A Appendix

A.1 Architectures and Training

Architectures: Nearly all network backbones follow either InfoGAN, if operating on handwritten characters, or DCGAN, if operating on natural images. Celeb-A networks include an additional half-width layer at the bottom of the encoder, and at the top of the generator, to account for increased image resolution. For convenience, we use the same network architecture for both AugIntAE encoders and auxiliary GAN discriminators. The only difference is that the discriminator has one output neuron in the final layer, while encoders have as many as there are latent dimensions.

We L_2-normalize our latent features and perform interpolation via spherical linear interpolation (SLERP). Latent dimensionality is set to 32 for handwritten character models, which, given their 28×28 resolution, amounts to an almost 25-fold reduction in dimensionality. Celeb-A dimensionality is set to 512, with a 128×128 resolution and three color channels, producing a reduction in dimensionality of almost two orders of magnitude. CIFAR images use the standard DCGAN resolution of 64×64 and 512 latent dimensions for an almost 25-fold dimensionality reduction, similar to the handwritten character models.

VAEs, WGAN-GPs, and classifiers all use the same architectures, plus necessary modifications to the number of input/output neurons. WGAN-GPs draw from the same latent dimensionality as the corresponding AE models, and receive L_2-normalized noise samples rather than samples from a standard Gaussian. VAE encoders have output dimensionality twice the above, as they predict both a mean μ and variance γ for each coordinate. Classifiers have as many output neurons as there are classes in the given task (2 for train/test dataset classifiers, 37 for EMNIST letter classifiers) and end in a softmax layer.

Training: Unless stated otherwise, all networks trained on real images use the Adam optimizer with initial step size .001. Models train for 100 passes over the given dataset. Models with no adversarial component cut the learning rate by a factor of 10 at epochs 35 and 70.

We chose these hyperparameters without a validation set, using convergence on the training data as our sole criterion.

GAN training: To stabilize learning, adversarial models do not change their learning rate. We also rescale all adversarial gradients so that their magnitudes match those of the reconstruction loss gradient. WGAN-GPs train for twice as long as other models, but update the generator only once per five discriminator updates.

On some datasets, we found that the auxiliary LSGAN discriminator could collapse and allow the generator to "win" the adversarial game. To prevent this, we introduce a scaling factor $k \in [0, 1]$ that updates dynamically based on discriminator performance. Specifically:

$$L_{AE} = L_{recon} + k * \gamma * L_{adv} \tag{2}$$

where

$$\gamma = \frac{\|\nabla L_{recon}\|_2}{\|\nabla L_{adv}\|_2} \tag{3}$$

calculated per-image, and k is adjusted with each gradient update according to the following rules:

$$k_0 = 1 \tag{4}$$

$$\bar{k} = k_t - .001 * (1 - (D(x_{real}) - D(x_{fake}))) \tag{5}$$

$$k_{t+1} = max(0, min(1, \bar{k})) \tag{6}$$

This update scheme for k ensures that whenever the scores coming from the discriminator D for real and fake images are separated by less than 1 on average, k decreases. The generator then focuses more on the reconstruction task and becomes less competitive, allowing the discriminator network

Input $3 \times 512 \times 512$ RGB image
4×4 conv. 32 BN ELU stride 2
4×4 conv. 64 BN ELU stride 2
4×4 conv. 128 BN ELU stride 2
4×4 conv. 256 BN ELU stride 2
4×4 conv. 512 BN ELU stride 2
4×4 conv. 1024 BN ELU stride 2
FC 512

Table 4: Inversion network architecture for PGAN.

to "catch up" until the margin of separation is greater than 1 again. At that point k increases back to 1, at which point the reconstruction and adversarial losses are equally weighted once more.

Augmentation Parameters: We sample data augmentation parameters ρ uniformly over predefined ranges of values. For handwritten character datasets, we sample rotations in the range $[-20, 20]$, translations in the range $[-4, 4]$, scaling in the range $[.8, 1.2]$, and shear in the range $[-6, 6]$. These values were chosen heuristically: we picked the largest values that would not occlude or obscure the character.

Natural image AugIntAEs sample from half the range of rotation and skew, and double the range of translation (though because of the higher resolution, this comes out to much smaller displacement overall). These values were chosen so as not to introduce large regions of empty space into the image, while staying as large as possible. CIFAR images are symmetric-padded; Celeb-A images are center cropped and do not require padding. Additionally, we sample a range of color jitter parameters for AugIntAEs handling 3-channel RGB images. We sample brightness/contrast/saturation adjustments from the range $[.75, 1.25]$ and hue from the range $[-5\%, +5\%]$. We again chose these values to be as large as possible without producing unrealistic or unparsable images.

Similar to Sainburg et al. (2018), we found that the network learned interpolations near the seed images more easily than near the midpoint of the interpolation path. We therefore biased our sampled α toward the midpoint by sampling the mean of two uniform random draws.

To ensure that our baselines are trained on the same data distribution as our AugIntAEs, we use the α-weighted interpolated image as the training image, even when the network does not use interpolative training. This only applies to AE and IntAE models.

A.2 GAN INVERSION

Obtaining generalization results for a GAN involves inverting the generator, which we attempt via a combination of a learned inversion network and direct SGD optimization on the latent codes for each target image. For the PGAN (fig. 1), we use a publicly available pretrained model provided by Facebook Research[1]. For the MNIST/EMNIST generalization experiments (fig. 2) we use our own implementations, constructed and trained using the procedure described above. The learned inversion network for the PGAN generator uses the architecture described in Table 4, while the inversion network for the MNIST WGAN-GP uses the same encoder as in other experiments. Both networks are trained by sampling images from the generator, and attempt to reconstruct the corresponding latent code for each image using mean squared error. We use SGD with an initial step size of .0001 and momentum equal to .1. We train for 6400 iterations and cut the learning rate by a factor of 10 after every 1280 iterations.

The subsequent stage of inversion involves direct refinement of the latent codes provided by the inversion network via SGD in latent space. In both cases we use 1000 iterations of SGD, with a learning rate of .01 and no momentum. The loss is an L1 pixel reconstruction loss, resulting in the final images displayed in figures 1 and 2 of the main paper.

