. Thus, with this placement jitter, we force our model to be robust to sticker misalignment. We found that when we did not include this simulated human placement error, performance suffered.

6.2 Localizer

Before being passed through the localizer, a container image has a chance of a small random perspective warp being performed. The image also has a chance of being augmented to include random changes to brightness, contrast, and saturation, all between $\pm 50\%$; Gaussian blur with sigma between 0.1 and 1; and/or random Gaussian noise. Each type of augmentation independently has a 75% chance of being added to an image.

For localization, inspired by the localization network in [1], we use a variant of HRNet [3, 5, 6]. We removed the last convolutional layer of the network, such that the network takes a $3 \times 128 \times 128$ tensor as input and outputs a $32 \times 32 \times 32$ tensor. That tensor is then fed to the localization head.

The localizer is trained to output a heatmap for each of the predicted corners of the encoded image. Thus, it produces a $4 \times 32 \times 32$ tensor. To get the predicted coordinate values from these heatmaps, we upsample them to be $4 \times 128 \times 128$ and then take the maximum value along each of the 4 channels. If those maximum values pass our minimum threshold, we consider a corner to have been found. If all four corners have been found, then we perform a perspective warp to make the predicted subsection $3 \times 64 \times 64$. We compute the ground truth heatmaps by taking a Gaussian distribution around each of the ground truth corner locations. If 4 valid corners have not been predicted, then a valid sticker was not found in the image.

6.3 Decoder

The decoder network takes in a $3 \times 64 \times 64$ subsection and outputs 100 decoded bits. These bits are then compared to the original message bits encoded in the subsection.

7 Hyperparameter Analysis

Recall that training takes place in three stages and the loss function we use is:

$$L = \lambda_1 L_{gtLocDecode} + \lambda_2 L_{heat} + \lambda_3 L_{predLocDecode} + \lambda_4 L_{cover}. \quad (5)$$

We conducted two experiments where we started training from scratch using stage two hyperparameter weights, or using stage three weights. In both cases, the training ended with the model producing blank stickers. Accordingly, in stage one, only λ_1 is non-zero, because those experiments showed that setting other weights to non-zero values would result in blank stickers. Thus, it was necessary that the model only optimize for decoding accuracy (at the ground truth sticker location) in the first stage.

In stage two, λ_4 increases from 0.01 to 40 in steps, roughly doubling at each step of 10-50 epochs. This gradual increase scheme is necessary because setting λ_4 to a larger number from the start placed too much emphasis on the cover loss, resulting in fully blank stickers and random decode accuracy.

In stage three, we roughly balance the loss components, so λ_2 is introduced as a seemingly large number, 10000, because the localization loss drops quickly and becomes very small, needing a large weight to balance it with the other losses. Thus, λ_2 eventually being set to 100000 does not diminish the training effectiveness of other loss components. λ_3 is also introduced in stage three because we found that it led to better real-world results when the model was optimized to decode based on the predicted sticker location, in addition to the ground truth location. λ_1 and λ_3 start out equal with values of 1.0, and then λ_1 is set to 0.2 and λ_3 to 2.0, once sticker localization is relatively accurate, placing slightly more emphasis on the decoding accuracy at the predicted location.

8 Real-World Encoding/Decoding Pipeline

In Fig. 10, we present pipeline diagrams of how encoding, placing, and decoding a sticker works in the real world.

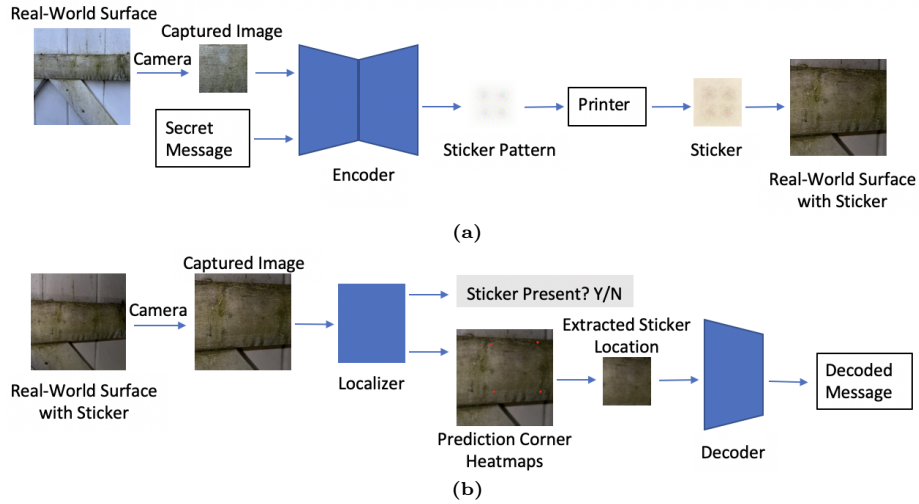


Fig. 10: Real-World Pipelines. Real world processes for (a) generating an encoded sticker given a secret message and a surface and (b) capturing an image of a surface with a sticker and decoding the secret message. Note that the location a sticker is generated for is not necessarily the exact location that the sticker is placed, and imaging conditions might change between sticker generation and placement.

References

1. Jia, J., Gao, Z., Zhu, D., Min, X., Zhai, G., Yang, X.: Learning invisible markers for hidden codes in offline-to-online photography. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2263–2272 (2022). <https://doi.org/10.1109/CVPR52688.2022.00231>
2. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. pp. 234–241. Springer International Publishing (2015)
3. Sun, K., Xiao, B., Liu, D., Wang, J.: Deep high-resolution representation learning for human pose estimation. In: CVPR (2019)
4. Tancik, M., Mildenhall, B., Ng, R.: Stegastamp: Invisible hyperlinks in physical photographs. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020)
5. Wang, J., ke, S., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., Liu, W., Xiao, B.: Deep high-resolution representation learning for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PP**, 1–1 (04 2020). <https://doi.org/10.1109/TPAMI.2020.2983686>
6. Xiao, B., Wu, H., Wei, Y.: Simple baselines for human pose estimation and tracking. In: European Conference on Computer Vision (ECCV) (2018)