

## Appendix Summary

We start with a brief overview of the content of the Appendix.

- In Appendix A, we give a more detailed description of diffusion models in general, cross-attention conditioning, classifier-free guidance, and our diffusion guidance via optimization.
- Next, in Appendix B, we provide further details on our experiments from Sec. 4. In particular, the shape bias of adversarially robust models (Fig. 14), errors of zero-shot CLIP (Fig. 15) and different biases of ViT and ConvNeXt architectures (Fig. 16).
- The process of collecting real images to validate the detected zero-shot CLIP errors (see Fig. 5) is explained in Appendix C.
- Appendix D is an extension of Section 5 from the main paper. We give further details about our visual counterfactual generation. Additionally, we show more VCEs for ImageNet (Fig. 20), CUB (Fig. 21), Food-101 (Fig. 22), Cars (Fig. 23) and FFHQ (Fig. 24) as well as EVA02 error visualizations in Fig. 25.
- In Appendix E, we provide details on the user study comparing DVCEs [5] and our UVCEs, including all images that were used in the study (see Fig. 27).
- Appendix F contains more details and examples for our synthetic neural visualizations as well as neuron counterfactuals. We explore a quantitative metric to discriminate spurious from core neurons in Appendix F.3.
- In Appendix G we give more details about the NPCA [54] optimization to validate harmful spurious features.
- Finally, Appendix H describes limitations and failure cases.

## A. Background and Method Details

### A.1. Diffusion Models

Diffusion models are a class of generative models that learn to sample from a data distribution  $q(x)$ . We thereby differentiate between the forward process which, given a real data point, adds noise at every timestep  $t \in \{1, \dots, T\}$  until the noisy sample can no longer be distinguished from a normally distributed random variable, and the reverse process, which, given a latent from a normal distribution, removes noise at every timestep such that at the final time step, we generate a sample  $x \sim q(x)$ . In short, the forward process takes a real data point to the latent space and the reverse process generates a real datapoint from a latent vector. For this section, we follow the notation from [79].

In this work, we focus on discrete-time diffusion models where both the reverse and forward process correspond to Markov Chains of length  $T$  and refer readers to [80] for the time-continuous case. While the first wave of image diffusion models [39, 78] were generating samples directly in

pixel space, it has been shown [64, 83] that it can be beneficial to instead work inside the latent space of a variational autoencoder (VAE). Instead of generating the image directly, latent diffusion models (LDM) generate a latent  $z_0$  inside the VAE latent space and then use the VAE decoder  $\mathcal{D}$  to transform  $z_0$  into pixel space to produce the final image  $x = \mathcal{D}(z_0)$ . As our experiments are based on Stable Diffusion (SD) [64], for the rest of this section, we assume that we are working with a latent diffusion model where the goal is to sample a VAE latent  $z_0$  using the diffusion process.

Thus let  $q(z_0)$  be the distribution of the VAE latents that can be obtained from the image distribution in pixel space  $q(x)$  via the VAE encoder  $\mathcal{E}$ . The goal is to learn a model distribution  $p_\theta(z_0)$  that is similar to the data distribution, i.e.  $p_\theta(z_0) \approx q(z_0)$ , and is easy to sample from. Denoising Diffusion Probabilistic Models (DDPM) [39] are defined via the forward process that uses Gaussian transitions  $q(z_t|z_{t-1})$  to incrementally add noise to a noise-free starting latent  $z_0$ :

$$q(z_t|z_{t-1}) = \mathcal{N}\left(z_t; \frac{\sqrt{\alpha_t}}{\sqrt{\alpha_{t-1}}}z_{t-1}, \left(1 - \frac{\alpha_t}{\alpha_{t-1}}\right)\mathbf{I}\right) \quad (7)$$

with a fixed decreasing sequence  $\alpha_{1:T} \in (0, 1]^T$  that determines the noise-level at each time step  $t$ . Given  $z_0$ , this defines a distribution over the other time steps  $z_{1:T}$  via:

$$q(z_{1:T}|z_0) = \prod_{t=1}^T q(z_t|z_{t-1}). \quad (8)$$

Due to the Gaussian nature of the transitions  $q(z_t|z_{t-1})$ , given  $z_0$ , it is possible to sample from  $q(z_t|z_0)$  in closed-form instead of following the Markov chain  $t$  times via:

$$q(z_t|z_0) = \mathcal{N}\left(z_t, \sqrt{\alpha_t}z_0, (1 - \alpha_t)\mathbf{I}\right), \quad (9)$$

from which it follows that:

$$z_t = \sqrt{\alpha_t}z_0 + \sqrt{(1 - \alpha_t)}\epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, \mathbf{I}). \quad (10)$$

This makes it obvious that, as long as  $\alpha_T$  is chosen sufficiently close to 0, we have that  $q(z_T|z_0) \approx \mathcal{N}(0, \mathbf{I})$ , i.e. the forward process transforms the original distribution  $q(z_0)$  into a standard Normal distribution. Thus one defines  $p_\theta(z_T) = \mathcal{N}(0, \mathbf{I})$  as the prior distribution for the generative model. Our parameterized distribution over the noise-free latents  $p_\theta(z_0)$  is then defined as:

$$p_\theta(z_0) = \int p_\theta(z_{0:T}) dz_{1:T} \quad (11)$$

with  $p_\theta(z_{0:T}) = p_\theta(z_T) \prod_{t=1}^T p_\theta^{(t)}(z_{t-1}|z_t)$ .

The goal in training a diffusion model is thus to optimize the parameters  $\theta$  that are used to parameterize the *reverse* transitions  $p_\theta^{(t)}(z_{t-1}|z_t)$ , which intuitively remove some of the noise from  $z_t$ , such that  $p_\theta(z_0) \approx q(z_0)$ . One key finding from [78] is that in the limit of  $T \rightarrow \infty$ , the reverse transitions become Gaussians with diagonal covariance matrix, thus in practice all *reverse* transitions  $p_\theta^{(t)}(z_{t-1}|z_t)$  are assumed to be diagonal Gaussian distributions where the mean and covariance are parameterized using a DNN. Originally, diffusion models were trained by optimizing the parameters of the model that is used to predict the means and covariance matrices of those reverse transitions to maximize the variational lower bound [78].

[39] found that, if one uses fixed covariances for the *reverse* transitions, it is possible to instead optimize a loss function that resembles a weighted denoising objective:

$$L(\theta) = \sum_{t=1}^T \gamma_t \mathbb{E}_{z_0 \sim q(z_0), \epsilon \sim \mathcal{N}(0, \mathbf{I})} \left[ \|\epsilon_\theta^{(t)}(\sqrt{\alpha_t}z_0 + \sqrt{(1-\alpha_t)}\epsilon) - \epsilon\|_2^2 \right]. \quad (12)$$

Here,  $\epsilon_\theta^{(t)}$  is a denoising model that, given a noisy latent  $\sqrt{\alpha_t}z_0 + \sqrt{(1-\alpha_t)}\epsilon$  at time step  $t$ , tries to predict the added noise  $\epsilon$ , and  $(\gamma_t)_{t=1}^T$  is a sequence of weights for the individual time steps that depend on  $(\alpha_t)_{t=1}^T$ . In practice, all  $\epsilon_\theta^{(t)}$  are parameterized using a single U-Net which is given the current time step  $t$  as additional input, i.e.  $\epsilon_\theta^{(t)}(z) := \epsilon_\theta(z, t)$ .

Once  $\epsilon_\theta$  has been trained, there are multiple samplers that allow us to obtain a new latent  $z_0$ . In all cases, one starts by sampling from the prior distribution  $z_T \sim \mathcal{N}(0, \mathbf{I})$ . For this work, we focus on the DDIM solver, which is a deterministic solver, i.e. all the randomness of the process lies in  $z_T$  whereas the rest of the chain  $z_{0:(T-1)}$  is fully determined by  $z_T$ . The update rule for DDIM is:

$$z_{t-1} = \frac{\sqrt{\alpha_{t-1}}z_t - \sqrt{1-\alpha_t}\epsilon_\theta(z_t, t)}{\sqrt{\alpha_t}} + \sqrt{1-\alpha_{t-1}}\epsilon_\theta(z_t, t). \quad (13)$$

DDIM can best be understood from Eq. (10) by assuming that  $\epsilon = \epsilon_\theta(z_t, t)$  and solving for  $z_0$ . Intuitively, this is equivalent to skipping all intermediate time steps and jumping directly from  $z_t$  to  $z_0$ :

$$z_0 = \frac{z_t - \sqrt{(1-\alpha_t)\epsilon_\theta(z_t, t)}}{\sqrt{\alpha_t}}. \quad (14)$$

Now if we apply Eq. (10) to our estimate of  $z_0$  to get to time step  $t-1$  and again use our noise estimate  $\epsilon = \epsilon_\theta(z_t, t)$ ,

we can recover the DDIM update rule. More formally, DDIM sampling is related to solving the probability flow ODE introduced in [80] using the Euler method, see Proposition 1 in [79]. Considering the connection between ODEs and ResNets described in [16], it is not surprising that the DDIM updates have the residual connection that allows for easy gradient flow through diffusion graphs:

$$z_{t-1} = \frac{\sqrt{\alpha_{t-1}}}{\sqrt{\alpha_t}} z_t + F(z_t, t),$$

where  $F(z_t, t) = \left(1 - \frac{\sqrt{\alpha_{t-1}}}{\sqrt{\alpha_t}}\right) \sqrt{1 - \alpha_{t-1}} \epsilon_\theta(z_t, t).$  (15)

## A.2. Conditional Diffusion Models

While the previous Section introduced unconditional latent diffusion models, i.e. models that learn a distribution  $p_\theta(z)$ , in practice it is often desirable to work with conditional models that give the user control over the output of the diffusion model. For example, if we are using an image dataset like ImageNet, the conditioning could be the target class we want to generate, or for the popular text-to-image models like Stable Diffusion [64], the conditioning will be a text prompt that tells the diffusion model what image it should generate.

## A.3. Classifier-Free Guidance and Cross-Attention Conditioning

Classifier-free guidance (CFG) [36] was introduced as an alternative to classifier guidance [9, 45, 56, 80]. [18] already used a class-conditional denoising model  $\epsilon_\theta(x_t, t, y)$  that was given the target class as additional input. The class label  $y$  was thereby integrated into the model via adaptive group normalization layers. They introduced classifier guidance to enforce the generation of the correct target class by strengthening the influence of  $y$  on the output of the generative process. Classifier-free guidance is an alternative that also strengthens the impact of the conditioning signal in combination with a conditional denoising model  $\epsilon_\theta(x_t, t, y)$  without the requirement of an external classifier.

In the following, we will first introduce cross-attention (XA) conditioning that is used by Stable Diffusion [64] to condition the denoising model  $\epsilon_\theta$  not only on class labels but also other modalities such as text prompts or depth maps. Then we will introduce classifier-free Guidance as a solution to strengthen the impact of the conditioning signal.

### A.3.1 Cross-Attention Conditioning

As our work is based on text-to-image Stable Diffusion [64], we restrict ourselves to text conditioning in the following Section. Thus assume that we are given a text prompt  $P$ , for example, "an image of a dog on the beach". The first step

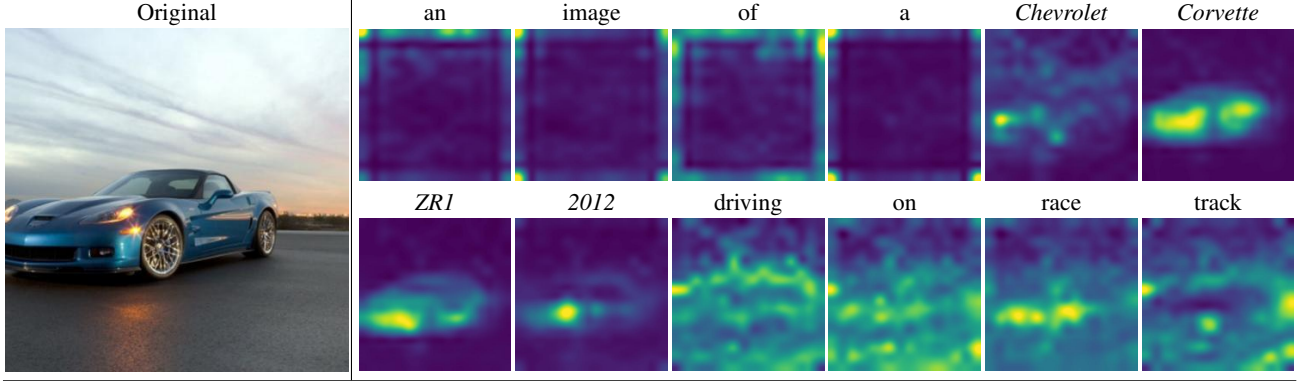


Figure 12. Visualization of the cross-attention maps produced from an image from the Cars validation set that was captioned by OpenFlamingo as "an image of a Chevrolet Corvette ZR1 2012 driving on a race track" and inverted via Null-Text inversion [53]. Given the starting latent  $z_T$  and the null-text sequence,  $(\emptyset_t)_{t=1}^T$  from the inversion, we reconstruct the image using 50 DDIM steps and save the XA maps  $M$  from the cross-attention layers inside the denoising U-Net. We show the cross-attention maps corresponding to each word for the first half of the diffusion process ( $T : (T/2)$ ) obtained at spatial resolution  $16 \times 16$  inside the U-Net averaged across all attention heads, normalized to  $[0, 1]$  and upsampled to  $512 \times 512$ . Note that the XA maps corresponding to the class name "Chevrolet Corvette ZR1 2012" can be used to locate the car in the image.

in creating a text-to-image diffusion model is to encode the prompt using a domain-specific encoder  $\tau$ . In the case of Stable Diffusion 1.4,  $\tau$  is a pre-trained CLIP [59] ViT-L/14 [19] text encoder as suggested in the Imagen paper [67]. Using  $\tau$ , one can transform the prompt  $P$  into a conditioning matrix  $C \in \mathbb{R}^{N_c \times d_\tau}$ , where  $N_c$  corresponds to the number of tokens that the prompt  $P$  is split into and  $d_\tau$  is the output feature dimension of the CLIP encoder.

