

Transforming and Projecting Images into Class-conditional Generative Networks

Minyoung Huh^{1,2*} Richard Zhang² Jun-Yan Zhu²
 Sylvain Paris² Aaron Hertzmann²

¹MIT CSAIL ²Adobe Research

Abstract. We present a method for projecting an input image into the space of a class-conditional generative neural network. We propose a method that optimizes for transformation to counteract the model biases in generative neural networks. Specifically, we demonstrate that one can solve for image translation, scale, and global color transformation, during the projection optimization to address the object-center bias and color bias of a Generative Adversarial Network. This projection process poses a difficult optimization problem, and purely gradient-based optimizations fail to find good solutions. We describe a hybrid optimization strategy that finds good projections by estimating transformations and class parameters. We show the effectiveness of our method on real images and further demonstrate how the corresponding projections lead to better editability of these images. The project page and the code is available at <https://minyoungg.github.io/pix2latent>.

1 Introduction

Deep generative models, particularly Generative Adversarial Networks (GANs) [26], can create a diverse set of realistic images, with a number of controls for transforming the output, e.g., [55, 9, 34, 37, 55, 7, 30]. However, most of these methods apply only to synthetic images that are generated by GANs in the first place. In many real-world cases, a user would like to edit their own image. One approach is to train a network for each separate image transformation. However, this would require a combinatorial explosion of training time and model parameters.

Instead, a user could “project” their image to the manifold of images produced by the GAN, by searching for an appropriate latent code [70]. Then, any transformations available within the GAN could be applied to the user’s image. This could allow a powerful range of editing operations within a relatively compact representation. However, projection is a challenging problem. Previous methods have focused on class-specific models, for example, for objects [70], faces [53, 12], or specific scenes such as bedrooms and churches [8, 10]. With the challenges in both optimization and generative model’s limited capacity, we wish to find a generic method that can fit *real* images from diverse categories into the same generative model.

*Work started during an internship at Adobe Research.

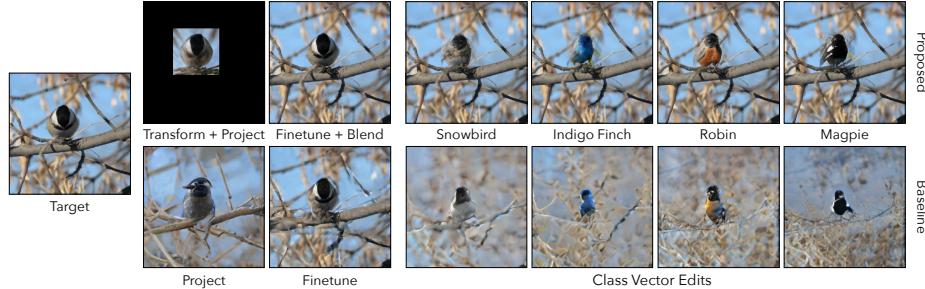


Fig. 1. Given a pre-trained BigGAN [11] and a target image (left), our method uses gradient-free BasinCMA to transform the image and find a latent vector to closely reconstruct the image. Our method (top) can better fit the input image, compared to the baseline (bottom), which does not model image transformation and uses gradient-based ADAM optimization. Finding an accurate solution to the inversion problem allows us to further fine-tune the model weights to match the target image without losing downstream editing capabilities. For example, our method allows for changing the class of the object (top row), compared to the baseline (bottom).

This paper proposes the first method for projecting images into class-conditional models. In particular, we focus on BigGAN [11]. We address the main problems with these tasks, mainly, the challenges of optimization, object alignment, and class label estimation:

- To help avoid local minima during the optimization process, we systematically study choices of both gradient-based and gradient-free optimizers and show Covariance Matrix Adaptation (CMA) [28] to be more effective than stand-alone gradient-based optimizers, such as L-BFGS [47] and Adam [39].
- To better fit a real image into the latent space, we account for the model’s center bias by simultaneously estimating both spatial image transformation (translation, scale, and color) and latent variable. Such a transformation can then be inverted back to the input image frame. Our simultaneous transformation and projection method largely expands the scope and diversity of the images that a GAN can reconstruct.
- Finally, we show that estimating and jointly optimizing the continuous embedding of the class variable leads to better projections. This ultimately leads to more expressive editing by harnessing the representation of the class-conditional generative model.

We evaluate our method against various baselines on projecting real images from ImageNet. We quantitatively and qualitatively demonstrate that it is crucial to simultaneously estimate the correct transformation during the projection step. Furthermore, we show that CMA, a non-parametric gradient-free optimization technique, significantly improves the robustness of the optimization and leads to better solutions. As shown in Figure 1, our method allows us to fine-tune our model to recover the missing details without losing the editing capabilities of the generative model.

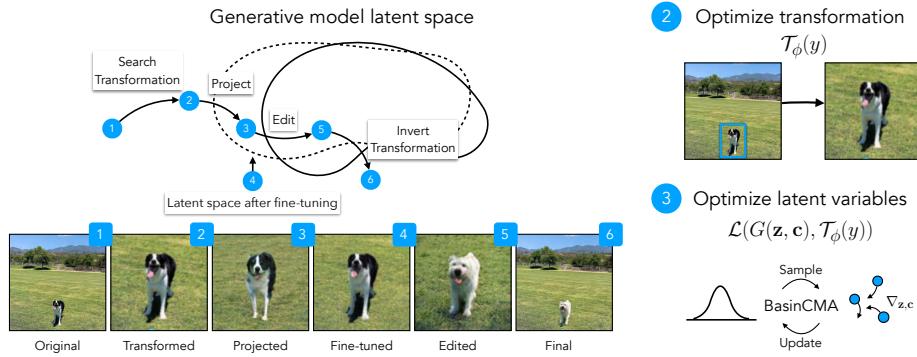


Fig. 2. Overview: Our method first searches for a transformation to apply to the input target image. We then solve for the latent vector that closely resembles the object in the target image, using our proposed optimization method, also referred to as “projection”. The generative model can then be further fine-tuned to reconstruct the missing details that the original model could not generate. Finally, we can edit the image by altering the latent code or the class vector (e.g., changing the border collie to a west highland white terrier), and invert and blend the edited image back into the original image.

2 Related Work

Image editing with generative models. Image editing tools allow a user to manipulate a photograph according to their goal while producing realistic visual content. Seminal work is often built on low-level visual properties, such as patch-based texture synthesis [21, 32, 20, 6], gradient-domain image blending [54], and image matting with locally affine color model [44]. Different from previous hand-crafted low-level methods, several recent works [70, 12] proposed to build editing tools based on a deep generative model, with the hope that a generative model can capture high-level information about the image manifold.

For example, iGAN [70] proposes to reconstruct and edit a real image using GANs. The method first projects a real photo onto a latent vector using a hybrid method of encoder-based initialization and per-image optimization. It then modifies the latent vector using various editing tools such as color, sketch, and warping brushes and generates the final image accordingly. Later, Neural Photo Editing [12] proposes to edit a face photo using VAE-GANs [43]. The same image prior from deep generative models has also been used in face editing, image inpainting, colorization, and deblurring [53, 66, 4, 27, 58]. Recently, GANPaint [8] proposes to change the semantics of an input image by first projecting an image into GANs, then fine-tuning the GANs to reproducing the details, and finally modifying the intermediate activations based on user inputs [9]. However, all the above systems focus on a single object category. Our work presents new ways of embedding an image into a class-conditional generative model, which allows the same GAN to be applied to many more in-the-wild scenarios.

Inverting networks. Our work is closely related to methods for inverting pre-trained networks. Earlier work proposes to invert CNN classifiers and intermediate features for visualizing recognition networks [48, 18, 49, 50]. More

recently, researchers adopted the above methods to invert generative models. The common techniques include: (1) Optimization-based methods: they find the latent vector that can closely reconstruct the input image using gradient-based method (e.g., ADAM, LBFGS) [70, 12, 46, 66, 57, 3, 13] or MCMC [61], (2) Encoder-based methods: they learn an encoder to directly predict the latent vector given a real image [15, 19, 70, 53, 12, 16], (3) Hybrid methods [70, 8, 10]: they use the encoder to initialize the latent vector and then solve the optimization problem. By using the ground truth class label, Neural Collage [60] turn a BigGAN into a single-class GAN and adopts single-class image projection [70]. Although the optimized latent vector roughly approximates the real input image, many important visual details are missing in the reconstruction [8]. To address the issue, GANPaint [8] gently fine-tunes the pre-trained GAN to adapt itself to the individual image. Image2StyleGAN [1] optimizes StyleGAN’s intermediate representation rather than the input latent vector. Unfortunately, the above techniques still cannot handle images in many scenarios due to the limited model capacity [10], the lack of generalization ability [1], and their single-class assumption. As noted by prior work [1], the reconstruction quality severely degrades under simple image transformation, and translation has been found to cause most of the damage. Compared to prior work, we consider two new aspects in the reconstruction pipeline: image transformation and class vector. Together, these two aspects significantly expand the diversity of the images that we can reconstruct and edit.

3 Image projection methods

We aim to project an image into a class-conditional generative model (e.g., BigGAN [11]) for the purposes of downstream editing. We first introduce the basic objective function that we slowly build upon. Next, since BigGAN is an object-centric model for most classes, we infer an object mask from the input image and focus on fitting the pixels inside the mask.

Furthermore, to better fit our desired image into the generative model, we propose to optimize for various image transformation (scale, translation, and color) to be applied to the target image. Lastly, we explain how we optimize the aforementioned objective loss function.

3.1 Basic Loss Function

Class-conditional generative model A class-conditional generative network can synthesize an image $\hat{\mathbf{y}} \in \mathbb{R}^{H \times W \times 3}$, given a latent code $\mathbf{z} \in \mathbb{R}^Z$ that models intra-class variations and a one-hot class-conditioning vector $\tilde{\mathbf{c}} \in \Delta^C$ to choose over C classes. We focus on the 256×256 BigGAN model [11] specifically, where $Z = 128$ and $C = 1,000$ ImageNet classes.

