Appendix for: Transforming and Projecting Images into Class-conditional Generative Networks

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

A Training and run-time details

For computation, we use a single NVIDIA V100 GPU. The run-time below is with respect to a single GPU. We use a total of 18 seeds in our main paper.

- **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 [7] 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 [14].
- **Encoder**: We follow the encoder-based initialization methods [14, 3] 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 [10]. 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 [4]. 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 15.
- **Transformation**: Transformation parameter ϕ is optimized by alternating CMA updates on ϕ and ADAM updates on $\mathbf{z}$. We initialize the mean of the

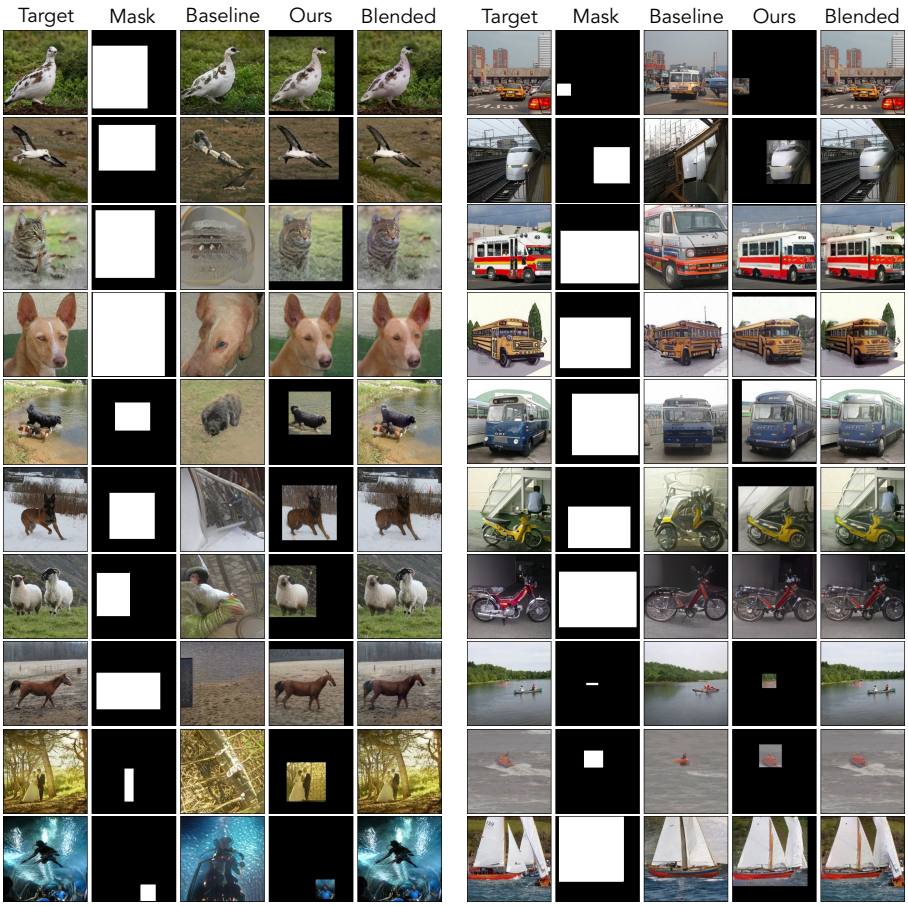

Fig. 11. **Additional results:** Comparison between ADAM and our final method. Our method is optimized using BasinCMA, and spatial and color transformation. The results shown above are not fine-tuned.

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.

- **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.

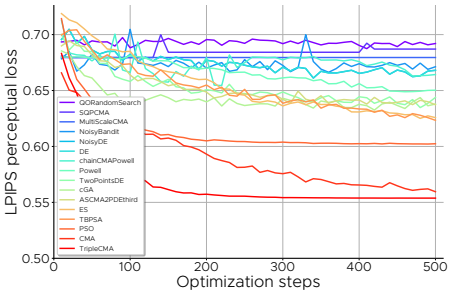

Fig. 12. **Gradient-free optimizers:** Experiments with various gradient-free optimizers. We use the implementations from Rapin and Teytaud [9]. The legend and the color are sorted by performance.