In SD, the conditioning  $C$  is fed into to the denoising U-Net [65] model  $\epsilon_\theta(z_t, t, C)$  via cross-attention (XA) layers [84]. In those XA layers, the visual features of the internal representations of the current latent  $z_t$  inside the U-Net are fused with the encoded text conditioning  $C$  to generate a noise estimate  $\epsilon_\theta(z_t, t, C)$  that will not only lead us to the image manifold but also incorporate the text features. In detail, let  $\phi_i(z_t)$  denote the intermediate representations inside the U-Net of the latent  $z_t$  at time step  $t$  that are fed into the  $i$ -th XA layer. As usual in attention layers,  $\phi_i(z_t)$  is decoded into a query matrix  $Q^{(i)}$  via a linear transformation with weight matrix  $W_Q^{(i)}$ . Similarly, the conditioning  $C$  is projected into key and value matrices  $K^{(i)}$  and  $V^{(i)}$  using the weight matrices  $W_K^{(i)}$  and  $W_V^{(i)}$ . The XA operation for query, key and value matrices  $Q, K, V$  is then defined as:

$$\text{XA}(Q, K, V) = M \cdot V, \quad (16)$$

$$\text{where } M = \text{softmax} \left( \frac{QK^T}{\sqrt{d}} \right).$$

During training, the SD model is trained on a dataset containing image-text pairs and the conditioning vector  $C$  obtained from the text prompt is given to the denoising model.

This leads  $\epsilon_\theta$  to learn to use the information in  $C$  to generate a noise estimate that points to images corresponding to the conditioning information instead of the general image manifold. In practice, each attention Layer in the U-Net is implemented as multi-head attention where the attention is done multiple times in parallel and then combined to the final output via an additional linear transformation. Intuitively, as  $Q$  is a representation of the visual features from  $z_t$  and  $K$  is a representation of the textual features from the original prompt  $P$ , the output of the softmax function  $M$  can be interpreted as a similarity between visual features and text features. In particular, large entries in  $M$  correspond to spatial locations that are heavily influenced by a particular text token. We show a visual example for this in Figure 12, where we plot the XA maps obtained from reconstructing an inverted image from the Cars validation set that we use for visual counterfactual generation in Figure 7. We use the strong spatial localization in the XA maps to generate a foreground segmentation mask for our distance regularization when creating VCEs (See Section 5 and D).

### A.3.2 Classifier-Free Guidance

Even with the conditional denoising model  $\epsilon_\theta(z_t, t, C)$ , it can happen that the generated images do not follow the conditioning  $C$  close enough. Classifier-free guidance was therefore introduced to strengthen the impact of  $C$ . To do so, the denoising model is jointly trained on images *without* text prompt and the conditioning  $C$  for all of those images is replaced by the CLIP encoding of the empty string to create the null-token  $\emptyset := \tau("")$ . Intuitively  $\epsilon_\theta(z_t, t, C)$  then points to the direction of noise-free images that correspond

to the prompt  $C$  whereas  $\epsilon_\theta(z_t, t, \emptyset)$  is an unconditional noise-estimate. The estimated noise  $\epsilon$  in Eq. (1) is then replaced with the classifier-free version  $\hat{\epsilon}$

$$\hat{\epsilon}(z_t, t, C, \emptyset) = \epsilon_\theta(z_t, t, C) + w (\epsilon_\theta(z_t, t, C) - \epsilon_\theta(z_t, t, \emptyset)), \quad (17)$$

where  $w$  in Eq. (17) corresponds to the classifier-free guidance strength.

#### A.4. Diffusion Guidance via Optimization

Next, we present some additional details about our diffusion optimization. Remember from Sec. 3.2 that our goal is to find inputs to the diffusion process  $z_T, (C_t)_{t=1}^T, (\emptyset_t)_{t=1}^T$  which optimize an objective like Eq. (2).

As usual, we want to use a first-order optimizer like ADAM which requires us to calculate the gradients of the loss with respect to the input variables. Since DDIM requires at least 20 steps to yield high-quality images, it is not possible to store the entire diffusion graph for backpropagation due to memory limitations. This problem can easily be circumvented by using gradient checkpointing which allows us to calculate the exact gradients of the objective with respect to the optimization variables.

In addition, some readers might recognize the similarity between our optimization formulation and that of adversarial attacks. In general, we found the diffusion model to be a strong prior for the creation of meaningful changes instead of adversarial perturbations. Note that this behavior is not unexpected as it has been demonstrated that diffusion models can be used for adversarial purification [57]. This means that the combination of a non-adversarially robust classifier and a denoising diffusion model yields a classification pipeline with non-trivial robustness to adversarial attacks and it has been demonstrated that robust models have certain generative properties [6]. To further prevent the generation of adversarial examples, we found it helpful to use test-time augmentations on our generated images before forwarding them through the classifier  $f$  for gradient computations. In particular, we found that generating different views of the same input image and averaging the loss over all of them yields more meaningful changes. In this work, we combine two types of augmentations. First, we randomly cutout different crops from the image [89] and then add Gaussian noise to each crop. In Fig. 13, we demonstrate that this yields gradients (with respect to the input image in pixel space) that are much more localized on the class object of interest.

## B. Classifier Disagreement

In Fig. 14, we extend our analysis of the shape bias of adversarially robust models. In addition to the images from Fig. 3, we also show results from maximizing the standard

model while minimizing the robust one. The generated images show a richer texture and the shape differs significantly from the Stable Diffusion initialization which is in line with our findings in Sec. 4. Fig. 15 shows additional results for the zero-shot CLIP where we used a ViT-B as second classifier instead of a ConvNeXt-B. The results show that the choice of the second classifier has only little influence on the detected errors. A reason for this is that the zero-shot model extends the original class to a large set of out-of-distribution images (see Fig. 4) which is not the case for models that were trained or fine-tuned on ImageNet. As described in Sec. 4, we show the results for the different biases of a ViT-B and a ConvNeXt-B in Fig. 16.

### B.1. Hyperparameters

Resolution	512
Guidance Scale	3.0
DDIM steps	25
Optimizer	ADAM
Optimization steps	15
$C_t, \emptyset_t$ stepsize	0.025
$z_T$ stepsize	0.00025
Scheduler	cosine
Gradient Clipping	0.05
Num. cutouts	16
Cutout Noise $\sigma$	0.05

## C. Validation of zero-shot CLIP errors

To validate the errors found in Fig. 5, we collected similar real images from the LAION-5B dataset using the CLIP retrieval tool<sup>1</sup>. The used retrieval queries were of the form “an image of ...” and resemble the detected failure cases: “... a waffle” for “waffle iron”, “... an arch bridge” for “steel arch bridge”, “... a spoon on a wooden table” for “wooden spoon” and “... a bar in space” for “space bar”. For “steel arch bridge” and “wooden spoon”, this procedure finds many images confirming the observed failure case. In the case of “waffle iron”, some kinds of waffles also produce a high confidence for the ConvNeXt as this feature is probably also spuriously correlated in the ImageNet training data. The “space bar” example is very specific and the retrieval procedure returns only few images fitting the pattern.

## D. Visual Counterfactual Explanations

### D.1. Method Details

We start by giving a more detailed description of our universal visual counterfactual explanation (UVCE) method and motivate our design choices. As in the main paper, we assume we are given a starting image from the validation

<sup>1</sup><https://knn5.laion.ai>



---

**Algorithm 1** Diffusion Guidance via Optimization
 

---

```

Input: Loss function  $L$ , Initial Prompt  $P$ , number of iterations  $K$ 
 $z_T \sim \mathcal{N}(0, 1)$  ▷ Draw starting latent
 $C = \tau(P)$  ▷ Encode prompt
 $\emptyset = \tau(" ")$  ▷ Generic null-text

for  $t = 1, \dots, T$  do ▷ Initialize time step-dependent variables
   $C_t = C$ 
   $\emptyset_t = \emptyset$ 
end for

optim = Adam( $z_T, C_1, \dots, C_T, \emptyset_1, \dots, \emptyset_T$ ) ▷ Define the optimizer

for  $k = 1, \dots, K$  do ▷ Optimization loop
   $z = z_T$ 
  for  $t = T, \dots, 1$  do ▷ Denoising DDIM loop
    with gradient_checkpointing():
       $\hat{\epsilon} = \epsilon_\theta(z, t, C_t) + w (\epsilon_\theta(z, t, C_t) - \epsilon_\theta(z, t, \emptyset_t))$  ▷ CFG update (17)
       $z = \sqrt{\alpha_t} \frac{z - \sqrt{1 - \alpha_t} \hat{\epsilon}}{\sqrt{\alpha_t}} + \sqrt{1 - \alpha_t} \hat{\epsilon}$  ▷ DDIM step (13)
    end for

     $x = \mathcal{D}(z)$  ▷ Decode final latent using VAE decoder
     $l = L(x)$  ▷ Calculate loss  $l$ 
     $l.backward()$  ▷ Calculate gradients

    optim.step()
    optim.zero_grad()
  end for
return  $z_T, (C_t)_{t=1}^T, (\emptyset_t)_{t=1}^T$ 

```

---

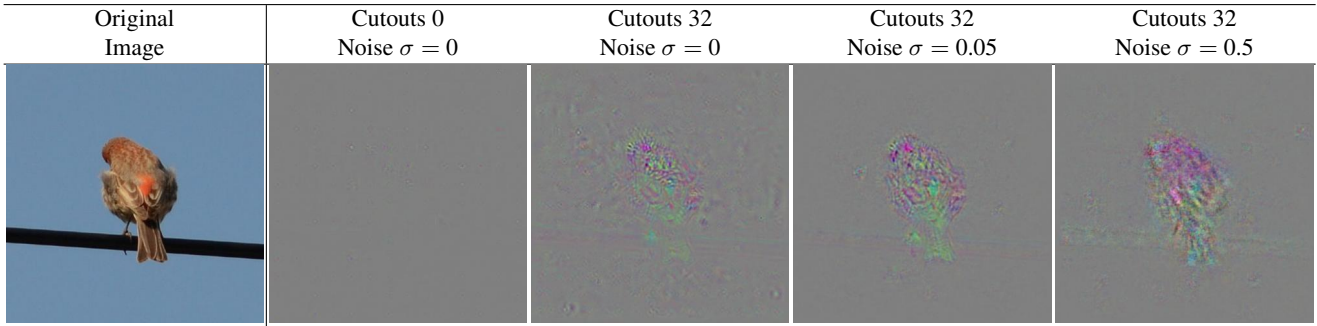


Figure 13. We plot the gradient  $\nabla_x p_f(y|x)$  with different test-time augmentations, including Cutout and Gaussian Noise with two standard deviations. The classifier  $f$  is a ViT and the original image is an ImageNet validation image for the class "house finch" and the target class  $y$  is "gold finch". Note that the gradient without augmentation is very noisy and not located on the bird. If we average the gradient across slightly perturbed images, we can achieve localization on the foreground object. While adding noise on top of the Cutout augmentation can further improve localization, too much noise ( $\sigma = 0.5$ ) leads to very coarse gradients that are no longer usable for optimization. Each gradient is separately rescaled to fit in  $[0, 1]$  and grey values of 0.5 correspond to a zero gradient.

set  $\hat{x}$  belonging to class  $\hat{y}$  and our goal is to create a VCE  $x$  that is classified as target class  $y$  by the classifier  $f$ . In the next subsections, we go over the individual steps of the UVCE process. The UVCE generation can be split into the following parts:

- i) Create a caption of the image using OpenFlamingo
- ii) Invert the image using Null-Text inversion
- iii) Obtain XA maps and compute foreground mask
- iv) Optimize the confidence into the target class and

		$p_f$ : Confidence Robust ViT-S vs. $p_g$ : Confidence ViT-S							
		Head Cabbage ( $p_f / p_g$ )		Koala ( $p_f / p_g$ )		Brown Bear ( $p_f / p_g$ )		Dugong ( $p_f / p_g$ )	
	SD Init.	0.57 / 0.95	0.70 / 0.95	0.79 / 0.96	0.76 / 0.97	0.76 / 0.96	0.67 / 0.96	0.01 / 0.01	0.14 / 0.92
	$p_f \uparrow - p_g \downarrow$	0.82 / 0.00	0.79 / 0.00	0.86 / 0.00	0.92 / 0.06	0.80 / 0.00	0.76 / 0.00	0.66 / 0.02	0.78 / 0.00
	$p_g \uparrow - p_f \downarrow$	0.00 / 0.96	0.02 / 0.98	0.45 / 0.94	0.06 / 0.96	0.09 / 0.97	0.00 / 0.99	0.06 / 0.96	0.08 / 0.97

Figure 14. **Classifier disagreement: shape bias of adversarially robust models (extended)**. This is an extended version Fig. 3 where we additionally show images maximizing the confidence of the standard model and minimizing the confidence of the robust one (third row) while starting from the same initial Stable Diffusion image. In contrast to the second row, these images show significant shape changes and a richer texture compared to the ones of the second row (maximizing/minimizing confidence of the robust/standard model). In particular, the images of the second row are mainly “cartoon”-like versions of the SD initializations with little texture.

		Waffle Iron ( $p_f / p_g$ )		Steel Arch Bridge ( $p_f / p_g$ )		Wooden Spoon ( $p_f / p_g$ )		Space Bar ( $p_f / p_g$ )	
	SD Init.	1.00 / 0.51	1.00 / 0.76	0.71 / 0.01	0.86 / 0.00	0.99 / 0.93	0.72 / 0.85	0.09 / 0.01	0.02 / 0.00
	$p_f \uparrow - p_g \downarrow$	1.00 / 0.01	1.00 / 0.00	1.00 / 0.00	1.00 / 0.00	0.98 / 0.00	0.92 / 0.04	1.00 / 0.00	0.99 / 0.00
	$p_f \uparrow - p_g \downarrow$	1.00 / 0.04	0.99 / 0.01	0.99 / 0.01	1.00 / 0.01	0.99 / 0.04	0.26 / 0.07	0.99 / 0.08	0.97 / 0.00

Figure 15. **Detected zero-shot CLIP errors are independent of the minimized classifier:** We show the results for maximizing the zero-shot CLIP while minimizing ConvNeXt-B (second row, as in Fig. 5) and minimizing a ViT-B (third row). The zero-shot CLIP extends the original classes to much larger sets of out-of-distributions images compared to models trained or fine-tuned on ImageNet. Therefore, the failure cases discovered by maximizing classifier disagreement do not depend on the choice of the minimized classifier.

background similarity to the original image

Algorithm 2.