[1]https://pytorch.org/hub/facebookresearch_pytorch-gan-zoo_pgan

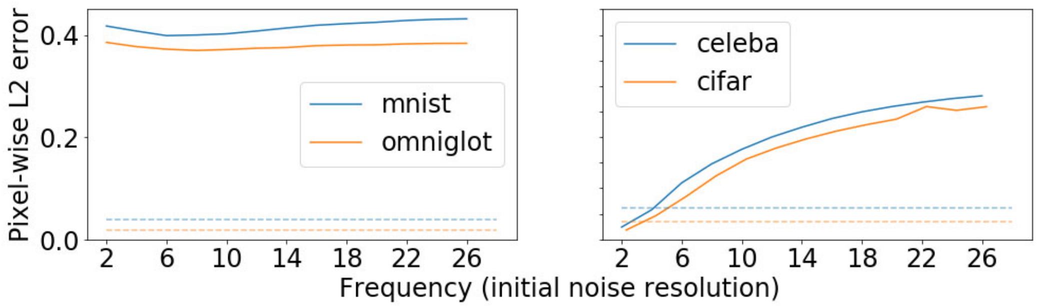

Figure 11: L2 pixel reconstruction error as a function of noise frequency, $[0, 1]$-scaled. Dotted lines represent reconstruction loss on novel domain images (EMNIST for the MNIST model, CIFAR-100 for the CIFAR-10 model, etc.

A.3 Quantitative Evaluation

Fréchet Inception Distance (FID) (Heusel et al., 2017) compares the distributions of embeddings of real ($p_r(x)$) and generated ($p_g(x)$) images. Both these distributions are modeled as multi-dimensional Gaussians parameterized by their respective mean and covariance. The distance measure is defined between the two Gaussian distributions as:

$$d^2((\mathbf{m}_r, \mathbf{C}_r), (\mathbf{m}_g, \mathbf{C}_g)) = \|\mathbf{m}_r - \mathbf{m}_g\|^2 + Tr(\mathbf{C}_r + \mathbf{C}_g - 2(\mathbf{C}_r\mathbf{C}_g)^{\frac{1}{2}}) \tag{7}$$

where $(\mathbf{m}_r, \mathbf{C}_r)$ and $(\mathbf{m}_g, \mathbf{C}_g)$ denote the mean and covariance of real and generated image distributions respectively (Shmelkov et al., 2018). We use a port of the official implementation of FID to PyTorch[2]. The default pool3 layer of the Inception-v3 network (Szegedy et al., 2016) is used to extract activations. We use 5k generated images to compute the FID scores, and sample 3 uniformly distributed points along each sampled interpolation path. To ensure domain fidelity of generated images, we use a binary classifier to distinguish whether the images come from the train distribution or the test set. Classifiers have the same architecture as other experiments, as described in Section A.1.

A.4 Noise Reconstruction

We analyze the ability of a trained AE to reconstruct uniform pixel noise of different frequencies. We simulate noise frequency by sampling noise at small resolutions and scaling up the resulting map to the desired image resolution (28×28 for handwritten characters, 64×64 or 128×128 for 3-channel color images). Fig. 11 plots the ability of trained AEs to reproduce noise patterns over given frequencies, averaged over 1000 trials. Networks have a clear low-frequency bias - though interestingly, the handwritten character datasets reach their minimum reconstruction error at a frequency level of 6-8, a possible manifestation of a learned bias toward penstroke thicknesses or certain-sized pockets of negative space associated with handwritten characters. Most tellingly, reconstruction error for novel images (dotted lines) is significantly lower than for noise of any frequency for handwritten character models, and most frequencies for natural image models. This suggests clearly that the network has learned a particular image distribution that is not reflected by uniform noise - an image prior.

It is also worth noting in what ways the AEs fail when reconstructing noise. Figs 12 and 13 show reconstruction attempts for random sampled noise at the given frequencies. It is clear that the handwritten character models struggle to abandon a strong, learned penstroke prior. Natural image networks are better at encoding noise, but also demonstrate a clear failure mode at high frequencies where they extract low-contrast, lower-frequency patterns and ignore the higher-frequency input. The Celeb-A model attempts to compensate for this by adding high-frequency ripple artifacts, visible also at some lower frequencies, probably reflecting a learned hair prior.

[2]https://github.com/mseitzer/pytorch-fid

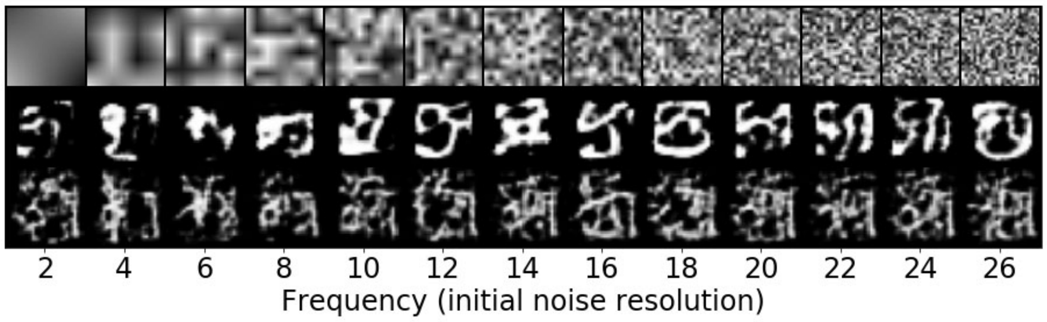

Figure 12: Attempted reconstructions of single-channel uniform noise. First row is noise, second is the MNIST model, third is the Omniglot model. Both models clearly struggle to abandon a learned penstroke prior.