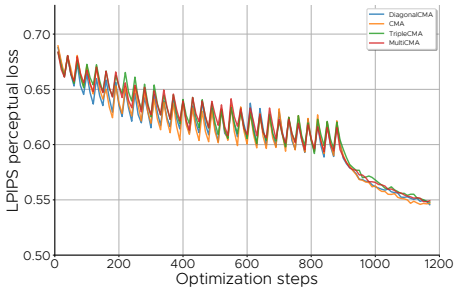

Fig. 13. **Basin-CMA variants:** Hybrid optimization with different CMA variants. We extended upon the implementations from Rapin and Teytaud [9]. All the CMA variants lead to similar results.

B Weighted Perceptual Loss

We formulate the weighted LPIPS loss discussed in Section 3.2. Given an input image $\mathbf{y}$, a generated image $\hat{\mathbf{y}}$, we extract the image features from a pre-trained model to compute the loss. The features are extracted from pre-specified L convolutional layers [13]. 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 $F^l_{hw}(\cdot) \in \mathbb{R}^{C_l}$. 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:

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

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 11. Our method is optimized using BasinCMA with spatial and color transformation. We provide the ADAM baseline along with our blended result using Poisson blending [8].

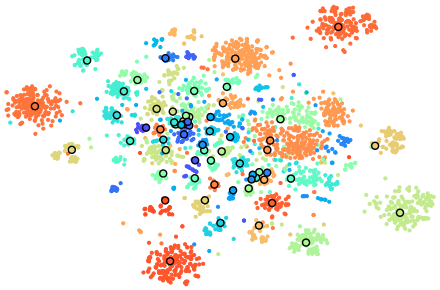

Fig. 14. **Class vector t-SNE:** The t-SNE embedding of the optimized class vector after optimization. The color represents the class and the circles with black border are the original classes.

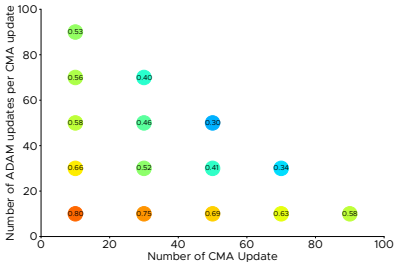

Fig. 15. **BasinCMA update ablation:** We plot the VGG L-PIPS score when we vary the number of CMA updates (x-axis) and the number of ADAM updates (y-axis). Lower is better.

D Additional Analysis

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 [12]. 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.

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, and the inner loop is the number of gradient descent updates to be applied before applying the CMA update. In Figure 15 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.

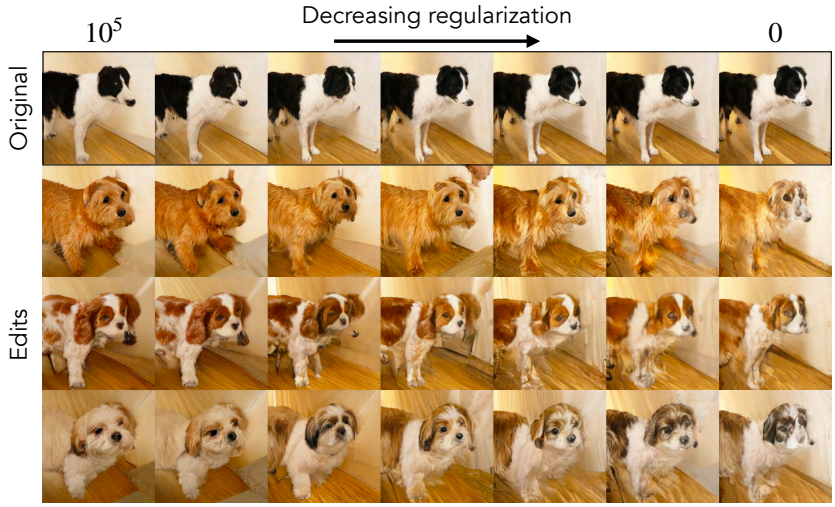

Fig. 16. **How fine-tuning affects editing:** We demonstrate how varying the regularization weight effects the edit-ability of the projected image. Images on the top are the original fine-tuned images with varying regularization weight, and the corresponding images below are class edited results. Decreasing the regularization weight allows us to fit the original image better, but introduces more editing artifacts.

We observed that initializing $\mathbf{z}$ by re-using the statistics from the previous iteration can speed up optimization by requiring fewer gradient updates. After thorough testing, we found that sampling from a variance-scaled multivariate Gaussian centered around the average of the previous iteration to work the best: $\{\mathbf{z}_i^{t+1}\}_{i=1}^n \sim \mathcal{N}(\frac{1}{n}\sum_{i=1}^n \mathbf{z}_i^t, 0.5 \cdot \mathbf{I})$, where $\mathbf{z}_0$ is zero-centered and t is indicates the CMA iteration.