Additionally, we show a diagram titrhuguublvrllkgvt-bvnrkhuvvlin Fig. 17 and give an algorithmic overview in












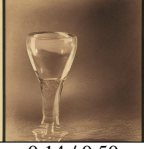

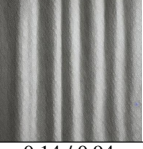




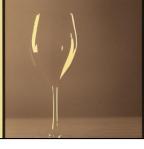



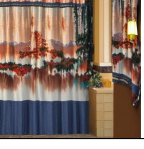
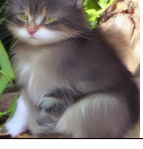

		$p_f$ : Confidence ViT-B vs. $p_g$ : Confidence ConvNeXt-B							
		Goblet ( $p_f / p_g$ )		Vase ( $p_f / p_g$ )		Shower Curtain ( $p_f / p_g$ )		Tabby ( $p_f / p_g$ )	
SD Init.		0.96 / 0.93	0.86 / 0.68	0.83 / 0.76	0.95 / 0.44	0.69 / 0.89	0.99 / 0.92	0.21 / 0.03	0.18 / 0.32
									
		0.99 / 0.81	0.86 / 0.68	0.83 / 0.05	0.96 / 0.04	0.99 / 0.06	0.99 / 0.02	0.88 / 0.05	0.83 / 0.02
$p_f \uparrow - p_g \downarrow$									
		0.07 / 0.90	0.14 / 0.50	0.18 / 0.89	0.20 / 0.76	0.14 / 0.94	0.07 / 0.96	0.07 / 0.71	0.14 / 0.79
$p_g \uparrow - p_f \downarrow$									

Figure 16. **Classifier disagreement: ViT vs. ConvNeXt.** For a given class label  $y$ , the first row shows the output of Stable Diffusion for “a photograph of  $y$ ”. The images in the other rows have been optimized to maximize the difference of the confidence between a ViT-B and a ConvNeXt-B. Empty wine glasses are classified as “goblet” by the ConvNeXt-B, whereas the ViT-B predicts “red wine”. For the class “vase”, realistic images without flower blossoms (high confidence for ViT-B) and paintings with more pronounced blossoms (high confidence for ConvNeXt-B) result in a large difference of confidence. Only the ConvNeXt, but not the ViT, predicts “shower curtain” for colorful exemplars and the opposite holds for the gray ones. A close-up of a cat face with large green eyes triggers only the ViT’s prediction of “tabby cat”, while only the ConvNeXt model assigns a high confidence to a zoomed-out version without eyes.

### D.1.1 Captioning

As every DDIM inversion requires a prompt, we first have to generate a prompt that describes  $\hat{x}$ . As we are going to use the XA maps to create a foreground segmentation map, it is important to have an accurate description of both the foreground object but also the background, such that in the XA layers, only the spatial locations in the image belonging to the class object attend to words from the class name corresponding to  $\hat{y}$ , which we call  $\langle \text{ORIGINAL CLASSNAME} \rangle$ . We found that using the generic caption “an image of a  $\langle \text{ORIGINAL CLASSNAME} \rangle$ ” results in worse post-inversion reconstruction qualities and can result in words contained in the class name attending to locations in the background of the image as these background objects do not have matching descriptions in the generic caption. We, therefore, use Open-Flamingo [3, 7] to enhance the generic captions. In particular, we manually label less than 30 images from the training set and always use the form: “an image of a  $\langle \text{ORIGINAL CLASSNAME} \rangle$   $\langle \text{BACKGROUND DESCRIPTION} \rangle$ ”, for example, “an image of a koala hanging on a tree”. We can then use the Flamingo model to take the image  $\hat{x}$  with the generic prompt “an image of a  $\langle \text{ORIGINAL CLASSNAME} \rangle$ ” as input and add a background description that resembles our handcrafted ones. We call the resulting prompt  $\hat{P}$ . In particular, due to its construction,  $\hat{P}$  is guaranteed to contain the name of the starting class.

To use  $\hat{P}$  as conditioning within the Stable Diffusion pipeline, we then encode the prompt  $\hat{P}$  into its representation  $\hat{C} = \tau(\hat{P})$  using the CLIP text encoder  $\tau$ .

### D.1.2 Inversion:

Next, we have to invert  $\hat{x}$ , i.e. find a latent  $z_T$  that, together with the conditioning  $\hat{C}$  reconstructs the original image. The standard DDIM inversion [79] often results in bad inversions that do not recreate  $\hat{x}$ . We, therefore, use Null-Text inversion [53], which uses the DDIM inversion with its latent  $z_T$  as initialization and then optimizes the null-text tokens  $(\emptyset_t)_{t=1}^T$  such that the image resulting from the diffusion process matches the original image  $\hat{x}$ , i.e.  $\hat{x} \approx \mathcal{D}(z_0(z_T, \hat{C}, (\emptyset_t)_{t=1}^T))$ , where we use  $z_0$  for the function that takes a starting latent  $z_T$ , conditioning matrix  $\hat{C}$  and the null-text sequence  $(\emptyset_t)_{t=1}^T$  and returns the final latent obtained from running the entire diffusion process.

### D.1.3 Initialization using XA-injections

Our objective is to create an image  $x$  that is similar to  $\hat{x}$  but shows an object from the new target class  $y$ . To achieve this, we can make use of the knowledge contained in SD to find a better initialization in the CLIP encoding space. A good initialization is important because our optimization problem is highly non-convex, thus the initialization will di-

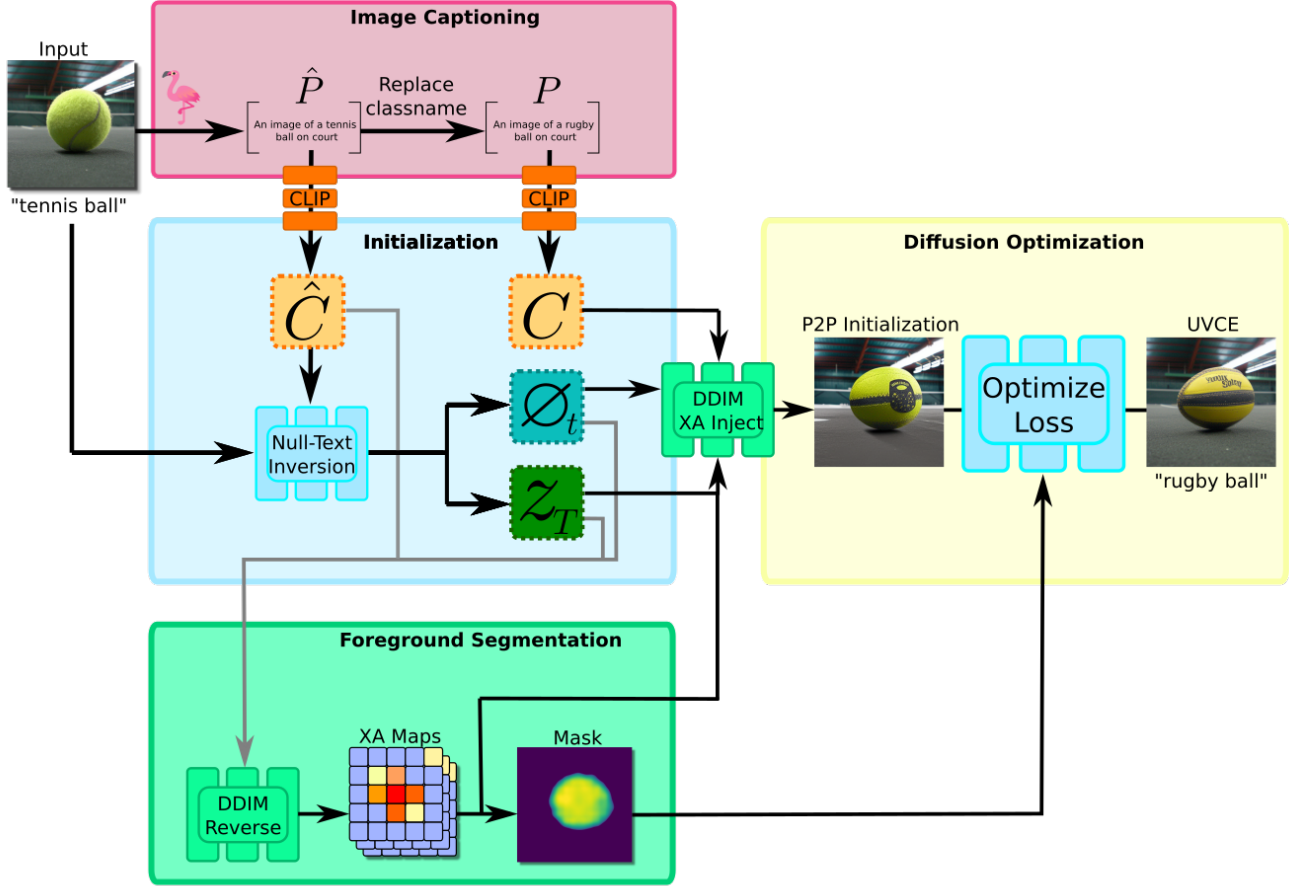


Figure 17. A graphical representation of our DiG-IN UVCE generation. We start with the input image and caption it using OpenFlamingo to get a prompt  $\hat{P}$ . A new prompt  $P$  containing the target class name is generated via string replacement. Both are encoded via the CLIP text model to get the conditionings  $\hat{C}$  and  $C$  belonging to the original and target class names. We then use Null-Text inversion with the *original* prompt  $\hat{C}$  to get a starting latent  $z_T$  and null-text sequence  $(\varnothing_t)_{t=1}^T$  which can be used to re-generate the input image. We then run the standard DDIM denoising process using  $z_T$ ,  $\hat{C}$  and  $(\varnothing_t)_{t=1}^T$  which will restore the original image and allows us to capture the Cross-Attention (XA) maps. These can be used to produce a point prompt for a segmentation model to create a segmentation map of the foreground object. The initialization for our optimization is obtained by replacing the original conditioning  $\hat{C}$  with the new conditioning  $C$  and by using prompt-to-prompt-like XA injection using the stored XA maps. As the resulting image will often have low confidence in the target class and/or be too far away from the input image, we optimize  $z_T$ ,  $(C_t)_{t=1}^T$  and  $(\varnothing_t)_{t=1}^T$  using the ADAM optimizer to obtain our final UVCE.

rectly influence the resulting image as we can not guarantee convergence to the global minimum and also, we are interested in producing images with as few optimization steps as possible. It is thus natural to take the original prompt  $\hat{P}$  and create a new prompt  $P$  by replacing the name of the starting class <ORIGINAL CLASSNAME> with the name of the target class <TARGET CLASSNAME>. After encoding using the CLIP encoder  $\tau$ , we then get an additional conditioning  $C = \tau(P)$ , corresponding to the prompt containing the label of the target class  $y$ .

Note that the Null-Text inversion naturally results in a time-step-dependent sequence of null-text tokens  $(\varnothing_t)_{t=1}^T$ , which is why we also adopted time-step-dependent condi-

tioning  $(C_t)_{t=1}^T$  to have the same degrees of freedom in both the null-text and the conditioning during optimization. We initialize  $C_t = C$  for all  $t$ .

The issue is that even local changes in the conditioning tend to have a global impact on the final image, which will lead to  $\mathcal{D}(z_0(z_T, C, (\varnothing_t)_{t=1}^T))$  looking very different from  $\mathcal{D}(z_0(z_T, \hat{C}, (\varnothing_t)_{t=1}^T))$ , not only in the foreground but also in the background (we refer readers to Figure 5 in the original Prompt-to-Prompt paper [34] for a visualization). As our goal is to create a VCE that resembles the original image, this is highly undesirable as we would have to spend many optimization steps to minimize the distance in the background between our new image and the starting image.



[34] found that the overall image structure is mostly dictated by the first diffusion steps and the XA maps inside the denoising U-Net  $\epsilon$ . It is thus possible to preserve the overall image structure by injecting the XA maps that lead to the creation of one image when creating a new image with a modified prompt. Recall from Section A.3.1 that inside the  $i$ -th XA layer in the U-Net, we compute a weight matrix  $M^{(i)}$  that measures similarity between the U-Net encoded spatial features from the current latent  $\phi_i(z_t)$  and the encoded text prompt  $C_t$ . In detail,  $M^{(i)}$  corresponds to the softmax-normalized similarity between:

- The query matrix  $Q^{(i)}$ , i.e. the projected internal representation of  $z_t$  inside the U-Net  $W_Q^{(i)} \cdot \phi_i(z_t)$  where the number of rows corresponds to the spatial resolution of the output of  $\phi_i$ , e.g.  $16 \times 16 = 256$ . We call this spatial resolution  $N_{\phi_i}$ .
- The key matrix  $K^{(i)}$ , i.e. the projected conditioning  $C_t$  at time step  $t$ :  $W_K^{(i)} \cdot C_t$ . The number of rows in  $K^{(i)}$  corresponds to the number of tokens  $N_c$  that the prompt was split into in the tokenizer of the CLIP encoder.

As  $M^{(i)}$  is defined as the post softmax output of  $Q^{(i)} \cdot (K^{(i)})^T$ ,  $M^{(i)}$  is a matrix of size  $N_{\phi_i} \times N_c$ . The  $(j, k)$ -th entry can therefore be interpreted as the similarity between the spatial features at position  $j$  in the flattened version of  $\phi_i(z_t)$  and the  $k$ -th token in the conditioning matrix  $C_t$ . Now let  $\hat{M}_t^{(i)}$  correspond to the XA maps that can be obtained from the  $i$ -th XA layer inside the denoising U-Net at time step  $t$  when running the diffusion process with the *original* conditioning  $\hat{C}$ . Due to the null-text inversion, this diffusion process will nearly perfectly reconstruct the original image  $\hat{x}$  and thus the XA maps  $\hat{M}_t^{(i)}$  will capture the structure of the *original* image.

During optimization, we now want to re-inject those XA maps when using our modified conditioning sequence  $(C_t)_{t=1}^T$ . Let  $M_t^{(i)}$  denote the *new* XA maps at time step  $t$  at the  $i$ -th XA layer of the U-Net that corresponds to the similarity between the spatial features and tokens belonging to the current conditioning  $C_t$  being optimized instead of the original conditioning  $\hat{C}$ . [34] found that it is not necessary to inject the original XA maps  $\hat{M}$  throughout the entire diffusion process and therefore only did the XA injection for a certain part of the diffusion process.

Note that the original Prompt-to-Prompt implementation only supports the replacement of words in a 1-to-1 fashion. However, in our case,  $\langle \text{ORIGINAL CLASSNAME} \rangle$  and  $\langle \text{TARGET CLASSNAME} \rangle$  can have a different number of words. We, therefore, calculate a similarity matrix that measures the cosine distance between all words in both strings in CLIP embedding space (if a word is encoded into multiple tokens, we average all of them to get a word-level representation in the CLIP latent space) and use these distances to reshape  $\hat{M}_t^{(i)}$  into a matrix that has the same size as  $M_t^{(i)}$

via weighted averaging.