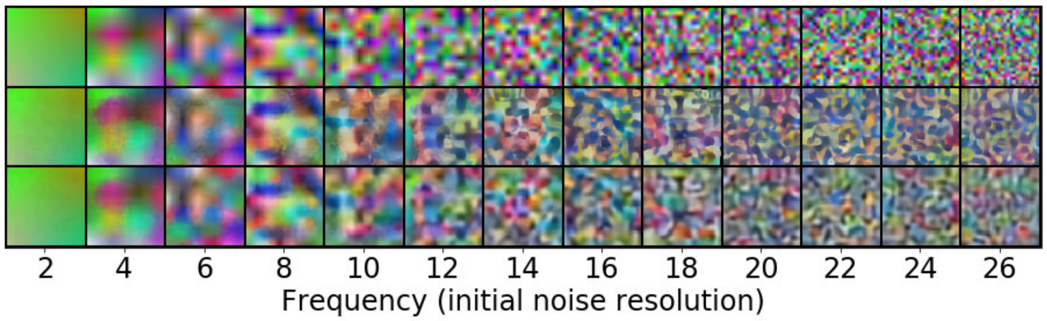

Figure 13: Attempted reconstructions of three-channel uniform noise. First row is noise, second is the Celeb-A model, third is the CIFAR-10 model. As noise frequency climbs, the networks progressively abandon the input signal and attempt to extract a lower-frequency pattern.

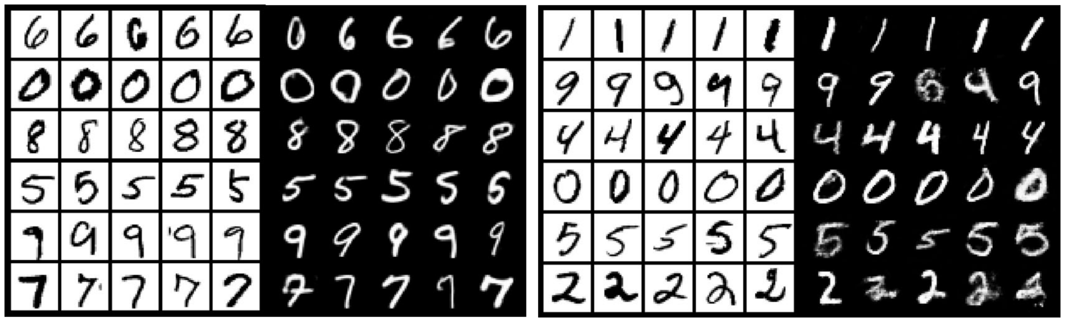

Figure 14: Black digits on white background are seed images, and white digits on black background are novel synthesized images. Neural statistician (left) and DAGAN (right) produce the desired behavior on training classes. Thus failures on novel classes clearly represent a failure to generalize.

A.5 Few-Shot Generation Baselines

We re-implement DAGAN using the architectures from our other experiments, with hyperparameters the same as for WGAN-GP training. The sole difference is that the critic network now takes in a pair of images, so the number of input channels is 2 instead of 1. We implement the critic network as a WGAN-GP. Fig. 14 shows that the network converges nicely on the training data: it successfully produces novel images from the same class as the initial seed images.

Neural statistician uses a much more complex network architecture, and involves many additional hyperparameters, making direct re-implementation difficult. Instead we run the publicly available code[3] as-is, keeping all hyperparameters the same. We run the omniglot experiment, and simply replace the omniglot training dataset with MNIST. Fig. 14 shows that like DAGAN, the network converges nicely and produces the desired behavior on the training data. The failure of these approaches on EMNIST is thus truly a failure of generalization.

A.6 Additional Illustrative Examples

Selected sets of interpolated image pairs from each of our four dataset regimes, demonstrating that AugIntAE performs smooth and intuitive interpolations where AEs and IntAEs produce artifacts. Images are organized as in the paper, with rows in each cell correspond to pixel fade, AE, IntAE, and AugIntAE. These are followed by four sets of randomly chosen pairs illustrating average-case performance, again one for each of our four dataset regimes. Begins on following page.

