An Analysis of Image-to-Image Translation with Conditional Adversarial Networks

CSCI 4622 - Machine Learning Final Project

https://github.com/clbeggs/Final_Project

Chris Beggs *
christopher.beggs@colorado.edu

Soroush Khadem*
soroush.khadem@colorado.edu

Greg Lund*
greg.lund@colorado.edu

Connor Thompson*
connor.thompson-1@colorado.edu

1 Motivation

Generative Adversarial Networks (GANs) have been a hot topic of research over the past six years, and have seemingly endless applications. GANs have been used in a wide range of applications from generating photo-realistic human faces [1] to generating feasible robot trajectories for path planning [2]. As students and machine learning enthusiasts, we wanted to learn more about GANs and how they work and most importantly get some hands-on experience working with them. In order to achieve this goal, we will investigate these exciting generative models through a literature review, implementation of a simple GAN, and through actual hands-on training of state of the art models.

1.1 Overview of GANs

Generative Adversarial Networks (GANs) fall into the class of unsupervised, generative models. The term adversarial is used to describe the general structure of the model, where one model generates novel data with the aim of fooling a separate discriminative model. In the case of GANs both the generator and the discriminator are implemented as neural networks. The generator produces new data, which in the simplest case comes as a result of random inputs to the network through some noise distribution [3]. This data is then fed into the discriminator which makes a binary classification of the input on whether it is a real example or a fake example from the generator. During training the generator tries to fool the discriminator in what is essentially a two-player minimax game. As both models are trained via backpropagation and stochastic gradient descent (or similar optimizations such as Adam), if both models were updated at the same time there would be no sense of stationarity and the training would be very unstable. To remedy this, the models are trained in batches where either the generator or the discriminator (but not both) is updated in a given batch. A combination of generated and real data is used to train the discriminator.

Training proceeds in this way, with both the generator and discriminator iteratively achieving better and better results. Output from the generator that is deemed fake by the discriminator gets added to the training set for negative examples. In an ideal GAN, we should see our discriminators accuracy slowly decrease before plateauing around 50% symbolizing that the discriminator is tossing a coin to determine if the data presented to it is real or fake. While this is the basic training outline for GANs, many small adaptions can be made to make learning more effective (such as including skip connections, training the discriminator at a slower rate than the generator, using ADAM, etc).

*University of Colorado, Boulder

2 Methods

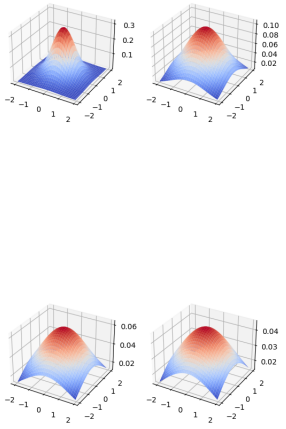
Aside from literature review to learn more about the fundamentals of GANs, we focused on two main objectives that could be used to analyze and understand GANs at a deeper level. The first of these objectives was to build a simple GAN from scratch, where 2D and 3D data was generated. Secondly, we trained two state of the art GAN architectures used in image-to-image translation. For each case we visualized training progress which revealed interesting insight onto the training process. We also qualitatively and quantitatively compared the two methods on the same set of images.

2.1 Initial Exploration

As an initial exploration into GANs we decided to explore two very basic and small architectures to get intuitive and interpretable results. The two architectures used were a fully connected GAN and a deep convolutional GAN.

2.1.1 Data

To maintain simplicity, both GAN architectures were trained on surfaces in $\mathbb{R}^3$. An example of the training data is as shown:



The surfaces are Normal distribution densities parameterized by:

$$\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix} \qquad \text{such that } \alpha \in \mathbb{N}$$

Because of the different architectures used, the data was formatted in 3 different ways:

1. A single surface point cloud: $N \times 3$, where each seperate point is the input
2. Mutliple surfaces: $S \times N \times 3$, where the entire point cloud is an input
3. Multiple surfaces that are meshgrids of points: $S \times 3 \times N \times N$, where the dimension 3 is synonymous to 3 RGB channels in an image.

The meshgrids were used for DCGAN, so that the training data could be in the same format as images: $N \times C \times H \times W$, as using 1D convolutional layers on $N \times 3$ point clouds may change the quality of generated data.