#### D.1.4 Distance regularization and Optimization

Given  $z_T$ ,  $(C_t)_{t=1}^T$  and  $(\emptyset_t)_{t=1}^T$  and the original XA maps  $\hat{M}_t^{(i)}$  we can finally define our optimization objective as in Eq. (5). Our new initialization will show class features from  $y$  and be relatively similar to  $\hat{x}$ , however, as we show in Figure 18a, the resulting images often have quite low confidence in the target class  $y$  and non-localized changes. Optimizing the confidence can again be done by maximizing  $\log p_f(y|x)$ . For the distance regularization, we use the segmentation-based regularization described in the main paper in Eq. (4).

The idea behind our distance regularization is to allow the UVCE process to do larger color changes on the object itself if necessary but keep the background as close to the original image as possible. If we know the pixel locations corresponding to the foreground object in the original image, i.e. we are given a foreground segmentation mask  $S_{\text{PX}}$  it is easy to define the distance regularization in pixel space as was done in the main paper:

$$\|(1 - S_{\text{PX}}) \odot (\mathcal{D}(z) - \hat{x})\|_2^2. \quad (18)$$

Since we know the class name associated with the foreground object, we could in principle use a segmentation model with text prompting to obtain such a mask. However, we noticed that off-the-shelf text prompted models often yield unreliable segmentation masks. To overcome this, we first generate a mask estimate using the XA maps and then use this to generate a point prompt for HQ-SAM [42] to generate a segmentation mask in pixel space. We found point prompting with pixel locations belonging to the foreground object to yield much more reliable segmentation masks. As noted previously, we can use the XA maps to get an idea of the location of the object in the image. Typically, the overall structure is captured best by the XA maps corresponding to the earlier diffusion steps. We, therefore, average the initial XA maps  $\hat{M}_t^{(i)}$  from the first half of the diffusion process at resolution  $16 \times 16$  in the U-Net that belong to all tokens that correspond to the words in  $\langle \text{ORIGINAL CLASSNAME} \rangle$  to approximate the location of the object in the image (see also Figure 12). For example, for the dog breed "Cocker Spaniel", we average the XA maps that correspond to the 3 tokens that the CLIP tokenizer uses to encode this class name. We then upscale this initial segmentation mask to the resolution of the original image, normalize it to have a maximum value of 1, and set all values below a pre-defined background threshold to 0 to obtain a first segmentation mask. Due to the low resolution and high amount of noise in the XA maps, this initial segmentation can be quite inaccurate. We therefore randomly sample 5 points from this initial mask and use it to prompt the HQ-SAM model. This gives us a binary

---

**Algorithm 2** DiG-IN UVCE Generation

---

**Input:** Input image  $\hat{x}$ , Starting class  $\hat{y}$ , Target class  $y$ , number of iterations  $K$ , Classifier  $f$

start\_classname = ClassNames[ $\hat{y}$ ] ▷ Create prompts  
target\_classname = ClassNames[ $y$ ]  
 $\hat{P}$  = open\_flamingo( $\hat{x}$ , "an image of a" + start\_classname)  
 $P$  =  $\hat{P}$ .replace(start\_classname, target\_classname)

$\hat{C} = \tau(\hat{P})$  ▷ CLIP Encode prompts  
 $C = \tau(P)$

$z_{\text{Original}}, \emptyset_1, \dots, \emptyset_T = \text{null\_text\_inversion}(\hat{x}, \hat{C})$  ▷ Invert Image  
 $z_T = z_{\text{Original}}.\text{clone}().\text{detach}()$

xa\_maps = ddim\_loop\_extract\_xa( $z_T, \hat{C}, \emptyset_1, \dots, \emptyset_T$ ) ▷ Extract Cross-Attention maps  
 $S_{\text{PX}}, S_{\text{VAE}} = \text{make\_segmentation\_from\_xa}(\text{xa\_maps})$  ▷ Get Pixel and latent space masks

**for**  $t = 1, \dots, T$  **do** ▷ Initialize time step-dependent conditioning  
     $C_t = C$   
**end for**

optim = Adam( $z_T, C_1, \dots, C_T, \emptyset_1, \dots, \emptyset_T$ ) ▷ Define the optimizer

**for**  $k = 1, \dots, K$  **do** ▷ Optimization loop  
     $z = z_T$   
     $z_0 = \text{ddim\_loop\_xa\_inject}(z_T, C_1, \dots, C_T, \emptyset_1, \dots, \emptyset_T, \text{xa\_maps})$

$x = \mathcal{D}(z_0)$  ▷ Decode final latent using VAE decoder  
     $l_{\text{CE}} = -\log p_f(y|x)$  ▷ Calculate losses  
     $l_d = w_{\text{VAE}}\|(1 - S_{\text{VAE}}) \odot (z_0 - z_{\text{Original}})\|_2^2 + w_{\text{PX}}\|(1 - S_{\text{PX}}) \odot (x - \hat{x})\|_2^2.$   
     $l = l_{\text{CE}} + l_d$   
     $l.\text{backward}()$  ▷ Calculate gradients

    optim.step()  
    optim.zero\_grad()

**end for**  
**return**  $z_T, (C_t)_{t=1}^T, (\emptyset_t)_{t=1}^T$

---

mask in the size of the original image. We post-process this mask using erosion and dilation filtering as well as Gaussian blurring to obtain the mask in pixel space  $S_{\text{PX}}$ . We show examples of our masks in Fig. 18b. We found it beneficial to also regularize the distance in the VAE latent space and therefore downsample  $S_{\text{PX}}$  by the VAE downsampling factor to obtain the VAE latent mask  $S_{\text{VAE}}$ . Given those masks, we can define our regularizer as:

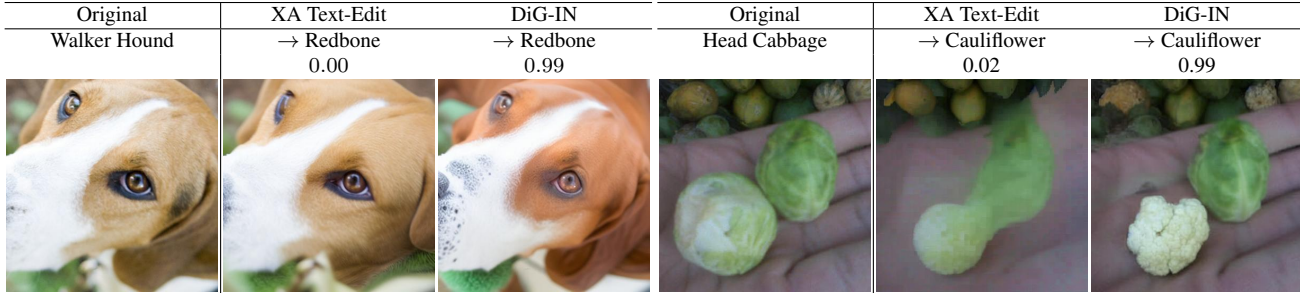
$$d(z, \hat{x}) = w_{\text{VAE}}\|(1 - S_{\text{VAE}}) \odot (z - \mathcal{E}(\hat{x}))\|_2^2 + w_{\text{PX}}\|(1 - S_{\text{PX}}) \odot (\mathcal{D}(z) - \hat{x})\|_2^2. \quad (19)$$

With this masked distance regularizer, we allow our optimization to arbitrarily change the foreground object, for example, it allows us to have large color changes that are not

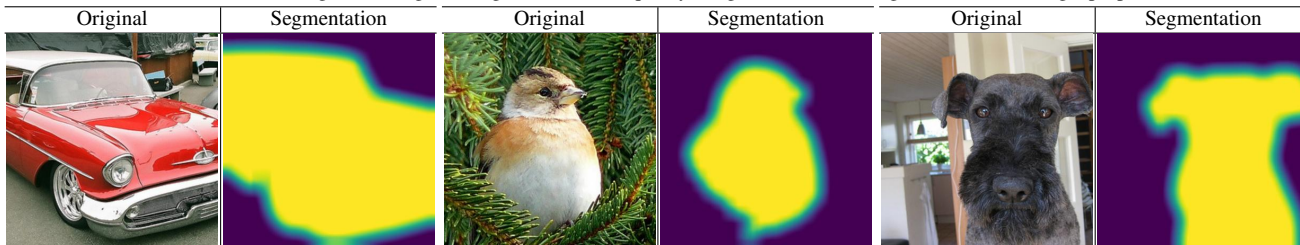
achievable with standard  $l_p$  regularization while still enforcing a strong similarity in the background of the image. We emphasize that it is important to have an accurate description of the background to make sure that background pixels are not captured by words in the class name, which is why it is important to use detailed Flamingo captions instead of generic ones.

Similar to the original Prompt-to-Prompt paper [34], we also found it beneficial to use mask blending outside of the foreground mask to further enforce background similarity to the original image in the first steps of the diffusion process.

Given the foreground mask, we optimize the starting latent  $z_T$ , our modified prompt embedding  $(C_t)_{t=1}^T$ , and the null-text sequence  $(\emptyset_t)_{t=1}^T$  to maximize the objective given in Eq. (5).



(a) **Comparison of UVCEs to text-guided changes with XA injection that we use as initialization:** We show examples where the P2P-style initialization fails. For the image on the left, P2P can preserve the overall image structure, however, the word replacement from "Walker Hound" to "Redbone" in the prompt is not sufficient for generating an image that is labeled as "Redbone" with a high confidence. Our DiG-IN optimization is able to add the missing features and achieve 0.99 confidence. For the image on the right, P2P generates a low-quality image and blurs the fingers without creating a proper Cauliflower.



(b) **Foreground segmentation masks** created via point-prompting HQ-SAM [42].

Figure 18

### D.1.5 Hyperparameters

The hyperparameters are *identical* across all UVCE tasks and images presented in this paper. This shows that our method can adapt to a large variety of image configurations and supports new classifiers as well as very different image datasets without any hyperparameter tuning.

Resolution	512
Guidance Scale	3.0
DDIM steps	20
Optimizer	ADAM
Optimization steps	20
$C_t, \varnothing_t$ stepsize	0.01
$z_T$ stepsize	0.001
Scheduler	$\times$
Gradient Clipping	$\times$
$w_{VAE}$	25.0
$w_{PX}$	250.0
Num. cutouts	16
Cutout Noise $\sigma$	0.005

## D.2. Qualitative Result

First, we demonstrate the impact of the guiding classifier on the resulting DiG-IN UVCEs in Fig. 19 using 4 state-of-the-art classifiers. In addition to the ViT-B [81] pre-trained on IN21K we used for the ImageNet VCEs in the main paper, we evaluate a ConvNeXt-L [48] pre-trained with CLIP loss on Laion-2B [70], a ConvNeXt-V2-H [91] pre-trained on

IN21K and a EVA02-L [23] trained on MIM38M. All models are fine-tuned on IN1K. As can be seen, the guiding classifier can have a strong impact on the resulting image which shows the impact of our optimization even when using the Prompt-to-Prompt initialization.

In Fig. 20 we show additional qualitative results on ImageNet where we compare our UVCEs to DVCEs [5] in the generation of classes that are close in the WordNet hierarchy. As the classifier, we use the same ViT-B as in the main paper. We again demonstrate that we can generate highly realistically looking UVCEs with minimal background changes that achieve high confidence in the target class. Unlike DVCEs, we cannot only handle texture changes but also more complex class changes that require editing the geometry of the image. For a selection of *random* images, please also refer to Fig. 27 and Fig. 28 which show the images for the user study.

Additionally, in Fig. 21, Fig. 22 and Fig. 23, we present UVCEs in a more fine-grained context for the CUB, Food-101 and Cars datasets. The classifiers are the same as in Fig. 7, namely a CAL-ResNet101 [62] classifier for Cars, a fine-tuned ViT-B [30] for CUB and a fine-tuned ViT-B on Food-101. Our broad model selection shows that our UVCEs cannot only be used to explain standard vision transformers but also support models with different pre-training strategies as well as convolutional neural networks without adjusting hyperparameters. Note that DVCEs only support ImageNet due to the requirement for a diffusion model trained on

that dataset and a robust classifier trained on the specific dataset which are not available for the CUB, Cars, or Food-101 datasets. Similar to ImageNet, we are able to create highly realistic VCEs that can handle very fine-grained class changes which cannot be achieved via textural changes that also preserve the image background due to our background distance regularization.

### D.2.1 EVA02 Error UVCEs

It can also be interesting to use UVCEs as a tool to understand prediction errors on the validation set. For this, we first evaluate the test set accuracy and save the indices corresponding to all images where the predicted class does not match the target class. We then create a UVCE into the target class by optimizing equation Eq. (5). Since we already use the target class name for the initial prompt  $\hat{P}$  and during inversion, we cannot use Prompt-to-Prompt for such error UVCEs. Nevertheless, as Fig. 25 shows, our DiG-IN UVCE method can generate meaningful images that achieve a high confidence by optimizing the cross-entropy into the target class. The EVA02-L we used to generate these UVCEs classifier achieves an ImageNet-1K validation accuracy of 90.05%, however, as our UVCEs show, the real accuracy is likely higher, since most errors are either caused by ambiguous labels (*i.e.* multiple objects on the image) or wrongly labeled validation images.

### D.2.2 Zero-Shot Attribute UVCEs

Lastly, we demonstrate UVCEs for zero-shot attribution classification using a CLIP model.

Assume we want to create a counterfactual that changes a certain source attribute to a target attribute in an image, for example, "smiling" to "looking sad" for a zero-shot attribute classifier. We first describe how we turn the CLIP model into a binary zero-shot classifier for this particular image  $\hat{x}$  and attribute. We again start with a prompt  $\hat{P}$  that contains the textual description of the source attribute and image, for example, "a closeup portrait of a man smiling". We then replace the source with the target attribute to obtain  $P$ , in this case, "a closeup portrait of a man looking sad". Note that we use the same prompts  $\hat{P}$  and  $P$  for the zero-shot classification as well as the DDIM inversion and DiG-IN generation.

Next, both prompts are encoded by the *text* encoder of the CLIP model  $f_{\text{txt}}$ . Given an input image  $x$ , we decode it using the *image* encoder  $f_{\text{im}}$ . For both encoders, we assume that the outputs are normalized as per usual. Now we can calculate the logits for the two classes (corresponding to the target and source attribute) as:

$$\langle f_{\text{txt}}(P), f_{\text{im}}(x) \rangle \quad \text{and} \quad \langle f_{\text{txt}}(\hat{P}), f_{\text{im}}(x) \rangle \quad (20)$$

The log-probability of the binary zero-shot classifier detecting the target attribute is thus given by:

$$\log \frac{\exp(\langle f_{\text{txt}}(P), f_{\text{im}}(x) \rangle)}{\exp(\langle f_{\text{txt}}(P), f_{\text{im}}(x) \rangle) + \exp(\langle f_{\text{txt}}(\hat{P}), f_{\text{im}}(x) \rangle)}. \quad (21)$$

For the examples in Fig. 24, we use face images from FFHQ [41] and the OpenCLIP [38] implementation of the CLIPA [46] ViT-H/14 trained on DataComp-1B [25]. Faces are generally challenging for VCEs as humans are naturally good at recognizing minor modifications between the generated and original image. Still, our examples demonstrate that we can create highly realistic UVCEs that only change the source to the target attribute while preserving the overall facial structure. In particular, for all examples, the person in the generated image can clearly be identified to be the same as the one in the starting image and the faces look realistic without any artifacts.