[3]https://github.com/conormdurkan/neural-statistician

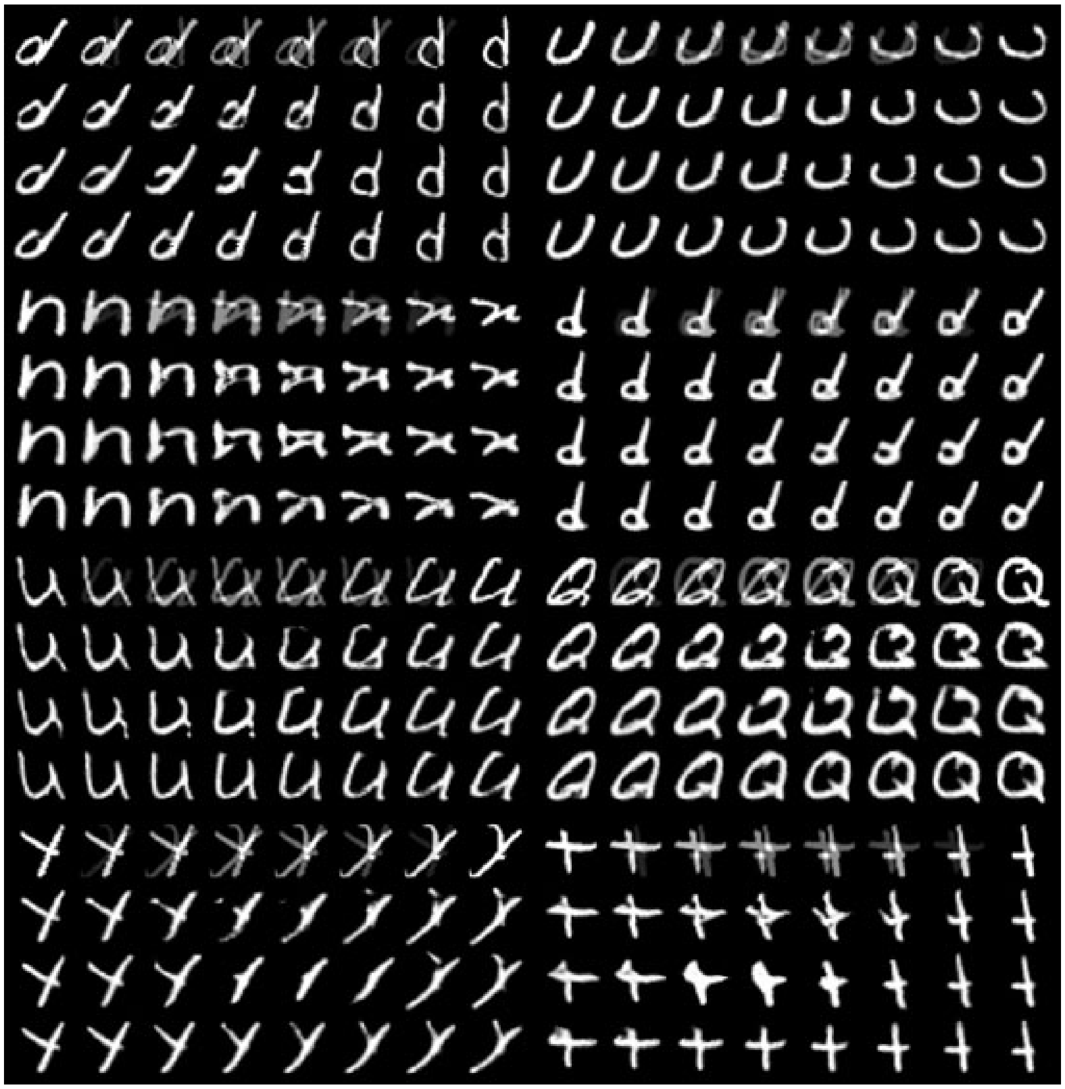

Figure 15: **Illustrative EMNIST pairs**. Each image pair contains artifacts in the AE/IntAE models that AugIntAE is able to avoid.

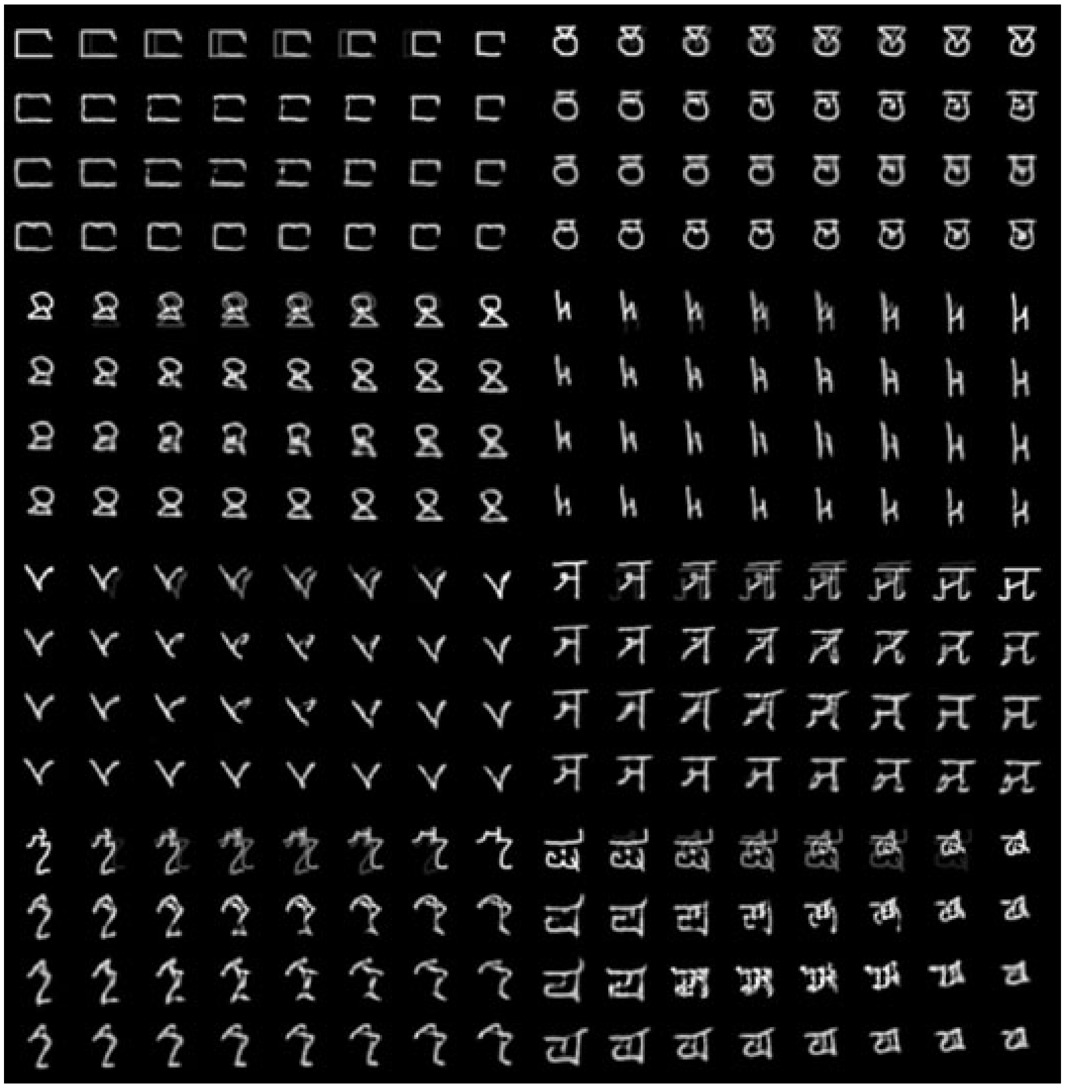

Figure 16: **Illustrative Omniglot pairs**. Each image pair contains artifacts in the AE/IntAE models that AugIntAE is able to avoid.