The BigGAN architecture first maps the one-hot $\tilde{\mathbf{c}}$ into a continuous vector $\mathbf{c} \in \mathbb{R}^{128}$ with a linear layer $\mathbf{W} \in \mathbb{R}^{128 \times 1000}$, before injecting into the main network G_θ , with learned parameters θ .

$$\hat{\mathbf{y}} = G_\theta(\mathbf{z}, \mathbf{c}) = G_\theta(\mathbf{z}, \mathbf{W}\tilde{\mathbf{c}}). \quad (1)$$

Here, a choice must be made whether to optimize over the discrete $\tilde{\mathbf{c}}$ or continuous \mathbf{c} . As optimizing a discrete class vector is non-trivial, we optimize over the continuous embedding.

Optimization setup. Given a target image \mathbf{y} , we would like to find a \mathbf{z}^* and \mathbf{c}^* that generates the image.

$$\mathbf{z}^*, \mathbf{c}^* = \arg \min_{\mathbf{z}, \mathbf{c}} \mathcal{L}(G_\theta(\mathbf{z}, \mathbf{c}), \mathbf{y}) \quad \text{s.t. } C(\mathbf{z}) \leq C_{\max}. \quad (2)$$

During training, the latent code is sampled from a multivariate Gaussian $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Interestingly, recent methods [11, 40] find that restricting the distribution at *test time* produces higher-quality samples. We follow this and constrain our search space to match the sampling distribution from Brock et al. [11]. Specifically, we use $C(\mathbf{z}) = \|\mathbf{z}\|_\infty$ and $C_{\max} = 2$. During optimization, elements of \mathbf{z} that fall outside the threshold are clamped to $+2$, if positive, or -2 , if negative. Allowing larger values of \mathbf{z} produces better fits but compromises editing ability.

Loss function. The loss function \mathcal{L} attempts to capture how close the approximate solution is to the target. A loss function that perfectly corresponds to human perceptual similarity is a longstanding open research problem [64], and evaluating the difference solely on a per-pixel basis leads to blurry results [68]. Distances in the feature space of a pre-trained CNN correspond more closely with human perception [35, 17, 23, 69]. We use the LPIPS metric [69], which calibrates a pre-trained model using human perceptual judgments. Here, we define our basic loss function, which combines per-pixel ℓ_1 and LPIPS.

$$\mathcal{L}_{\text{basic}}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{HW} \|\hat{\mathbf{y}} - \mathbf{y}\|_1 + \beta \mathcal{L}_{\text{LPIPS}}(\hat{\mathbf{y}}, \mathbf{y}). \quad (3)$$

In preliminary experiments, we tried various loss combinations and found $\beta = 10$ to work well. We now expand upon this loss function by leveraging object mask information.

3.2 Object Localization

Real images are often more complex than the ones generated by BigGAN. For example, objects may be off-centered and partially occluded, or multiple objects appear in an image. Moreover, it is possible that the object in the image can be approximated by GANs but not the background.

Accordingly, we focus on fitting a single foreground object in an image and develop a loss function to emphasize foreground pixels. We automatically produce a foreground rectangular mask $\mathbf{m} \in [0, 1]^{H \times W \times 1}$ using the bounding box of an object detector [31]. Here, we opt for bounding boxes for simplicity, but one could consider using segmentation mask, saliency maps, user-provided masks,

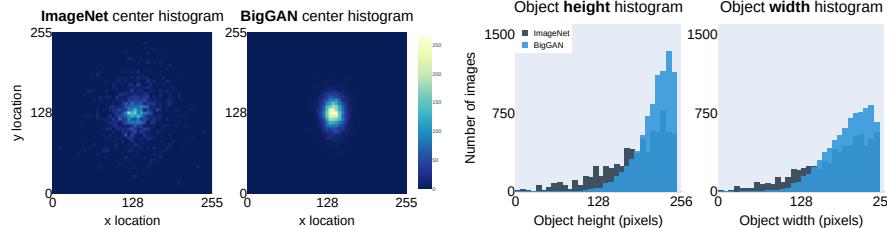


Fig. 3. Object center comparison: We use an object detector to compute the histogram of object locations. Note that ImageNet (left) is biased towards the center (right). But BigGAN (right) exhibits a long-tail, whereas the BigGAN (blue) accentuates the mode.

etc. The foreground and background values within mask \mathbf{m} are set to 1 and 0.3, respectively. We adjust the objective function to spatially weigh the loss:

$$\mathcal{L}_{\text{mask}}(\mathbf{y}, \hat{\mathbf{y}}, \mathbf{m}) = \frac{1}{M} \|\mathbf{m} \odot (\hat{\mathbf{y}} - \mathbf{y})\|_1 + \beta \mathcal{L}_{\text{mLPIPS}}(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{m}), \quad (4)$$

where normalization parameter $M = \|\mathbf{m}\|_1$ and \odot represents element-wise multiplication across the spatial dimensions. Given a mask of all foreground (all ones), the objective function is equivalent to Equation 3. We calculate the masked version of the perceptual loss $\mathcal{L}_{\text{mLPIPS}}(\hat{\mathbf{y}}, \mathbf{y}, \mathbf{m})$ by bilinearly downsampling the mask at the resolution of the intermediate spatial feature maps within the perceptual loss. The details are described in Appendix B. With the provided mask, we now explore how one can optimize for image transformation to better fit the object in the image.

3.3 Transformation Model and Loss

Generative models may exhibit biases for two reasons: (a) inherited biases from the training distribution and (b) bias introduced by mode collapse [25], where the generative model only captures a portion of the distribution. We mitigate two types of biases, *spatial* and *color* during image reconstruction process. A concurrent work [2] proposes that surrogate network with spatial transformers could be used to partially alleviate this aforementioned distributional shift at test-time.

Studying spatial biases. To study spatial bias, we first use a pre-trained object detector, MaskRCNN [31], over 10,000 real and generated images to compute the statistics of object locations. We show the statistics regarding the center locations and object sizes in Figures 3 and 4, respectively.

Figure 3 (left) demonstrates that ImageNet images exhibit clear center bias over the location of objects, albeit with a long tail. While the BigGAN learns to mimic this distribution, it further accentuates the bias [10, 34], largely forgoing the long tail to generate high-quality samples in the middle of the image. In Figure 4, we see similar trends with object height and width. Abdal et al. [1] noted

that the quality of image reconstruction degrades given a simple translation in the target image. Motivated by this, we propose to incorporate spatial alignment in the inversion process.

Searching over spatial alignments. We propose to transform the generated image using $\mathcal{T}_\psi^{\text{spatial}}(\cdot)$, which shifts and scales the image using parameters $\psi = [s_x, s_y, t_x, t_y]$. The parameters ψ are used to generate a sampling grid which in turn is used by a grid-sampler to construct a new transformed image [33]. The corresponding inverse parameters are $\psi^{-1} = [\frac{1}{s_x}, \frac{1}{s_y}, -\frac{t_x}{s_x}, -\frac{t_y}{s_y}]$.

Transforming the generated image allows for more flexibility in the optimization. For example, if G can perfectly generate the target image, but at different scales or at off-centered locations, this framework allows it to do so.

Searching over color transformations. Furthermore, we show that the same framework allows us to search over color transformations $\mathcal{T}_\gamma^{\text{color}}(\cdot)$. We experimented with various color transformations such as hue, brightness, gamma, saturation, contrast, and found brightness and contrast to work the best. Specifically, we optimize for brightness, which is parameterized by scalar γ with inverse value $\gamma^{-1} = -\gamma$. If the generator can perfectly generate the target image, but slightly darker or brighter, this allows a learned brightness transformation to compensate for the difference.

Final objective. Let transformation function $\mathcal{T}_\phi = \mathcal{T}_\psi^{\text{spatial}} \circ \mathcal{T}_\gamma^{\text{color}}$ be a composition of spatial and color transformation functions, where transformation parameters ϕ is a concatenation of spatial and color parameters ψ, γ , respectively. The inverse function is $\mathcal{T}_{\phi^{-1}}$. Our final optimization objective function, with consideration for (a) the foreground object and (b) spatial and color biases, is to minimize the following loss:

$$\arg \min_{\mathbf{z}, \mathbf{c}, \phi} \mathcal{L}_{\text{mask}}(\mathcal{T}_{\phi^{-1}}(G_\theta(\mathbf{z}, \mathbf{c})), \mathbf{y}, \mathbf{m}) \quad \text{s.t. } C(\mathbf{z}) \leq C_{\max} \quad (5)$$

Our optimization algorithm, described next, has a mix of gradient-free and gradient-based updates. Alternatively, instead of inverse transforming the *generated* image, we can transform the *target* and mask images during gradient-based updates and compute the following loss: $\mathcal{L}_{\text{mask}}(G_\theta(\mathbf{z}, \mathbf{c}), \mathcal{T}_\phi(\mathbf{y}), \mathcal{T}_\phi(\mathbf{m}))$. We will discuss when to use each variant in the next section.

3.4 Optimization Algorithms

Unfortunately, the objective function is highly non-convex. Gradient-based optimization, as used in previous inversion methods, frequently fall into poor local minima. Bau et al. [10] note that recent large-scale GAN models [36, 37] are significantly harder to invert due to a large number of layers, compared to earlier models [55]. Thus, formulating an optimizer that reliably finds good solutions is a significant challenge. We evaluate our method against various baselines and

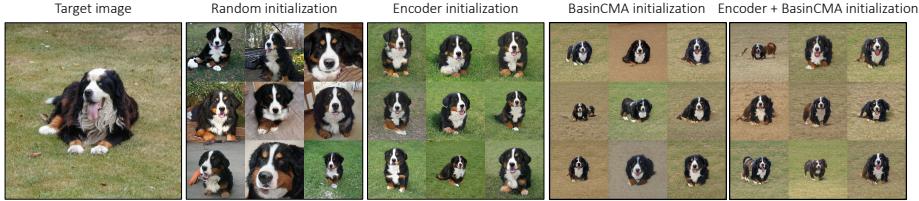


Fig. 5. Initialization from various methods: We show samples drawn from different methods, before the final gradient descent optimization. In “random initialization”, seeds are drawn from the normal distribution; the results show higher variation. For the “encoder initialization”, we use a trained encoder network to predict the latent vector and apply a minor perturbation. Our method uses CMA to find a good starting distribution. For “Encoder+BasinCMA”, we initialize CMA with the output of the encoder. The results are more consistent and better reconstruct the target image.

ablations in Section 4. Given the input image \mathbf{y} and foreground rectangular mask \mathbf{m} (which is automatically computed), we present the following algorithm.

