A. Video Results

Please see our project page at www.wpeebles.com/gangealing for animated results, augmented reality applications and visual comparisons against other methods.

B. Implementation Details

Below we provide details regarding the implementation of GANgealing. Our code is also available at www.github.com/wpeebles/gangealing.

B.1. Spatial Transformer

Our full Spatial Transformer T is composed of two individual Spatial Transformers: T_{sim}—a network that takes an image as input and regresses and applies a similarity transformation to the input—and T_{flow}—a network that takes an image as input and regresses and applies an unconstrained dense flow field to the input. The warped output image produced by T_{sim} is fed directly into T_{flow}. Our final composed Spatial Transformer is given by $T = T_{\text{flow}} \circ T_{\text{sim}}$, where we compose the warps predicted by the two individual networks and apply it to the original input to obtain the final output.

The architectures of T_{sim} and T_{flow} are identical up to their final layers. Specifically, they each use a ResNet backbone [30], closely following the design of the ResNet-based discriminator from StyleGAN2 [43]; in particular, we do not use normalization layers. We do *not* use weight sharing for any layers of the two networks, and both modules are trained together jointly from scratch.

Similarity Spatial Transformer. The final parametric layer of T_{sim} takes the spatial features produced by the ResNet backbone, flattens them, and sends them through a fully-connected layer with four output neurons o_1, o_2, o_3 and o_4. We construct the affine matrix M representing the similarity transform as follows:

$$r = \pi \cdot \tanh(o_1) \tag{7}$$

$$s = \exp(o_2) \tag{8}$$

$$t_x = o_3 \tag{9}$$

$$t_y = o_4 \tag{10}$$

$$M = \begin{bmatrix} s \cdot \cos(r) & -s \cdot \sin(r) & t_x \\ s \cdot \sin(r) & s \cdot \cos(r) & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

To obtain the warped output image, we apply the affine matrix M to an identity sampling grid, and apply the resulting transformed sampling grid to the input.

Unconstrained Spatial Transformer. The features produced from T_{flow}'s ResNet backbone have spatial resolution 16×16. These features are fed into two separate, small convolutional networks: (1) a network that outputs a coarse flow of spatial resolution 16×16 with 2 channels (the first channel contains the horizontal flow while the second contains the vertical flow); (2) a network that outputs weights used to perform learned convex upsampling as described in RAFT [75]. Both of these networks consist of two conv layers (each with 3×3 kernels and unit padding) with a ReLU in-between. After performing $8\times$ convex upsampling on the coarse flow, we obtain our higher resolution flow $\mathbf{g}_{\text{dense}}$ of resolution 128×128. This flow field can be further densified with any type of interpolation.

In the case of the composed Spatial Transformer, we directly compose $\mathbf{g}_{\text{dense}}$ with the affine matrix M predicted by T_{sim}. We then sample from the original, unwarped input image according to this composed flow.

B.2. Clustering

For the clustering-variant of GANgealing ($K > 1$), our architectures as described above are only slightly changed as follows. T_{sim}'s final fully-connected layer has $4K$ output neurons that are used to build a total of K affine matrices $\{M_k\}_{k=1}^{K}$, one warp per cluster.

Similarly, the only change to T_{flow} is that the two convolutional networks at the end of the network each have K times as many output channels—each cluster predicts its own coarse-resolution 16×16 flow and its own set of weights used for convex upsampling.

Cluster Initialization. In contrast to the $K = 1$ case (where we initialize $\mathbf{c}$ as $\bar{\mathbf{w}}$, the average $\mathbf{w}$ vector), we randomly-initialize the $\mathbf{c}_i$. We do this using K-means++ [5] (only the initialization part). In standard K-means++, one selects centroids from points in the input dataset. In our case, we generate a pool of 50,000 random $\mathbf{w}$ vectors to apply K-means++ to. The first centroid $\mathbf{c}_0$ is selected uniformly at random from the pool. Recall that K-means++ requires a distance function so it can gauge how well-represented the points in the pool are by the currently selected centroids. We define the distance between two latent vectors as $d(\mathbf{w}_1, \mathbf{w}_2) = \ell(G(\text{mix}(\mathbf{w}_1, \bar{\mathbf{w}})), G(\text{mix}(\mathbf{w}_2, \bar{\mathbf{w}})))$. The motivation for feeding-in $\bar{\mathbf{w}}$ into the later layers of StyleGAN is to ensure that the two latents are being compared based on the poses they represent when decoded to images, *not* the appearances they represent.

B.3. Smoothly-Congealed Target Images

	PCK@$\alpha_{\text{bbox}} = 0.1$
w/o target annealing	59.3%
w/ target annealing	67.2%

Table 4. **The effects of smooth target annealing for LSUN Cats.** Smooth target annealing improves GANgealing performance.

