A. CLIP Backdoor [7]

Figure 11. An image with a 16×16 backdoor used in [7].

CLIP Backdoor [7] demonstrated a backdoor attack on a multimodal contrastive model CLIP [33]. It uses a patch backdoor: a 16×16 grid of black and white pixels. We train a backdoored CLIP following the steps described in [7]. We visualize a sample backdoored image in Figure 11. The backdoored CLIP model we trained has a backdoor ASR of 99.99%. We then cast this CLIP model as a standard image classifier in a zero-shot manner and apply SmoothInv *w/o diffusion* only (due to limited computation resources). We test this on 10 random clean images from ImageNet testset and the average ASRs of the reversed backdoors are 73.16% for ($\epsilon = 5$) and 93.51% for ($\epsilon = 10$). We show the reversed backdoored images in Figure 12. We can see that some patterns with high color contrast appear in the top-left part.

B. Limitations

One major limitation of SmoothInv is that it does not generalize to more advanced backdoors. In this work, we consider patch-based backdoor in particular. However, other forms of backdoor are shown possible by previous work, e.g. image wrapping [27], adaptive imperceptible perturbation [48] and instagram filters [16]. Our method does not apply to those backdoors and we believe the likely reason is that ℓ_2 based perturbations are only suitable for reversing patch-based backdoors. For advanced backdoors, we suspect that one would need to design the perturbation space of SmoothInv more carefully. For instance, we can model instagram filters with a per-pixel position-dependent transformation implemented by a neural network [20]. It would be interesting future work to extend our approach beyond patch based backdoors.

C. Runtime and Resource Considerations

The major time bottleneck of our approach SmoothInv comes from back-propagating through the smoothed classifier G_b. SmoothInv *w/o diffusion* is generally fast as diffusion model is not involved. Here we mainly study the resource consumption of SmoothInv *w/ diffusion*. First we would like to clarify that for SmoothInv *w/ diffusion*, we are not using the full standard reverse diffusion process but the one-shot denoising approach proposed in [8], where we apply only one diffusion step to obtain an estimate of the denoised image. All our experiments were run on four RTX A6000 GPU machines with 48685MiB GPU memory each. In Table 4, we report the time to synthesize an image using SmoothInv with various number of noise vectors N. We compare with the baseline "Standard" where we back-propagate through the standard base classifier (ResNet-18). We can see that the time spent scales linearly with the number of noisy vectors. In our experiments, we find that $N = 10$ noise vectors are usually enough for a stable synthesis result. In this case, SmoothInv takes roughly around 5 mins to synthesize one backdoored image in one GPU. It would be interesting future work to investigate methods to speedup our synthesis process when a diffusion denoiser is used.

Figure 12. SmoothInv on a backdoored CLIP model.

	Standard	$N = 1$	$N = 5$	$N = 10$	$N = 20$	$N = 40$
#GPUs	1	1	1	1	2	4
Time/sec	9.00	103.04	344.61	339.82	691.79	1014.72

Table 4. Time (in seconds) and resource taken to synthesize an image (400 PGD iterations) for SmoothInv *w/ diffusion*. We report the results with different number of noisy samples N. "Standard" corresponds to the PlainAdv baseline.

Algorithm 1 SmoothInv (PyTorch-style)

```
# model: the backdoored classifier
# diffusion: class-unconditional diffusion model
# x_orig: a single clean image from victim class
# y_t: target class
# delta: perturbation vector δ to be optimized
# sigma: noise level σ for RS procedure [8,10]
# n: number of Monte Carlo noise samples
# eps, alpha, steps: PGD hyper-parameters


def backdoor_tracing(f, diffusion, x_orig, y_t):
   for _ in range(steps):
      x = x_orig + delta
      x_n = x.repeat(n,1,1,1)
      x_noise = x_n + torch.randn_like(x_n) *
          sigma # add isotropic Gaussian noise

      x_denoised = diffusion.denoise(x_noise) #
          optional
      y_prob = model(x_denoised)
      y_est = y_prob.mean(dim=0) # estimated
          output of the smoothed classifier

      loss = criterion(y_est, y_t)
      loss.backward()

      delta += alpha * l2_normalize(delta.grad)
      delta = project(delta, eps)
      delta.grad.zero()

   return x_orig + delta
```

D. Additional Visualization Results

In Figure 16, we show some backdoored images with the true backdoors listed in Table 2. Notice that for Blind-P and Blind-S, the backdoors are placed in the top left region of the images (the injected backdoor may be hard to identify unless zooming in the specific part).

