Generating Furry Face Art from Sketches using a GAN (Shitpost for 6.869, December 2019)

Andrew Yu
acyu@mit.edu

Abstract

I generate furry face artwork from color sketches. The sketches are procedurally generated from a data set of furry artwork. Sketches are translated back into artwork via a Generative Adversarial Network. I implement the GAN using a U-Net autoencoder with encoder-decoder skip connections and experiment with adding adaptive instance normalization into upsampling layers. The results show effective mapping of training and dev set sketches back to their input style. However, the model does not perform as effectively on novel user sketches and often fails to add stochastic textures like hair details.

1. Introduction

While there are highly successful prior works on generating animu waifus [15, 5], there have been no successful prior efforts on generating furries. This problem appears more difficult, as there is greater variation in the "structure" and "style" of furry content relative to animu (Figure 1). I refer to "structure" as the global features of a scene, such as shapes, geometry and colors. I refer to "style" as how features are filled, such as the shading and fine details. In this metric, animu and furry would refer to a specific mix of structure and style.

When generating artwork, we want control of both structure and style. Generating from user inputted sketches provides control over the global structure and content. Style can either be deduced from an input image or tuned from a generative model. For this work I generate furry faces from rough line and color sketches using a generative adversarial network (GAN) to get a specific stylistic output [4, 8].

GANs are increasingly powerful at creating both photorealistic and highly varied human faces, objects, and environments [4, 1, 10, 11]. However their problem from a user perspective is that the input is a poorly understood and arbitrary "latent space". Typically GAN inputs are a random noise vector sampled from this latent space.

Replacing the latent space with user inputted images turns the problem into image translation, which is a mix of generation and style transfer. Traditional image style transfer infers the style of one image then transfers it onto the content inferred in another image [3, 7]. For image translation, the model is simply a map between input and output spaces. However, this makes it challenging to generate multiple, varied style outputs from the same input [8]. This work explores different techniques in upsampling layers to tune image generation.

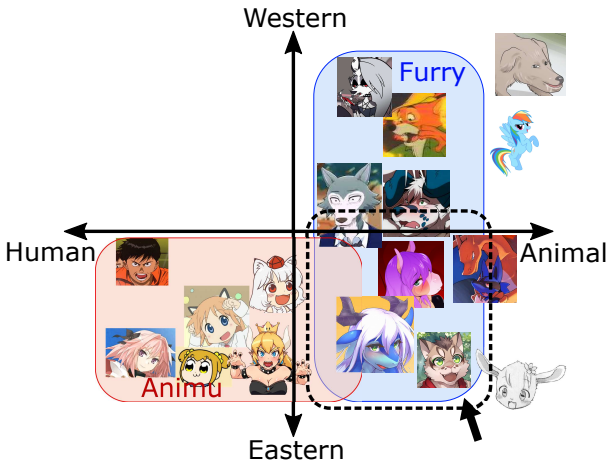

Figure 1: Animu and furry subspaces of artwork projected on Eastern-Western style and Human-Animal content axes. Dashed box indicates the style-content subspace of the data set used in this work.

2. Data Set

I scraped 17K raw images from the image gallery site `e621.net` from a curated subset of artists with a consistent style as indicated in Figure 1. From these, I hand-selected and labeled faces to form a 4K image data set for training.

2.1. Sketch Generation

The art "sketch" for the GAN input is to consist of simple black outlines with flat color fills. I choose this representa-

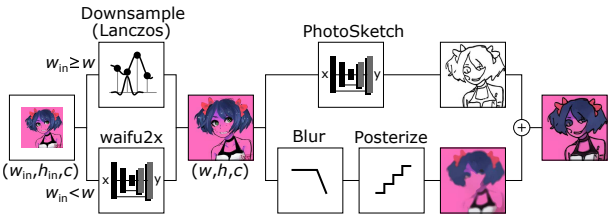

Figure 2: Sketch generation pipeline. PhotoSketch [12] and waifu2x [14] are neural net models.

tion because it's a common flow for artists to first segment the structure of their scene with lines, then fill with basic colors. Figure 2 depicts the procedure to generate sketches from raw artwork.

The input is a square crop with size (w_{in}, h_{in}, c) where $w_{in} = h_{in}$ are dimensions and c are color channels. If the crop is not equal to the target model input size (w, h, c), different techniques are used for downscaling and upscaling. Downscaling uses a conventional Lanczos downsampler. For upscaling, I use waifu2x, a super-resolution neural network trained for animu-style art [14]. For most cartoon art, waifu2x is far better at preserving stylistic content and avoiding artifacts of conventional upsampling techniques (bilinear, bicubic).