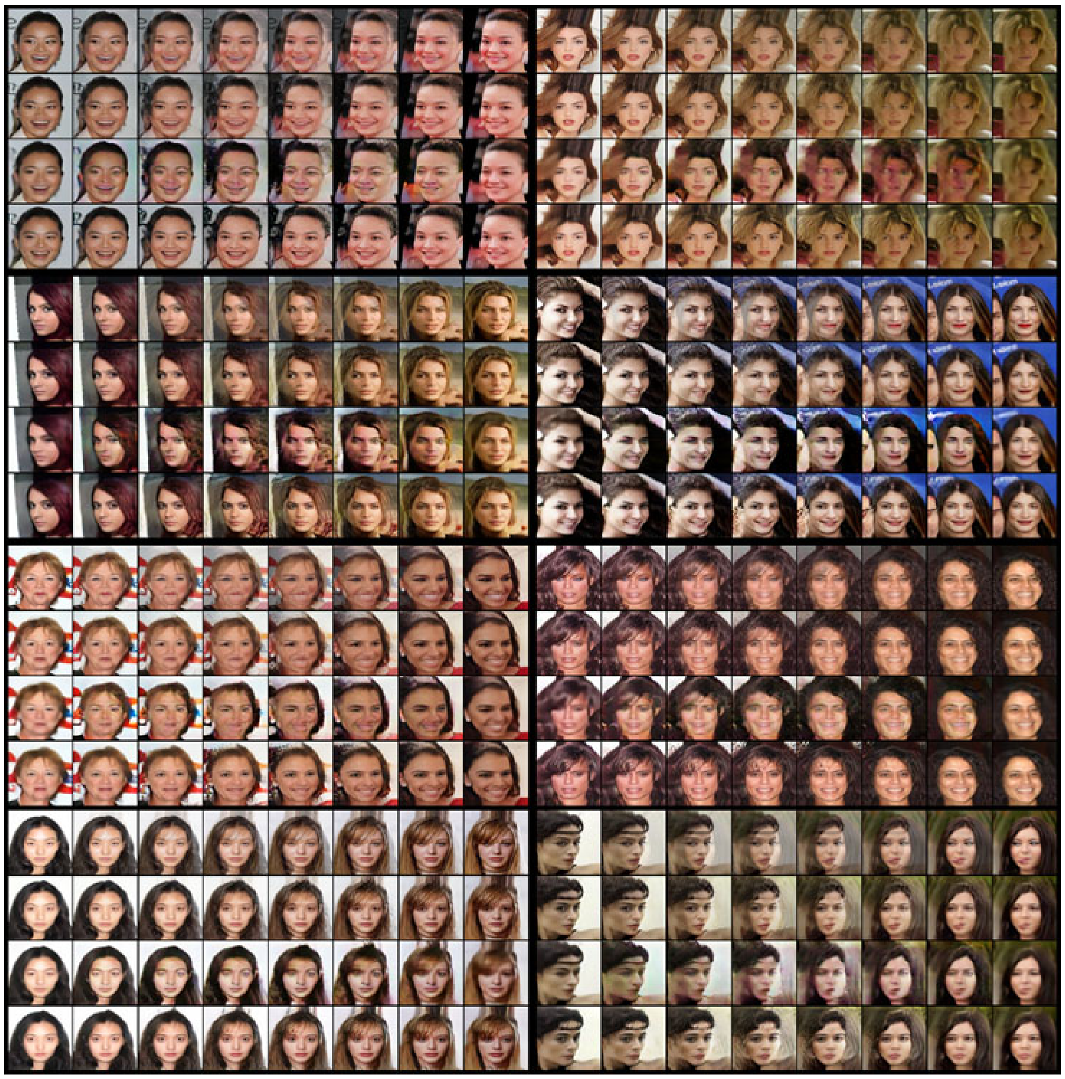

Figure 17: **Illustrative Celeb-A pairs**. Each image pair contains artifacts in the AE/IntAE models that AugIntAE is able to avoid. AEs generally produce transparency and silhouette artifacts around the hairline, mouth, and chin, while IntAEs produce nonsmooth interpolations, unrealistic head contours, and/or color artifacts. We note that AugIntAE is prone to smoothing away fine details, sometimes to a greater degree than AE.