We provide a comparison of SmoothInv *w/ diffusion* and *w/o diffusion* in Figure 13. We can see that for TrojAI, SmoothInv *w/o diffusion* tends to generate more regions of interest on the background while the synthesized patterns appear more often in the foreground for SmoothInv *w/ diffusion*. For HTBA, SmoothInv *w/o diffusion* tends to have vague artifacts while the backdoor patterns are more distinctive for SmoothInv *w/ diffusion*. This suggests that while SmoothInv *w/o diffusion* may generate more effective backdoors, but using a diffusion denoiser may lead to better visualization results.

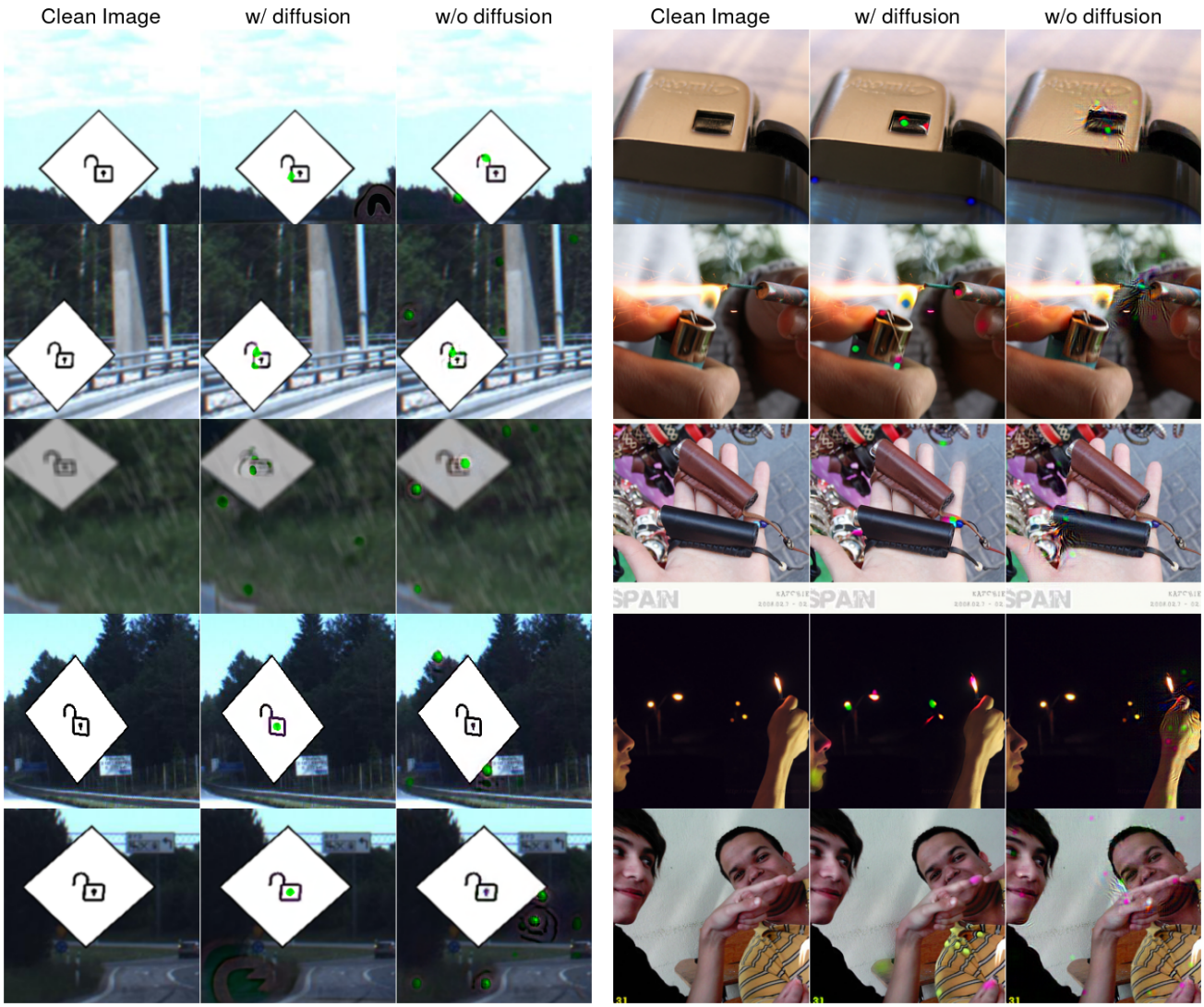
We include more visualization results ($\epsilon = 10$) on the Blind-P and Blind-S models in Figure 14 and Figure 15, where we show the synthesized images under various noise level $\sigma \in \{0.25, 0.50, 1.00\}$. We find no distinction between *w/ diffusion* and *w/o diffusion* visually so here we show the results of SmoothInv *w/o diffusion* for these two models. We can see that in general, smoothed classifiers constructed with larger noise levels tend to give better visualization results.