## E. User Study

For the user study, we collected 30 pairs of original and target classes at random from a pool of more than 3000 VCEs that were generated for similar classes in the ImageNet hierarchy and asked 30 participants to answer the four questions described in Sec. 5. Each participant assessed 1 to 30 image pairs. They participated voluntarily (without payment) and had not seen the generated images before.

During the random selection of examples, we disregarded cases where the optimization failed for one of the methods by thresholding the confidence in the target class at 0.8. We also provided four random training images of the target class as a reference as well as the original image. The VCEs were displayed as "Counterfactual A" and "Counterfactual B" (see Fig. 26). For each participant, the order of the examples, as well as the (per example) assignment of the two methods to "A" and "B" were chosen at random. All images used in the user study along with the individual results can be found in Fig. 27 and Fig. 28.

## F. Neuron Activations

### F.1. Synthetic Neuron Visualizations

For synthetic neuron visualizations, we start by computing the activations over the train set. For a given neuron  $n$ , we then ask CogAgent [37] to list the objects in the 5 most activating train images via the prompt "list the most important objects in the image in a list format starting with [ and ending with ] without a full sentence". For each object, we use Stable Diffusion to generate 8 images using the prompt "a photograph of a <OBJECT>" and use the conditioning from the encoded prompt of the object that achieves the highest mean activation for neuron  $n$  as initialization  $C$  for our optimization.




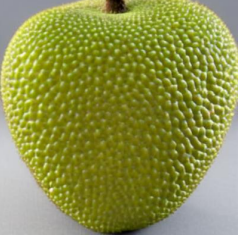






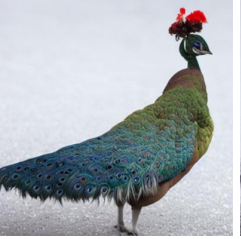






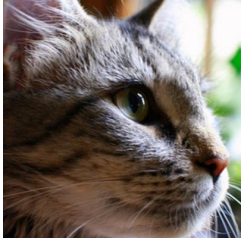


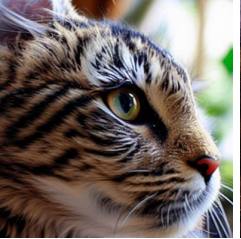
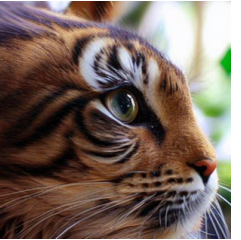





ImageNet Validation Image	Guidance: ViT-B	Guidance: ConvNeXt-L	Guidance: ConvNeXt-V2-H	Guidance: EVA02-L
Custard Apple	→: Jackfruit 0.99 / 0.90 / 0.95 / 0.72	→: Jackfruit 0.41 / 0.93 / 0.92 / 0.67	→: Jackfruit 0.93 / 0.93 / 0.96 / 0.72	→: Jackfruit 0.03 / 0.22 / 0.28 / 0.81
				
Ptarmigan	→: Peacock 0.99 / 0.86 / 0.90 / 0.67	→: Peacock 0.86 / 0.92 / 0.69 / 0.64	→: Peacock 0.97 / 0.87 / 0.72 / 0.73	→: Peacock 0.95 / 0.87 / 0.80 / 0.73
				
Kit Fox	→: Arctic Fox 0.98 / 0.78 / 0.85 / 0.39	→: Arctic Fox 0.96 / 0.97 / 0.91 / 0.74	→: Arctic Fox 0.98 / 0.91 / 0.94 / 0.70	→: Arctic Fox 0.18 / 0.74 / 0.86 / 0.87
				
Egyptian Cat	→: Persian Cat 0.95 / 0.86 / 0.95 / 0.70	→: Persian Cat 0.65 / 0.96 / 0.88 / 0.66	→: Persian Cat 0.70 / 0.89 / 0.98 / 0.59	→: Persian Cat 0.75 / 0.65 / 0.76 / 0.87
				
Jeep	→: Model T 0.99 / 0.90 / 0.95 / 0.68	→: Model T 0.07 / 0.91 / 0.20 / 0.56	→: Model T 0.52 / 0.88 / 0.95 / 0.66	→: Model T 0.09 / 0.88 / 0.39 / 0.75
				

Figure 19. Differences between DiG-IN UVCES when using different SOTA ImageNet models as guiding classifier. We use a ViT-B [81] pre-trained on IN21K, a ConvNeXt-L [48] pre-trained with CLIP loss on Laion-2B [70], a ConvNeXt-V2-H [91] pre-trained on IN21K and a EVA02-L [23] trained on MIM38M. All models are fine-tuned on IN1K. The confidences into the target classes are given as: confidence ViT-B / ConvNeXt-L / ConvNeXt-V2-H / EVA02-L.



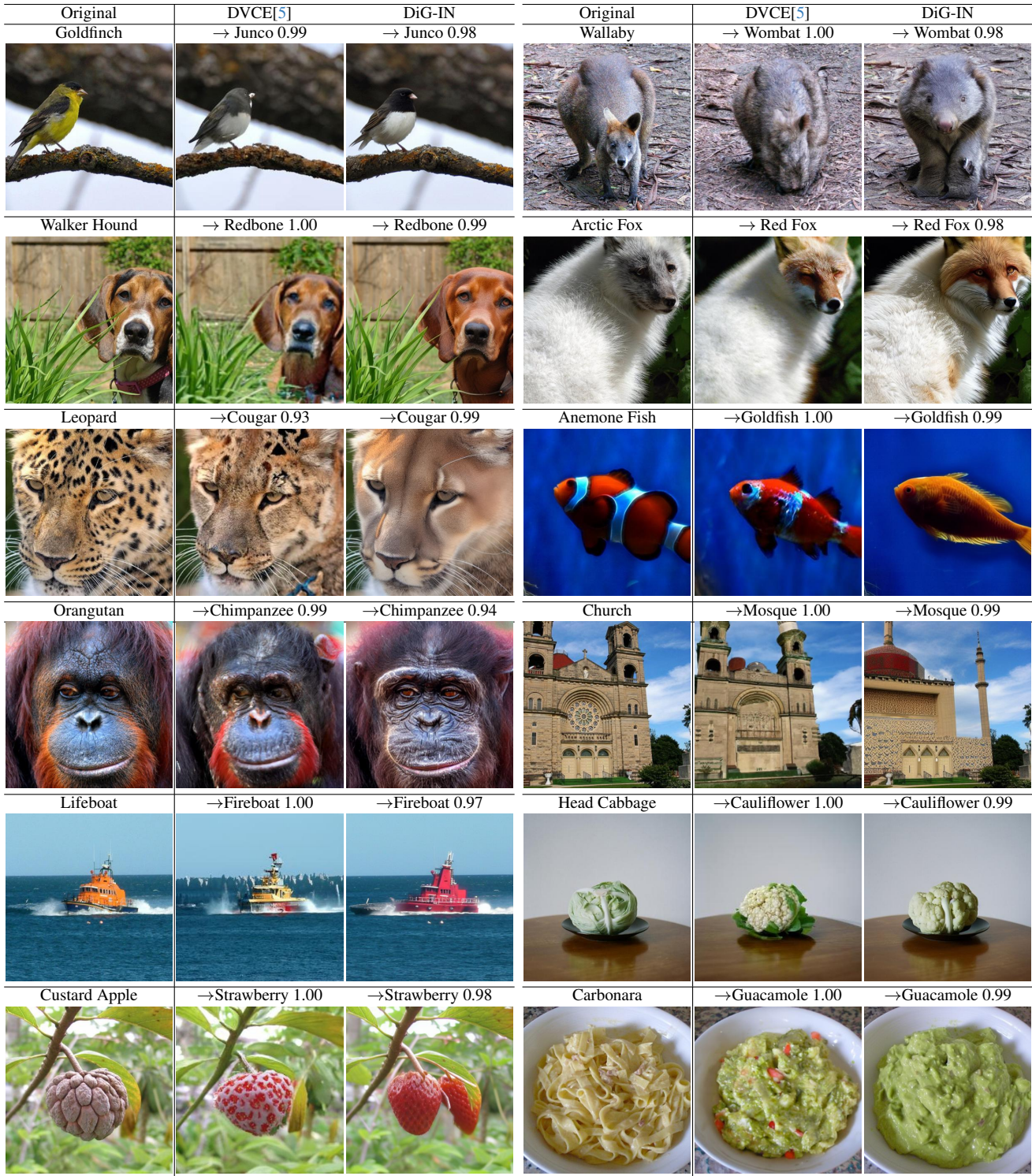


Figure 20. **ImageNet-1K** [66] DiG-IN UVCEs and DVCEs [5] for a ViT-B/16 AugReg [19, 81] pretrained on ImageNet-21K. Note that UVCEs are better at preserving the background (the branch for "Goldfinch → Junco", the ground for "Wallaby → Wombat") while also being able to do more complex geometry changes that can be required to transfer one class into another ("Church → Mosque", "Custard Apple → Strawberry") and generally yield a higher image quality and more meaningful features ("Orangutan → Chimpanzee", "Anemone Fish → Goldfish").



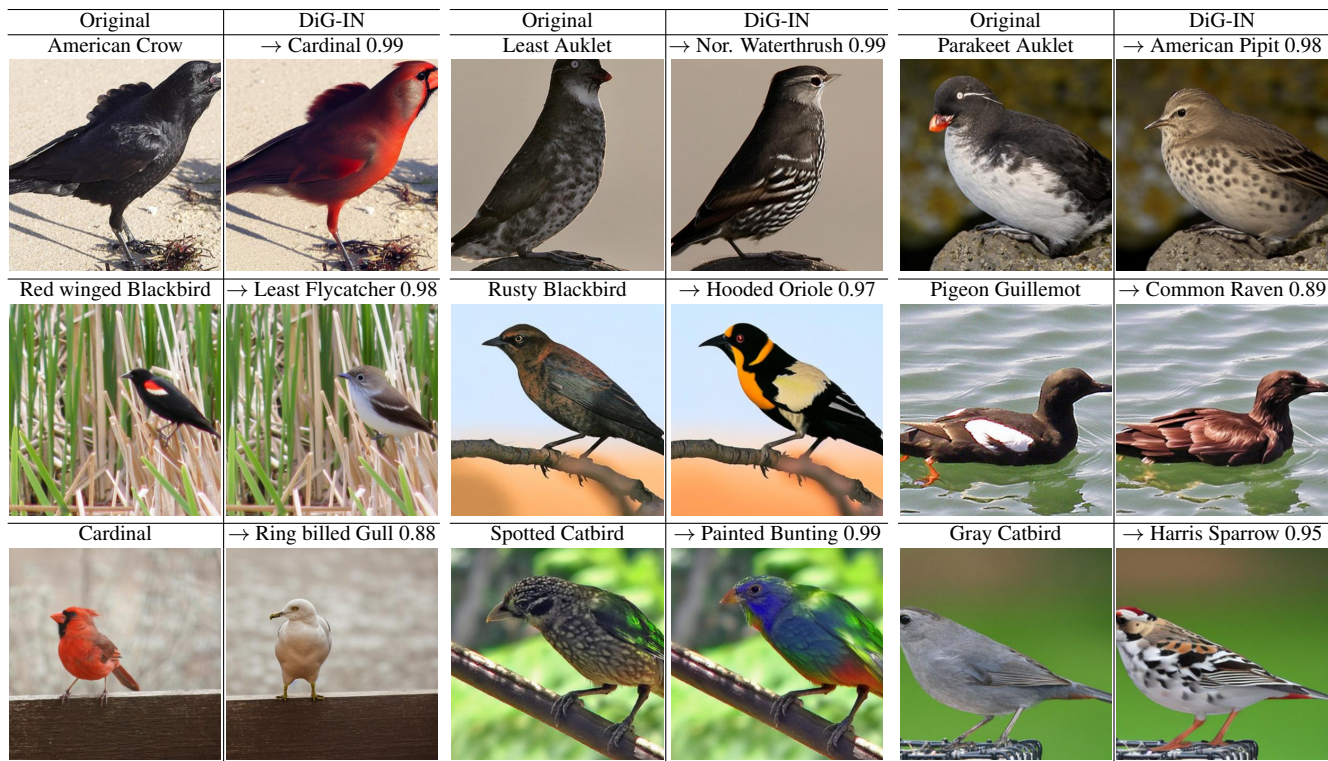


Figure 21. CUB-200-2011 [88] UVCEs for a for a ViT-B/16 AugReg [19, 81] pretrained on ImageNet-21K and fine-tuned on CUB.



Figure 22. Food-101 [11] UVCEs for a for a ViT-B/16 AugReg [19, 81] pretrained on ImageNet-21K and fine-tuned on Food-101.











Original	DiG-IN	Original	DiG-IN	Original	DiG-IN
GMC Terrain SUV 2012	→ Hyundai Sonata 2012 0.99	Audi S5 Convertible 2012	→ Dodge Challenger SRT8 2011 0.97	Ferrari California Convertible 2012	→ A.M. Virage Convertible 2012 0.99
					
A.M. V8 Vantage Coupe 2012	→ Bentley Arnage Sedan 2009 0.99	Acura TL Sedan 2012	→ Mercedes S-Class Sedan 2012 0.99	BMW 3 Series Sedan 2012	→ Bentley Continental GT Coupe 2007 0.99
					
Bentley Continental GT Coupe 2012	→ Ford Mustang Convertible 2007 0.99	Buick Verano Sedan 2012	→ Honda Accord Sedan 2012 0.99	Lamborghini Aventador Coupe 2012	→ Jaguar XK XKR 2012 0.99
					

Figure 23. Stanford Cars [43] DiG-IN UVCEs for a for a CAL-ResNet101 [62] trained on the Cars dataset.

Original	DiG-IN	Original	DiG-IN	Original	DiG-IN
"...eyes closed..."	→ "...eyes open..."	"...shaved..."	→ "...moustache..."	"...looking sad..."	→ "...smiling..."
					
"...red hair..."	→ "...blonde hair..."	"...smiling..."	→ "...looking sad..."	"...old man..."	→ "...young boy..."
					

