StegoPatch Image Watermarking

Eugene Francisco

June 2026

1 Overview

Watermarking refers to the process of embedding information in generated data so as to label that data as having been generated by a model. This paper introduces StegoPatch, a novel way to watermark images, robust to common image transformations and lossy compression and virtually undetectable to the human eye. Inspired by Bui et al. [1], we train a *message encoder* to embed hidden bit sequences in the latent representation of images from a fixed autoencoder; alongside this, we train a *secret decoder* to recover messages from possibly corrupted images. We take advantage of convolutional autoencoders to allow for the spatial embedding of these watermarks; alongside this, we use techniques from Tancik et al. [3] to make these watermarkers robust to spatial transformations like cropping and rotating, as well as regular image transformations like noising, blurring, and JPEG.

Our code can be found here.k

2 Related Work

RoSteALS, introduced by Bui et al. [1], is the most direct inspiration for StegoPatch. Rather than learning an image-to-image encoder and decoder from scratch, RoSteALS watermarks within the latent space of a fixed, image autoencoder. A small neural network maps bit sequences to additive perturbations of the latent variable, which is then decoded back into image space. StegoPatch adopts this latent-space embedding strategy and extends it to embed messages spatially, exploiting the convolutional structure of the autoencoder so that the watermark is repeated across the image and can survive cropping.

Tancik et al. [3] introduced StegaStamp. Their model allowed watermarks to survive the physical printing and recapture of an image. To accomplish this, StegaStamp is trained against a set of differentiable image perturbations like noise, blur, color shifts, and perspective warps, between the encoder and decoder during training, forcing the decoder to recover messages even from heavily corrupted inputs. StegoPatch borrows this idea of embedding a noise layer between encoding and decoding: we expose the secret decoder to spatial transformations like cropping and rotating, as well as noising, blurring, and JPEG compression, so that messages remain recoverable after these corruptions. Earlier, Zhu et al. [4] proposed HiDDeN, one of the first end-to-end deep watermarking frameworks; we adopt their ideas for decoding messages by aggregating message predictions across the image by vote.

3 Methods

Watermarking derives from the much older technique of steganography, where the challenge is to conceal messages within another medium. Stated concretely, the goal of image watermarking is to take a so-called *cover image* c and message m and generate a *stego image* $\tilde{c}$ in which m is somehow embedded; later on, a verifier should be able to take $\tilde{c}$ (which has potentially been compressed or transformed in some way) and recover a message $\hat{m}$ which should be similar to m.

StegoPatch is a so-called *post-hoc* watermarker, meaning it adds watermarks to existing images. This is as opposed to an *ad-hoc* watermarker, which embeds messages into the generated content at the same time that the content is being generated.

3.1 Architecture

StegoPatch trains two neural networks:

(i) A message encoder $F_\theta : \{0,1\}^\ell \to \mathbb{R}^{P\times P\times C}$ which is used to encode messages and generate watermarked images.

(ii) A secret decoder $D_\varphi : \mathbb{R}^{H\times W\times C} \to \{0,1\}^\ell$ which is used to recover a message $\hat{m}$ from a (possibly corrupted) $\tilde{c}$.

The message encoder F_θ takes advantage of a fixed image autoencoder to embed messages. Let