E. Pseudo-code

We provide a PyTorch-style pseudo-code for our approach SmoothInv in Algorithm 1, where we apply projected gradient descent [25] to synthesize backdoored patterns given a single image. We use a pre-trained diffusion model to build a robust smoothed classifier, following [8].

F. Sanity Check

Following the initial sanity check result in Figure 4, we report the results on all four backdoored classifiers in Table 5. We show both the clean accuracy and backdoor ASR of the smoothed classifiers *w/* and *w/o diffusion*. We can see that using a diffusion denoiser can significantly improve the clean accuracy of the resulting smoothed classifiers for all four backdoored classifiers. For backdoor ASR, we can see that the backdoor remains effective for smoothed classifiers both *w/* and *w/o diffusion* for some values of noise level σ. Note that for TrojAI, the results should have large variance as only five clean images are provided by the data publisher.



(a) TrojAI-R4-131 (b) HTBA

Figure 13. Comparison of SmoothInv *w/ diffusion* and *w/o diffusion* ($\epsilon = 10$).

Figure 14. Additional results on the Blind-P model.

Figure 15. Additional results on the Blind-S model.

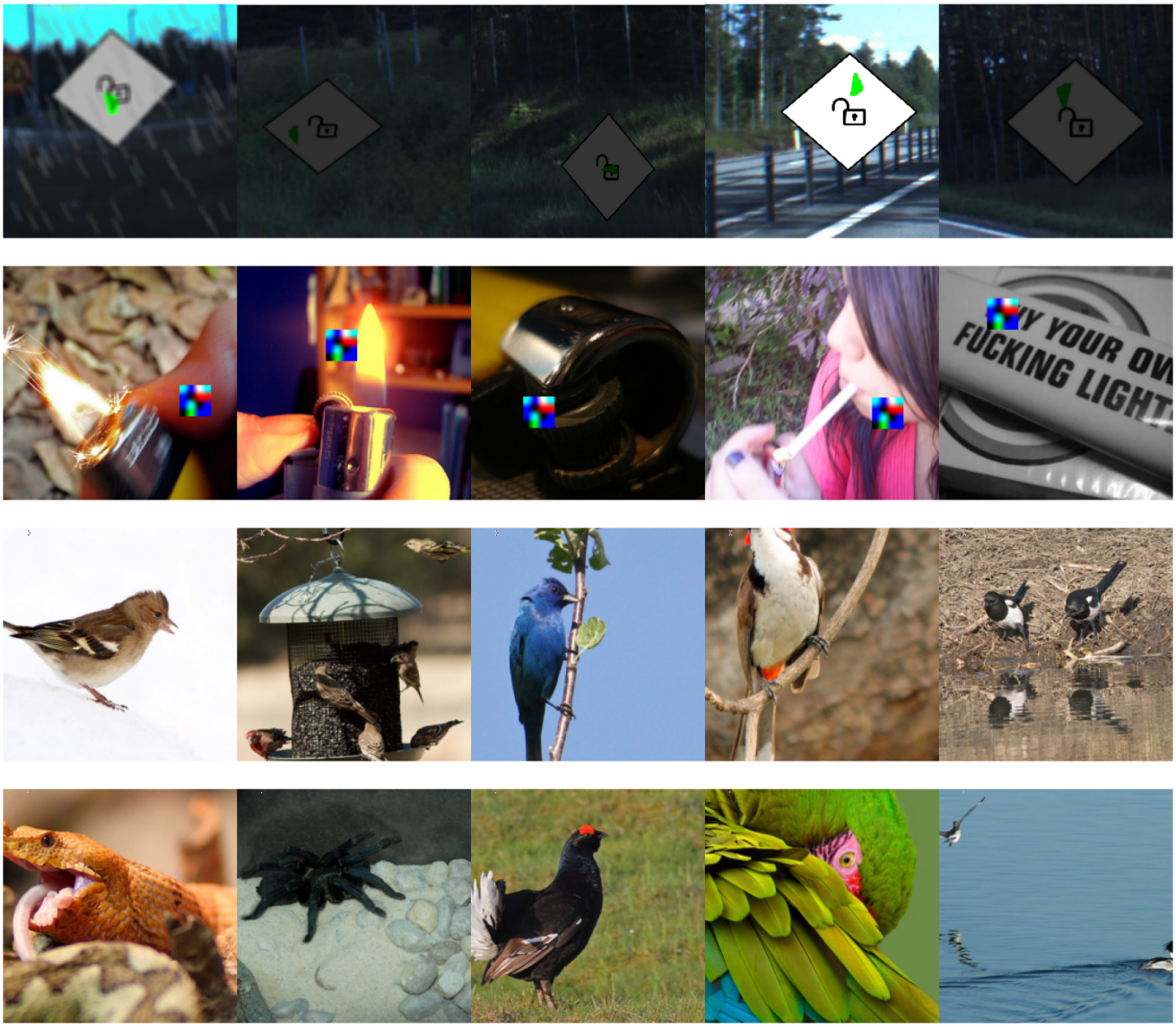