To generate a simplified color structure of the input image, I apply a blur followed by a mean shift posterization filter. The blur removes the solid black outlines common in cartoon-style artwork. A naive posterization is to uniformly quantize the color space and map colors to their nearest threshold level. The problem with this technique is that it mismatches the sketch color with the ground truth color. A better technique is mean shift filtering, which maps colors to local cluster averages [2]. This provides much closer mapping to the ground truth color.

Black outlines are generated using a pre-trained PhotoSketch, a GAN that generates sketch-style black outlines from an input image [12]. While PhotoSketch was trained on photos, it works sufficiently well on artwork.

I finish generating the sketch by overlaying the outlines on top of the simplified color structure.

3. Network Architecture

The baseline generator architecture is a U-Net (Figure 3) which uses skip connections between autoencoder layers. The overall baseline image translation GAN architecture is a pix2pix architecture with 512 autoencoder latent space features, which I will simply refer to as "pix2pix" [9]. I use the standard conditional GAN loss functions with

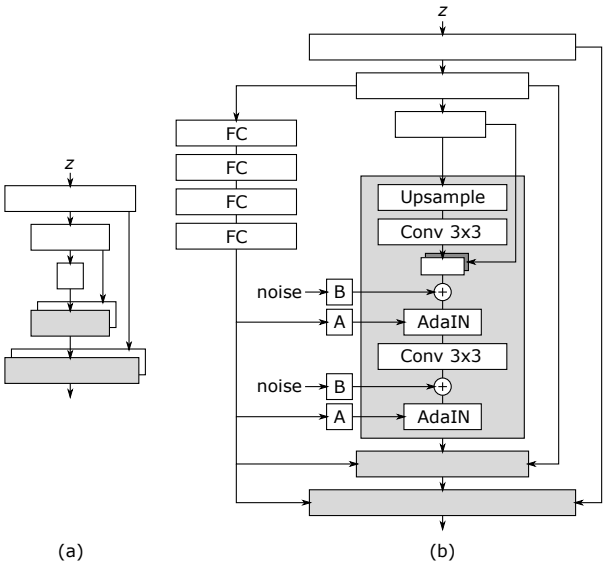

Figure 3: Network architectures. (a) Conventional U-Net with skips. (b) U-Net with AdaIN upsampling blocks and skips.

Method	Parameters	FID
pix2pix	57.2M	122.3
style-pix2pix	49.1M	133.2

Table 1: Fréchet Inception Distance (FID, lower is better) of two models. Note this metric is weak because the data set and output samples size are small.

L1 regularization [9],

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}\left[\log D(x, y)\right] + \mathbb{E}_{x,z}\left[\log(1 - D(x, G(x, z)))\right] \quad (1)$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}\left[||y - G(x, z)||_1\right] \quad (2)$$

for generator G, discriminator D, real image x, input sketch z, and generated y. The total loss is $\mathcal{L} = \mathcal{L}_{cGAN} + \lambda\mathcal{L}_{L1}$ where λ is the weight of L1 regularization.

Karras et al. adopt Adaptive Instance Normalization (AdaIN) in their GAN upsampling layers [11], a technique first used in feed-forward style transfer by Huang et al. [7] AdaIN is defined as:

$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{\mathbf{s},\mathbf{i}}\frac{\mathbf{x}_i - \mu(\mathbf{x}_\mathbf{i})}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i} \quad (3)$$

This is an instance normalization but the affine parameters $\mathbf{y} = (\mathbf{y}_s, \mathbf{y}_b)$ are mapped from the latent space through a series of fully connected layers. This is supposed to let the input modulate the "style" of each feature activation vector