A significant benefit of GAN-Supervision is that it naturally admits a smooth learning curriculum. Rather than force T to learn the (often complex) mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$ at the onset of training, we can instead smoothly vary the latent vector used to generate the target $\boldsymbol{y}$ from $\mathbf{w}$ to $\text{mix}(\mathbf{c}, \mathbf{w})$. Early in training, $\boldsymbol{y} \approx \boldsymbol{x}$ and T thus only has to learn very small warps where there are large regions of overlap between its input and target. As training proceeds, we gradually anneal the target latent $\mathbf{w} \rightarrow \text{mix}(\mathbf{c}, \mathbf{w})$ with cosine annealing [53] over the first 150,000 gradient steps. As this occurs, the learned $\boldsymbol{y}$ target images gradually become aligned across all $\mathbf{w}$ and require T to predict increasingly intricate warps. As shown in Table 4, excluding this smooth annealing degrades performance from 67.2% to 59.3%.

B.4. Horizontal Flipping

Because T_{sim} regresses the log-scale of a similarity transformation (o_2), it is incapable of performing horizontal flipping. While T_{flow} is in principle capable of learning how to horizontally (or vertically) flip an image, in practice, we find that it is beneficial to explicitly parameterize horizontal flips. We found two methods effective.

Unimodal Models. Our unimodal models ($K = 1$) are not trained with any flipping mechanism, and we introduce the capability only at test time. This is done with the flow smoothness trick as described above in Supplement G.2—we query T with both $\boldsymbol{x}$ and $\text{flip}(\boldsymbol{x})$, choosing whichever yields the smoothest flow field. This simple heuristic is surprisingly reliable.

Clustering Models. Clustering models provide a slightly more direct way to handle flipping. During training, when we evaluate the perceptual reconstruction loss on an input fake image $G(\mathbf{w})$, we also evaluate the loss on $\text{flip}(G(\mathbf{w}))$ (the loss is computed against $\boldsymbol{y} = G(\text{mix}(\mathbf{c}, \mathbf{w}))$ for both inputs). We only optimize the minimum of those two losses. At test time, we need to train a classifier to predict whether an input image should be flipped. To this end, we have the cluster classifier (as described in Section 3.2) directly predict both cluster assignment as well as whether or not the input should be flipped by simply doubling the number of output classes.

B.5. Training Details

As with most Spatial Transformers, we initialize the final output layers of our networks such that they predict the identity transformation. For T_{sim}, this is done by initializing the final fully-connected matrix as the zeros matrix (and zero bias); for T_{flow}, the convolutional kernel that outputs the coarse flow is set to all-zeros.

We jointly optimize the loss with respect to both T and $\mathbf{c}$ using Adam [45] (we do not use alternating optimization).

Dataset	$\mathcal{W}^+$ cutoff	N	K	GAN Resolution
LSUN (unimodal)	5	1	1	256×256
CUB	5	1	1	256×256
LSUN (clustering)	6	5	4	256×256
In-The-Wild CelebA	6	512	1	128×128

Table 5. **GANgealing Hyperparameters**.

We use a learning rate of 0.001 for T and a learning rate of 0.01 for $\mathbf{c}$. We apply the cosine annealing with warm restarts scheduler [53] to both learning rates. We use a batch size of 40. Finally, we note that we do not make use of any data augmentation—GANgealing uses raw samples directly from the generator as the training data.

B.6. Hyperparameters

We use $\lambda_{TV} = 2500$ for all models trained with LPIPS and $\lambda_{TV} = 1000$ for all models trained with the self-supervised perceptual loss. All models use $\lambda_I = 1$. We detail other hyperparameters in Table 5.

N controls how many degrees of freedom $\mathbf{c}$ has in choosing the target pose that T congeals towards. As shown in Table 3, small N are *critical* for some models (LSUN Cats with $N = 512$ gets less than 1% PCK on SPair Cats whereas $N = 1$ achieves 67%). However, StyleGAN2 generators with less expressive $\mathcal{W}$ spaces (such as those trained on In-The-Wild CelebA) can function well without any constraints ($N = 512$).

Padding Mode. One subtle hyperparameter is the padding mode of the Spatial Transformer which controls how T samples pixels beyond image boundaries. We found that `reflection` padding is the "safest" option and seems to work well in general. We also found `border` padding works well on some datasets (e.g., LSUN Cats and Dogs, CUB, In-The-Wild CelebA), but can be more prone to degenerate solutions on datasets like LSUN Bicycles and TVs. We recommend `reflection` as a default choice.

B.7. Direct Image-to-Image Correspondence

Our method is able to find dense correspondences directly between any pair of images $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$. Figure 8 gives an overview of the process. At a high-level, this merely involves applying the forward warp that maps points in $\boldsymbol{x}_A$ to points in $T(\boldsymbol{x}_A)$ and composing it with the reverse warp that maps points in the congealed coordinate space back to $\boldsymbol{x}_B$. We go into detail about this procedure in this section.