2.1.2 Architectures

FCGAN:

- Generator Architecture: Fully Connected, 2 hidden layers with sizes (128, 128), LeakyReLU activations, final Tanh output

- Discriminator Architecture: Fully Connected, 2 hidden layers with sizes (256, 256), LeakyReLU activations, final Sigmoid output

DCGAN

- Generator Architecture: Transpose convolution layers increasing in size, stride of 2, final Tanh
- Discriminator Architecture: Convolution layers decreasing in size, stride 2, final Sigmoid output

2.1.3 Training

For both models, Binary Cross Entropy with Logits was used as a loss function:

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$

and the Adam optimizer was used for both the generator and discriminator. Both models were trained on Nvidia GeForce GTX 1050 Ti Mobile. Because the models were trained on a laptop, the size of the dataset and networks had to be kept at a minimum.

FCGAN:

FCGAN was very unstable to train, even with various stability tricks implemented. We found that the following worked best with both model architectures:

- Training discriminator more
- Adding noise to discriminator inputs
- Label smoothing
- Larger learning rate for discriminator

With these, we were able to get reproducible results:

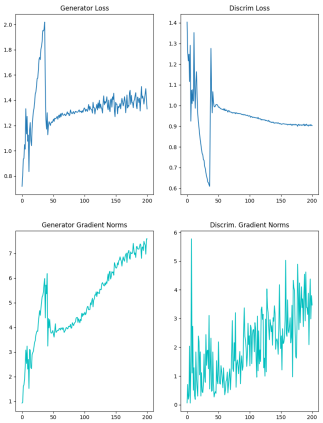


Figure 1: FCGAN training results. **Top Left:** Generator Loss, **Top Right:** Discriminator Loss, **Bottom Left:** Generator Gradient norms, **Bottom Right:** Discriminator Gradient norms

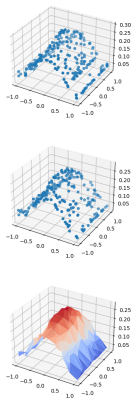


Figure 2: FCGAN generated surfaces

DCGAN: The Deep Convolutional GAN was more stable to train, however still remained a bit finnicky. The same stability tricks used on FCGAN were used to train DCGAN, with the exception of training the discriminator more.

It was able to get good results:

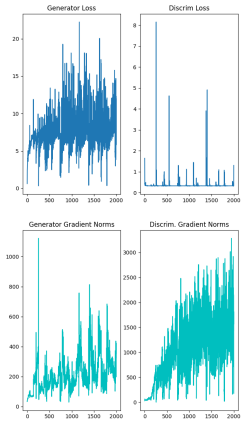


Figure 3: DCGAN training results. **Top Left:** Generator Loss, **Top Right:** Discriminator Loss, **Bottom Left:** Generator Gradient norms, **Bottom Right:** Discriminator Gradient norms

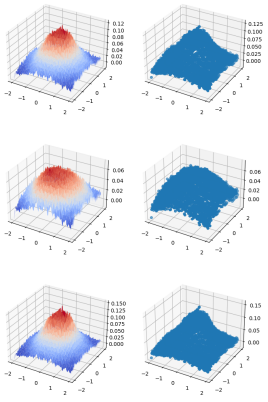


Figure 4: DCGAN generated surfaces

2.2 GANs for Image-to-Image Translation

Alongside the implementation of a GAN from scratch we thought that it would be worthwhile to train multiple state of the art models on our own data to understand the process of applying GANs to more complex tasks. For this section of the project we decided to focus on image-to-image translation, the problem of developing a mapping between different styles or representations of image data. The two models that we investigated were Pix2Pix and CycleGAN [4, 5]. Both models were trained on similar data and were qualitatively and quantitatively compared.