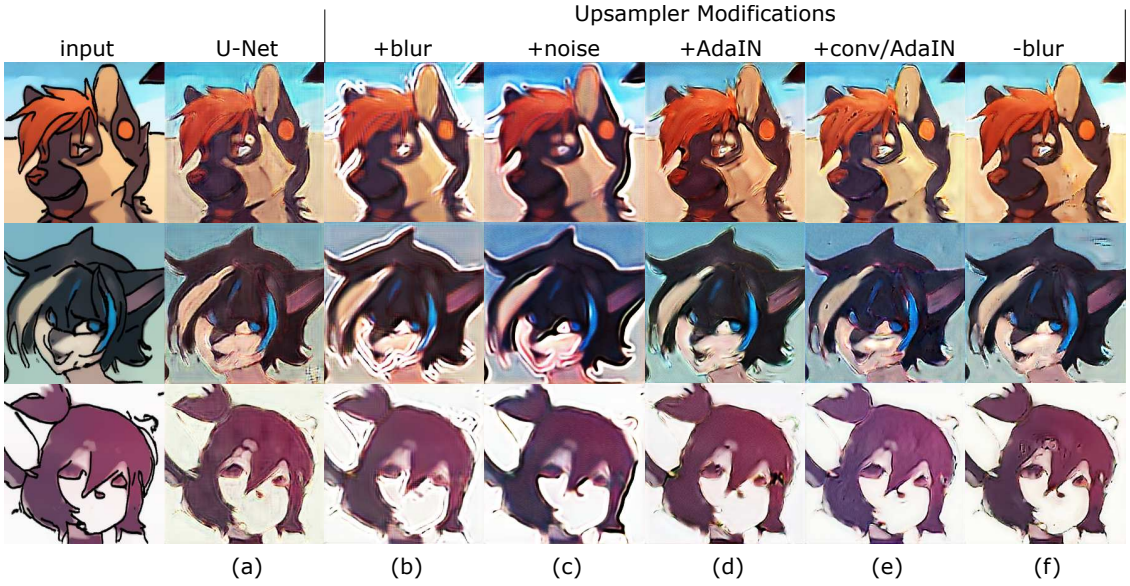

Figure 4: Effects of adding in different upsampler techniques in early training stages (∼0.04M images). (a) is a U-Net baseline. (b)–(f) sequentially add new techniques to the upsampler. Images are from the dev set.

$\mathbf{x}_i$ through these AdaIN blocks [7]. AdaIN replaces batch norm, pixel norm, or pure instance norm.

As shown in Figure 3, I replace the normal U-Net upsampling blocks with those used in StyleGAN. The pix2pix model uses batch norms while these new layers use AdaIN instead. The 'A' blocks in Figure 3 are fully connected layers that generate separate $(\mathbf{y}_s, \mathbf{y}_b)$ for each AdaIN.

I continue to use skips which concatenate downsampler activations with the upsampled activations. I find that the skips help seed the output with global color and structure based on the input image. This creates the global output image structure without using multi-scale approaches like progressively growing the generated output [10].

Additionally, the upsampling uses bilinear image resizing instead of a transposed convolution. I apply anti-aliasing blurs to feature activations after concatenating the skip connection inputs [16]. I inject noise into the post-concatenated features with a learned weighting 'B'. According to Karras et al. the noise improves finely detailed stochastic texture generation [11].

I refer to this architecture as "style-pix2pix." This architecture is quite similar to that used by Huang et al. in [8]. However, they use a separate downsampling network to feed the fully connected layers that map to AdaIN parameters, whereas I re-use the same downsampler and pull features from one of the middle downsampler layers.

Both pix2pix and style-pix2pix use the same convolutional Markovian discriminator described in [9], which detects if local patches are real or fake.

Table 1 gives the number of trainable model parameters and the commonly used Fréchet Inception Distance (FID) metric of the generated results [6] after training for ∼0.8M images. However, FID is a poor metric on small few thousand image sample sizes due to variance in the data set. Relative values show the style-pix2pix is performing slightly worse, which will be discussed in Section 4

3.1. Training Details

The total data set contains ∼4.2K images, which are shuffled into a ∼4.1K image train set and ∼100 image dev set. I apply random horizontal left-right image flips during training. Test set cases are personal hand drawn images. All image inputs and outputs are sized 256x256.

I implement both models in Figure 3 using Tensorflow 2.0. I use a minibatch size of 16. Both the generator and discriminator use an Adam optimizer with learning rate of 0.0002, $\beta_1 = 0.6$, and $\beta_2 = 0.99$. Layer weights are all initialized using a random normal $\mathcal{N}(0, 1)$. I did not use a training scheduler. I train using a Nvidia GTX 1060.

4. Results and Discussion

To diagnose how the output is being generated, Figure 4 shows the effects of different upsampler techniques during the early stages of training (after ∼0.04M images). Blurring upsampled images separately in each activation channel (b) creates smoother texture but heavily distorts lines. Additive noise (c) is initialized with 0 weight (i.e. no noise), so the impact is not seen well in early training. Adding the first AdaIN (d) corrects the distortions from blurred acti-

Figure 5: Generation versus total training images.