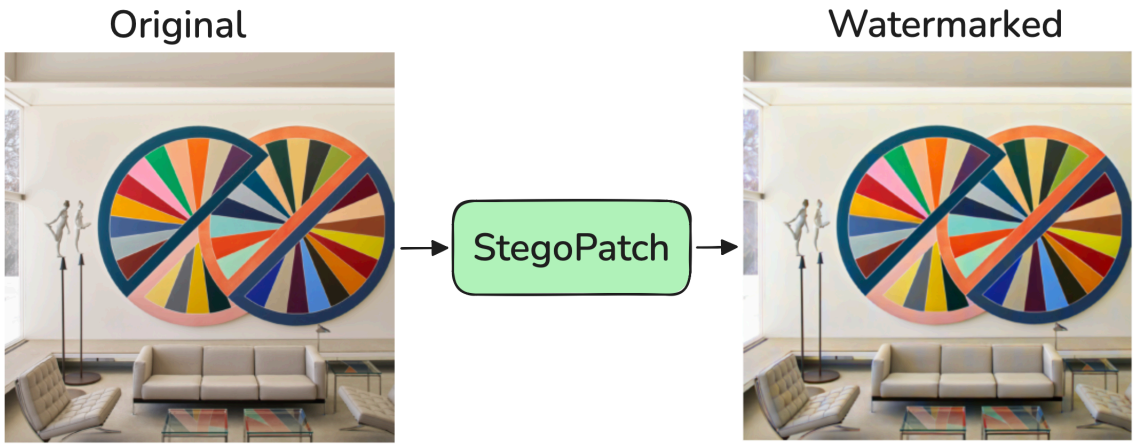

Figure 1: StegoPatch is a *post-hoc* watermarker, meaning it adds watermarks to existing images. StegoPatch works on images much larger than what it was originally trained on, like here, where StegoPatch has been used to watermark a Frank Stella painting that is $2\times$ larger resolution than used in training. Image courtesy of Christie's [7].

(E, G) be an image autoencoder. E is an image encoder that sends images $c \in \mathbb{R}^{H\times W\times C}$ to $z = E(c) \in \mathbb{R}^{H'\times W'\times C}$ where H', W' are a constant factor smaller than H and W. And G is an image generator that takes latent variables $z \in \mathbb{R}^{H'\times W'\times C}$ and sends them to $c = G(z) \in \mathbb{R}^{H\times W\times C}$. It is important that (E, G) are convolutional so that they preserve spatial information. (E, G) are fixed in advance and aren't trained.

The networks F_θ and D_φ encode in the following way:

(i) Message Encoder: given a cover image c of size $H \times W$ and a message m of length ℓ, we first embed c into its latent space to make latent variable $z \leftarrow E(c)$. Assign $\delta \leftarrow F_\theta(m) \in \mathbb{R}^{P\times P\times C}$; this will encode watermarked information. Partition z into patches of size $P \times P \times C$; let $\mathcal{P}$ be the set of these patches.

For each patch $p \in \mathcal{P}$, create the watermarked patch $\tilde{p} \leftarrow p + \delta$. Using the watermarked patches, stitch together a watermarked latent $\tilde{z}$. Finally, return $G(\tilde{z})$ as the watermarked cover. See Algorithm 1.

The neural network F_θ is a very small convolutional neural network implemented in a similar way to Bui et al.

(ii) Secret Decoder: Given a (possibly corrupted) watermarked image $\tilde{c}$, the secret decoder returns $\hat{m} = D_\varphi(\tilde{c})$. The neural network D_φ is implemented as a modified ResNet50 with the final flattening layer replaced by a 1×1 CNN whose output is ℓ channeled. An average is taken across the width and height dimensions to retrieve an ℓ length vector of logits which are thresholded and returned as $\hat{m}$. See Algorithm 2

This last *voting* step is taken from Zhu et al.'s HiDDeN [4].

Here, θ and φ are trainable parameters; note in particular that the autoencoder (E, G) remain fixed and are not trained.

In this architecture, it is important that the patching and periodic pasting of the latent variable δ happens within the latent space. Watermarking performance worsens significantly if instead we choose to watermark each patch individually and then stitch together the patches in image space. Doing so leads to more obvious stitch marks and much higher quality loss during training.

3.2 Training Details

We use the same training objectives as RoSteALS [1]. F_θ is trained to minimize

$$\mathcal{L}_{\text{quality}} = \mathcal{L}_{\text{LPIPS}}(\tilde{c}, c) + \alpha\|\gamma(c) - \gamma(\tilde{c})\|_2^2 \quad (1)$$

where $\mathcal{L}_{\text{LPIPS}}$ is the *Learned Perceptual Image Patch Similarity* loss from Zhang et al. [6] and γ is a differentiable map from RGB to YUV color space. α is