2.2.1 Data

Although Pix2Pix and CycleGAN are fundamentally different models, with differing strengths, for the most part we used the same data to train each. The dataset used was the Sketchy Database [6], which provides hand sketched images of various animals and objects paired with real photos used as inspiration for those drawings. For training Pix2Pix drawing photo pairs were combined for training while for CycleGAN there was no paired relationship between drawings and photos, so all sketches and photos were shuffled. In addition, as a means to explore training on more difficult datasets, we hand crafted a new dataset for unpaired images for trees. In order to do so, sketches were taken from an open source sketch dataset from the Google Quick Draw game [7]. There were two versions of this dataset tested. Version one (Quickdraw v1) used sketches from Quick Draw, and images from sketchy. This resulted in bad results (see Fig. 10). The most likely reason is that the types of trees drawn were quite different than the images of trees. This lead us to experiment with a new source for images, which led to version two (Quickdraw v2). The images for this dataset were pulled from ImageNet [8], a dataset of millions of images belonging to thousands of classes. The `oak` class was used, since that type of tree qualitatively looks very similar to the sketches for the `tree` class of sketch. See Fig. 5.

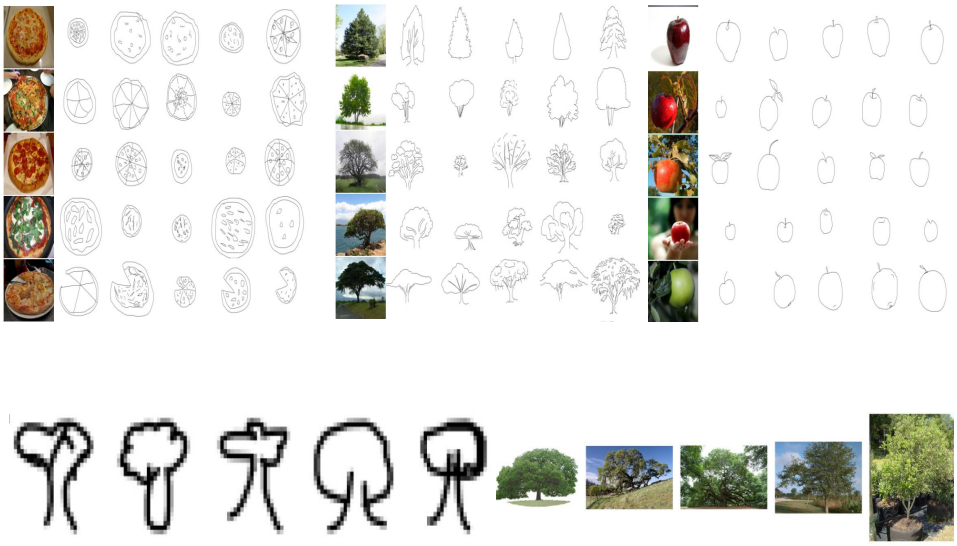


Figure 5: Samples taken from the datasets used. **Top:** paired images from the sketchy database [6] for pizza, tree, and apple respectively. **Bottom:** Sketches from Google Quick Draw dataset [7] and images from ImageNet [8]

2.2.2 Pix2Pix

Pix2Pix is a Conditional Generative Adversarial Network, meaning that its output is conditioned on its input in a more meaningful way than with standard GANs. Instead of feeding the GAN random inputs in order to generate a random output in the desired feature space, we feed Pix2Pix an image of one style (drawing, segmentation map, grayscale etc.) and the output is conditioned on the input resulting in an image sharing defining features with the input but in a new style (color photo). The generator in Pix2Pix uses an encoder-decoder model, where the input image is compressed via many convolutional layers into a bottleneck as can be seen in figure 6. This bottleneck can be thought of housing the latent space of the generator. After this bottleneck, many transposed convolutional layers are used, in increasing size until the size of the output image. In reality Pix2Pix uses a slightly more complicated encoder-decoder model where skip connections are made between layers i and $n - i$ for $i = 1, ..., \frac{n}{2}$. This architecture is known as U-Net and allows for low-level information transfer across the entire network.

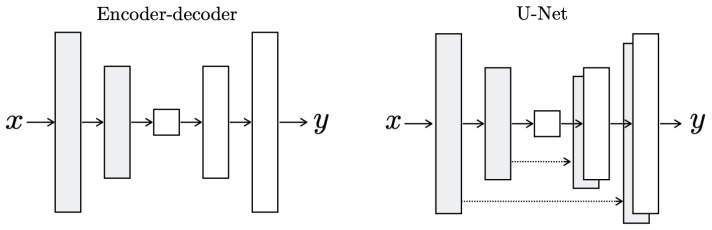


Figure 6: A standard encoder-decoder model alongside the U-Net architecture used for Pix2Pix. Taken from Isola *et al.* [4]

The implementation of CycleGAN and Pix2Pix comes directly from the original authors and is open source [9]. The models are created in PyTorch, and it was very straightforward to prepare our data for training. Three classes from the Sketchy Database were selected: apples, pizza and trees. 100 drawing photo pairs were taken for each of these three classes, and separate weights were trained for

each class. All of the Pix2Pix training was done on a computer with a single GTX 1050Ti and each class was trained for 1600 epochs. Every set of 1600 epochs took around 8 hours to train.