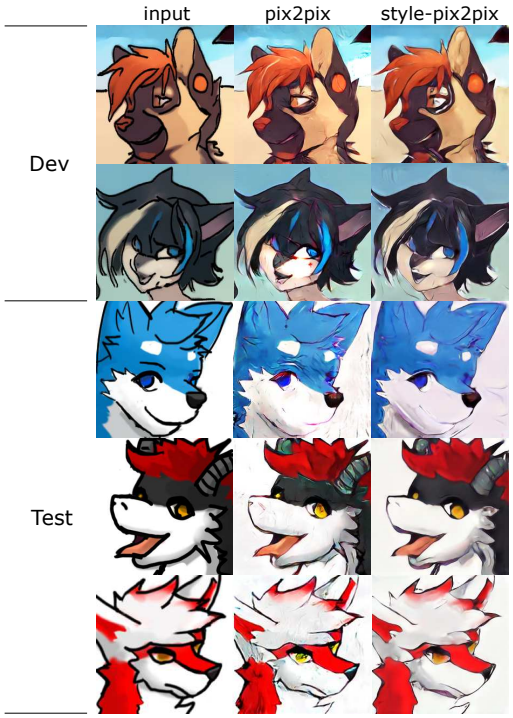

Figure 6: Results on dev and test examples after training for ∼0.8M images

vations and appears to tune the colors generated. Adding the second conv/AdaIN block (f) appears to add more texture variation. Removing the activation blurring in (b) but keeping the other elements in (c)-(e) appears to cause some additional rough patches (f).

I proceeded to train style-pix2pix model with techniques in Figure 4e. Figure 5 compares the generation over ∼0.8M training images between baseline pix2pix and style-pix2pix. The style-pix2pix appears to initially generate strong texture noisiness as seen in the hair, but eventually loses this. By the end of training the textures are highly smoothed. In contrast, the pix2pix is able to add in some

Figure 7: Failure cases, typically certain colors and samples with little color variation. Solid color inputs on the right show artifacts.

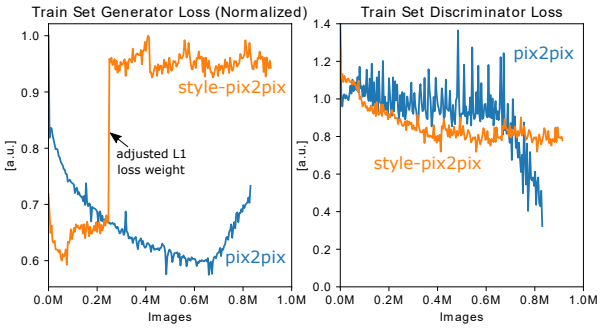

Figure 8: Training loss, showing pix2pix mode collapse.

new texture as seen in the hair. However, both results are still relatively smooth. A better discriminator or loss function could help improve texture generation.

Figure 6 shows results after ∼0.8M training images on dev and test set examples. As shown before, the style-pix2pix resulted in smoother texture outputs. Two issues with test set results are lack of texture generation and the model appears quite sensitive to thickness of the black lines. This is likely because PhotoSketch outputs the same thickness black lines for each image, and the model may overfit on these exact lines. As can be seen in the test set examples in Figure 6, using slightly thicker black lines causes thicker and often rougher outlines in the output. And the blue wolf's mouth disappeared. So in the future it may be better to simply train on only colors and let the model infer outlines.

4.1. Failure Modes

The worst dev set results are shown in Figure 7. Certain colors (especially darker tones) and images with little color variation tend to produce garbage results, especially patches of sharp, bright colors. Testing with a solid color input shows texture artifacts at the edges. The style-pix2pix

model did not generate these patches. Instead style-pix2pix suffers from overly smooth textures.

Figure 8 shows the training loss for both models. The sharp decrease in discriminator loss for the pix2pix model suggests it is suffering from mode collapse and overfitting on certain features. The style-pix2pix model did not collapse, but instead had difficulty fitting to the output, with the loss functions simply oscillating.

This work only tunes the generator and shows that reasonably varied outputs can be generated. However, the failure cases, overly smooth colors, and lack of variance suggest the loss function and discriminator need to be greatly improved to get results with proper texture generation [13].