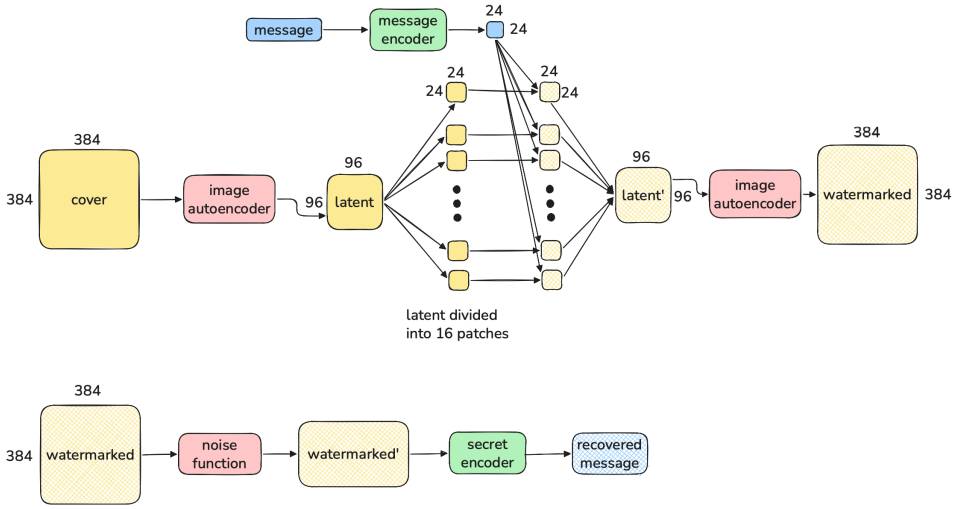

Figure 2: StegoPatch Watermarking and Decoding Architecture. The message encoder refers to F_θ and the secret decoder refers to D_φ.

Algorithm 1 Message Encoder F_θ

Require: cover image $c \in \mathbb{R}^{H\times W\times C}$, message $m \in \{0,1\}^\ell$
Ensure: stego image $\tilde{c}$

$z \leftarrow E(c)$
$\delta \leftarrow F_\theta(m)$
$\mathcal{P} \leftarrow \textsc{Partition}(z)$ ▷ into $P \times P \times C$ patches
for each patch $p \in \mathcal{P}$ **do**
    $\tilde{p} \leftarrow p + \delta$
end for
$\tilde{z} \leftarrow \textsc{Stitch}(\{\tilde{p}\})$
return $G(\tilde{z})$

Algorithm 2 Secret Decoder D_φ

Require: (possibly corrupted) stego image $\tilde{c} \in \mathbb{R}^{H\times W\times C}$
Ensure: recovered message $\hat{m} \in \{0,1\}^\ell$

$L \leftarrow \textsc{ModifiedResNet50}_\varphi(\tilde{c})$ ▷ Last layer is a $1\times 1\times$ CNN, ℓ channels. $L \in \mathbb{R}^{H'\times W'\times \ell}$
$\bar{L} \leftarrow \textsc{Mean}_{H',W'}(L)$
$\hat{m} \leftarrow \mathbf{1}[\bar{L} > 0]$
return $\hat{m}$

a hyperparameter to control objective weight. D_φ is trained to minimize the binary cross entropy between m and $\hat{m}$:

$$\mathcal{L}_{\text{recovery}} = \mathcal{L}_{\text{BCE}}(m, \hat{m}). \tag{2}$$

The combined training objective is then

$$\mathcal{L} = \beta \mathcal{L}_{\text{quality}} + \mathcal{L}_{\text{recovery}} \tag{3}$$

where β is a weighting hyperparameter.

We use curriculum learning to train F_θ and D_φ. Learning is split into the following phases:

1. Fix a single minibatch of images from the dataset. Train F_θ and D_φ on this minibatch until bit accuracy of m against $\hat{m}$ crosses t_1. Sample m's bits uniformly and randomly. Before passing $\tilde{c}$ into D_φ, we choose with probability $1/2$ whether to crop $\tilde{c}$, in which case we crop $\tilde{c}$ to a random patch with sides of at least $3P$ in length.