Recall that T outputs a sampling grid $\mathbf{g}$ which maps points in congealed space to points in the input image. Without loss of generality, say we wish to know where a point (i, j) in $\boldsymbol{x}_A$ corresponds to in $\boldsymbol{x}_B$. First, we need to determine where (i, j) maps to in the congealed image $T(\boldsymbol{x}_A)$—i.e., we need to *congeal* point (i, j). This is given by the value at pixel coordinate (i, j) in the **inverse** of the sampling

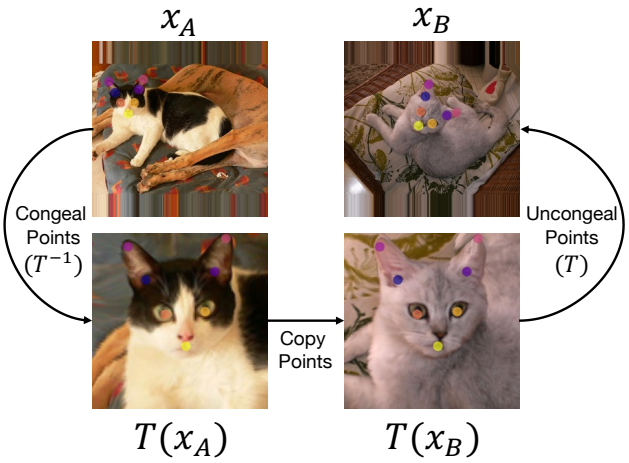

Figure 8. **The process for finding correspondences between two images with GANgealing**.

grid produced by T for $\boldsymbol{x}_A$. When T produces similarity transformations, we can analytically compute this inverse by inverting the affine matrix representing the similarity transform and applying it to the identity sampling grid. Unfortunately, for the unconstrained flow case we cannot analytically determine $\mathbf{g}^{-1}$. There are many ways one could go about obtaining this inverse. We opt for the simplest solution—inversion via nearest neighbors. Specifically, we can approximate the quantity by using nearest neighbors to find the pixel coordinates that give rise to the coordinates closest to (i, j) in $\mathbf{g}$. Recall that $\mathbf{g}_{i,j}$ is the input pixel coordinate that gets mapped to pixel (i, j) in congealed coordinate space. Then we can approximate $\mathbf{g}^{-1}_{i,j} \approx \arg\min_{i',j'} ||\mathbf{g}_{i',j'} - [i, j]||_2$.

Now that we know where points in $\boldsymbol{x}_A$ map to in $T(\boldsymbol{x}_A)$, the last step is to determine where the congealed points ($\mathbf{g}^{-1}_{i,j}$) map to in $\boldsymbol{x}_B$—i.e., we need to *uncongeal* $\mathbf{g}^{-1}_{i,j}$. This is the easy step: we need only query the sampling grid (produced by applying T to $\boldsymbol{x}_B$) at location $\mathbf{g}^{-1}_{i,j}$.

C. Clustering Results

Figure 9 shows our $K = 4$ learned clusters for our LSUN Cars and Horses models. We show dense correspondence results for various clusters in Figure 10.

D. Visualizing GAN-Supervised Training Data

We show examples of our paired GAN-Supervised training data in Figure 11. In Table 3, we showed that learning the fixed mode can be essential for some datasets (e.g., LSUN Cats). Figure 11 illustrates one potential reason why it is so critical: often, the initial truncated target mode in StyleGAN2 models produces *unrealistic* images. For example, truncated bicycles are surrounded with incoherent texture and have an unnatural structure, truncated TVs are unintelligible and truncated cats have unnatural bodies. Congealing all images to these poor targets could produce erroneous correspondences; hence, *learning* the target mode is in general very important.

Furthermore, the initial mode is often unsuitable as a dataset-wide congealing target. For example, not all LSUN Cat images (real or fake) feature the full-body of a cat, but most do feature a cat's upper-body. Hence, GANgealing updates the target to a more suitable target "reachable" by the broader distribution. Finally, also observe that our Spatial Transformer can be successfully trained even in the presence of *imperfect targets*: the target bicycles sometimes do not retain the color of the corresponding input fake bicycle, for instance.

E. Accelerating GAN Training with Learned Pre-Processing

A natural application of GANgealing is unsupervised dataset alignment for downstream machine learning tasks. In this section, we use our trained Spatial Transformer networks to align and filter data for GAN training.

Filtering. The first step is dataset filtering. We would like to remove two types of images: (1) images for which our Spatial Transformer makes a mistake (i.e., produces an erroneous alignment) and (2) images that are *unalignable*. For example, several images in LSUN Cats do not actually contain any cats. And, some images contain cats that cannot be well-aligned to the target mode (e.g., due to significant out-of-plane rotation of the cat's face). We can automatically detect these problematic images by examining the *smoothness* of the flow field produced by the Spatial Transformer for a given input image: images with highly un-smooth flows usually correspond to one of these types of problematic images. Please refer to Supplement G.2 below for a detailed explanation of this procedure. Here, we experiment with dropping 75% of real images (those with the least-smooth flow fields), reducing LSUN Cats' size from 1,657,264 images to 414,316 images.