5. Conclusion

I have introduced a new high quality hand-labelled 4K image data set for furry face artwork, developed a pipeline for sketch generation from artwork, and implemented a GAN model for generating art from sketches. The conventional pix2pix GAN results are promising but suffer from the usual problems of mode collapse, difficult training, and lack of texture variation. A modified GAN with AdaIN blocks in the upsampler did not suffer from mode collapse but instead had issues converging and the resulting textures are too smooth. A selection of dev set results for both models are included in Figure 9 and 10 at the end of this paper.

A major issue with this current sketch approach is that the results tend to overfit on the sketch's input black lines. As a result, for the output to look good, the input should already look good. An improvement may be to just remove the black lines in the sketches and generate based solely on color, letting the model infer the outlines. Further experiments in decoupling the output could include removing the skip connections or adding a separate latent space input noise vector for generating AdaIN style parameters.

6. Resources

The figures and code used for image processing and neural net training are included in supplementary materials file `acyu.zip`. The code is setup with local paths and requires external third party software (waifu2x and PhotoSketch), so it cannot be easily run and is included only as reference.

References

[1] A. Brock, J. Donahue, and K. Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *ICLR*, 2019.

[2] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[3] L. A. Gatys, A. S. Ecker, and M. Bethge. Image Style Transfer Using Convolutional Neural Networks. In *CVPR*, pages 2414–2423, 2016.

[4] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[5] gwern. Making Anime Faces with StyleGAN, 2019. https://www.gwern.net/Faces.

[6] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *NIPS*, pages 6627–6638, 2017.

[7] X. Huang and S. Belongie. Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. In *ICCV*, pages 1510–1519, 2017.

[8] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz. Multimodal Unsupervised Image-to-Image Translation. In *ECCV*, volume 11207, pages 179–196. Springer International Publishing, 2018.

[9] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-Image Translation with Conditional Adversarial Networks. In *CVPR*, pages 5967–5976, 2017.

[10] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *ICLR*, 2018.

[11] T. Karras, S. Laine, and T. Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *CVPR*, 2019.

[12] M. Li, Z. Lin, R. Mech, E. Yumer, and D. Ramanan. Photo-Sketching: Inferring Contour Drawings From Images. In *WACV*, pages 1403–1412, 2019.

[13] L. Mescheder, A. Geiger, and S. Nowozin. Which Training Methods for GANs do actually Converge? In *ICML*, 2018.

[14] nagadomi. waifu2x, 2018. https://github.com/nagadomi/waifu2x.

[15] M. Sugimura. Fgo StyleGAN: This Heroic Spirit Doesn't Exist. https://towardsdatascience.com/fgo-stylegan-this-heroic-spirit-doesnt-exist-23d62fbb680e.

[16] R. Zhang. Making Convolutional Networks Shift-Invariant Again. In *ICML*, 2019.