2. Expose F_θ and D_φ to a much larger portion of the training set ($50{,}000$ images in our case). Train until bit accuracy reaches t_2 as this new exposure will likely drop the bit accuracy by a significant amount.

3. Begin incrementing β linearly by some amount δ_β until it reaches some $\beta_{\max}$, at which point continue training until bit accuracy reaches t_3.

4. Reveal the full dataset to F_θ and D_φ and train until bit accuracy reaches t_4.

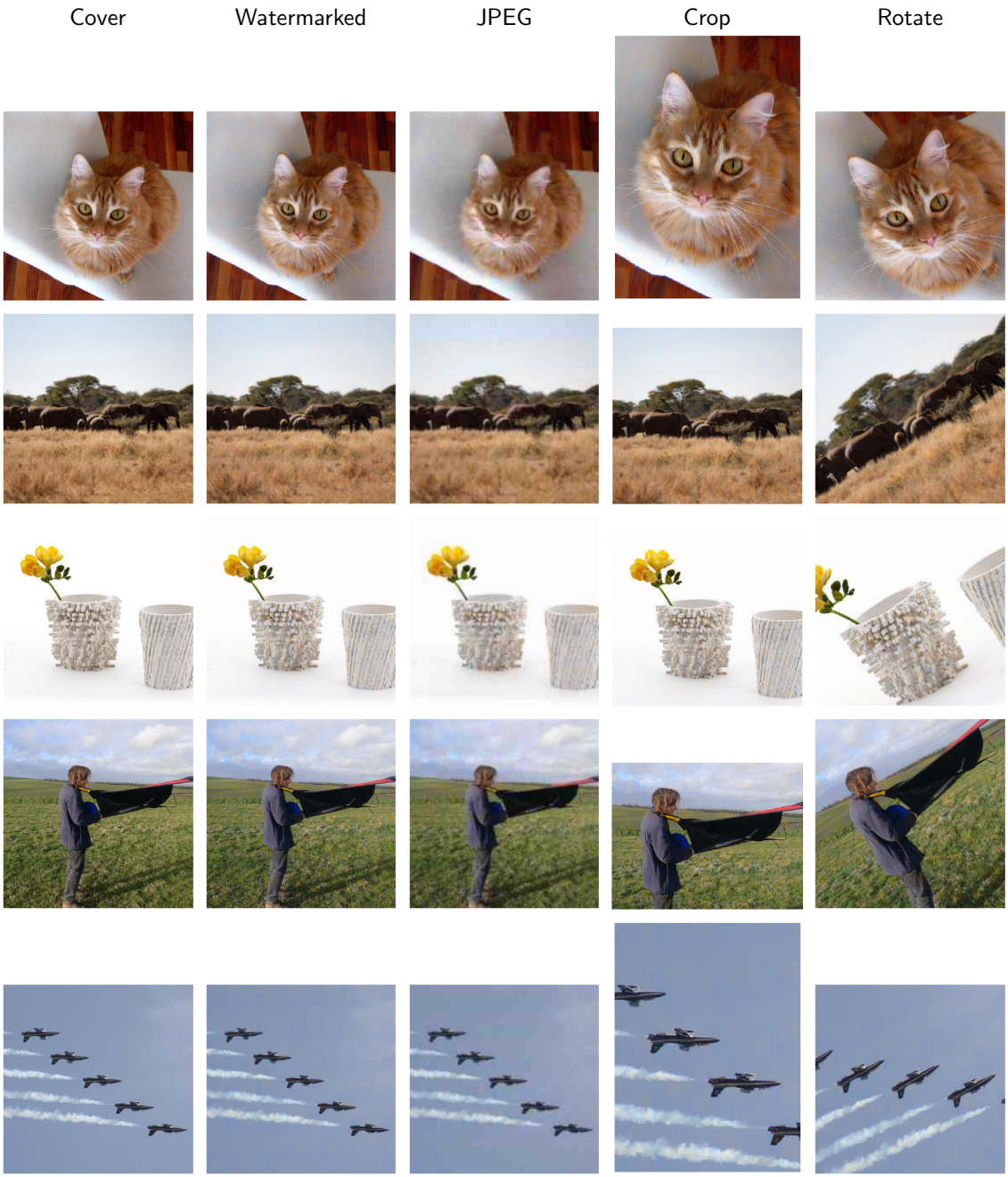

Figure 3: Qualitative results. Each row is an image class (cat, elephants, flower, kite, planes) and each column is a transformation applied to the watermarked image (cover and watermarked shown for reference). Watermarked images without any noise applied achieve 100% bit accuracy. For JPEG, crop, and rotation results, check the whisker plot below.