Figure 16. Backdoored images, from top to bottom: polygon/TrojAI, square/HTBA, pattern backdoor/Blind-P and single pixel/Blind-S.

Model	Diffusion	0.00	0.12	0.25	0.50	1.00
TrojAI	✗	100.00%	100.00%	80.00%	20.00%	0.00%
	✓	100.00%	100.00%	100.00%	100.00%	100.00%
HTBA	✗	95.00%	2.00%	0.00%	0.00%	0.00%
	✓	95.00%	90.00%	92.00%	98.00%	96.00%
Blind-P	✗	69.26%	33.40%	2.80%	0.00%	0.00%
	✓	69.26%	66.10%	63.10%	57.70%	47.10%
Blind-S	✗	68.06%	29.90%	2.30%	0.10%	0.10%
	✓	68.06%	65.60%	62.30%	56.70%	47.80%

(a) Clean Accuracy

Model	Diffusion	0.00	0.12	0.25	0.50	1.00
TrojAI	✗	100.00%	100.00%	40.00%	40.00%	20.00%
	✓	100.00%	100.00%	100.00%	60.00%	40.00%
HTBA	✗	54.00%	70.00%	100.00%	100.00%	100.00%
	✓	54.00%	64.00%	58.00%	64.00%	48.00%
Blind-P	✗	99.29%	99.80%	94.90%	40.10%	4.70%
	✓	99.29%	99.40%	87.70%	1.70%	0.10%
Blind-S	✗	79.73%	89.60%	88.40 %	81.70%	97.00%
	✓	79.73%	59.20%	21.50%	4.00%	0.00%

(b) Backdoor ASR

Table 5. Clean accuracy and backdoor ASR of the smoothed classifiers (*w/* and *w/o diffusion*) with various values of σ: 0.12, 0.25, 0.50 and 1.00. The $\sigma = 0$ column corresponds to the results of the base backdoored classifier. The results on TrojAI are computed on a limited number of 5 available clean images so they should have high variances.