Alignment. The second step is to align the filtered dataset. This is done by applying our Spatial Transformer to congeal every image in the input dataset. To avoid introducing excessive warping artifacts into the output distribution, we only use our similarity Spatial Transformer T_{sim} (which performs oriented cropping) in this step—we merely remove the unconstrained T_{flow} Spatial Transformer module from our trained T. To ensure a high quality output dataset, we remove some additional images during this procedure. (1) We removes images for which T_{sim} has to extrapolate a large number of pixels beyond image boundaries to avoid output images with lots of visible padding. (2) We remove images

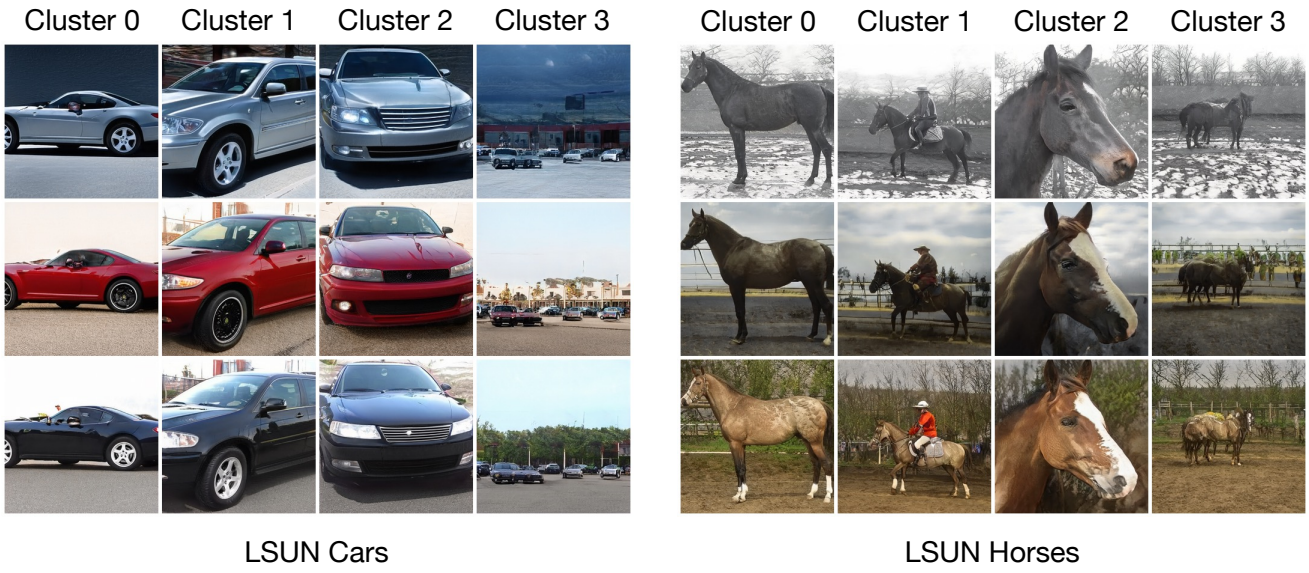

Figure 9. **Learned clusters**. We show three samples of target images $\boldsymbol{y}$ at the end of training which define our $K = 4$ learned clusters.

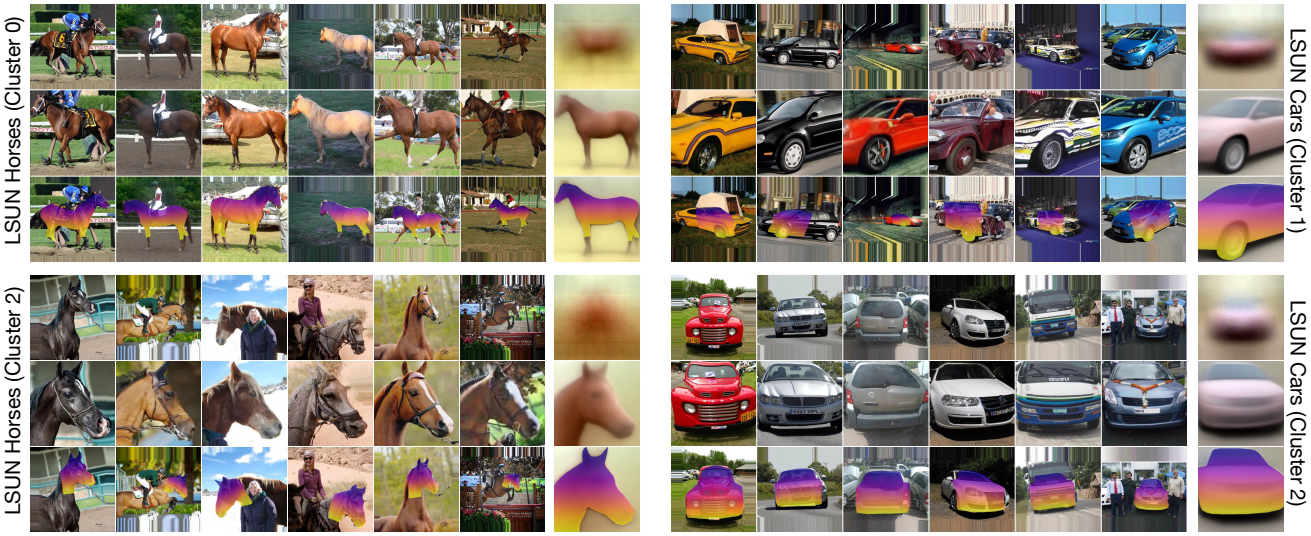

Figure 10. **Dense correspondence results for different clusters**. For each cluster, the top row shows unaligned real images assigned to that cluster by our cluster classifier; we also show the cluster's average image. The middle row shows our learned transformations of the input images. The bottom row shows dense correspondences between the images.

where T_{sim} zooms-in "too much." The motivation for this second criterion is that some images contain very small objects, and in these cases the congealed image will be low resolution (blurry). These two heuristics further reduce dataset size from 414,316 images to 58,814 high quality aligned images.