5. Introduce a noise function N which manipulates $\tilde{c}$ to generate a noised image $N(\tilde{c})$. Pass $N(\tilde{c})$ into D_φ. Train for an additional k epochs.

4 Implementation Details

For simplicity, we trained on square images of size 384×384 using a patch size of $P = 96$. All of our experiments are done with a message length of $\ell = 20$. We picked $\alpha = 1.5$ as in RoSteALS. We chose $\beta = 0.8$ and increment β linearly until $\beta_{\max} = 15$ in training phase 3. The bit accuracy thresholds are set to $t_1 = 0.9, t_2 = 0.8, t_3 = 0.95, t_4 = 0.98$. The last training phase we trained for $k = 3$ epochs. We used the first $100{,}000$ images from the COCO training dataset from 2017 [5]. We optimize the objective in (3) using stochastic gradient descent (in PyTorch), a minibatch size of 8, and the AdamW optimizer with a learning rate of $2 \cdot 10^{-5}$. Training was done on an NVIDIA A100 40GB GPU and took around 10 hours per experiment.

We use a very small neural network for F_θ with only $9{,}168$ parameters. The network is described by the following PyTorch Sequential network:

```
nn.Linear(20, 12 * 12 * 3),
nn.SiLU(),
nn.Unflatten(1, (3, 12, 12)),
nn.Upsample(scale_factor=2),
nn.Conv2d(3, 3, 3, padding = 1),
nn.Conv2d(3, 3, 1)
```

where, as in RoSteALS, the final layer is 0-initialized so that $\delta = 0$ at the start training.

We use the vq-f4 autoencoder [2] just like RoSteALS; notably, this autoencoder is fully convolutional which means that periodic patching in the latent space will be preserved in the image space.

We use several different noise transformations which can be seen in Figure 8. For those noise transformations which are non-differentiable, we use [1]'s trick of treating the noise as a constant by letting $N(c) = c + \langle N(c) - c \rangle$ where $\langle \cdot \rangle$ is treated as a constant (no computational graph or gradients computed); we also use [1] for the *differentiable* noise that is added, which is a layered combination of noises that is differentiable.

5 Results

Figure 3 shows qualitative results across five image classes (rows) under a range of transformations

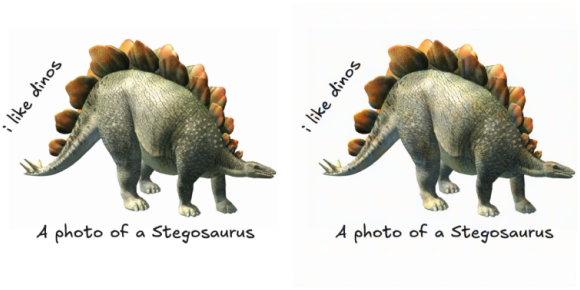

Figure 4: StegoPatch applied to an image with text.

(columns). For each class we show the original cover image, the watermarked image, and the watermarked image after JPEG compression, cropping, and rotation. The watermarked images are essentially identical to the original images, except in the case of the flower (row 3) where the white background betrays some faint watermarking splotches. Also notable is that all the images displayed here use a resolution $1.5\times$ larger than the one used for training (and Figure 2 is $2\times$ larger), demonstrating scalability to unseen dimensions.

Interestingly, StegoPatch seems to perform better on text (Figure 5) and faces than [1] even though both utilize an image encoder backbone. It is important that the patching for StegoPatch happens within the latent space since otherwise clear stitch marks can be seen between the patches, as can be seen in Figure 5.