Class and transform initialization We first predict the class of the image with a pre-trained ResNeXt101 classifier [65] and multiply it by \mathbf{W} to obtain our initial class vector \mathbf{c}_0 .

Next, we initialize the spatial transformation vector $\psi_0 = [s_{x_0}, s_{y_0}, t_{y_0}, t_{x_0}]$ such that the foreground object is well-aligned with the statistics of the BigGAN model. As visualized in Figures 3 and 4, $(\bar{h}, \bar{w}) = (137, 127)$ is the center of BigGAN-generated objects and $(\bar{y}, \bar{x}) = (213, 210)$ is the mode of object sizes. We define $(h_{\mathbf{m}}, w_{\mathbf{m}})$ to be the height and width and $(y_{\mathbf{m}}, x_{\mathbf{m}})$ to be center of the masked region. We initialize scale factors as $s_{y_0} = s_{x_0} = \max(\frac{h_{\mathbf{m}}}{h}, \frac{w_{\mathbf{m}}}{w})$ and translations as $(t_{y_0}, t_{x_0}) = (\frac{\bar{y}-y_{\mathbf{m}}}{2}, \frac{\bar{x}-x_{\mathbf{m}}}{2})$. Finally, initial brightness transformation parameter is initialized as $\gamma_0 = 1$.

Choice of optimizer. We find the choice of optimizer critical and that BasinCMA [10] provides better results than previously used optimizers for the GAN inversion problem. Previous work [70, 1] has exclusively used gradient-based optimization, such as LBFGS [47] and ADAM [39]. However, such methods are prone to obtaining poor results due to local minima, requiring the use of multiple random initial seeds. Covariance Matrix Adaptation (CMA) [28], a *gradient-free* optimizer, finds better solutions than gradient-based methods. CMA maintains a Gaussian distribution in parameter space $\mathbf{z} \sim \mathcal{N}(\mu, \Sigma)$. At each iteration, N samples are drawn, and the Gaussian is updated using the loss. The details of this update are described in Hansen and Ostermeier [28]. A weakness of CMA is that when it nears a solution, it is slow to refine results, as it does not use gradients. To address this, we use a variant, BasinCMA [63], that alternates between CMA updates and ADAM optimization, where CMA distribution is updated after taking M gradient steps.

Next, we describe the optimization procedure between the transformation parameters ϕ and latent variables \mathbf{z}, \mathbf{c} .

Algorithm 1 Transformation-aware projection algorithm

Input: Image \mathbf{y} , initial class vector \mathbf{c}_0 , mask \mathbf{m}
Output: Transformation parameter ϕ^* , latent variable \mathbf{z}^* , class vector \mathbf{c}^*

```

1: # Optimize for transformation  $\phi$ 
2: Initialize  $(\mu_\phi, \Sigma_\phi) \leftarrow (\phi_0, 0.1 \cdot \mathbf{I})$                                 ▷  $\phi_0$  precomputed in Section 3.3
3: for  $\mathbf{n}$  iterations do
4:    $\phi_{1:N} \sim \text{SampleCMA}(\mu_\phi, \Sigma_\phi)$                                          ▷ Draw  $N$  samples of  $\phi$ 
5:    $\mathbf{z}_{1:N} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ , reset  $\mathbf{c}_{1:N} \leftarrow \mathbf{c}_0$                                 ▷ Reinitialize  $\mathbf{z}$  and  $\mathbf{c}$ 
6:   for  $\mathbf{m}$  iterations do
7:     for  $i \leftarrow 1$  to  $N$  do
8:        $g_i \leftarrow \mathcal{L}_{\text{mask}}(G_\theta(\mathbf{z}_i, \mathbf{c}_i), \mathcal{T}_{\phi_i}(\mathbf{y}), \mathcal{T}_{\phi_i}(\mathbf{m}))$           ▷ This loop is batched
9:        $(\mathbf{z}_i, \mathbf{c}_i) \leftarrow (\mathbf{z}_i, \mathbf{c}_i) - \eta \cdot \nabla_{\mathbf{z}, \mathbf{c}} g_i$                                 ▷ Update each sample  $\mathbf{z}, \mathbf{c}$ 
10:       $g_{1:N}^{inv} \leftarrow \mathcal{L}_{\text{mask}}(\mathcal{T}_{\phi_{1:N}^{-1}}(G_\theta(\mathbf{z}_{1:N}, \mathbf{c}_{1:N})), \mathbf{y}, \mathbf{m})$     ▷ Recompute loss with inverse
11:       $\mu_\phi, \phi_{1:N} \leftarrow \text{UpdateCMA}(\phi_{1:N}, g_{1:N}^{inv}, \mu_\phi, \Sigma_\phi)$                                 ▷ Section 3.4
12:      Set  $\phi^* \leftarrow \mu_\phi$ 
13: # Optimize for latent variables  $\mathbf{z}, \mathbf{c}$ 
14: Initialize  $(\mu_{\mathbf{z}}, \Sigma_{\mathbf{z}}) \leftarrow (\mathbf{0}, \mathbf{I})$ 
15: for  $\mathbf{p}$  iterations do
16:    $\mathbf{z}_{1:M} \sim \text{SampleCMA}(\mu_{\mathbf{z}}, \Sigma_{\mathbf{z}})$ , reset  $\mathbf{c}_{1:M} \leftarrow \mathbf{c}_0$           ▷ Draw  $M$  samples of  $\mathbf{z}$ 
17:   for  $\mathbf{q}$  iterations do
18:     for  $i \leftarrow 1$  to  $M$  do
19:        $g_i \leftarrow \mathcal{L}_{\text{mask}}(G_\theta(\mathbf{z}_i, \mathbf{c}_i), \mathcal{T}_{\phi^*}(\mathbf{y}), \mathcal{T}_{\phi^*}(\mathbf{m}))$           ▷ This loop is batched
20:        $(\mathbf{z}_i, \mathbf{c}_i) \leftarrow (\mathbf{z}_i, \mathbf{c}_i) - \nabla_{\mathbf{z}, \mathbf{c}} g_i$ 
21:        $g_{1:N}^{inv} \leftarrow \mathcal{L}_{\text{mask}}(\mathcal{T}_{\phi_{1:N}^{-1}}(G_\theta(\mathbf{z}_{1:N}, \mathbf{c}_{1:N})), \mathbf{y}, \mathbf{m})$     ▷ Recompute loss with inverse
22:        $\mu_{\mathbf{z}}, \Sigma_{\mathbf{z}} \leftarrow \text{UpdateCMA}_{\mathbf{z}}(\mathbf{z}_{1:M}, g_{1:M}^{inv}, \mu_{\mathbf{z}}, \Sigma_{\mathbf{z}})$                                 ▷ Section 3.4
23:      Set  $\mathbf{z}^*, \mathbf{c}^* \leftarrow \arg \min_{\mathbf{z}, \mathbf{c}} (g_{1:M})$                                 ▷ Choose the best  $\mathbf{z}, \mathbf{c}$ 

```

Choice of Loss function In Equation 5, we described two variants of our optimization objective. Ideally, we would like to optimize the former variant $\mathcal{L}_{\text{mask}}(\mathcal{T}_{\phi^{-1}}(G_\theta(\mathbf{z}, \mathbf{c})), \mathbf{y}, \mathbf{m})$ such that the target image \mathbf{y} is consistent throughout optimization; and we do so for all CMA updates. However for gradient optimization, we found that back-propagating through a grid-sampler to hurt performance, especially for small objects. A potential reason is that when shrinking a generated image, the grid-sampling operation sparsely samples the image. Without low-pass filtering, this produces a noisy and aliased result [24, 51]. Therefore, for gradient-based optimization, we optimize the latter version $\mathcal{L}_{\text{mask}}(G_\theta(\mathbf{z}, \mathbf{c}), \mathcal{T}_\phi(\mathbf{y}), \mathcal{T}_\phi(\mathbf{m}))$.

Two-stage approach. Historically, searching over spatial transformations with reconstruction loss as guidance has proven to be a difficult task in computer vision [5]. We find this to be the case in our application as well, and that joint optimization over the transformation ϕ , and variables \mathbf{z}, \mathbf{c} is unstable. We use a two-stage approach, as shown in Algorithm 1, where we first search for ϕ^* and use ϕ^* to optimize for \mathbf{z}^* and \mathbf{c}^* . In both stages, a gradient-free CMA outer loop maintains a distribution over the variable of interest in that stage. In the inner loop, ADAM is used to quickly find the local optimum over latent variables \mathbf{z}, \mathbf{c} .