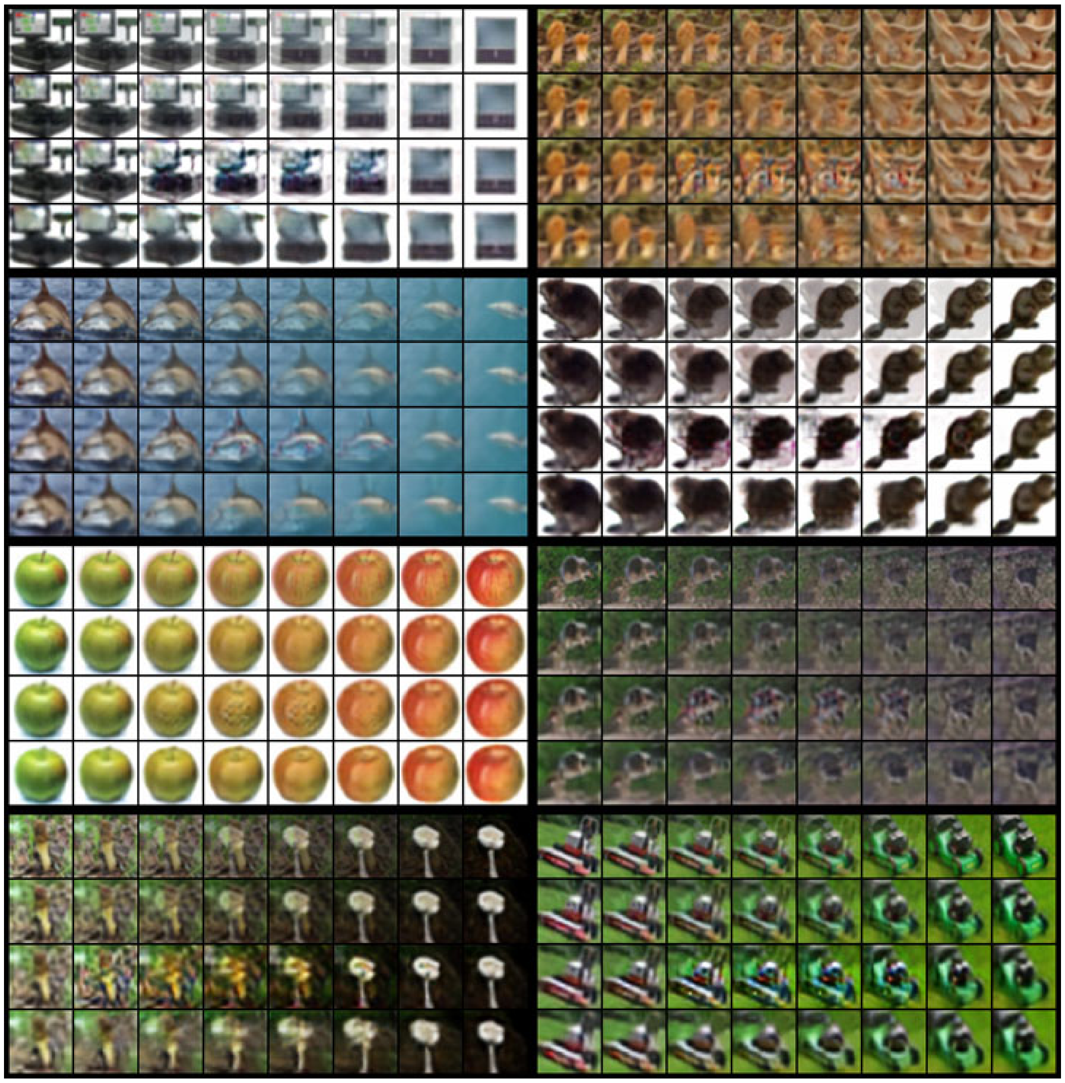

Figure 18: **Illustrative CIFAR-100 pairs**. AEs are largely indistinguishable from pixel fade, while IntAEs produce color and texture artifacts. AugIntAE does not always successfully preserve semantic content through the interpolation, but the interpolation at least usually makes sense. Of particular note are the apples on the left-hand side: rather than fade away the stem, the AugIntAE model retracts it into the apple! However, we again note that the more sensible AugIntAE interpolation comes at the cost of smoothing away fine details.

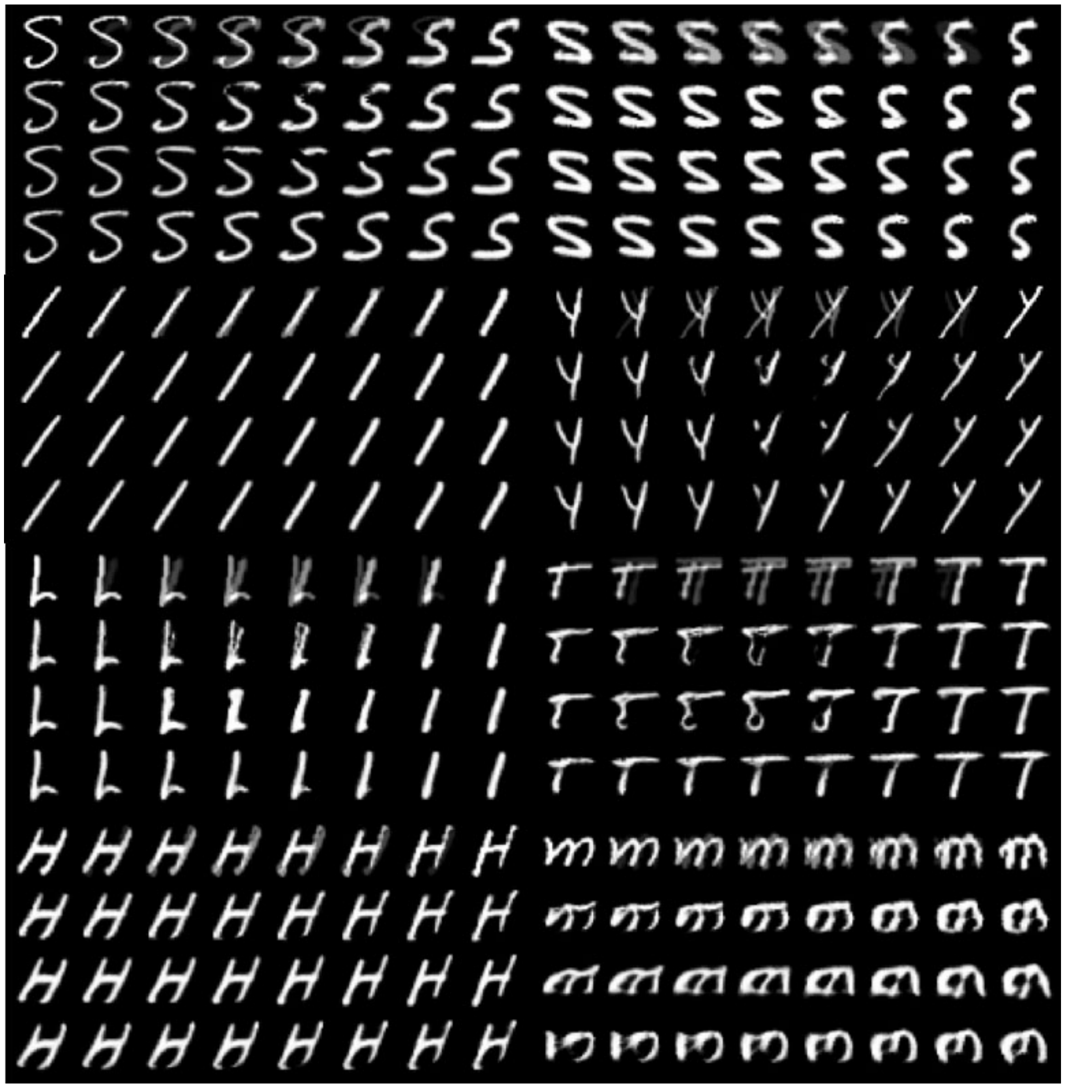

Figure 19: **Random EMNIST pairs**. AugIntAE is superior in some cases, and gives roughly equal performance in the remainder.