StegoPatch achieves very strong performance on most forms of image manipulation, as can be seen in Figure 5. We point in particular to JPEG performance where StegoPatch achieves performance comparable or stronger than [1] and [3]. However, StegoPatch still has notable weaknesses. For example, although StegoPatch does significantly better than guessing when crops are applied, it still has a statistical power of only around 0.75 (crops are sampled as a random height and random width, where height and widths must be a minimum of 3 patch-lengths). We note though that this power grows with image size as more patches become visible. Another disadvantage of StegoPatch is that, applied to images that have much larger resolution than what is trained on, StegoPatch seems to impart more discoloring on the image. Also, on images that have little detail, StegoPatch struggles to recover images that have been JPEG compressed (see future work). StegoPatch's repetitive watermarking can be seen in residuals of watermarked images (Figure 7). No-

Figure 5: A comparison of watermarking when stitching is done in the image space (middle) vs latent space (bottom). Clear stitch marks can be seen in the middle on the left person's face.

tably, residuals are stronger in places of the image where there is more noise, suggesting that watermark signal is highest in those parts of the image that contain texture and contrast.

6 Future Work

Training StegoPatch on even larger, higher resolution images, will likely allow for much stronger robustness against cropping. Part of the challenge with training StegoPatch is that training images cannot be significantly larger than their cropped counterparts due to the author's compute limitations. While we used images that were 4×4 patches across with a maximum crop down to 3×3 patches, an ideal setting would train StegoPatch on larger images where more patches can be spread out.

Along these lines, it may be interesting to see whether patch placement can be chosen more carefully rather than pure periodicity. One may be able to choose the places in the image where the watermark would be more hidden to place a stronger watermark. For example, the message encoder F_θ could take in the cover image too as input to give it context for how much to watermark each patch, as in StegaStamp [3].

References

[1] Tu Bui, Shruti Agarwal, Ning Yu, and John Collomosse. RoSteALS: Robust steganography using autoencoder latent space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2023.

[2] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[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)*, 2020.

[4] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. HiDDeN: Hiding data with deep networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