To optimize for the transformation parameter, we initialize CMA distribution for ϕ . The mean μ_ϕ is initialized with pre-computed statistics ϕ_0 , and Σ_ϕ is set to $0.1 \cdot \mathbf{I}$ (Alg. 1, line 2). A set of transformations $\phi_{1:N}$ is drawn from CMA, and

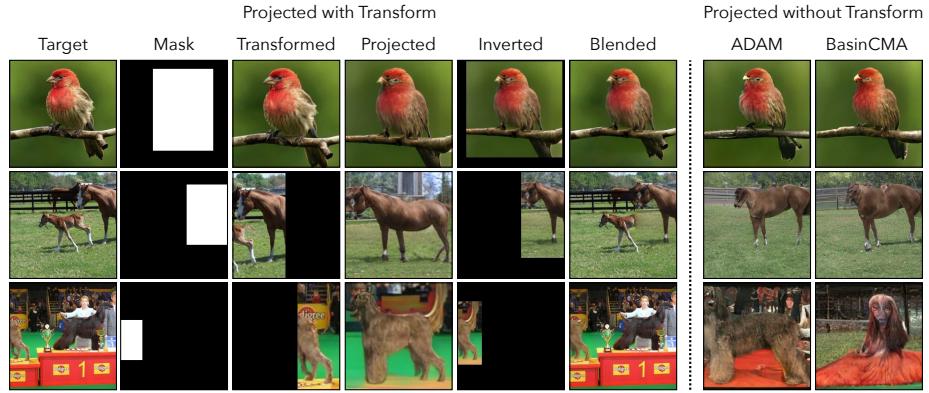


Fig. 6. Transformation search: Our method optimizes for a geometric and color transformation, such that the transformed image is more easily projected into the latent space. Transforming the image causes some pixels to be missing. To address this, we invert the results back to the original coordinates and poisson blend with the background after optimization. Results are not-finetuned.

latent variables $\mathbf{z}_{1:N}$ are randomly initialized (Alg. 1, line 4–5). To evaluate the sampled transformation, we take gradient updates w.r.t. $\mathbf{z}_{1:N}, \mathbf{c}_{1:N}$ for $m = 30$ iterations (Alg. 1, line 6–9). This inner loop can be interpreted as quickly assessing the viability of a given spatial transform. The final samples of $\mathbf{z}_{1:N}, \mathbf{c}_{1:N}, \phi_{1:N}$ are used to compute the loss for the CMA update (Alg. 1, line 10–11). This procedure is repeated for $n = 30$ iterations, and the final transformation ϕ^* is set to the mean of the current estimate of CMA (Alg. 1, line 12).

After solving for the transformation ϕ^* , a similar procedure is used to optimize for \mathbf{z} . We initialize CMA distribution for \mathbf{z} with $\mu_{\mathbf{z}} = \mathbf{0}$ and $\Sigma_{\mathbf{z}} = \mathbf{I}$ (Alg. 1, line 14). M samples of $\mathbf{z}_{1:M}$ are drawn from the CMA distribution and $\mathbf{c}_{1:M}$ is set to the initial predicted class vector (Alg. 1, line 16). The drawn samples are evaluated by taking $q = 30$ gradient updates w.r.t $\mathbf{z}_{1:M}$ and $\mathbf{c}_{1:M}$ (Alg. 1, line 17–20). The optimized samples are used to compute the loss for the CMA update (Alg. 1, line 21–22). This procedure is repeated for $p = 30$ iterations. On the final iteration, we take 300 gradient updates instead to obtain the final solution \mathbf{z}, \mathbf{c} (Alg. 1, line 23).

3.5 Fine-tuning

So far, we have located an approximate match within a generative model. We hypothesize that if a high-quality match is found, fine-tuning to fit the image will preserve the editability of the generative model. On the contrary, if a poor match is found, the fine-tuning will corrupt the network and result in low-quality images after editing. Next, we describe this fine-tuning process.

To synthesize the missing details that the generator could not produce, we wish to fine-tune our model after solving for the latent vector \mathbf{z} , the class vector \mathbf{c} , and transformation parameters ϕ . Unlike previous work [8], which proposed

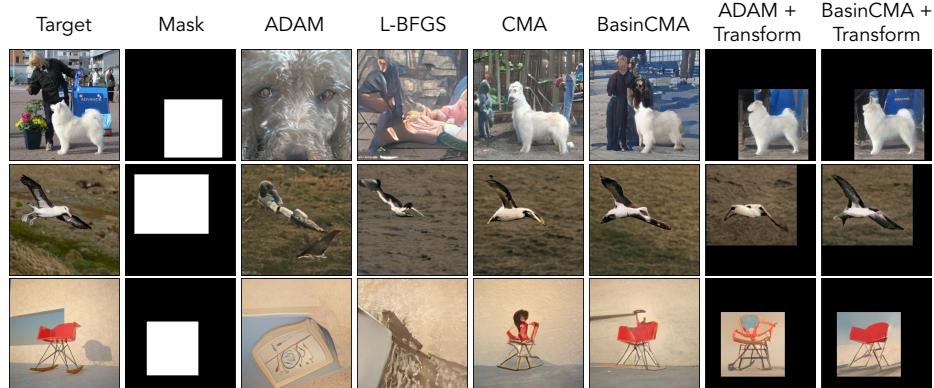


Fig. 7. ImageNet comparisons: Comparison across various methods on inverting ImageNet images without fine-tuning. A rectangular mask centered around the object of interest is provided for all methods using MaskRCNN [31]. The losses are weighted by the mask. BasinCMA+Transform is our full method.

to produce the residual features using a small, auxiliary network, we update the weights of the original GAN directly. This allows us to perform edits that spatially deform the image. After obtaining the values for $\phi, \mathbf{z}, \mathbf{c}$ in our projection step, we fine-tune the weights of the generative model. During fine-tuning, the full objective function is:

$$\arg \min_{\mathbf{z}, \mathbf{c}, \phi, \theta} \mathcal{L}_{\text{mask}}(\mathcal{T}_{\phi^{-1}}(G_{\theta}(\mathbf{z}, \mathbf{c})), \mathbf{y}, \mathbf{m}) + \lambda \|\theta - \theta_0\|_2 \quad \text{s.t. } C(\mathbf{z}) \leq C_{\max} \quad (6)$$

We put an ℓ_2 -regularization on the weights, such that the fine-tuned weights do not deviate too much from the original weights θ_0 . In doing so, we can prevent overfitting and preserve the generative model’s ability to edit the final image. We use $\lambda = 10^3$ for our results with fine-tuning.

4 Results

We demonstrate results on images from ImageNet [14], compare against baselines and ablations, examine cases that BigGAN cannot generate, and show failure cases. We further demonstrate the validity of our method on out-of-distribution data such as COCO and conduct perceptual studies on the edited images.

The ImageNet dataset consists of 1.3 million images with 1,000 classes. We construct a test set by using PASCAL [22] classes as super-classes. There are a total of 229 classes from ImageNet that map to 16 out of 20 classes in PASCAL. We select 10 images at random from each super-class to construct a dataset of 160 images. We run off-the-shelf Mask-RCNN [31] and take the highest activating class to generate the detection boxes. We use the same bounding box for all baselines, and the optimization hyper-parameters are tuned on a separate set of ImageNet images.

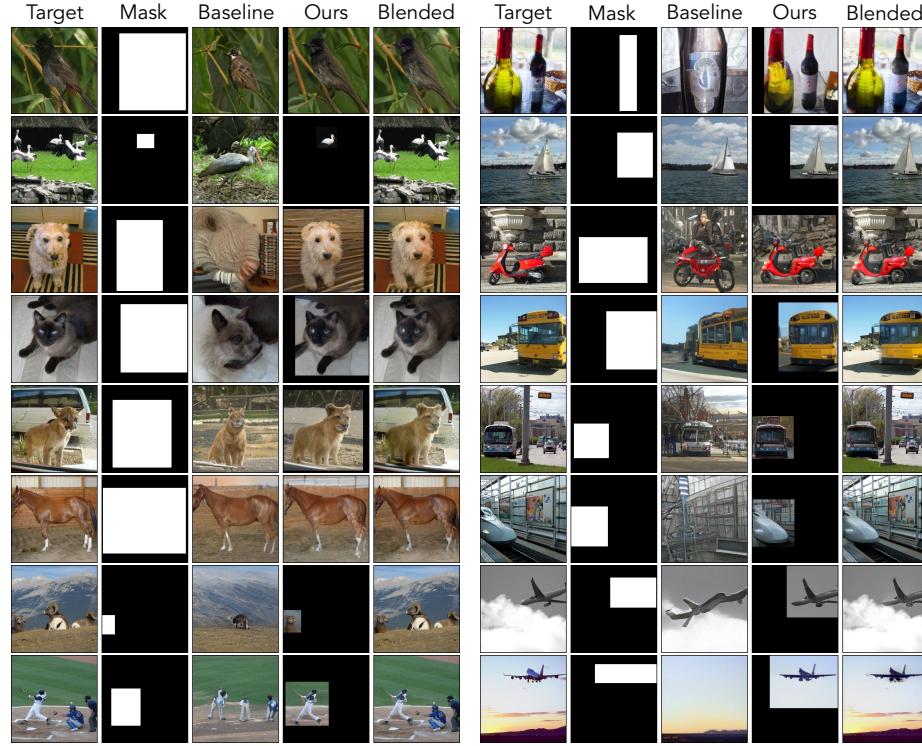


Fig. 8. **ImageNet results:** Results using our final method without fine-tuning. The final method uses BasinCMA as well as spatial and color transformation. Our generated results are inverted back for visualization. We also provide the ADAM baseline along with the blended result using Poisson blending [54].

Experimental details. We use a learning rate of 0.05 for \mathbf{z} and 0.0001 for \mathbf{c} . We use AlexNet-LPIPS [42, 69] as our perceptual loss for all our methods. We did observe an improvement using VGG-LPIPS [59, 69] but found it to be 1.5 times slower. In our experiments, we use a total of 18 seeds for each method. After we project and edit the object, we blend the newly edited object with the original background using Poisson blending [54].

For all of our baselines, we optimize both the latent vector \mathbf{z} and class embedding \mathbf{c} . We use the same mask \mathbf{m} , and the same loss function throughout all of our experiments. The algorithms we compare against are:

- ADAM [39]: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and optimized with ADAM. This method is used in Image2StyleGAN [1].
- L-BFGS [47]: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and optimized using L-BFGS with Wolfe line-search. This method is used in iGAN [70].
- CMA [28]: \mathbf{z} is optimized using CMA.
- ADAM + CMA [28]: \mathbf{z} is drawn from the optimized distribution of CMA. The seeds are further optimized with ADAM.