2.2.3 CycleGAN

CycleGAN is a novel style transfer architecture that does not need paired images. The data for CycleGAN is two sets of images, from different domains X and Y. This can be anything from paintings and photographs, horses and zebras, or in our case, hand-drawn sketches and photographs. CycleGAN uses *two* sets of GANs, a discriminator and generator for set X and a discriminator and generator for set Y. The goal is to learn a mapping $G : X \rightarrow Y$ and $F : Y \rightarrow X$. In order to do so, two discriminators are introduced: D_X, D_Y. These discriminators are used to train the generators in an adversarial fashion using a typical minimax loss. Note that any discriminator and any generator can be used for these networks.

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}[\log(1 - D_Y(G(x))]$$
$$\mathcal{L}_{GAN}(G, D_X, X, Y) = \mathbb{E}_{x \sim p_{data}(x)}[\log D_X(x)] + \mathbb{E}_{y \sim p_{data}(y)}[\log(1 - D_X(F(y))]$$

In other words, this loss is equivalent to the log likelihood over two classes for the classifiers D_X and D_Y, using x and y as positive examples, and $F(y)$ and $G(x)$ as negative examples, respectively. While this loss alone produces generators capable of creating samples from one distribution similar to those in the other, CycleGAN introduces another metric that aids in even better style transfer. Borrowing ideas from Natural Language Processing, a cycle loss is used. In natural language processing, cycle loss is the error that arises from translating text from one language into another, and then back. For CycleGAN, the structural difference between images is used:

$$\mathcal{L}_{cycle}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim p_{data}(y)}[||G(F(y)) - y||_1]$$

Where $|| \cdot ||_1$ represents the first norm. This loss penalizes discrepancies in image space between a sample x transformed into Y using G, and then transformed back into X using F. The analogous case for y applies as well. Using a combination of the three losses, the networks can be trained together, and the results are state of the art for style transfer.

For our implementation, we used the open source implementation [9]. In order to do a direct comparison with Pix2Pix, the same three datasets were used from [6]: pizza, trees, and apples. In addition, we used our own hand-crafted dataset combining ImageNet oak images and sketches from Google Quick Draw [7]. Since CycleGAN takes essentially double the time to train as Pix2Pix, a Google Cloud compute VM was used with 4 Tesla V100 GPUs. Each dataset was trained for 300 epochs, and took about 3 hours each.

3 Results

3.1 GUI

In order to analyze the results obtained from our GANs, we first needed a way to provide new test data to our models. In order to match the Sketchy Database data utilized for training, we determined the easiest way to provide this functionality would be through a Python GUI that offers a simple canvas to draw pictures in. We can then easily send these hand-drawn pictures from the GUI to our trained Pix2Pix and CycleGAN models to view the results in real time. The GUI created is shown below:

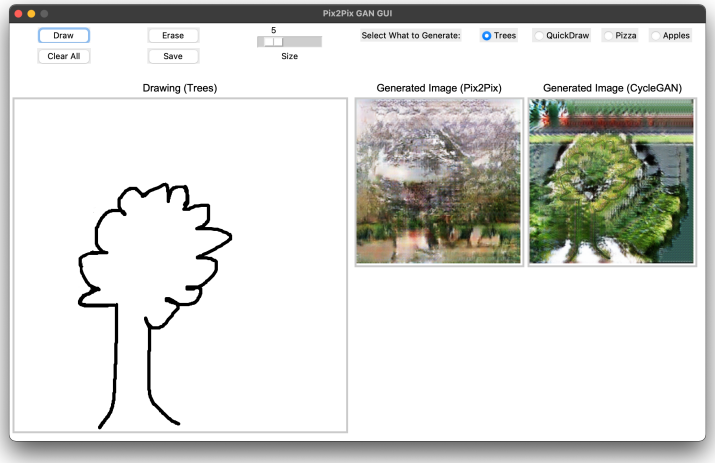


Figure 7: A simple GUI used to input new data to both our trained Pix2Pix and CycleGAN models. The pane in the left allows for the mouse to be used for creating a sketch, while the images on the right are the generated result. Controls at the top allow for changing brush size, toggling between eraser mode and drawing mode, and saving. The radio buttons configure which class is being generated.