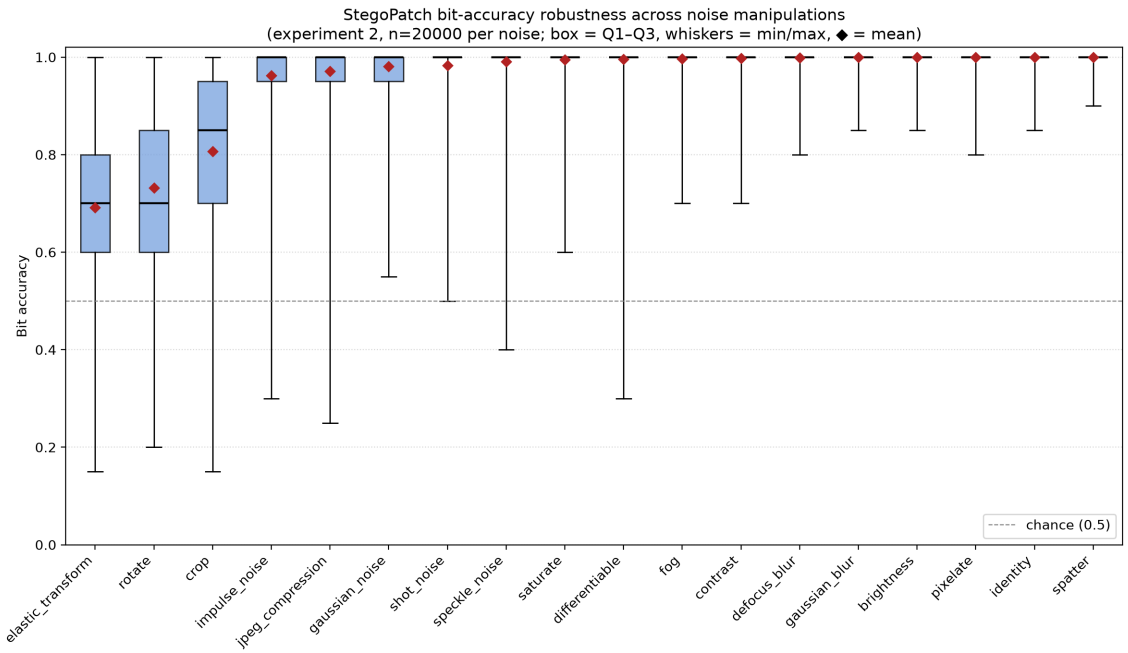

Figure 6: A whisker plot of StegoPatch's performance against different image manipulations.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[6] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[7] Christie's. 15 things to know about Frank Stella. linked here, accessed 2026.

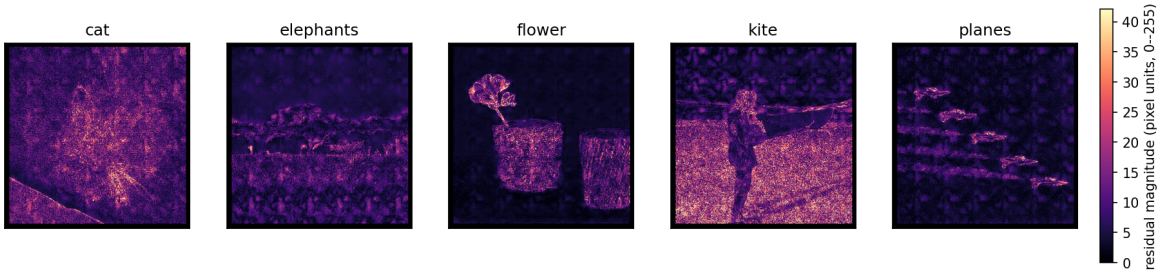

Figure 7: Watermark residuals for the five image classes of Figure 3. Each panel shows the per-pixel residual magnitude $\|\tilde{c} - c\|_2$ (L2 norm over the RGB channels, in 0–255 pixel units) between the cover image c and its watermarked counterpart $\tilde{c}$.