Quantitative Results. We apply our learned pre-processing procedure to the LSUN Cats dataset for downstream GAN training. As is common practice when training GANs on aligned datasets like AFHQ and FFHQ, we use mirror augmentations [42] when training on our aligned data. In Figure 12, we show that training with our learned pre-processing enables GANs to converge to good FID [31] faster by simplifying the training distribution. We also show that while *only* filtering the dataset to 58,814 images without alignment accelerates convergence (a conclusion similar to the one found by DeVries et. al [18]), both steps together provide the greatest speed-up. We stress that our gains in Figure 12 come from *simplifying the training distribution*. We leave showing how learned

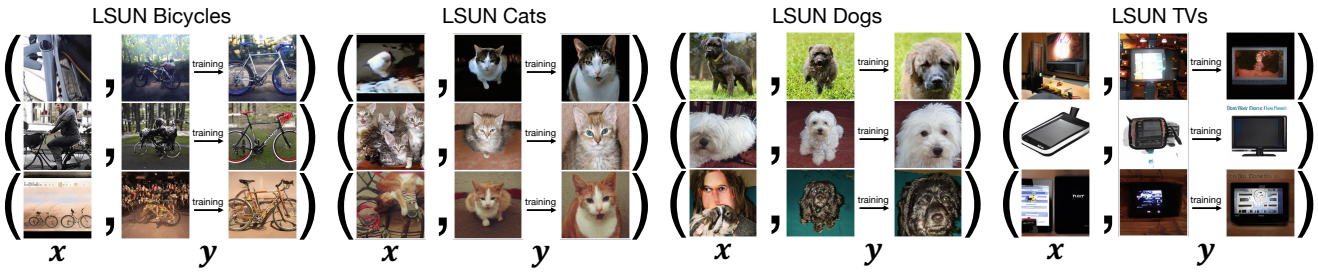

Figure 11. **Examples of GAN-Supervised paired data used in GANgealing**. For each dataset, we show random GAN samples $\boldsymbol{x} = G(\mathbf{w})$ used to train our Spatial Transformer. For each input $\boldsymbol{x}$, we show both the *initial* target image $\boldsymbol{y} = G(\text{mix}(\mathbf{c} = \bar{\mathbf{w}}, \mathbf{w}))$ as well as our *learned* target at the end of training $\boldsymbol{y} = G(\text{mix}(\mathbf{c}, \mathbf{w}))$. The initial target images (initialized with the truncation trick) are often unrealistic—for example, the initial LSUN TV targets are largely incoherent. Learning $\mathbf{c}$ is essential so images can be congealed to a coherent target. Note that we omit target annealing (Supplement B.3) in this visualization for clarity.

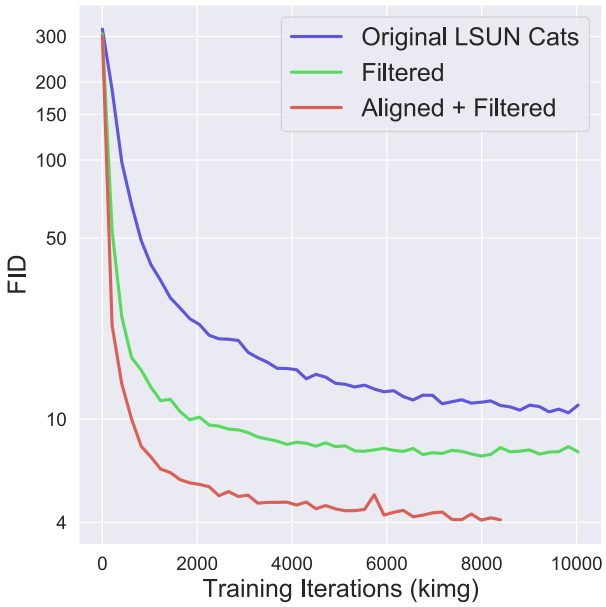

Figure 12. **The effect of aligning and filtering datasets before GAN training**. Each curve represents a StyleGAN2 trained on LSUN Cats with different data pre-processing. For each model, we compute FID against its corresponding pre-processed real distribution (i.e., the Original LSUN Cats curve computes FID against LSUN Cats, Filtered LSUN Cats computes FID against the filtered LSUN Cats distribution, etc.). Aligning and filtering images with our method yields significantly better FID early in training by simplifying the real distribution.

alignment can improve performance on the original, unaligned distribution to future work. Given the extent to which human-supervised dataset alignment and filtering is prevalent in the GAN literature [15, 40, 42], we hope our unsupervised procedure will be used in the future to automate these important steps.

Visual Results. In Figure 13 we show 96 uncurated, untruncated GAN samples for the baseline LSUN Cats model (FID versus LSUN Cats is 6.9); in Figure 14 we show uncurated samples from a StyleGAN2 trained on our learned aligned and filtered LSUN Cats (FID versus pre-processed LSUN Cats is 3.9). Our learned alignment improves the visual fidelity of the generator by reducing the complexity of the real distribution through alignment and filtering.