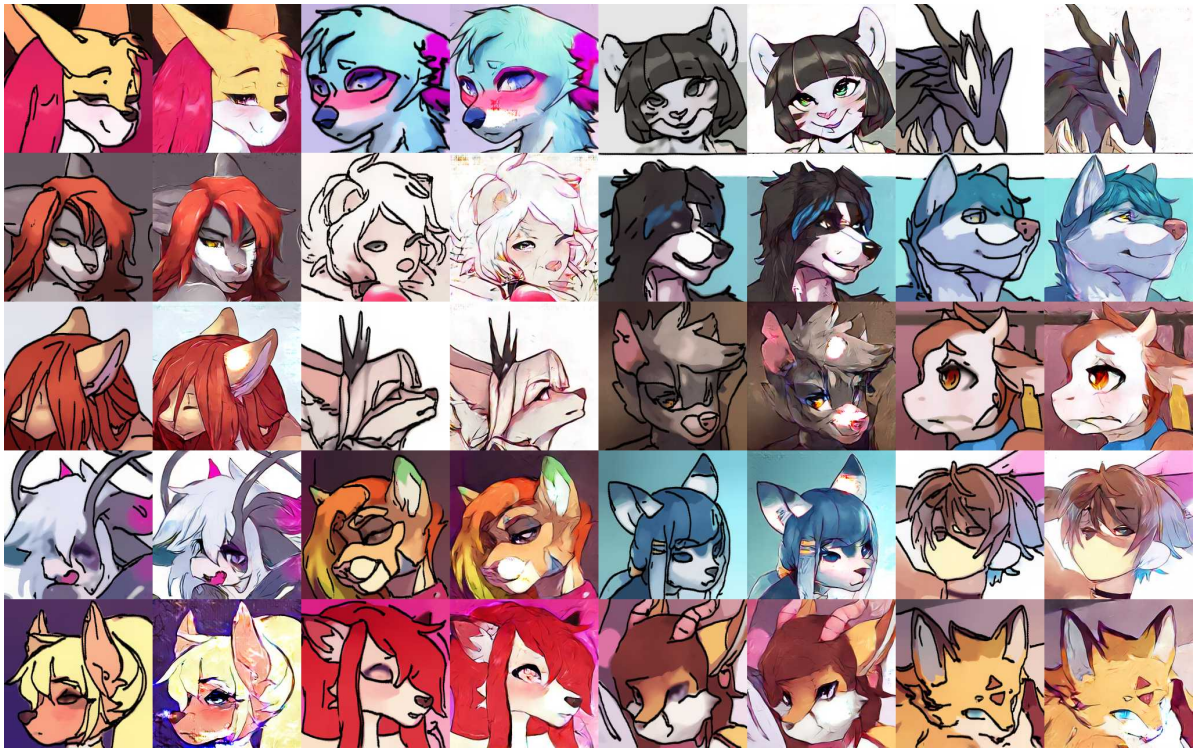

Figure 9: Dev set results using pix2pix after ∼0.8M training images

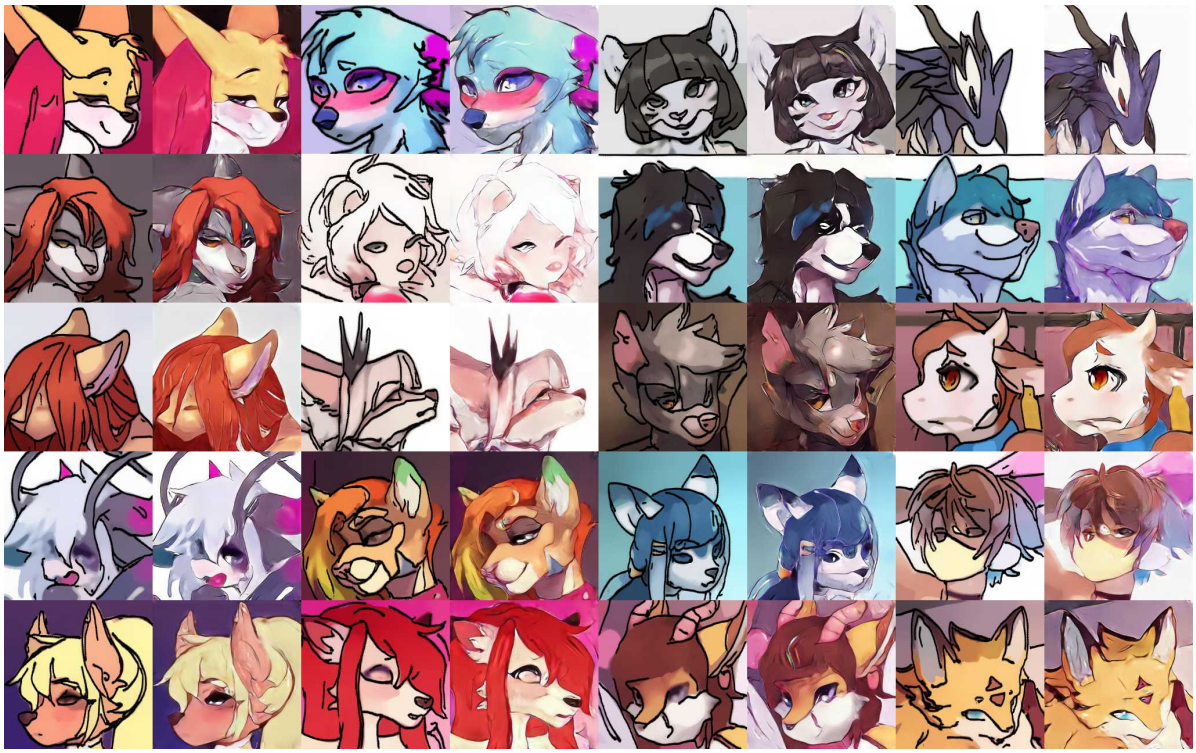

Figure 10: Dev set results using style-pix2pix after ∼0.8M training images