Learning Invisible Markers for Hidden Codes in Offline-to-online Photography Supplement

1. Implementation Details and Settings of the Distortion Network

In this paper, we simulate eight kinds of distortions in the distortion network during training. According to their sources, these distortions are divided into three categories: the distortion caused by environmental factors (brightness, contrast, and color distortions), the distortion caused by cameras (noise, image compression, and defocusing blur), and the distortion caused by photographers (motion blur, and geometric distortion). The detailed settings of these distortions are set as:

Brightness, Contrast, and Color Distortions We define the contrast offset as f_{con}, the brightness offset as f_{bri}, and the color offset as f_{col}. Thus, the distorted image I_{dis} caused by these three kinds of distortions is formulated as:

$$I_{dis} = (1.0 + f_{con}) * (I_{ori_{RGB}} + f_{col_{RGB}}) + f_{bri} \quad (1)$$

where f_{col} is a triple representing the color offsets of R, G, and B channels. When training, the pixel values of I_{ori} and I_{dis} are normalized to $[0.0, 1.0]$. Thus, f_{bri} is uniformly sampled from $[-0.3, +0.3]$, f_{con} is uniformly sampled from $[-0.3, +0.3]$, and the elements of f_{col} are uniformly sampled from $[-0.1, +0.1]$.

Noise, Image Compression, and Defocusing Blur To simulate noise, we add random noise to the generated image of the output with 50% probability. The noise range is $[-0.1, +0.1]$.

For image compression, we use JPEG compression to simulate the compression effect in the imaging process from screens to cameras. We use the differentiable approximation proposed by [2] to approximate the rounding operations:

$$rounding(x) = \begin{cases} x^3, & |x|<0.5 \\ x, & |x| \geq 0.5 \end{cases} \quad (2)$$

which has nonzero derivative almost everywhere. The quality factor q is randomly selected from $[50, 100]$.

We use Gaussian blur to simulate defocusing blur. The kernel size of the filtering window is $(3, 3)$ and the standard deviations of the horizontal and vertical directions to 0.1 and 1, respectively.

Motion Blur and Geometric Distortion For motion blur, the kernel size of the motion blur window is $(3, 3)$ and the angle range is $(-35°, +35°)$. In each iteration of training, we first apply Gaussian blur with 80% probability, and then motion blur with 80% probability.

The geometric distortion is simulated by a combination of 2D transformation and 3D transformation. For 2D geometric transformation, the range of rotation degrees is $[-15°, +15°]$ and the scaling range is $[0.8, 1.0]$. For 3D geometric transformation, The range of the perspective scale is $[0, 0.3]$. In each iteration of training, we first apply 2D transformation with 60% probability, and then 3D transformation with 60% probability. Thus, 36% of the training images are processed by the combination of these two transformations, 16% of the training images are without any geometric distortions, and 48% of the training images are processed by one of these two transformations.

The above operations are differentiable, implemented by PyTorch[1] and Kornia[2].

2. Details of Simulation-based Robustness Test

In this section, we describe the distortion settings in the experiments in Section 4.4 of the main body. The distorted images are generated by NumPy[3] and OpenCV[4] in these experiments. We present some examples of these distortions in Figure. 1.

Brightness, Contrast, and Color Distortions In this experiment, the offset range of I_{ori} and I_{dis} in Eq. 1 is [0,255]. Thus, the brightness offset f_{bri} in Eq. 1 includes 8 levels ranging from ± 10 to ± 80 and the color offset $f_{col_{RGB}}$ includes 6 levels ranging from ± 5 to ± 30. For adjusting contrast, the contrast offset f_{con} in Eq. 1 includes 5 levels ranging from ± 0.1 to ± 0.5. Adjusting brightness and contrast are divided into increase and decrease.