F. Uncurated Dense Correspondence Results

Below we show 120 *uncurated* dense correspondence results for each of our eight datasets. The results are best viewed zoomed-in.

G. Emergent Properties of GANgealing

We empirically find that our Spatial Transformer develops two useful test time capabilities from training with GANgealing.

G.1. Recursive Alignment

Recall that our Spatial Transformer consists of two submodules: a similarity warping Spatial Transformer T_{sim} and an unconstrained Spatial Transformer T_{flow}. Once trained, we find that T_{sim}'s primary role is to localize objects and correct in-plane rotation; T_{flow} brings the localized object into tight alignment by handling articulations, out-of-plane rotation, etc. For especially complex datasets (e.g., all LSUN categories), objects appear at many scales and are sometimes non-trivial to accurately localize. Surprisingly, we find that the T_{sim} network can be applied *recursively* multiple times to its own output at test time, significantly improving the ability of T to accurately align challenging images. This recursive processing appears to be relatively stable—i.e., T does not explode as we increase the number of recursions. We use three recursive iterations of T_{sim} for our LSUN models.

We suspect the reason this behavior emerges is because, over the course of training, it is likely that many generated in-

Figure 13. **Uncurated samples from an LSUN Cats StyleGAN2**.

Figure 14. **Uncurated samples from a StyleGAN2 trained on LSUN Cats pre-processed with our learned alignment and filtering**. Our method leads to a GAN with higher overall visual fidelity at the cost of reduced dataset complexity.

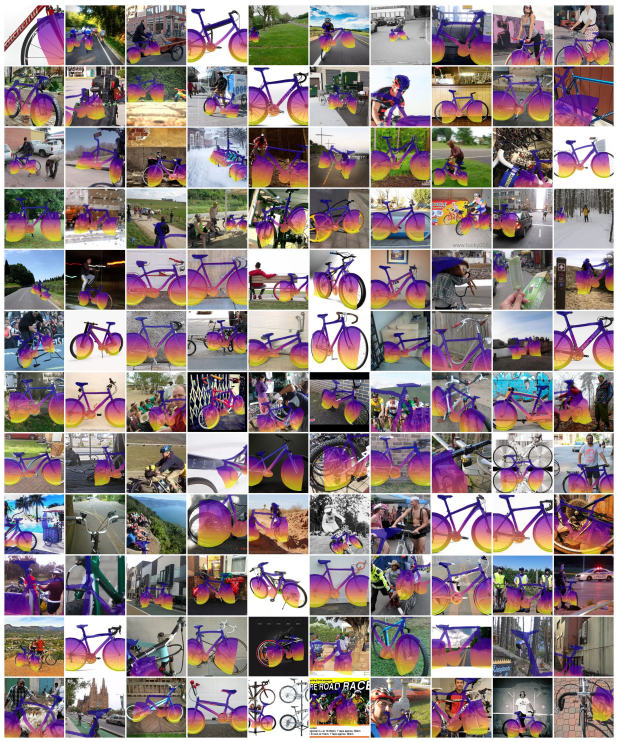

Figure 15. **LSUN Bicycles uncurated results**.

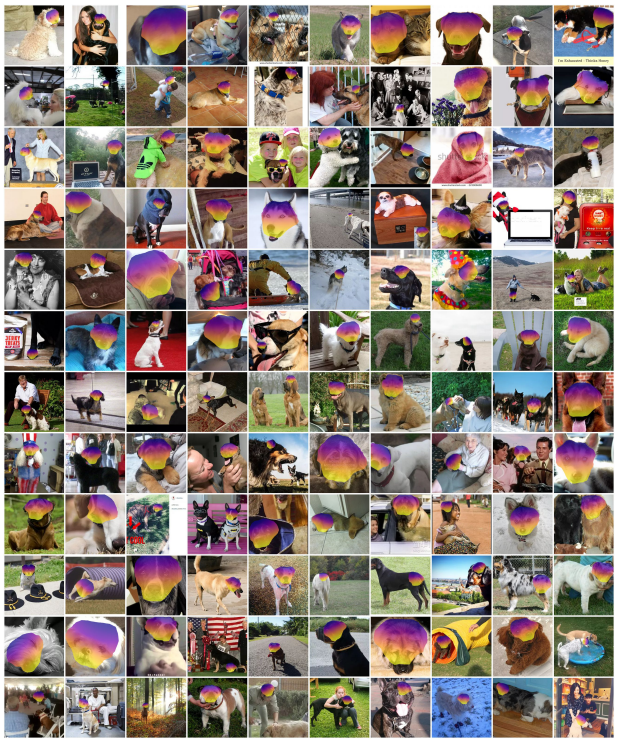

Figure 17. **LSUN Dogs uncurated results**.