Encoder networks. Bau et al. [2] 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 $\mathbf{z} \sim \mathcal{N}(E(\mathbf{z}), 0.5 \cdot \mathbf{I})$.

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 14. 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 [9] in Figure 12. With the default optimization hyper-parameters, we found that CMA and its variants to perform

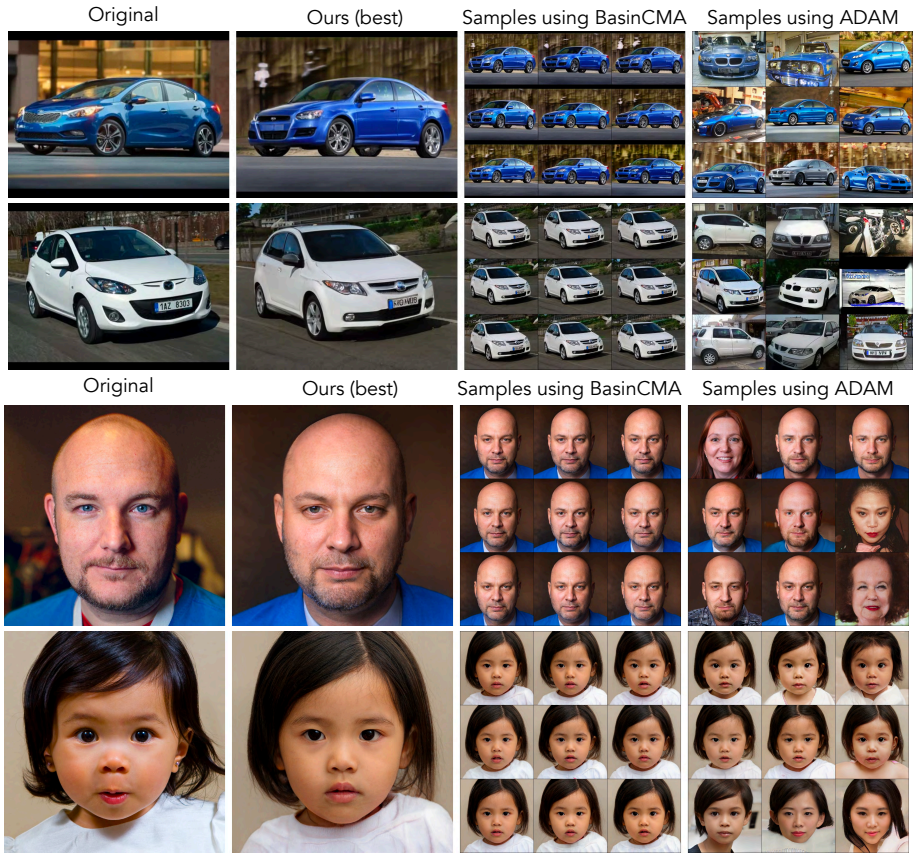

Fig. 17. **Inverting StyleGAN2 in z space.** We show results of projecting real images into StyleGAN2 using our BasinCMA method without transformation and fine-tuning. The images are inverted into the original input latent code $z \in \mathbb{R}^{512}$. The top results are from a model trained on 512×512 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.

the best. In Figure 13, we also experimented with hybrid optimization using various CMA variants but did not see a clear winner.

How fine-tuning effects editing. In Figure 1, 2, 10, 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 16, 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 Inverting unconditional generative model: StyleGAN2

StyleGAN [5] and StyleGAN2 [6] 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 [6] 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 17 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 [5] can often be inverted using gradient-descent methods; however, models trained on more challenging datasets such as LSUN cars [11] can often only be solved using BasinCMA.

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) 1, 7
2. 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) 5
3. 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. Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on Github (2019). https://doi.org/10.5281/zenodo.2559634 1
5. 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) 7
6. 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) 7
7. 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) 1
8. Pérez, P., Gangnet, M., Blake, A.: Poisson image editing. ACM Transactions on Graphics (TOG) **22**(3), 313–318 (2003) 3
9. Rapin, J., Teytaud, O.: Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad` (2018) 3, 5
10. 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) 1

11. 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) 7
12. Zhang, R., Isola, P., Efros, A.A.: Colorful image colorization. In: European Conference on Computer Vision (2016) 4
13. 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) 3
14. 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