Noise, JPEG Compression, and Defocusing Blur For noise, we test both random noise and Gaussian noise, where random noise is used during training and Gaussian noise is used to verify the generalization ability of our model for

[1] https://pytorch.org/
[2] https://kornia.github.io/
[3] https://numpy.org/
[4] https://opencv.org/

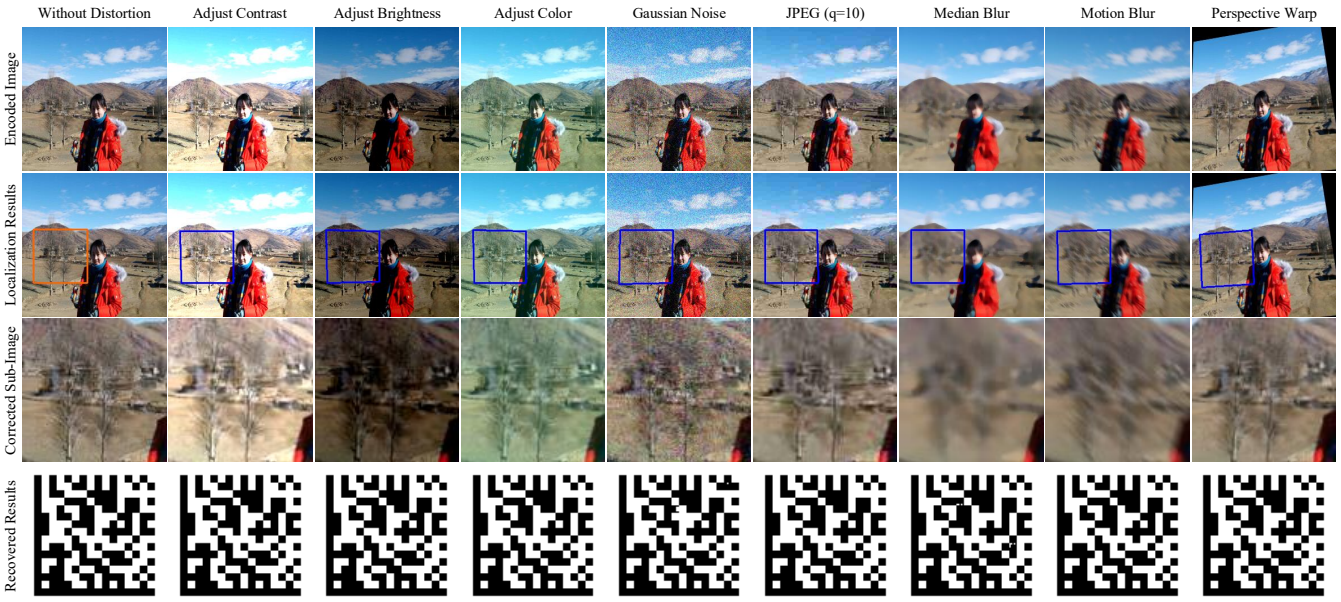

Figure 1. The samples of different simulated distortion. The distortion categories used in experiments include adjusting contrast, adjusting brightness, adjusting color, random noise, Gaussian noise, JPEG compression, median blur, mean blur, Gaussian blur, bilateral filtering, motion blur, and geometric distortion.

unknown distortion category. The intensity of random noise includes 10 levels ranging from 0.01 to 0.1. The intensity of Gaussian noise is controlled by the standard deviation σ that is divided into 10 levels ranging from 0.001 to 0.01.

The degree of JPEG compression is controlled by the quality factor q. In this experiment, the quality factor q of JPEG includes 9 levels ranging from 90 to 10.

For defocusing blur, we use Gaussian blur, mean filtering, median filtering, and bilateral filtering to generate test images. Except for Gaussian blur, the other three kinds of blur are unknown in training stages and are used to verify the generalization ability of our model. The degree of blur is positively correlated with the filtering kernel size. For Gaussian blur, mean filtering, and median filtering, the kernel size is divided into 3 levels: 3×3, 5×5, and 7×7. For bilateral filtering, the kernel size is divided into 3 levels including 5×5, 7×7, and 9×9. In addition, the σ parameter of color space and the σ parameter of coordinate space are set to 41.

Motion Blur and Geometric Distortion For motion blur, the kernel size is divided into 3 levels: 3×3, 5×5, and 7×7 and the motion angle include 3 values: $5°$, $15°$, and $30°$. The motion angle used in Table 1 of the main body is $15°$.

For geometric distortion, we add offsets to the coordinates of image vertices to represent geometric distortions, which is similar to StegaStamp [3]. The values of the offset d include 6 levels ranging from [-5,+5] to [-30,+30].