Optimizer	Method			Average of 18 seeds				Best of 18 seeds			
	Spatial Transform	Color Transform	Encoder	Per-pixel		LPIPS		Per-pixel		LPIPS	
				L1	L2	Alex	VGG	L1	L2	Alex	VGG
ADAM				0.98	0.62	0.41	0.58	0.83	0.47	0.33	0.51
L-BFGS				1.04	0.68	0.45	0.61	0.85	0.49	0.35	0.53
CMA				0.96	0.61	0.39	0.55	0.91	0.54	0.37	0.54
None			✓	1.61	1.39	0.62	0.68	1.35	1.00	0.55	0.64
ADAM	✓			0.96	0.60	0.39	0.56	0.82	0.46	0.32	0.51
ADAM		✓		0.98	0.62	0.42	0.58	0.83	0.47	0.33	0.51
ADAM	✓			0.90	0.54	0.44	0.57	0.76	0.41	0.36	0.50
ADAM	✓	✓	✓	0.88	0.52	0.42	0.55	0.76	0.40	0.36	0.49
CMA+ADAM				0.93	0.57	0.37	0.55	0.83	0.47	0.32	0.51
BasinCMA				0.82	0.48	0.29	0.51	0.78	0.43	0.26	0.49
BasinCMA			✓	0.82	0.47	0.29	0.50	0.78	0.43	0.26	0.49
BasinCMA		✓		0.81	0.46	0.29	0.50	0.77	0.42	0.25	0.49
BasinCMA	✓			0.72	0.38	0.33	0.48	0.69	0.35	0.31	0.46
BasinCMA	✓	✓	✓	0.71	0.37	0.32	0.47	0.68	0.34	0.31	0.46

Table 1. ImageNet: We compare various methods for inverting images from ImageNet (lower is better). The last row is our full method. The model is optimized using L1 and AlexNet-LPIPS perceptual loss. The mask and ground-truth class vector is provided for each method. We show the error using different metrics: per-pixel and perceptual [69]. We show the average and the best score among 18 random seeds. Methods that optimized for transformation are inverted to the original location and the loss is computed on the masked region for a fair comparison. All the results here are not fine-tuned.

- ADAM + BasinCMA [63]: \mathbf{z} is optimized by alternating CMA and ADAM updates (Section 3.4).
- + Encoder [70, 10]: \mathbf{z} is initialized with the output of the encoder. To generate variations in seeds, we add a Gaussian noise with a variance of 0.5. For BasinCMA, the mean of the CMA distribution is initialized with the output of the encoder.
- + Transform: Transformation parameter ϕ is optimized by alternating CMA updates on ϕ and ADAM updates on \mathbf{z} (Section 3.3).

Further optimization details are in the Appendix A.

Experiments. We show qualitative comparisons of various optimization methods for ImageNet images in Figure 7. We show results of our final method with blending in Figure 8. We then quantify these results by comparing against each method using various metrics in Table 1. For all methods, we do *not* fine-tune our results and we only compute the loss inside the mask for a fair comparison. For methods optimized with transformation, the projected images are inverted back before computing the loss. We further evaluate on COCO dataset [45] in Table 3, and observed our findings to hold true on out-of-distribution dataset. The success of hybrid optimization over purely gradient-based optimization techniques may indicate that the generative model latent space is locally smooth but not globally.

In Figure 6, we visualize how optimizing for spatial transformation allows us to better fit the target image. Without transforming the object, we observed that the optimization often fails to find an approximate solution, specifically when

Class search	Best of 18 seeds			
	Per-pixel		LPIPS	
	L1	L2	Alex	VGG
Random Gaussian	1.26	0.88	0.69	0.86
Random Class	0.88	0.51	0.40	0.59
Predicted	0.84	0.47	0.33	0.52
Ground Truth	0.83	0.47	0.33	0.51

Table 2. Class search: Given a fixed optimization method (ADAM), we compare different methods for initializing the class dataset (lower is better). Baselines are: initialized from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, a random class, and the ground truth class.

Method	Best of 18 seeds			
	Per-pixel		LPIPS	
	L1	L2	Alex	VGG
ADAM	0.96	0.57	0.32	0.56
ADAM + Transform	0.81	0.45	0.39	0.52
BasinCMA	0.93	0.18	0.81	0.53
BasinCMA + Transform	0.78	0.42	0.36	0.49

Table 3. Out-of-distribution: We compare different methods on the COCO-dataset (lower is better). BigGAN was not vector (lower is better). Baselines are: initialized on COCO images. The class labels are predicted using ResNext-101 and the masks are predicted using MaskRCNN.

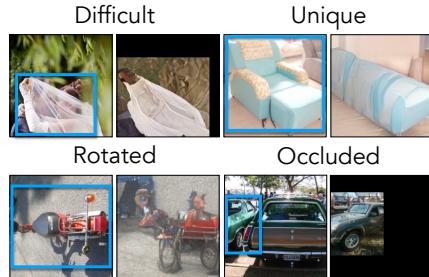


Fig. 9. Failure cases: Our method fails to invert images that are not well represented by BigGAN. The mask is overlayed on the target image in blue.

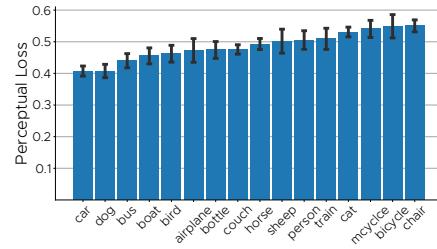


Fig. 10. Projection error by class: The average VGG-perceptual loss with standard error. The ImageNet images are sampled from the PASCAL super-class.

the objects are off-centered or contain multiple objects. We observed that optimizing over color transformation does not lead to drastic improvements. Possibly because BigGAN can closely match the color gamut statistics of ImageNet images. Nonetheless, we found that optimizing for color transformation can slightly improve visual aesthetics. Out of the experimented color transformations, optimizing for brightness gave us the best result, and we use this for color transformation throughout our experiments. We further experimented with composing multiple color transformations but did not observe additional improvements.

We found that using CMA/BasinCMA is robust to initialization and is a better optimization technique regardless of whether the transform was applied. Note that we did not observe any benefits of optimizing the class vectors \mathbf{c} with CMA compared to gradient-based methods, perhaps because the embedding between the continuous class vectors is not necessarily meaningful. Qualitatively, we often found the class embeddings to be meaningful when it is either in the close vicinity of original class embeddings or between the interpolation of 2 similar classes and not more. As a result, we use gradient descent to search within the local neighborhood of the initial class embedding space.



Fig. 11. Fine-tuned edits: Inversion results on various datasets. We use BasinCMA and transformation to optimize for the latent variables. After obtaining the projections, we fine-tune the model weights and perform edits in the latent and class vector space.

We also provide ablation study on how the number of CMA and ADAM updates for BasinCMA affects performance, and how other gradient-free optimizers compare against CMA in Appendix D. We further provide additional qualitative results for our final method in Appendix C.

Class initialization. In downstream editing application, the user may not know the exact ImageNet class the image belongs to. In Table 2, we compare different strategies for initializing the class vector. Here the classifier makes an incorrect prediction 20% of the time. We found that using the predicted class of an ImageNet classifier performs almost as well as the ground truth class. Since we optimize the class vector, we can potentially recover from a wrong initial guess if the predicted class is sufficiently close to the ground-truth.

Failure cases. Figure 9 shows some typical failure cases. We observed that our method fails to embed images that are not well modeled by BigGAN – outlier modes that may have been dropped. For example, we failed to project images that are unique, complicated, rotated, or heavily occluded. More sophisticated transformations such as rotations and perspective transformation could address many of these failure cases and are left for future work.

Which classes does BigGAN struggle to generate? Given our method, we analyze which classes BigGAN, or our method has difficulty generating. In Figure 10, we plot the mean and the standard error for each class. The plot is from the output of the method optimized with ADAM + CMA + Transform. We observed a general tendency for the model to struggle in generating objects with delicate structures or with large inter-class variance.

Image Edits. A good approximate solution allows us to fine-tune the generative model and recover the details easily. Good approximations require less fine-tuning and therefore preserve the original generative model editing capabilities. In Figure 11, we embed images from various datasets including CIFAR [41],

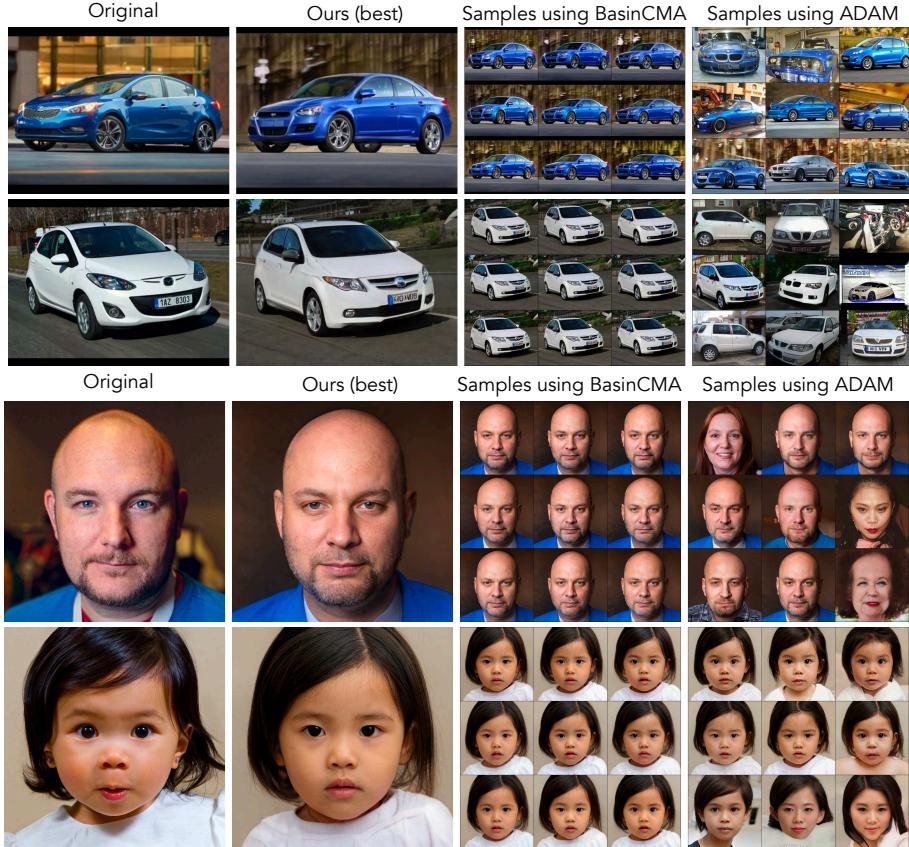


Fig. 12. Inverting StyleGAN2 in z space. We show results of projecting real images into StyleGAN2 using our BasinCMA method without transformation and fine-tuning. The images are inverted into the original input latent code $z \in \mathbb{R}^{512}$. The top results are from a model trained on 512×512 LSUN cars, and the results on the bottom are from a model trained on 1024×1024 FFHQ face dataset. We show results from the top 9 seeds for both BasinCMA and ADAM.