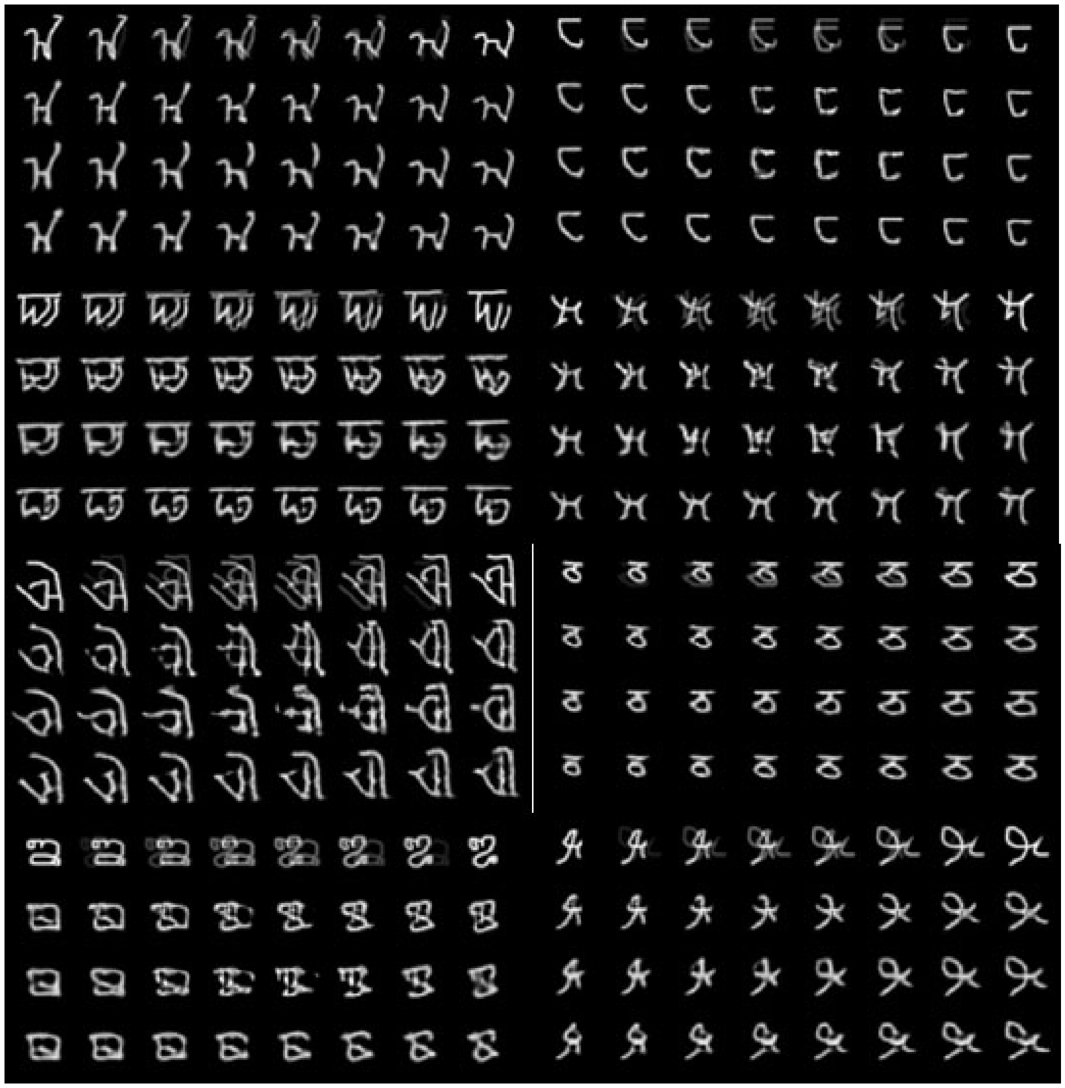

Figure 20: **Random Omniglot pairs**. AugIntAE successfully avoids many visual artifacts, but also tends to oversimplify complex shapes.