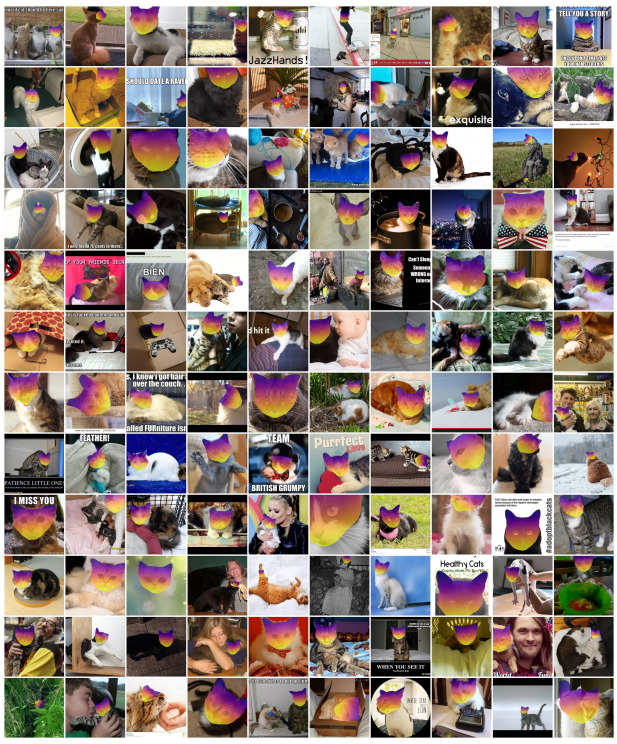

Figure 16. **LSUN Cats uncurated results**.

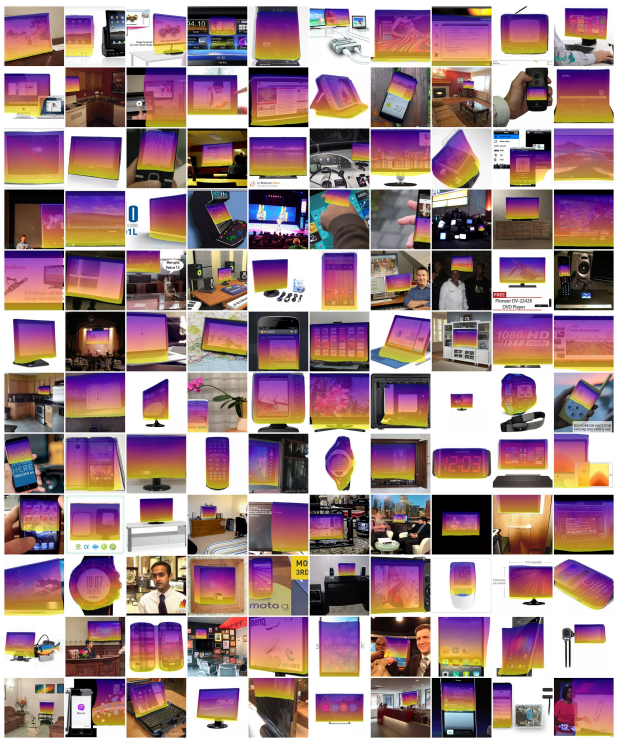

Figure 18. **LSUN TVs uncurated results**.

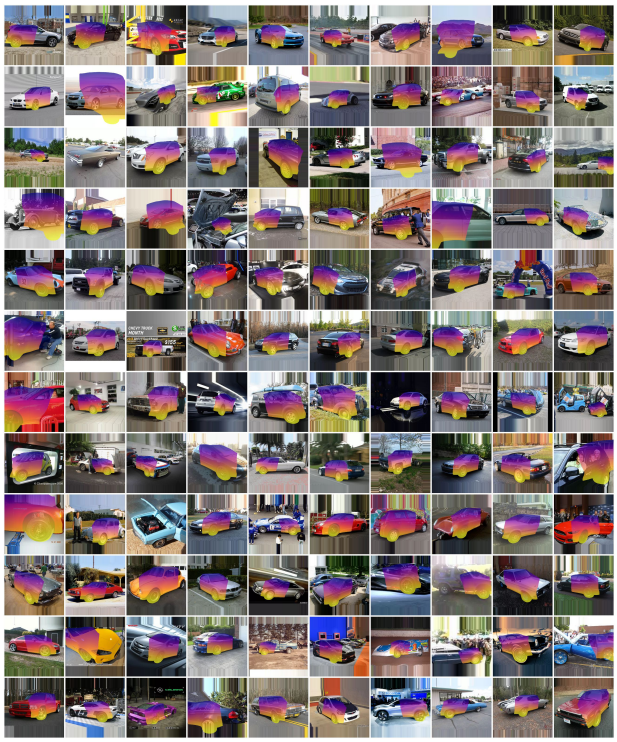

Figure 19. **LSUN Cars (cluster 1) uncurated results**.

Figure 20. **LSUN Horses (cluster 0) uncurated results**.

Figure 21. **In-The-Wild CelebA results**. The results are uncurated with the exception of replacing three potentially offensive images.