Figure 24. FFHQ UVCEs for a zero-shot attribute classifier based on a CLIPA [46] text and image encoder pair.











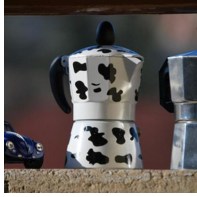
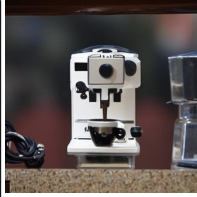
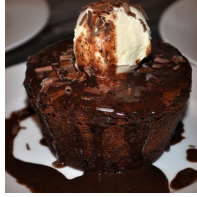
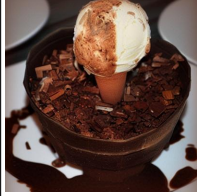






Misclassified Val. Image	DiG-IN UVCE	Misclassified Val. Image	DiG-IN UVCE	Misclassified Val. Image	DiG-IN UVCE
kit fox: 0.50 red fox: 0.16	kit fox: 0.01 red fox: 0.76:	stopwatch: 0.47 analog clock: 0.13	stopwatch: 0.02 analog clock: 0.75	recr. vehicle: 0.66 trailer truck: 0.08	recr. vehicle: 0.00 trailer truck: 0.74
					
tailed frog: 0.40 tree frog: 0.25	tailed frog: 0.06 tree frog: 0.70	coffeepot: 0.50 espresso maker: 0.16	coffeepot: 0.00 espresso maker: 0.79	choc. sauce: 0.54 ice cream: 0.07	choc. sauce: 0.01 ice cream: 0.75
					
wild boar: 0.48 hog: 0.24	wild boar: 0.09 hog: 0.60	paintbrush: 0.67 face powder: 0.02	paintbrush: 0.07 face powder: 0.30	beach wagon: 0.64 minivan: 0.03	beach wagon: 0.00 minivan: 0.84
					

Figure 25. **EVA02-L error UVCEs:** We generate DiG-IN UVCEs into the target class for images that are misclassified by an EVA02-L [23] (90.05% accuracy) according to the labels in the original IN1K validation set. Above each image, we give the confidence into the wrongly predicted class (top) and the correct/target class (bottom). We note that most "errors" according to ImageNet labels are results of ambiguous labels or straight-up labeling errors.

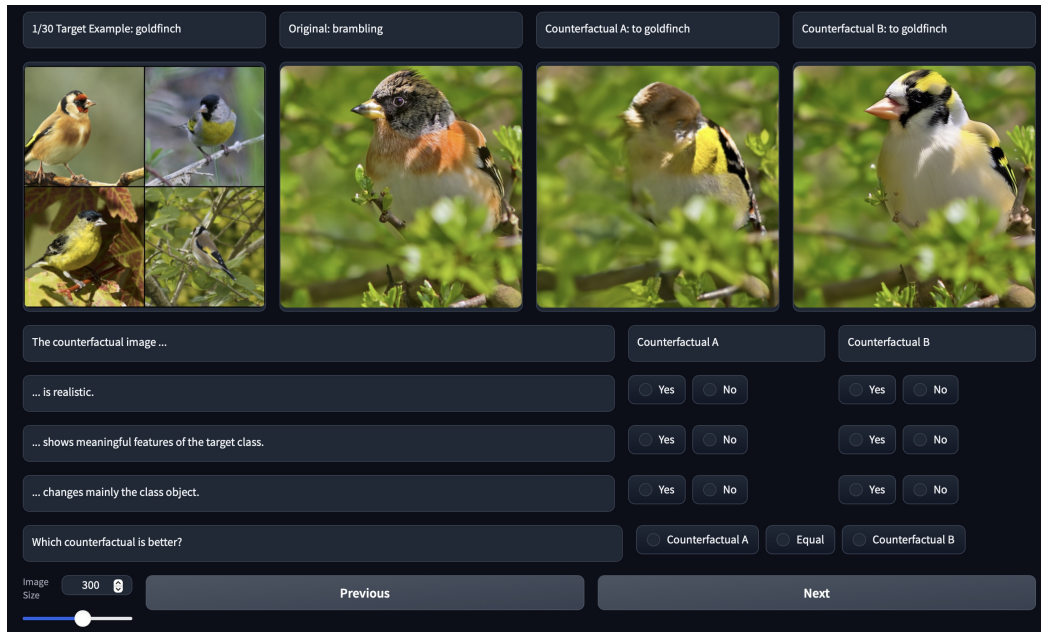


Figure 26. **User Study:** The participants were shown four training images of the target class, the original image and the two VCEs.



Target Example	Original	DVCE	UVCE (ours)	Target Example	Original	DVCE	UVCE (ours)
	n=19	0 / 79 / 63 / 0	95 / 95 / 95 / 89		n=19	58 / 95 / 42 / 26	63 / 84 / 95 / 26
	n=24	0 / 0 / 38 / 0	79 / 58 / 96 / 83		n=19	26 / 32 / 26 / 0	89 / 95 / 100 / 95
	n=21	10 / 76 / 86 / 19	90 / 14 / 95 / 38		n=21	62 / 95 / 100 / 81	81 / 71 / 100 / 10
	n=18	6 / 44 / 72 / 0	94 / 100 / 100 / 100		n=20	35 / 70 / 80 / 5	95 / 90 / 90 / 70
	n=23	30 / 30 / 87 / 4	100 / 87 / 87 / 87		n=21	86 / 76 / 90 / 24	67 / 86 / 90 / 48
	n=18	6 / 22 / 89 / 0	67 / 56 / 94 / 56		n=19	58 / 74 / 74 / 0	95 / 89 / 100 / 84
	n=18	67 / 100 / 94 / 39	61 / 94 / 89 / 33		n=16	50 / 69 / 56 / 0	75 / 81 / 100 / 75
	n=17	94 / 100 / 76 / 82	29 / 88 / 88 / 12		n=20	45 / 75 / 80 / 0	85 / 95 / 100 / 85

Figure 27. **User Study:** Examples 1-16 **Q1:** '... is realistic' **Q2:** '... is realistic' **Q3:** '... is realistic' **Better?:** 'Which counterfactuals is better?'



Target Example	Original	DVCE	UVCE (ours)	Target Example	Results: Q1 / Q2 / Q3 / Better? (in %)		
					Original	DVCE	UVCE (ours)
n=19	47 / 95 / 63 / 26	79 / 100 / 68 / 32		n=19	53 / 84 / 89 / 42	16 / 79 / 89 / 21	
n=19	26 / 42 / 63 / 0	89 / 89 / 100 / 89		n=22	91 / 95 / 91 / 73	32 / 55 / 77 / 0	
n=18	44 / 78 / 61 / 17	89 / 67 / 94 / 56		n=21	10 / 38 / 81 / 0	90 / 81 / 62 / 90	
n=22	9 / 9 / 77 / 5	91 / 91 / 73 / 91		n=20	55 / 100 / 70 / 20	50 / 95 / 65 / 25	
n=21	100 / 95 / 95 / 52	90 / 90 / 95 / 19		n=18	67 / 94 / 50 / 22	100 / 94 / 94 / 67	
n=17	71 / 82 / 94 / 6	88 / 82 / 94 / 59		n=17	12 / 24 / 76 / 0	47 / 88 / 88 / 82	
n=26	19 / 46 / 73 / 4	92 / 92 / 92 / 88		n=18	6 / 44 / 89 / 0	78 / 94 / 94 / 89	

Figure 28. **User Study:** Examples 17-30 **Q1:** '... is realistic' **Q2:** '... is realistic' **Q3:** '... is realistic' **Better?:** 'Which counterfactuals is better?'

We then maximize Eq. (6) to achieve prototypical examples that maximize this neuron. To visualize the necessity of our optimization and the benefits over manual inspection of the maximally activating train images and using text-guided Stable Diffusion without optimization we refer to Fig. 29. Note that our optimization can generate prototypical examples for a neuron that achieve higher activations than even the maximally activating train images that the model was

trained on. Additionally, we can visualize highly specialized neurons much more accurately than with text guidance only. For example, the highest activating object from CogAgent for neurons 494 and 798 of the SE-ResNet is "water". However, generating images using the default prompt does not result in large neuron activations. In contrast, DiG-IN can create highly active images without manual prompt tuning. We also highlight that given the target neuron  $n$ , our pipeline

is completely automatic and does not require humans in the loop.

We show additional examples of similar neurons, similar to the ones shown in the main paper in Fig. 30 and more individual neurons in Fig. 31. We highlight that we can generate visualizations for a diverse set of neurons that achieve higher activations than the most activating train images and that are easy to interpret. However, while we identify maximally activating visual concepts of neurons, we note that we are not aiming at achieving an exhaustive list of such concepts, but just visualize one per neuron.

## F.2. Neuron Counterfactuals

The creation of neuron counterfactuals is similar to that of our UVCEs. Instead of optimizing the confidence, we maximize or minimize the activation of the target neuron. On top of that, since we now want to keep the class object fixed while allowing for background changes, we no longer use the inverted masks  $(1 - S_{PX})$  but  $S_{PX}$  directly in the regularization term. The resulting objective is given by:

$$\begin{aligned} \max_{z_T, (C_t)_{t=1}^T, (\emptyset_t)_{t=1}^T} & \phi\left(\mathcal{D}(z_0(z_T, (C_t)_{t=1}^T, (\emptyset_t)_{t=1}^T))\right)_n \\ & - w_{VAE} \|S_{VAE} \odot (z - \mathcal{E}(\hat{x}))\|_2^2 \\ & - w_{PX} \|S_{PX} \odot (\mathcal{D}(z) - \hat{x})\|_2^2. \end{aligned} \quad (22)$$

Hyperparameters are identical to the ones used for UVCEs. We show additional examples for our neuron visualization for the spurious neurons found in [76] in Fig. 32 and Fig. 33.

## F.3. Quantitative Evaluation

We also quantitatively evaluate whether or not a given neuron is spurious, *i.e.* if it is activated by the class object or background features. To do this, we use HQ-SAM [42] with manual prompting and quality control to segment the foreground object in our neuron counterfactuals and remove images where the class object is no longer visible post-optimization or covers the entire image. We then calculate the HiResCAM [20] activation map where we use the neuron’s activation as loss for gradient calculations. This results in a heatmap with the size of the original input image with larger values in areas that activate the neuron. We then normalize the CAM map to sum to 1 and integrate it over our segmentation mask. A larger sum outside the segmentation mask implies that the neuron is spurious. The maximum value of 1 would correspond to the entire activation being on the background and the minimum value of 0 corresponds to the entire activation being on the class object.

Results for 4 spurious and 4 core neurons can be found in Tab. 2 and some examples in Fig. 34. Note that the core

neurons from [76] are more focused on the class object ( $\sum$  CAM outside mask closer to 0) whereas the corresponding sum is closer to 1 for spurious neurons. We note that this is the case even though our regularization term tries to preserve the foreground object, in which case the optimization tries to generate a background that activates the target neuron, which naturally increases the sum over the CAM map outside of the segmentation mask even for core neurons.

## G. NPCA counterfactuals and harmful spurious features

**Class-wise neural PCA (NPCA)** Instead of individual neurons, [54] introduce class-wise neural PCA components to identify spurious features: Let  $\phi(x) \in \mathbb{R}^D$  be the features of the penultimate layer of a neural network for an input  $x$  and  $w_k \in \mathbb{R}^D$  the last layer weights associated with a given class  $k$ . The NPCA components are computed by performing principle component analysis on  $\{\psi_k(x) := w_k \odot \phi(x)\}_{x \in \mathcal{D}_k}$  where  $\mathcal{D}_k$  are all images of class  $k$  in the ImageNet training set. The contribution of an NPCA component  $l$  to the logit of class  $k$  is

$$\alpha_l^{(k)}(x) = \langle \mathbf{1}, v_l \rangle \langle \psi_k(x) - \bar{\psi}_k, v_l \rangle,$$

where  $\bar{\psi}_k$  is the mean of  $\psi_k(x)$  over  $\mathcal{D}_k$  and  $v_l$  is the eigenvector corresponding to the principal component  $l$ . Code for NPCA and the spurious components are available at [https://github.com/YanNeu/spurious\\_imagenet](https://github.com/YanNeu/spurious_imagenet).

**NPCA counterfactuals** Analogous to the neuron counterfactuals, we maximize and minimize the logit contribution  $\alpha_l^{(k)}$  starting from training images of class  $k$  for several NPCA components that were labeled as spurious in [54] (see Fig. 35). In contrast to the neurons, these contributions can also attain negative values. The negative range can correspond to a different semantic feature.



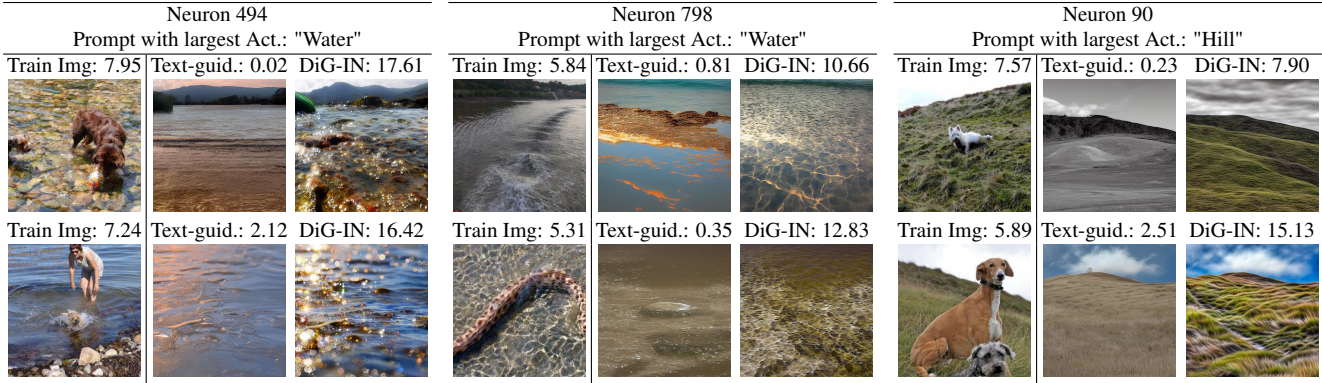


Figure 29. **Neuron visualization for a SE-ResNet-D 152 [90]:** We demonstrate the need for our Guided Diffusion optimization to properly explain a target neuron. While investigating highly activating images from the train set can give us an idea of which concepts are captured by a neuron, natural images often contain multiple objects which makes it unclear which object or concept in particular activates the target neuron. For the two leftmost neurons "932" and "494" (see Figure 9 for more examples), the prompt word "water" from CogAgent [37] achieves the highest average activation for images generated with text-guided Stable Diffusion without guidance. However, these neurons are highly specialized which makes it hard to generate strongly activating images with text guidance alone, resulting in images that achieve much lower activations than highly active images from the training set. Our DiG-IN guidance allows us to automatically create images that show prototypical neuron visualizations that highlight the subtle differences between those neurons and achieve much higher activations than even the most activating train images.