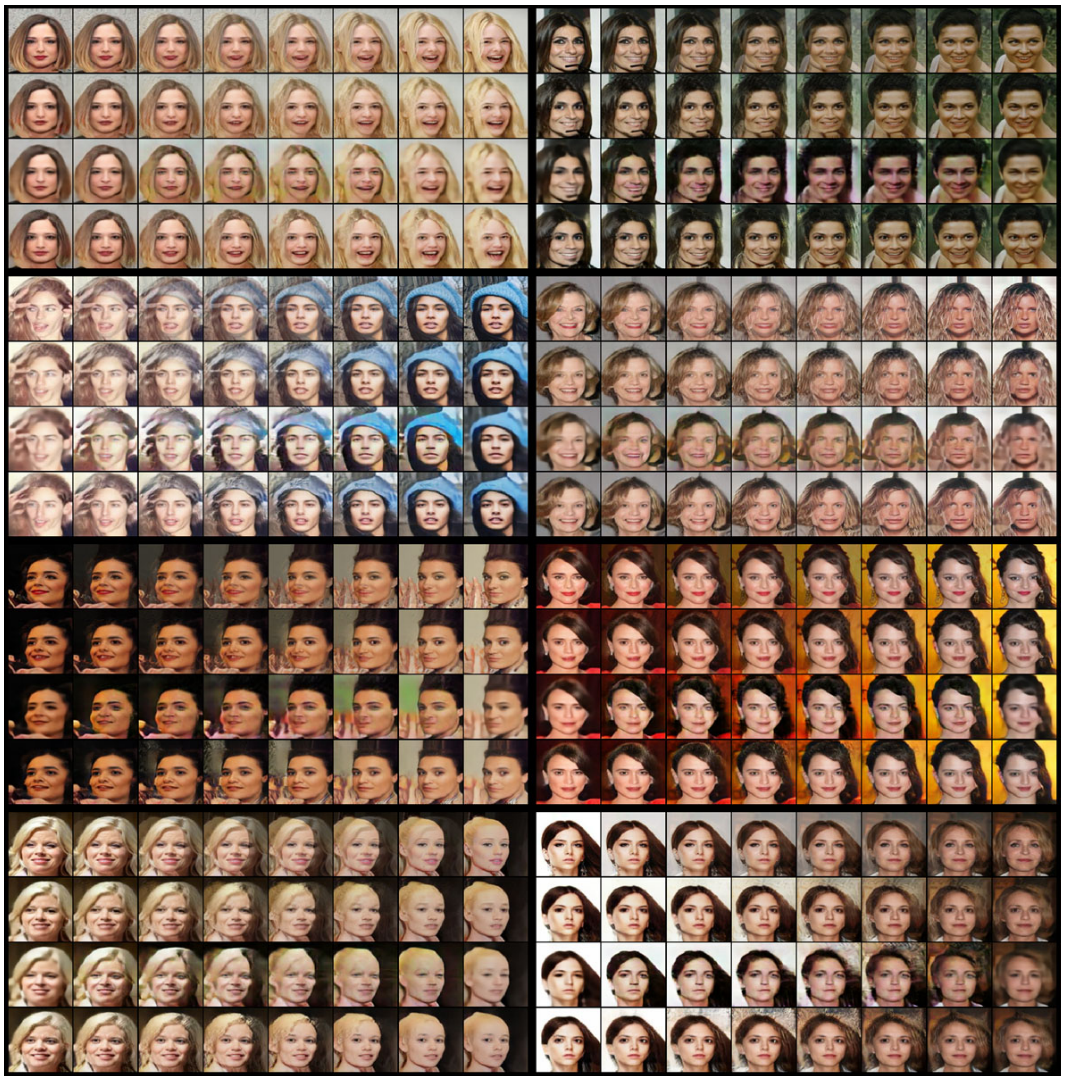

Figure 21: **Random Celeb-A pairs**. Pixel fade is actually a reasonably strong baseline on Celeb-A because of how the images are aligned, so in many cases the models all arrive at the same reasonably good interpolation.

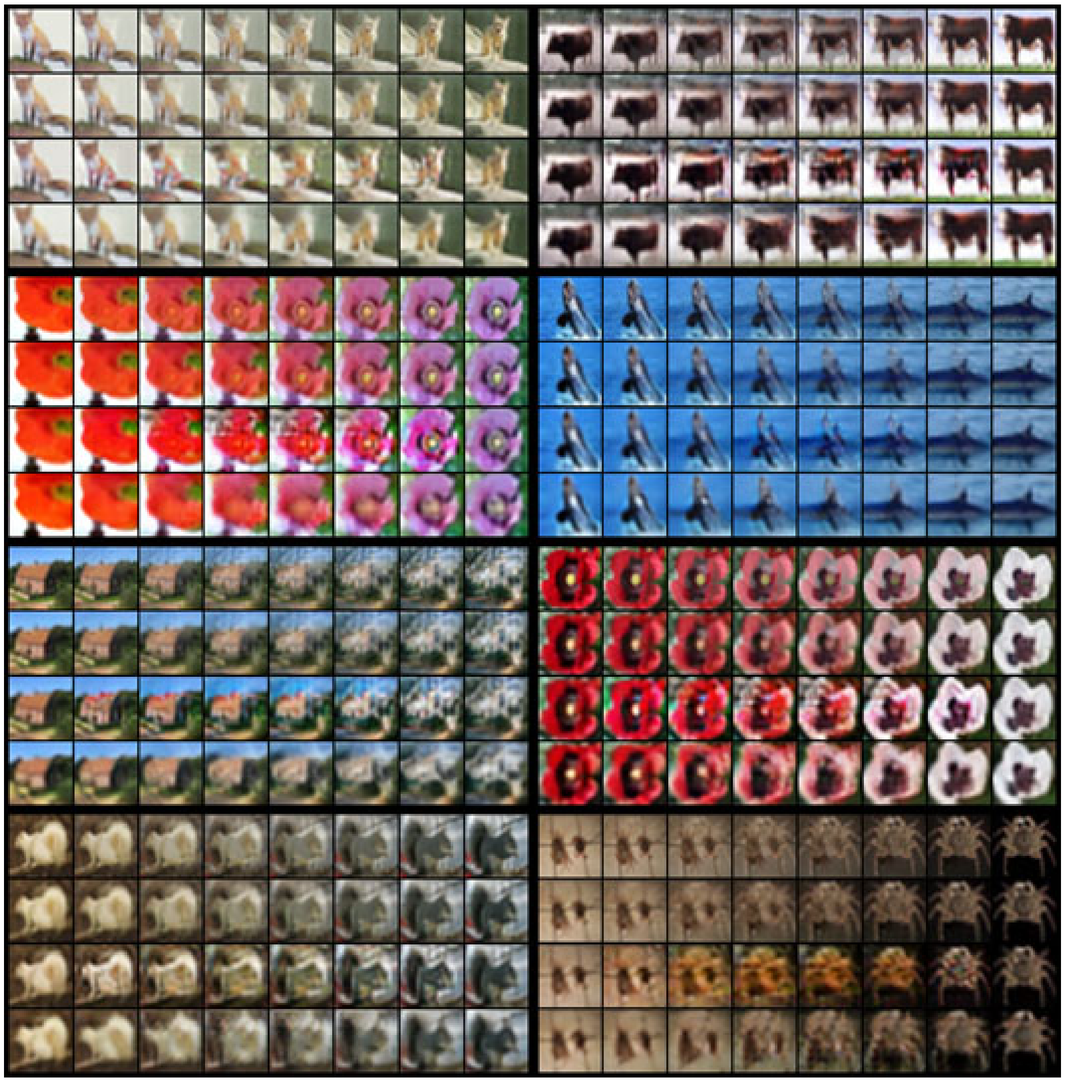

Figure 22: **Random CIFAR-100 pairs**. There is clear room for future work - in some cases the AugIntAE output is unacceptably blurry, and the AugIntAE interpolation frequently fails to preserve semantic content.