LSUN [67], and images in-the-wild. We then fine-tune and edit the results by changing the latent vector or class vector. Prior works [55, 34] have found that certain latent vectors can consistently control the appearance change of GANs-generated images such as shifting an image horizontally or zooming an image in and out. We used the “shift” and “zoom” vectors [34] to modify our images. Additionally, we also varied the class vector to a similar class and observed the editability to stay consistent. Even for images like CIFAR, our method was able to find good solutions that allowed us to edit the image. In cases like LSUN, where there is no corresponding class for the scene, we observed that the edits ended up being meaningless.

Perceptual Study. We verify the quality of the projected results with a perceptual study on edited projections. We fine-tune each projection to the same reconstruction quality across methods and apply edits to the latent variable \mathbf{z} . We show each image to an Amazon Mechanical Turker for 1 second and ask whether the edited image is real or fake, similar to [68]. Our method (BasinCMA + Transform) achieves 26% marked as real, while the baselines achieve 22% for ADAM, 23% for ADAM + Transform, and 25% for BasinCMA. This indicates that our design choices, adding transforms and choice of optimization algorithm, produces inversions that better enable downstream editing.

Inverting unconditional generative model: StyleGAN2. StyleGAN [37] and StyleGAN2 [38] are other popular choices of generative models for their ability to produce high fidelity images. Although these models can generate high-resolution images, they are restricted to generating images from a single class. Additionally, Abdal et al. [1] have demonstrated that it is difficult to project images into the original latent space $z \in \mathbb{R}^{512}$ using gradient-descent methods. Henceforth, Image2StyleGAN [1] and StyleGAN2 [38] has relied on inverting images into its intermediate representation, also known as the w^+ space. The w^+ space is \mathbb{R}^{699536} for generative model that outputs images of size 512×512 . Due to the large dimensionality of the intermediate representation, it is much easier to fit any real image into the generative model. Embedding the image into this intermediate representation drastically limits the ability to use the generative model to edit the projected images. On the contrary, we show in Figure 12 that CMA-based methods can invert the images all the way back to the original latent code $z \in \mathbb{R}^{512}$. We observed that models trained on well-aligned images such as FFHQ face dataset [37] can often be inverted using gradient-descent methods; however, models trained on more challenging datasets such as LSUN cars [67] can often only be solved using BasinCMA.

5 Discussion

Projecting an image into the “space” of a generative model is a crucial step for editing applications. We have systematically explored methods for this projection. We show that using a gradient-free optimizer, CMA, produces higher quality matches. We account for biases in the generative model by enabling spatial and color transformations in the search, and the combination of these techniques finds a closer match and better serves downstream editing pipelines. Future work includes exploring more transformations, such as local geometric changes and global appearance changes, as well as modeling generation of multiple objects or foreground/background.

Acknowledgements. We thank David Bau, Phillip Isola, Lucy Chai, and Erik Härkönen for discussions, and David Bau for encoder training code.

References

1. Abdal, R., Qin, Y., Wonka, P.: Image2stylegan: How to embed images into the stylegan latent space? In: International Conference on Computer Vision (2019) [4](#), [6](#), [8](#), [12](#), [17](#), [22](#)
2. Anirudh, R., Thiagarajan, J.J., Kaikhura, B., Bremer, P.T.: Mimicgan: Robust projection onto image manifolds with corruption mimicking. International Journal of Computer Vision (2020) [6](#)
3. Ankit, R., Li, Y., Bresler, Y.: Gan-based projector for faster recovery with convergence guarantees in linear inverse problems. In: International Conference on Computer Vision (2019) [4](#)
4. Asim, M., Shamshad, F., Ahmed, A.: Blind image deconvolution using deep generative priors. In: British Machine Vision Conference (2018) [3](#)
5. Baker, S., Matthews, I.: Lucas-kanade 20 years on: A unifying framework. International Journal of Computer Vision **56**(3), 221–255 (2004) [9](#)
6. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: A randomized correspondence algorithm for structural image editing. ACM Transactions on Graphics (TOG) (2009) [3](#)
7. Bau, D., Liu, S., Wang, T., Zhu, J.Y., Torralba, A.: Rewriting a deep generative model. In: European Conference on Computer Vision (2020) [1](#)
8. Bau, D., Strobelt, H., Peebles, W., Wulff, J., Zhou, B., Zhu, J.Y., Torralba, A.: Semantic photo manipulation with a generative image prior. ACM Transactions on Graphics (TOG) (2019) [1](#), [3](#), [4](#), [10](#), [25](#)
9. Bau, D., Zhu, J.Y., Strobelt, H., Zhou, B., Tenenbaum, J.B., Freeman, W.T., Torralba, A.: Gan dissection: Visualizing and understanding generative adversarial networks. In: International Conference on Learning Representations (2019) [1](#), [3](#)
10. Bau, D., Zhu, J.Y., Wulff, J., Peebles, W., Strobelt, H., Zhou, B., Torralba, A.: Seeing what a gan cannot generate. In: International Conference on Computer Vision (2019) [1](#), [4](#), [6](#), [7](#), [8](#), [13](#), [22](#)
11. Brock, A., Donahue, J., Simonyan, K.: Large scale gan training for high fidelity natural image synthesis. In: International Conference on Learning Representations (2019) [2](#), [4](#), [5](#)
12. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Neural photo editing with introspective adversarial networks. In: International Conference on Learning Representations (2017) [1](#), [3](#), [4](#)
13. Creswell, A., Bharath, A.A.: Inverting the generator of a generative adversarial network. IEEE Transactions on Neural Networks and Learning Systems **30**(7), 1967–1974 (2018) [4](#)
14. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition (2009) [11](#)
15. Donahue, J., Krähenbühl, P., Darrell, T.: Adversarial feature learning. In: International Conference on Learning Representations (2017) [4](#)
16. Donahue, J., Simonyan, K.: Large scale adversarial representation learning. In: Advances in Neural Information Processing Systems (2019) [4](#)
17. Dosovitskiy, A., Brox, T.: Generating images with perceptual similarity metrics based on deep networks. In: Advances in Neural Information Processing Systems (2016) [5](#)
18. Dosovitskiy, A., Brox, T.: Inverting visual representations with convolutional networks. In: IEEE Conference on Computer Vision and Pattern Recognition (2016) [3](#)

19. Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., Courville, A.: Adversarially learned inference. In: International Conference on Learning Representations (2017) [4](#)
20. Efros, A.A., Freeman, W.T.: Image quilting for texture synthesis and transfer. In: ACM SIGGRAPH (2001) [3](#)
21. Efros, A.A., Leung, T.K.: Texture synthesis by non-parametric sampling. In: International Conference on Computer Vision (1999) [3](#)
22. Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The pascal visual object classes challenge: A retrospective. International Journal of Computer Vision **111**(1), 98–136 (2015) [11](#)
23. Gatys, L.A., Ecker, A.S., Bethge, M.: Image style transfer using convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition (2016) [5](#)
24. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Pearson, 2nd edn. (1992) [9](#)
25. Goodfellow, I.: NIPS 2016 tutorial: Generative adversarial networks. arXiv preprint arXiv:1701.00160 (2016) [6](#)
26. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems (2014) [1](#)
27. Gu, J., Shen, Y., Zhou, B.: Image processing using multi-code gan prior. In: IEEE Conference on Computer Vision and Pattern Recognition (2020) [3](#)
28. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. evolutionary Computation (2001) [2, 8, 12](#)
29. Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on Github (2019). <https://doi.org/10.5281/zenodo.2559634> [22](#)
30. Härkönen, E., Hertzmann, A., Lehtinen, J., Paris, S.: Ganspace: Discovering interpretable gan controls. arXiv preprint arXiv:2004.02546 (2020) [1](#)
31. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: International Conference on Computer Vision (2017) [5, 6, 11](#)
32. Hertzmann, A., Jacobs, C.E., Oliver, N., Curless, B., Salesin, D.H.: Image analogies. In: ACM SIGGRAPH (2001) [3](#)
33. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks. In: Advances in Neural Information Processing Systems (2015) [7](#)
34. Jahanian, A., Chai, L., Isola, P.: On the “steerability” of generative adversarial networks. In: International Conference on Learning Representations (2020) [1, 6, 16](#)
35. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European Conference on Computer Vision (2016) [5](#)
36. Karras, T., Aila, T., Laine, S., Lehtinen, J.: Progressive growing of gans for improved quality, stability, and variation. In: International Conference on Learning Representations (2018) [7](#)
37. Karras, T., Laine, S., Aila, T.: A style-based generator architecture for generative adversarial networks. In: IEEE Conference on Computer Vision and Pattern Recognition (2019) [1, 7, 17](#)
38. Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T.: Analyzing and improving the image quality of StyleGAN. CoRR **abs/1912.04958** (2019) [17](#)
39. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2015) [2, 8, 12](#)
40. Kingma, D.P., Dhariwal, P.: Glow: Generative flow with invertible 1x1 convolutions. In: Advances in Neural Information Processing Systems (2018) [5](#)

41. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. Master's thesis, University of Toronto (2009) [15](#)
42. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012) [12](#)
43. Larsen, A.B.L., Sønderby, S.K., Larochelle, H., Winther, O.: Autoencoding beyond pixels using a learned similarity metric. In: International Conference on Machine Learning (2016) [3](#)
44. Levin, A., Lischinski, D., Weiss, Y.: A closed-form solution to natural image matting. IEEE Transactions on Pattern Analysis and Machine Intelligence **30**(2), 228–242 (2007) [3](#)
45. Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Doll'ar, P., Zitnick, C.L.: Microsoft COCO: common objects in context. In: European Conference on Computer Vision (2014) [13](#)
46. Lipton, Z.C., Tripathi, S.: Precise recovery of latent vectors from generative adversarial networks. ICLR workshop (2017) [4](#)
47. Liu, D.C., Nocedal, J.: On the limited memory bfgs method for large scale optimization. Mathematical programming **45**(1-3), 503–528 (1989) [2](#), [8](#), [12](#)
48. Mahendran, A., Vedaldi, A.: Understanding deep image representations by inverting them. In: IEEE Conference on Computer Vision and Pattern Recognition (2015) [3](#)
49. Olah, C., Mordvintsev, A., Schubert, L.: Feature visualization. Distill **2**(11), e7 (2017) [3](#)
50. Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., Mordvintsev, A.: The building blocks of interpretability. Distill **3**(3), e10 (2018) [3](#)
51. Oppenheim, A.V., Schafer, R.W., Buck, J.R.: Discrete-Time Signal Processing. Pearson, 2nd edn. (1999) [9](#)
52. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS 2017 Workshop (2017) [22](#)
53. Perarnau, G., Van De Weijer, J., Raducanu, B., Álvarez, J.M.: Invertible conditional gans for image editing. In: NIPS 2016 Workshop on Adversarial Training (2016) [1](#), [3](#), [4](#)
54. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Transactions on Graphics (TOG) **22**(3), 313–318 (2003) [3](#), [12](#), [24](#)
55. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks. In: International Conference on Learning Representations (2016) [1](#), [7](#), [16](#)
56. Rapin, J., Teytaud, O.: Nevergrad - A gradient-free optimization platform. <https://GitHub.com/FacebookResearch/Nevergrad> (2018) [24](#), [26](#)
57. Shah, V., H., C.: Solving linear inverse problems using gan priors: An algorithm with provable guarantees. In: ICASSP (2018) [4](#)
58. Shen, Y., Gu, J., Tang, X., Zhou, B.: Interpreting the latent space of gans for semantic face editing. In: IEEE Conference on Computer Vision and Pattern Recognition (2020) [3](#)
59. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015) [12](#)
60. Suzuki, R., Koyama, M., Miyato, T., Yonetsuji, T., Zhu, H.: Spatially controllable image synthesis with internal representation collaging. arXiv preprint arXiv:1811.10153 (2018) [4](#)

61. Tiantian, F., Schwing, A.: Co-generation with gans using ais based hmc. In: Advances in Neural Information Processing Systems (2019) [4](#)
62. de Vries, H., Strub, F., Mary, J., Larochelle, H., Pietquin, O., Courville, A.: Modulating early visual processing by language. In: Advances in Neural Information Processing Systems (2017) [22](#)
63. Wampler, K., Popović, Z.: Optimal gait and form for animal locomotion. ACM Transactions on Graphics (TOG) (2009) [8](#), [13](#)
64. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P., et al.: Image quality assessment: from error visibility to structural similarity. IEEE Transactions on Image Processing **13**(4), 600–612 (2004) [5](#)
65. Xie, S., Girshick, R., Dollr, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. arXiv preprint arXiv:1611.05431 (2016) [8](#)
66. Yeh, R.A., Chen, C., Yian Lim, T., Schwing, A.G., Hasegawa-Johnson, M., Do, M.N.: Semantic image inpainting with deep generative models. In: IEEE Conference on Computer Vision and Pattern Recognition (2017) [3](#), [4](#)
67. Yu, F., Zhang, Y., Song, S., Seff, A., Xiao, J.: Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv preprint arXiv:1506.03365 (2015) [16](#), [17](#)
68. Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: European Conference on Computer Vision (2016) [5](#), [17](#)
69. Zhang, R., Isola, P., Efros, A.A., Shechtman, E., Wang, O.: The unreasonable effectiveness of deep networks as a perceptual metric. In: IEEE Conference on Computer Vision and Pattern Recognition (2018) [5](#), [12](#), [13](#), [24](#)
70. Zhu, J.Y., Krähenbühl, P., Shechtman, E., Efros, A.A.: Generative visual manipulation on the natural image manifold. In: European Conference on Computer Vision (2016) [1](#), [3](#), [4](#), [8](#), [12](#), [13](#), [22](#)

A Training and run-time details

The experiments in the appendix were computed on a smaller subset of ImageNet images and are consistent within each other.

For computation, we use a single NVIDIA 2080 TI GPU. The run-time below is with respect to a single GPU. We use a total of 18 seeds in our main paper. The run-time is an over-estimate as we divide 18-seeds into 3 smaller batches to fit it into the GPU memory.

- **ADAM:** $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and optimized with ADAM. We optimize the latent vector for 500 iterations, roughly taking 5 minutes to invert a single image. We observed sharing momentum across random seeds can hurt performance, and we disentangle them in our runs. Furthermore, increasing the number of iterations does not significantly improve performance. This is the optimizer used in Image2StyleGAN [1].
- **L-BFGS:** $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and optimized using L-BFGS with Wolfe line-search. We use the PyTorch implementation [52] to optimize our latent vector for 500 iterations. We use the Wolfe line search with an initial learning rate set to 0.1. L-BFGS has an average run time of 5 minutes. This is the optimizer used in iGAN [70].
- **Encoder:** We follow the encoder-based initialization methods [70, 10] to train our encoder network on 10 million generated images, which took roughly 5 days to train. The encoder network was trained in a class-conditional manner, where the class information was fed into the network through the normalization layers [62]. We tried using the ImageNet pre-trained model to initialize the weights but found it to perform worse. It takes less than 1 second to run the encoder but requires additional gradient descent optimization steps for a reasonable result. We observed using an encoder still suffers the same problem as gradient-based methods and slightly improves the results. For our baseline (Encoder + ADAM), we still run ADAM for 500 iterations.
- **CMA:** \mathbf{z} is optimized using CMA. We use the python implementation of CMA [29]. For CMA-only optimization, we use 300 iterations. For CMA+ADAM, we use 100 CMA iterations and 500 ADAM updates. It takes roughly 0.2 seconds per CMA update.
- **BasinCMA:** \mathbf{z} is optimized by alternating CMA and ADAM updates. We use the same CMA implementation discussed above with 30 updates. For each update iteration, we evaluate after taking 30 gradient steps. The run-time is roughly 10 minutes per image. Increasing the number of updates and gradient descent steps does improve performance, see Figure 17.
- **Transformation:** Transformation parameter ϕ is optimized by alternating CMA updates on ϕ and ADAM updates on \mathbf{z} . We initialize the mean of the CMA using the statistics of generative images, as discussed in Section 3.4. We optimize for 30 iterations, where CMA is updated after 30 gradient updates on \mathbf{z}, \mathbf{c} . Optimizing for transformation adds an additional 5 minutes.

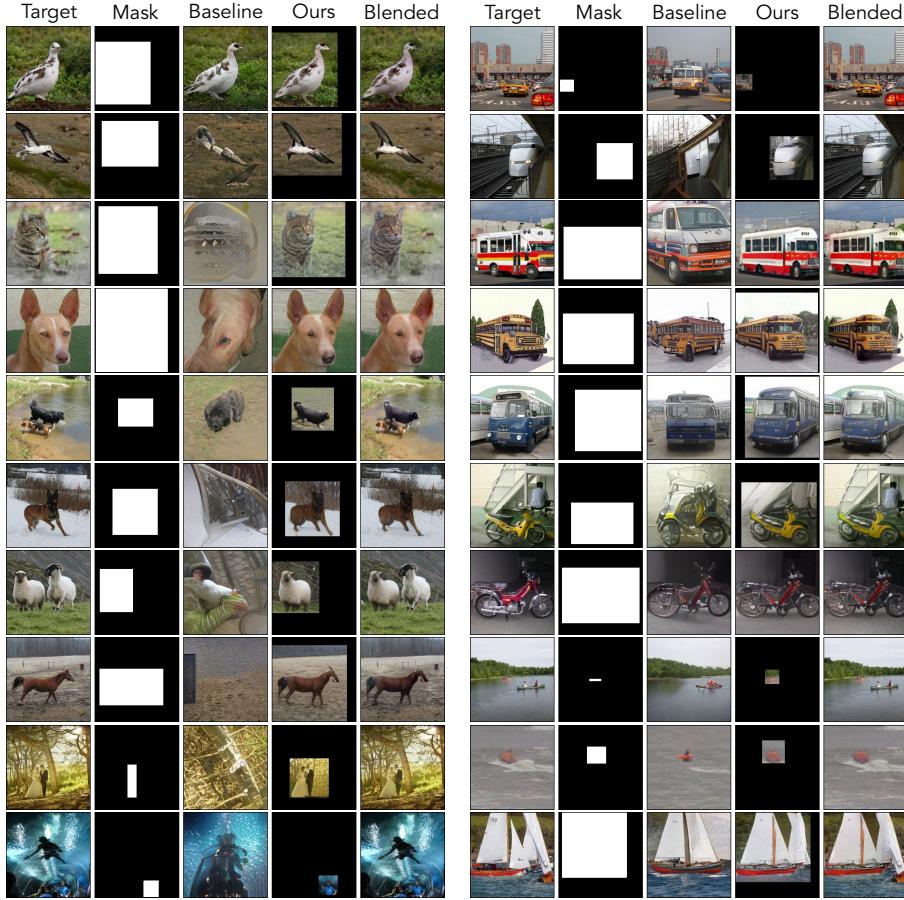


Fig. 13. **Additional results:** Comparison between ADAM and our final method. Our method is optimized using BasinCMA, and spatial and color transformation. The results shown above are not fine-tuned.