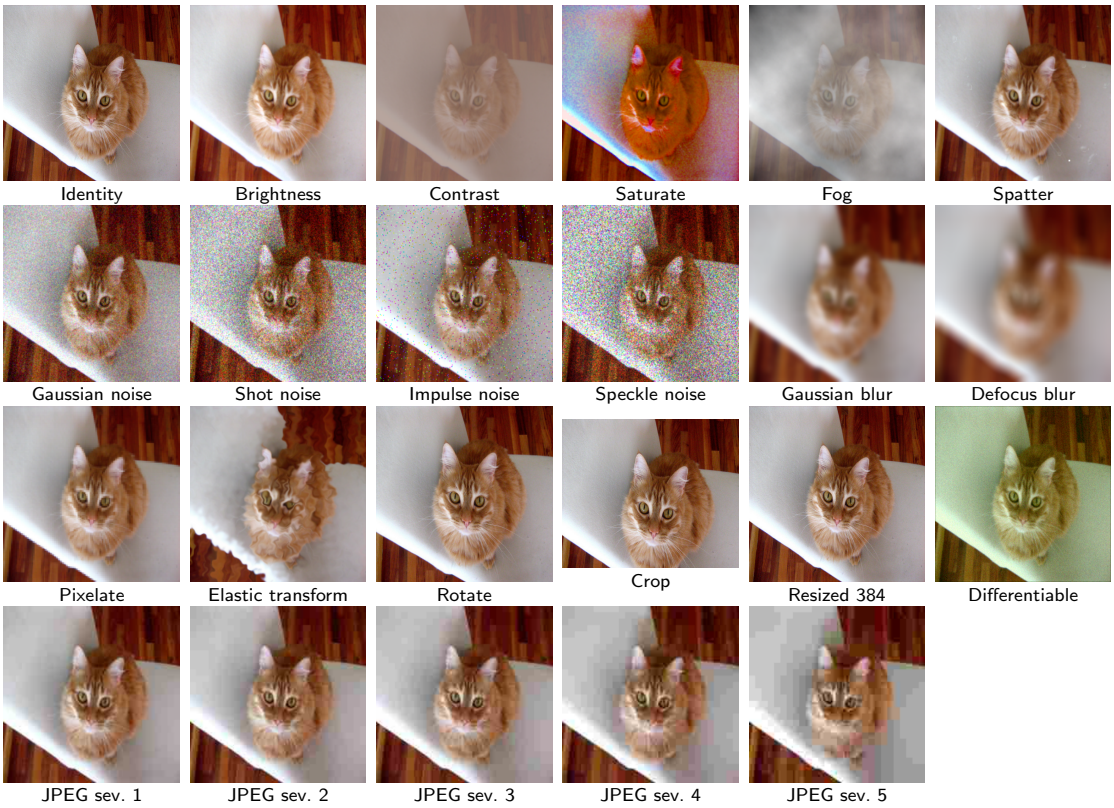

Figure 8: The full set of image manipulations (noises) applied to a single cover image during training and evaluation. Each panel shows the same image after a different distortion.

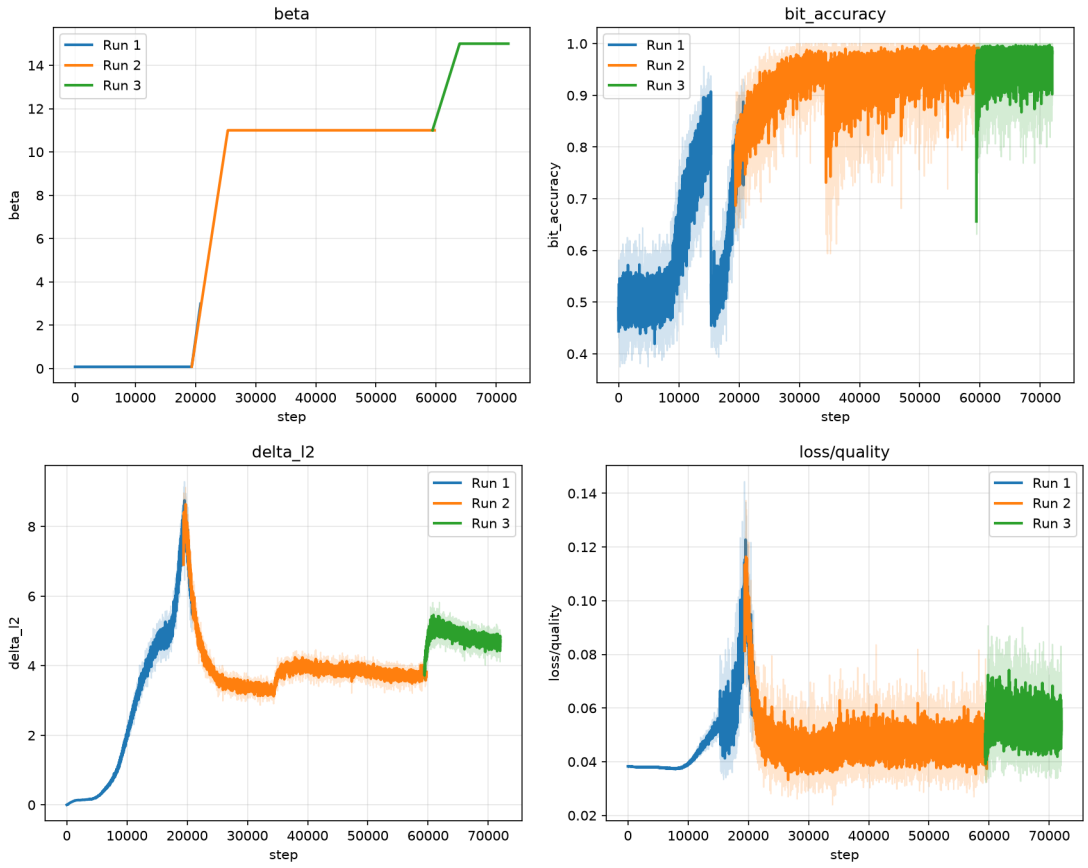

Figure 9: Performance curves during training. Training was done over the course of three runs.