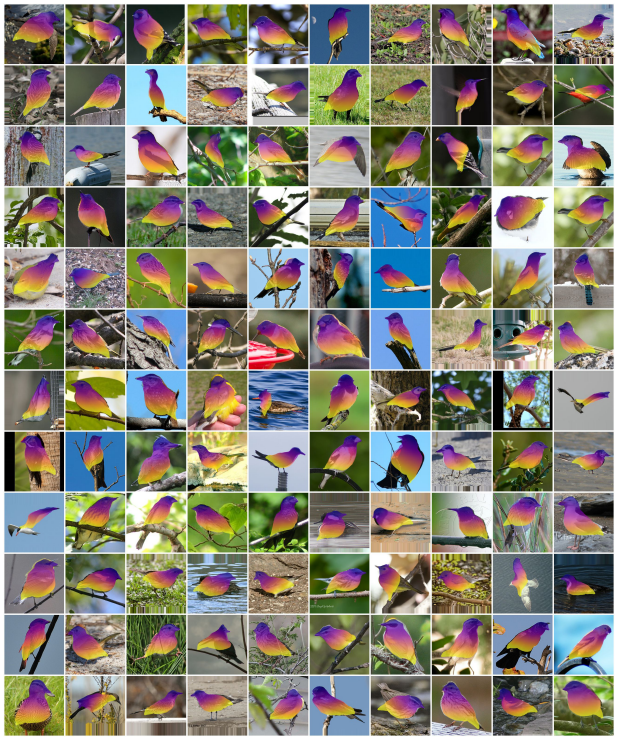

Figure 22. **CUB uncurated results**.

Figure 23. **GANgealing produces high frequency flow fields for unalignable images**. Each image in the top row shows a cat that either cannot be congealed to the learned mode or is a failure case of the Spatial Transfer. Our T produces high frequency flows to try and "fool" the perceptual loss for these hard cases. These examples are easily detected.

puts $\boldsymbol{x}$ will be sampled that are close to the target $\boldsymbol{y}$. In other words, it is likely that $\mathbf{w}$ get sampled such that $\mathbf{w} \approx \mathbf{c}$[3], and hence the Spatial Transformer must learn to produce approximately the identity function in these cases. Thus, aligned fake images happen to be *in-distribution* for T, meaning it can stably process increasingly aligned real images at test time without issue in this recursive evaluation.

G.2. Flow Smoothness

A particularly useful behavior developed by our T is a tendency to *fail loudly*. What happens when T is faced with an unalignable image or makes a mistake? As shown in Figure 23, T produces very high-frequency flow fields to try and fool the perceptual loss (e.g., by synthesizing cat ears and eyes from background texture). It turns out these failure cases are easily detectable by simply measuring how smooth the flow field produced by T is—this can be done by evaluating our $\mathcal{L}_{TV}$ total variation loss on the flow. This behavior enables several test time capabilities; for example, we can determine if an image should be horizontally flipped by merely running $\boldsymbol{x}$ and flip($\boldsymbol{x}$) through T and selecting whichever input yields the smoothest flow. Or, we can score real images by their flows' $\mathcal{L}_{TV}$ and drop a percentage of the dataset with the worst (most positive) scores—this is how we perform dataset filtering with our T as part of our learned pre-processing for downstream GAN training as described above in Supplement E.

H. Performance

Training. Our unimodal GANgealing models train for 1.3125M gradient steps; this takes roughly 48 hours on 8× A100 GPUs for 256×256 StyleGAN2 models. Over the course of training, our Spatial Transformers process more than 50M randomly-sampled GAN images.

[3]This is likely *as long as* $\mathbf{c}$ *is properly constrained*. When poorly constrained, $\mathbf{c}$ may correspond to an *out-of-distribution* image and hence T may not be used to processing images similar to sampled $\boldsymbol{y}$. Indeed, for our In-The-Wild CelebA model where $N = 512$ (i.e., $\mathbf{c}$ is not constrained at all), we find recursion is unstable. For all LSUN models where $N \leq 5$, we find it helps significantly.

Inference. In this section, we briefly discuss runtime comparisons between GANgealing and supervised baselines. We compare the time to perform the cartoon cat face augmented reality application online (batch size of 1, RTX 6000 GPU). This involves propagating over 3.9 million points every frame. GANgealing runs at 15 FPS, CATs [14] at 13 FPS, and RAFT [75] at 3 FPS. For time to perform keypoint transfers between pairs of images (SPair Cats), GANgealing runs at 31 pairs per second, and CATs runs at 70 pairs per second. We note that we did not optimize our implementation for inference, and so we expect it is possible to further improve performance.

I. Broader Impacts

As with all machine learning models, our Spatial Transformer is only as good as the data on which it is trained. In the case of GANgealing, this means our algorithm's success in finding correspondence depends on the underlying generative model which generates its training data. Concern over GANs' tendency to drop modes is well-documented (e.g., [9]). However, as latent variable generative models continue to improve—including efforts to improve mode coverage—we expect that GANgealing will improve as well.

J. Assets

- Cat Batman Mask
- Bird Soldier
- Rudolph Dog
- CelebA Pokémon Tattoo 1
- CelebA Pokémon Tattoo 2
- Unicorn Horn
- Horse Saddle
- Car Dragon Decal

Acknowledgements. We thank Tim Brooks for his antialiased sampling code and helpful discussions. We thank Tete Xiao, Ilija Radosavovic, Taesung Park, Assaf Shocher, Phillip Isola, Angjoo Kanazawa, Shubham Goel, Allan Jabri, Shubham Tulsiani and Dave Epstein for helpful discussions. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE 2146752. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Additional funding provided by Adobe and Berkeley Deep Drive.