3.2 Performance

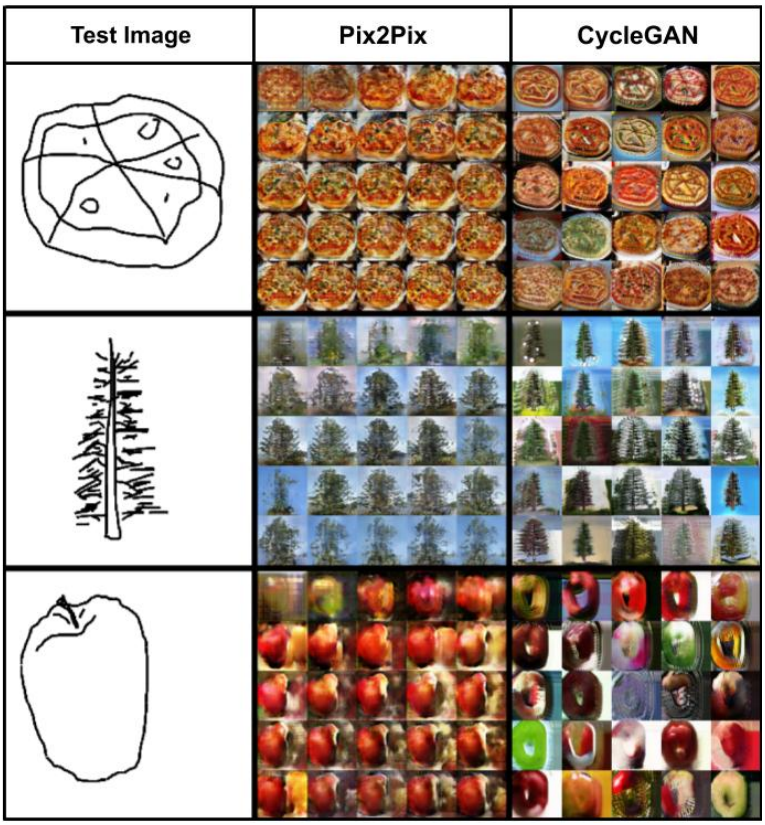


Figure 8: Training progress for Pix2Pix and CycleGAN. The same test image was used to generate a sample every 10 epochs, and can be seen of the far left. Training progress for each epoch is seen in the mosaics where epoch 10 is the top left and the final trained model is at the bottom right. Pix2Pix is in the middle, and CycleGAN is on the right

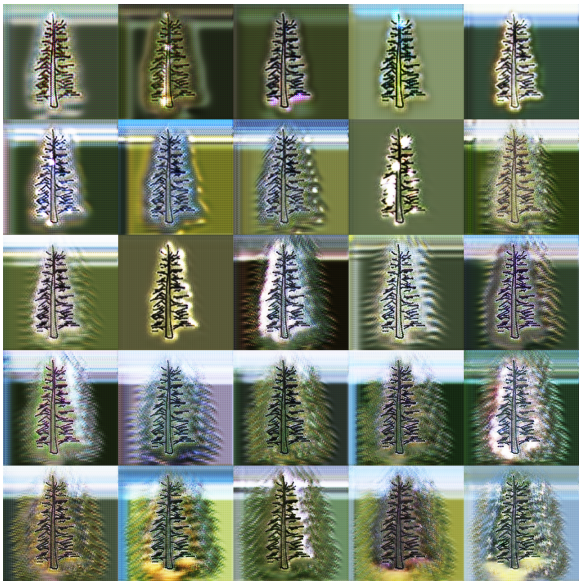


Figure 9: Training progress for CycleGAN on the more challenging combination of sketches from QuickDraw and images from ImageNet. The input is the same sketch as Fig. 8