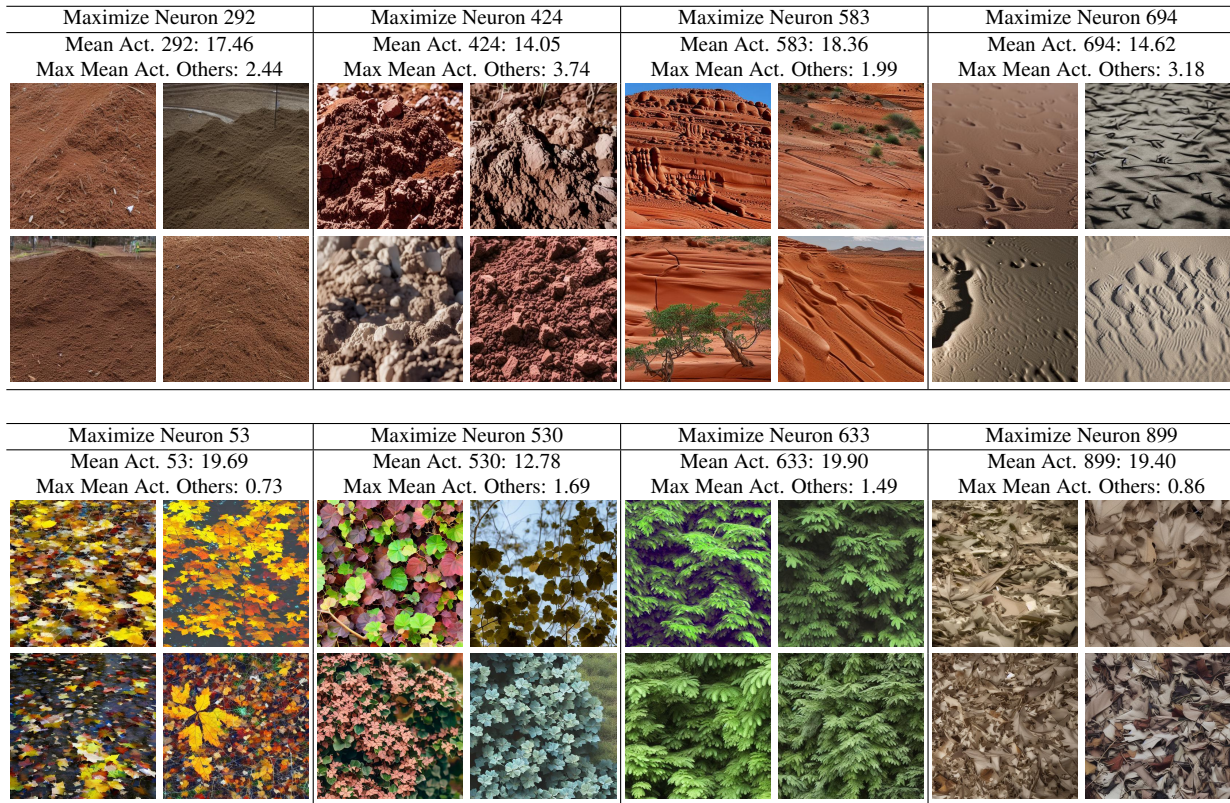


Figure 30. **Neuron visualization for a SE-ResNet-D 152 [90]:** More examples of closely related neurons that capture similar but slightly different concepts similar to Figure 9. The neurons in the top part of the image all seem to be activated by red sand, however, while neuron 292 seems to capture piles of reddish sand, neuron 424 is activated by larger chunks. The lower part of the Figure shows different "leaf" neurons, ranging from neurons activated by yellow leaves on the ground, dried and withered leaves, to green leaves in bushes.



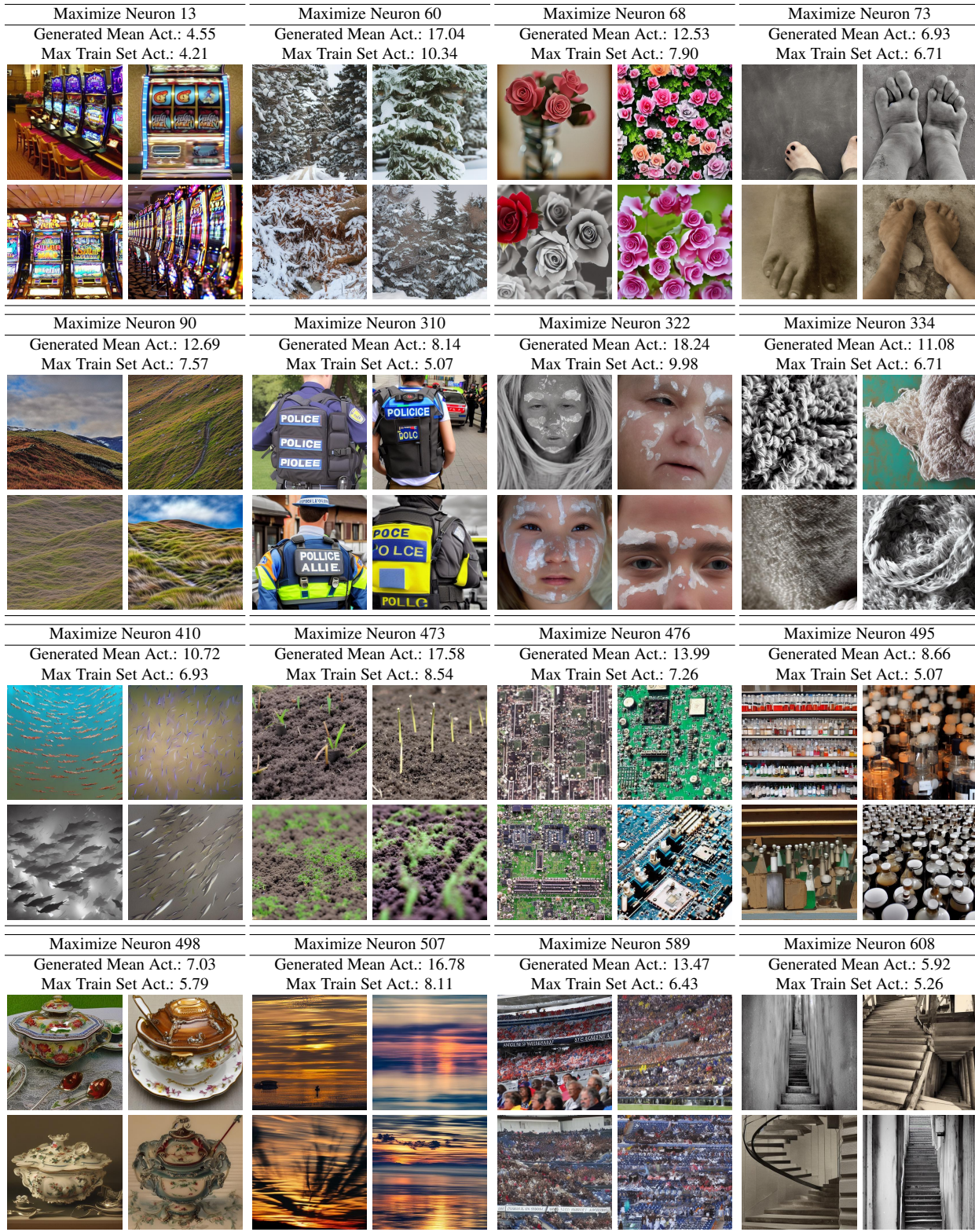


Figure 31. **Additional Neuron visualizations for a SE-ResNet-D 152 [90]:** We provide additional examples of coherent concepts captured by certain neurons.





















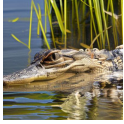


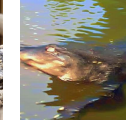





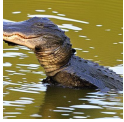
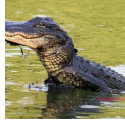
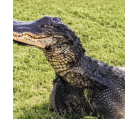
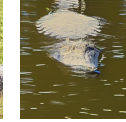
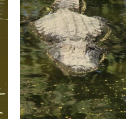




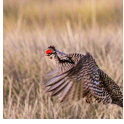
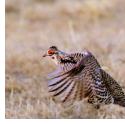
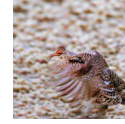
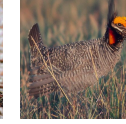

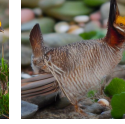
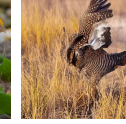
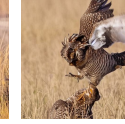
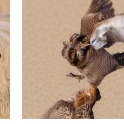



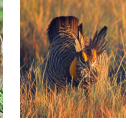
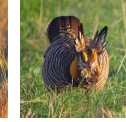
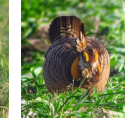

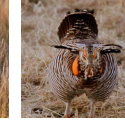
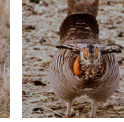








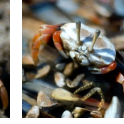
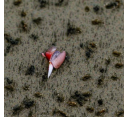




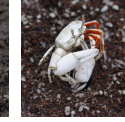


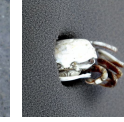
Neuron 1697 (Conf. class Great White Shark)								
Maximize	← ImageNet →		Maximize	← ImageNet →		Maximize	← ImageNet →	
Neuron 1697	Initialization	Neuron 1697	Neuron 1697	Initialization	Neuron 1697	Neuron 1697	Initialization	Neuron 1697
<b>5.86</b> (0.63)	<b>3.20</b> (0.36)	<b>1.47</b> (0.13)	<b>5.66</b> (0.79)	<b>1.16</b> (0.35)	<b>0.09</b> (0.07)	<b>5.75</b> (0.95)	<b>1.90</b> (0.92)	<b>0.19</b> (0.50)
								
<b>6.10</b> (0.80)	<b>4.07</b> (0.56)	<b>2.27</b> (0.31)	<b>5.41</b> (0.87)	<b>2.54</b> (0.71)	<b>0.91</b> (0.45)	<b>6.97</b> (0.36)	<b>4.69</b> (0.55)	<b>0.32</b> (0.01)
								
Neuron 341 (Conf. American Alligator)								
Maximize	← ImageNet →		Maximize	← ImageNet →		Maximize	← ImageNet →	
Neuron 341	Initialization	Neuron 341	Neuron 341	Initialization	Neuron 341	Neuron 341	Initialization	Neuron 341
<b>4.67</b> (0.57)	<b>0.81</b> (0.26)	<b>0.00</b> (0.01)	<b>6.15</b> (0.11)	<b>0.36</b> (0.04)	<b>0.02</b> (0.01)	<b>6.73</b> (0.28)	<b>0.67</b> (0.02)	<b>0.3</b> (0.01)
								
<b>7.80</b> (0.07)	<b>3.36</b> (0.60)	<b>0.19</b> (0.08)	<b>7.71</b> (0.01)	<b>1.75</b> (0.09)	<b>0.02</b> (0.01)	<b>4.38</b> (0.40)	<b>0.20</b> (0.06)	<b>0.02</b> (0.00)
								
Neuron 565 (Conf. Prairie Chicken)								
Maximize	← ImageNet →		Maximize	← ImageNet →		Maximize	← ImageNet →	
Neuron 565	Initialization	Neuron 565	Neuron 565	Initialization	Neuron 565	Neuron 565	Initialization	Neuron 565
<b>5.88</b> (0.97)	<b>3.23</b> (0.87)	<b>0.08</b> (0.01)	<b>6.15</b> (0.99)	<b>2.53</b> (0.99)	<b>0.31</b> (0.64)	<b>6.78</b> (0.80)	<b>3.28</b> (0.57)	<b>0.32</b> (0.00)
								
<b>7.02</b> (0.99)	<b>3.70</b> (0.99)	<b>0.37</b> (0.11)	<b>7.82</b> (0.98)	<b>2.60</b> (0.97)	<b>0.09</b> (0.02)	<b>6.68</b> (0.99)	<b>2.51</b> (0.99)	<b>0.05</b> (0.41)
								
Neuron 870 (Conf. Fiddler Crab)								
Maximize	← ImageNet →		Maximize	← ImageNet →		Maximize	← ImageNet →	
Neuron 870	Initialization	Neuron 870	Neuron 870	Initialization	Neuron 870	Neuron 870	Initialization	Neuron 870
<b>5.74</b> (0.99)	<b>2.24</b> (0.93)	<b>0.02</b> (0.04)	<b>4.12</b> (0.99)	<b>1.88</b> (0.99)	<b>0.14</b> (0.34)	<b>4.48</b> (0.99)	<b>2.31</b> (0.97)	<b>0.03</b> (0.22)
								
<b>5.44</b> (0.99)	<b>3.41</b> (0.90)	<b>0.2</b> (0.00)	<b>4.19</b> (0.98)	<b>1.43</b> (0.86)	<b>0.07</b> (0.30)	<b>3.10</b> (0.95)	<b>1.31</b> (0.86)	<b>0.17</b> (0.16)
								

Figure 32. Neuron counterfactuals for spurious neurons from [76].