3. Comparison Details with the SOTA Methods

In this section, we supplement some experiments to compare our method with StegaStamp [3] and RIHOOP [1] under different distortion types in addition to geometric distortion. Table 1 shows that both our models and the SOTA methods are robustness to blur, motion blur, color distortion, and noise. For JPEG compression, when the quality factor (q) decreased to 10, the localization module performs poorly, which significantly increases the error rate (BER=26.17%). These indicate that our localization network is sensitive to serious JPEG compression. However, when we directly input the compressed sub-images (q=10) to our decoder as StegaStamp (BER=7.60%) and RIHOOP (BER=4.55%), the error rate (4.62%) is similar to the SOTA methods.

For geometric distortion, StegaStamp and RIHOOP do not design specific module to process it and their decoders cannot learn enough robustness to geometric distortions. Thus, the proposed localization module is specifically designed to locate codes under geometric distortions, which is also the motivation of hiding codes in a localized region because locating is the premise of decoding in displaying/printing-camera applications. For the distortions other than geometric distortion, the distortion layers in training can enhance the robustness of the localization module and the decoder. For example, when we retrain our model without motion blur, the bit error rates under motion blur increased by 3.04%.

Figure 2. Examples from different shooting conditions.

4. Analysis to Occlusion

We also analyze the robustness of our model to occlusion in this supplementary materials. When occluding the encoded sub-images, our model fails since we need to locate and decode the invisible code. While, our model is not affected when occluding the regions outside the encoded sub-images. In addition, we occlude the boundaries of the entire images by perspective transformation and zero padding, because occlusions in boundaries are more general in real applications such as scanning barcodes. In this experiment, the decoding rates of StegaStamp and RIHOOP are low (StegaStamp: 8% and RIHOOP: 0%). However, our model can still maintain 74% decoding rate (decoded by barcode recognizers (pylibdmtx). Our model failed only when the encoded sub-images appear at the edges of the entire images.

5. Supplemental Videos

We provide two supplemental videos to demonstrate the in-the-wild applications of our model. The attached videos are MPEG-4 files with H.264 encoding. We would like to apologize that, due to the commercial value of the proposed model, our source code is currently confidential. If the readers need codes for academic research, please contact the first author.

Method	Bit Error Rate ↓ (Distortion Levels)				
	blur (7X7)	motion (4X4)	light (30)	noise (0.1)	JPEG (q=20)
StegaStamp	0.30%	0.42%	0.65%	0.52%	1.05%
RIHOOP	0.32%	0.42%	0.30%	0.58%	0.90%
Ours	0.09%	0.31%	0.07%	0.08%	2.65%

Table 1. The Comparison Results with SOTA Methods under Distortions

- **displaying.mp4:** The first video presents the application in display-camera scenarios (https://youtu.be/B37Qt9hwyzs).
- **printing.mp4:** The first video presents the application in print-camera scenarios (https://youtu.be/X2ZfmPeRxSw).

Each video is concatenated by three sub-videos side by side. The left sub-video represents the user perspective where the red box, simulating the box in the scanning app, is to roughly tell where the user should scans. The middle sub-video represents the real-time back-end results:

(1) **Cropped Input**: according to the position of the red box, we crop a sub-region from each frame, resize this sub-region to 256×256, and input it to the model in users' sides (the localization network and the decoder).

(2) **Locating Results**: the locating results of the localization network.

(3) **Corrected Sub-Image**: According to the locating results, we correct the encoded sub-image from the cropped input with geometric distortion and input this corrected sub-image to the decoder.

(4) **Recovery Results**: the recovery results of the decoder.

In these videos, we normalize the recovered data matrices as follows: if more than half of the pixels representing the same bit unit are 255, the unit is 1; otherwise, it is 0.

6. Additional Example Results

We provide more examples of our model in Figure. 2. For good visual quality, we select the areas containing rich texture features as the sub-images to hide data matrices. We can find that the locating results are related to the relative geometric distortions of the sub-images, but not to the absolute backgrounds of the entire images.

References

[1] Jun Jia, Zhongpai Gao, Kang Chen, Menghan Hu, Xiongkuo Min, Guangtao Zhai, and Xiaokang Yang. RIHOOP: Robust Invisible Hyperlinks in Offline and Online Photographs. *IEEE Transactions on Cybernetics*, pages 1–13, 2020. 2

[2] Richard Shin and Dawn Song. Jpeg-resistant Adversarial Images. In *NeurIPS Workshop on Machine Learning and Computer Security*, 2017. 1

[3] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible Hyperlinks in Physical Photographs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2117–2126, 2020. 2