- **Encoder:** \mathbf{z} is initialized with the output of the encoder. To generate variations in seeds, we add a Gaussian noise with a variance of 0.5. For BasinCMA, the mean of the CMA distribution is initialized with the output of the encoder.
- **Fine-tuning:** We fine-tune the generative model using ADAM with a learning rate of 10^{-4} until the reconstruction loss falls below 0.1. We use the regularization weight of 10^3 , and we fix the batch-norm statistics during fine-tuning. The whole process takes roughly 1 minute.

B Weighted Perceptual Loss

We formulate the weighted LPIPS loss discussed in Section 3.4. Given an input image \mathbf{y} , a generated image $\hat{\mathbf{y}}$, we extract the image features from a pre-trained

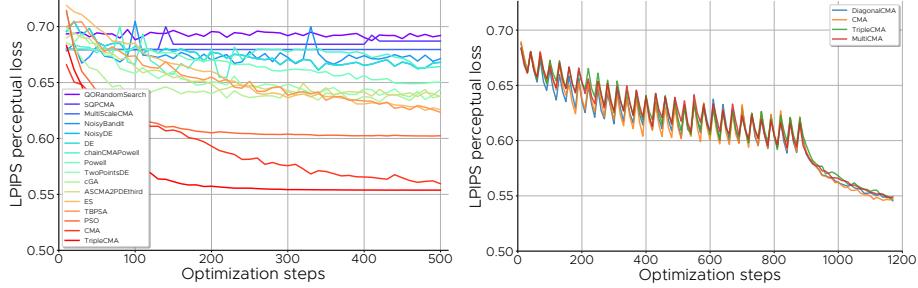


Fig. 14. Gradient-free optimizers: Experiments with various gradient-free optimizers. We use the implementations from Rapin and Teytaud [56]. The legend and color sorting by performance.

Fig. 15. Basin-CMA variants: Hybrid optimization with different CMA variants. We extended upon the implementations from Rapin and Teytaud [56]. All the CMA variants lead to similar results.

model to compute the loss. The features are extracted from pre-specified L convolutional layers [69]. We denote the intermediate feature extractor for layer $l \in L$ as $F^l(\cdot)$. The features extracted from a real image \mathbf{y} can be written as $F^l(\mathbf{y}) \in \mathbb{R}^{H_l \times W_l \times C_l}$ and similarly for $\hat{\mathbf{y}}$. A feature vector at a particular position is written as $F_{hw}^l(\cdot) \in \mathbb{R}^{C_l}$. LPIPS also provides a per-layer linear weighting $w_l \in \mathbb{R}_+^{C_l}$ to accentuate channels that are more “perceptual”. To weight the features spatially, we bilinearly resize the mask to match the spatial dimensions of each layer $m^l \in [0, 1]^{H_l \times W_l}$. Then the spatially weighted loss for LPIPS can be written as:

$$\mathcal{L}_{\text{mLPIPS}}(\mathbf{y}, \hat{\mathbf{y}}, \mathbf{m}) = \sum_{l \in L} \frac{1}{M^l} \sum_{h,w} m_{hw}^l \|w_l \odot (F_{hw}^l(\mathbf{y}) - \hat{F}_{hw}^l(\hat{\mathbf{y}}))\|_2^2 \quad (7)$$

Here \odot indicates elementwise multiplication in the channel direction and M^l is the sum of all elements in the mask.

C Additional results

We provide additional results for our method without fine-tuning in Figure 13. Our method is optimized using BasinCMA with spatial and color transformation. We provide the ADAM baseline along with our blended result using Poisson blending [54].

D Additional Analysis

Inner-outer optimization steps. Our optimization method maintains a CMA distribution of \mathbf{z} in the outer loop and is sampled to be optimized in the inner loop with gradient descent. Here the outer loop is the number of CMA updates,

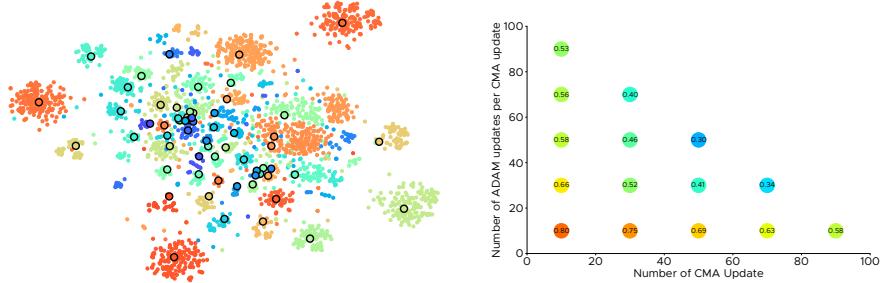


Fig. 16. **Class vector t-SNE:** The t-SNE embedding of the optimized class vector after optimization. The color represents the class and the circles with black border are the original classes.

Fig. 17. **BasinCMA update ablation:** We plot the VGG L-PIPS score when we vary the number of CMA updates (x-axis) and the number of ADAM updates (y-axis). Lower is better.

and the inner loop is the number of gradient descent updates to be applied before applying the CMA update. In Figure 17 we ablate the number of optimization steps and observed having the right balance of 1 : 1 ratio between CMA and gradient updates leads to the best result. The performance in the figure is mapped by color, with blue indicating the best and red indicating the worst. Although using 50 CMA update with 50 ADAM update performs the best, it requires more than 20-minutes to project a single image. We found 30 CMA updates and 30 gradient updates to be a sweet spot for run-time and image quality and is used in all our experiments. We observed the same trend when optimizing for the transformation.

Speeding up transformation search. Optimizing for transformation requires the model to quickly search over \mathbf{z} and \mathbf{c} given the sampled transformation F . Here \mathbf{z} and \mathbf{c} are reinitialized at the beginning of the CMA iteration. We observed that initializing \mathbf{z} by re-using the statistics from the previous iteration can speed up optimization by requiring fewer gradient updates. We found this to be quite important to have the algorithm run efficiently. After thorough testing, we found that sampling from a multivariate normal distribution with the mean centered around an exponential moving average of the best performing seeds to work the best. With the decay rate for the exponential moving average set to 0.5.

Encoder networks. Bau et al. [8] proposed an approach to efficiently train a model-specific encoder E to predict \mathbf{z} given a generated image \mathbf{y} , $E(\mathbf{y}) = \hat{\mathbf{z}}$. The encoder network is trained only on generated images, and therefore projecting real images often lead to incorrect predictions and require further optimization. Although initializing the optimization with the encoder does not lead to better results, we found that the optimization can converge 20% faster for gradient optimization and 40% faster for hybrid-optimization when $\mathbf{z} \sim \mathcal{N}(E(\mathbf{z}), 0.5 \cdot \mathbf{I})$.

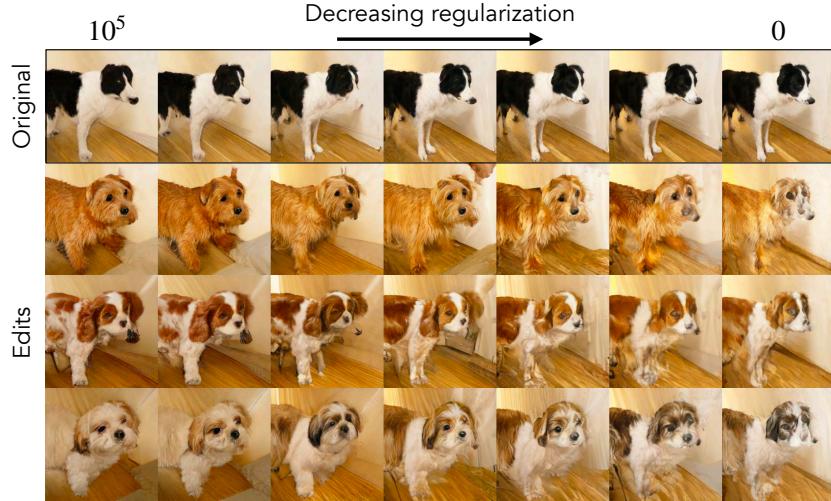


Fig. 18. How fine-tuning affects editing: We demonstrate how varying the regularization weight effects the edit-ability of the projected image. Images on the top are the original fine-tuned images with varying regularization weight, and the corresponding images below are class edited results. Decreasing the regularization weight allows us to fit the original image better, but introduces more editing artifacts.

Class-vector embedding. Optimizing for the class embedding allows the model to better fit the image into the generative model. We provide visualization of optimized class embedding using t-SNE in Figure 16. The classes are mapped by color and the original classes have black border. We observed that similar classes are embedded closer and class cross-overs are more common during optimization.

Gradient-free methods. We experimented with various gradient-free optimization methods using the Nevergrad library [56] in Figure 14. With the default optimization hyper-parameters, we found that CMA and its variants to perform the best. In Figure 15, we also experimented with hybrid optimization using various CMA variants but did not see a clear winner.

How fine-tuning effects editing. In Figures 1, 2, 11, we demonstrated having good projection allows us to fine-tune the weights to better fit the image without losing the editing capabilities of the generative model. In Figure 18, we visualize how such editing capability is affected by the fine-tuning process. We vary the regularization weight of the fine-tuning objective function that limits the deviation from the original weight. We observed that getting a better initial fit of the image requires us to relax the regularization weight, which in turn introduces additional editing artifacts. Therefore, we found it is crucial to approximate a good initial fit for real image editing.

E Changelog

v1 Initial preprint release.

v2 ECCV 2020 camera-ready version. (1) Add related work. (2) Add StyleGAN2 experiments.