**Neuron 1697** (Conf. class Albatross)

Maximize Neuron 1697	← ImageNet → Initialization	Minimize Neuron 1697	Maximize Neuron 1697	← ImageNet → Initialization	Minimize Neuron 1697	Maximize Neuron 1697	← ImageNet → Initialization	Minimize Neuron 1697
<b>4.97</b> (0.99)	<b>1.52</b> (0.97)	<b>0.39</b> (0.40)	<b>5.54</b> (0.99)	<b>3.36</b> (0.99)	<b>0.50</b> (0.41)	<b>5.00</b> (0.99)	<b>1.63</b> (0.97)	<b>0.63</b> (0.85)
<b>4.27</b> (0.99)	<b>2.09</b> (0.99)	<b>0.22</b> (0.09)	<b>5.60</b> (0.99)	<b>0.72</b> (0.66)	<b>0.21</b> (0.17)	<b>2.46</b> (0.98)	<b>0.21</b> (0.93)	<b>0.03</b> (0.61)

**Neuron 595** (Conf. Bee)

Maximize Neuron 595	← ImageNet → Initialization	Minimize Neuron 595	Maximize Neuron 595	← ImageNet → Initialization	Minimize Neuron 595	Maximize Neuron 595	← ImageNet → Initialization	Minimize Neuron 595
<b>14.61</b> (0.65)	<b>4.13</b> (0.91)	<b>0.15</b> (0.26)	<b>15.67</b> (0.22)	<b>9.19</b> (0.55)	<b>0.65</b> (0.02)	<b>9.96</b> (0.35)	<b>0.13</b> (0.46)	<b>0.03</b> (0.26)
<b>10.80</b> (0.78)	<b>4.89</b> (0.52)	<b>0.91</b> (0.76)	<b>11.90</b> (0.71)	<b>5.39</b> (0.90)	<b>0.99</b> (0.74)	<b>10.69</b> (0.84)	<b>3.12</b> (0.91)	<b>0.47</b> (0.45)

**Neuron 0** (Conf. Dogsled)

Maximize Neuron 0	← ImageNet → Initialization	Minimize Neuron 0	Maximize Neuron 0	← ImageNet → Initialization	Minimize Neuron 0	Maximize Neuron 0	← ImageNet → Initialization	Minimize Neuron 0
<b>4.87</b> (0.99)	<b>1.88</b> (0.98)	<b>0.89</b> (0.82)	<b>5.14</b> (0.98)	<b>2.90</b> (0.94)	<b>0.73</b> (0.46)	<b>4.41</b> (0.99)	<b>0.93</b> (0.99)	<b>0.17</b> (0.95)
<b>6.55</b> (0.99)	<b>1.41</b> (0.98)	<b>0.25</b> (0.60)	<b>4.23</b> (0.91)	<b>1.62</b> (0.79)	<b>0.51</b> (0.40)	<b>6.56</b> (0.99)	<b>3.17</b> (0.95)	<b>0.48</b> (0.18)

**Neuron 1772** (Conf. Gondola)

Maximize Neuron 1772	← ImageNet → Initialization	Minimize Neuron 1772	Maximize Neuron 1772	← ImageNet → Initialization	Minimize Neuron 1772	Maximize Neuron 1772	← ImageNet → Initialization	Minimize Neuron 1772
<b>4.59</b> (0.99)	<b>0.30</b> (0.96)	<b>0.15</b> (0.87)	<b>4.03</b> (0.99)	<b>0.72</b> (0.99)	<b>0.46</b> (0.99)	<b>2.53</b> (0.99)	<b>1.00</b> (0.99)	<b>0.38</b> (0.99)
<b>5.25</b> (0.99)	<b>1.82</b> (0.99)	<b>0.13</b> (0.48)	<b>3.97</b> (1.00)	<b>1.61</b> (0.99)	<b>0.44</b> (0.98)	<b>6.56</b> (0.99)	<b>4.16</b> (0.99)	<b>0.22</b> (0.90)

Figure 33. Neuron counterfactuals for spurious neurons from [76].

Class	Neuron	Spurious [76]	$\sum$ CAM outside mask	Mean Activation
prairie chicken	565	✓	0.82	6.45
fiddler crab	870	✓	0.86	3.14
great white shark	1697	✓	0.77	5.22
American alligator	341	✓	0.83	5.76
prairie chicken	1297	✗	0.44	5.29
fiddler crab	952	✗	0.50	7.07
koala	1571	✗	0.33	6.73
leonberg	1065	✗	0.31	5.86

Table 2. **Neuron countfactuals:** Quantitative evaluation for background neuron activations in our DiG-IN Neuron counterfactuals.

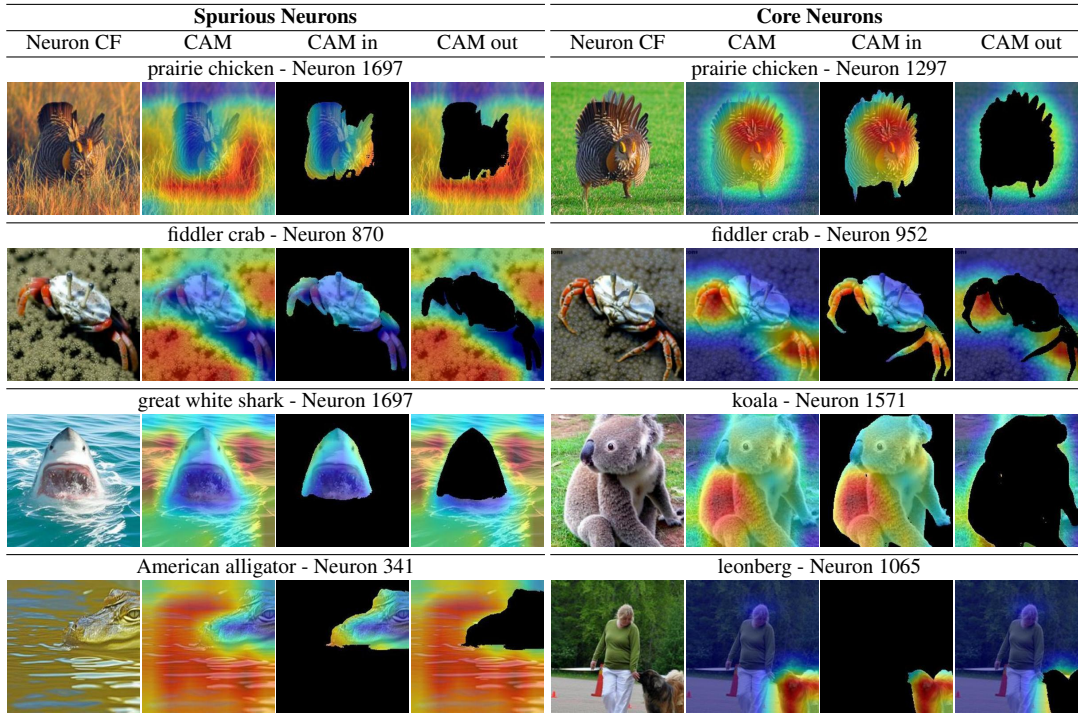


Figure 34. **Neuron countfactuals:** Visualization of the CAM maps and segmentation masks used for generating Tab. 2. Note that for spurious neurons, most of the activation is on the background while for core neurons, it is on the class object.











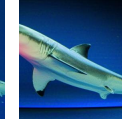

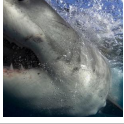
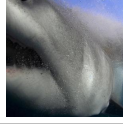
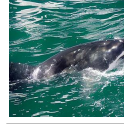





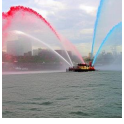
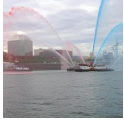
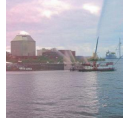
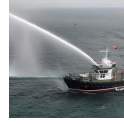





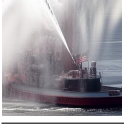
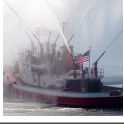
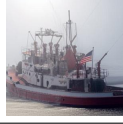







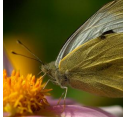


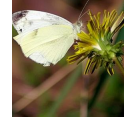
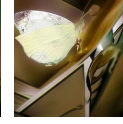
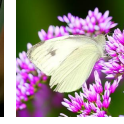

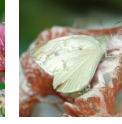


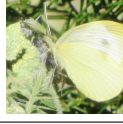






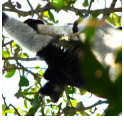
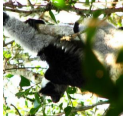
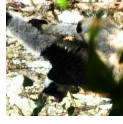
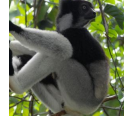
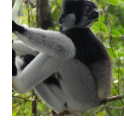
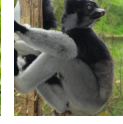
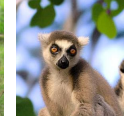
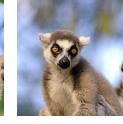
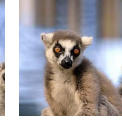
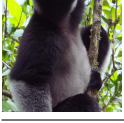
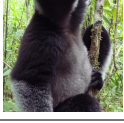
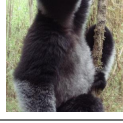
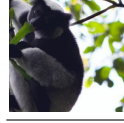
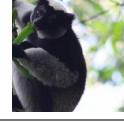
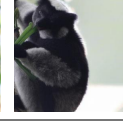
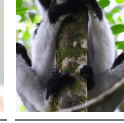


Class 2 (Great White Shark) - NPCA Comp. 1 (Conf. class Great White Shark)								
Maximize NPCA 1	← ImageNet → Initialization	Minimize NPCA 1	Maximize NPCA. 1	← ImageNet → Initialization	Minimize NPCA. 1	Maximize NPCA. 1	← ImageNet → Initialization	Minimize NPCA. 1
5.01 (0.58)	1.01 (0.37)	-2.78 (0.04)	-0.36 (0.54)	-3.96 (0.50)	-5.27 (0.33)	3.70 (0.39)	-0.70 (0.19)	-3.75 (0.05)
								
5.21 (0.93)	-1.59 (0.57)	-4.34 (0.15)	10.11 (0.78)	5.02 (0.55)	-3.40 (0.07)	2.44 (0.47)	-4.38 (0.01)	-5.61 (0.00)
								
Class 554 (Fireboat) - NPCA Comp. 2 (Conf. Fireboat)								
Maximize NPCA 2	← ImageNet → Initialization	Minimize NPCA 2	Maximize NPCA 2	← ImageNet → Initialization	Minimize NPCA 2	Maximize NPCA 2	← ImageNet → Initialization	Minimize NPCA 2
4.18 (1.00)	0.30 (0.75)	-1.00 (0.09)	3.59 (1.00)	0.04 (0.95)	-1.30 (0.27)	2.51 (1.00)	0.31 (1.00)	-0.98 (0.94)
								
4.47 (1.00)	0.66 (0.89)	-1.14 (0.30)	1.02 (1.00)	-1.25 (0.78)	-1.37 (0.69)	4.82 (0.99)	-0.40 (0.32)	-1.18 (0.05)
								
Class 324 (Cabbage Butterfly) - NPCA Comp. 3 (Conf. Cabbage Butterfly)								
Maximize NPCA 3	← ImageNet → Initialization	Minimize NPCA 3	Maximize NPCA 3	← ImageNet → Initialization	Minimize NPCA 3	Maximize NPCA 3	← ImageNet → Initialization	Minimize NPCA 3
8.32 (0.80)	2.10 (0.55)	-2.46 (0.03)	4.61 (0.90)	-0.04 (0.90)	-3.11 (0.03)	13.99 (0.96)	3.61 (0.83)	-2.89 (0.00)
								
8.16 (0.78)	0.81 (0.68)	-2.94 (0.34)	4.30 (0.93)	-0.90 (0.78)	-2.60 (0.07)	9.05 (0.96)	0.84 (0.92)	-2.97 (0.01)
								
Class 384 (Indri) - NPCA Comp. 6 (Conf. Indri)								
Maximize NPCA 6	← ImageNet → Initialization	Minimize NPCA 6	Maximize NPCA 6	← ImageNet → Initialization	Minimize NPCA 6	Maximize NPCA 6	← ImageNet → Initialization	Minimize NPCA 6
2.21 (0.47)	0.95 (0.35)	-0.82 (0.04)	1.49 (0.99)	0.21 (0.99)	-0.91 (0.92)	0.54 (0.52)	-0.88 (0.11)	-1.40 (0.04)
								
1.27 (0.98)	-0.38 (0.71)	-1.21 (0.13)	1.66 (0.87)	0.39 (0.71)	-0.82 (0.09)	0.86 (0.95)	-0.15 (0.83)	-0.94 (0.23)
								

Figure 35. NPCA counterfactuals for harmful spurious features identified in [54]. Outgoing from a generated image of the class the corresponding NPCA component of this class is maximized respectively minimized. While the class object is not changing much, changing the spurious feature alone can increase/decrease the confidence in the class significantly. According to [54] the spurious features for each class are: water/foam for great white shark, water jet for fireboat, flowers for cabbage butterfly, branches/leaves for indri. The NPCA components are available at [https://github.com/YanNeu/spurious\\_imagenet](https://github.com/YanNeu/spurious_imagenet).

## H. Limitations

One potential downside of our method is the increase in computational cost compared to text-guided Stable Diffusion. A possible way to overcome this is to use a distilled model [47, 69] which can generate images in 1 to 8 steps which would reduce overall computational costs.

We inherit systematic issues from the diffusion model, for example, in Fig. 31, the images for neuron 73 show the typical issues that Stable Diffusion has with producing feet and hands. While we did not encounter this during our evaluations, if a concept is completely unknown to Stable Diffusion, it is possible that we fail to uncover potential vulnerabilities of the generating classifier due to SD not being able to generate the corresponding subgroup.

Especially for Img2Img tasks, there are several failure modes that can occur which will reduce the quality of the resulting image. If the HQ-SAM segmentation mask is off, we will regularize similarity in the wrong parts of the image which can result in the generation being too restricted or allow for too many changes, however, overall we found the segmentation model to be sufficiently robust. Even if the mask is correct, it can sometimes be difficult to maximize the main objective (confidence in the target class or neuron activation) while simultaneously preserving the image structure in 20 optimization steps. Lastly, for our DiG-IN UVCEs, we found that Prompt-to-Prompt sometimes leads to very large changes and our optimization is not always able to lead the generation back to the original image. If the diffusion model does not know the name of the target class, it can sometimes generate the wrong object. Due to the non-convexity of the optimization objective, generating the right object from a conditioning vector that SD does not associate with this object can fail with few optimization steps. We demonstrate some UVCE failure cases in Fig. 36















Original	P2P	DiG-IN	Original	P2P	DiG-IN
Welsh springer spaniel	→ Clumber 0.00	→ Clumber 0.03	Audi S6 2011	→ Volvo 240 1993 0.16	→ Volvo 240 1993 0.99
					
ice bear	→ brown bear 0.76	→ brown bear 0.92	ice bear	→ American black bear 0.68	→ American black bear 0.85
					

Figure 36. **UVCE failure cases:** For the image on the top left, the word "Clumber" is not associated with a dog breed in Stable Diffusion which leads P2P to generate a human instead. While our optimization is able to recover the background, it cannot generate the proper class object, likely due to the distance between the CLIP encoding of the word "Clumber" to that of a matching dog breed.

The top right image shows how P2P can sometimes cause large changes in the image structure. Since our mask covers the blue car in the original image, our distance regularization only enforces similarity in the background and our optimization does not return to the structure of the original image.

In the bottom row, we show 2 UVCEs from the same starting image of an "ice bear" into "brown bear" and "American black bear". Notice how the P2P initialization leads to large changes in image structure in both cases. For the first image, our optimization can recover the original image structure and produce a valid UVCE whereas it produces an image that is too different from the original one in the second case.