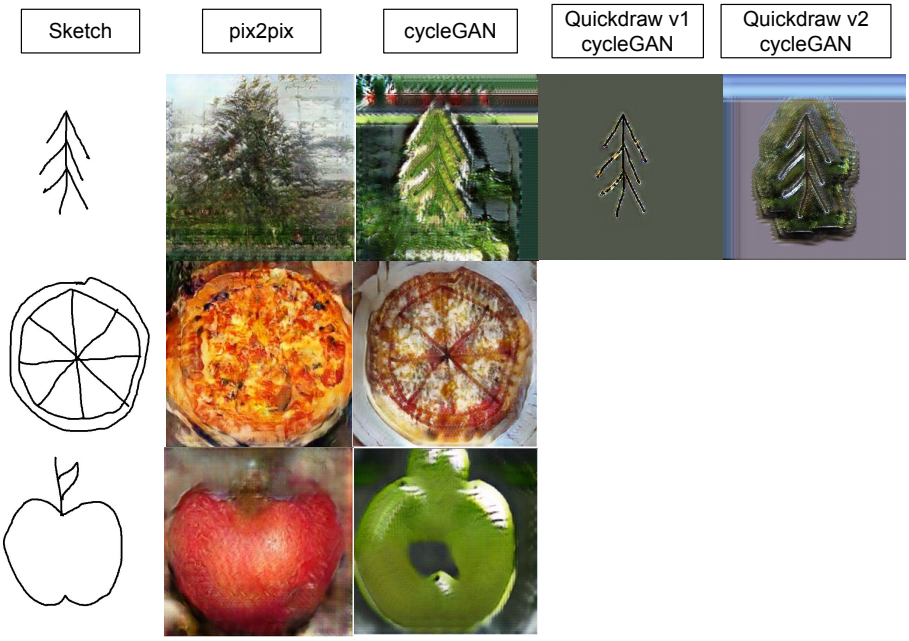


Figure 10: A direct comparison between models on the same sketch

As can be seen qualitatively, CycleGAN and Pix2Pix produced quite different results. It is interesting that in the CycleGAN generated images, the lines that are present in the sketch are also present in the image, while in the Pix2Pix generated image, the sketch is more obfuscated. The differences between Quickdraw v1 and Quickdraw v2 are also significant, and it seems the model benefited from a more consistent set of tree images.

3.3 Error Analysis

In addition to the qualitative assessment on the Pix2Pix and CycleGAN models, we wanted to calculate some quantitative error measures for comparison. The first metric that comes to mind is the mean-squared-error between generated image and ground truth image, however this metric fails to account for general structure in the image. After all both Pix2Pix and CycleGAN were developed to translate general structure between feature spaces (edges, gradients etc). Instead of using MSE, we opted to use the Structure Similarity Index Measure (SSIM) [10]. This metric works by comparing the luminance, contrast and structure across many windows between the compared images. This metric should give a clearer indication of the similarity between generated images and the ground truth images. The results are shown in table 1 below.

	Pix2Pix	CycleGAN
Pizza	0.083	0.088
Trees	0.140	0.095
Apples	0.223	0.258
Quickdraw Trees	N/A	0.272

Table 1: SSIM values for Pix2Pix and CycleGAN. Note that Pix2Pix was not trained on the Quickdraw dataset, since those images are unpaired

It is interesting to note that Pix2Pix and CycleGAN follow similar trends in SSIM performance, where they performed the best on the apples dataset and worst on the pizza dataset. This is most likely due to characteristics of the data, and similarities between the two models. CycleGAN also performed better than any other configuration when trained on the Quickdraw dataset. Although the SSIM values in general are far from optimal, it is clear that the GANs did learn mappings that preserve a decent bit of structure.

3.4 Challenges Faced

Training CycleGAN with the new dataset combination of sketches and ImageNet images provided its own unique challenges. Because the sketches were stored as 28x28 images, they had to be upscaled to be large enough to be able to pass through the network. This caused the images to be blurry, which resulted in less than optimal results due to the differences between the images used to train and the images drawn in our GUI. To combat this, we experimented with different algorithms to upscale the images (using OpenCV). The default cubic interpolation was replaced with the `INTER_AREA` algorithm which re-samples using pixel area relation, which gives moire'-free results, and is thus preferred for upscaling images. The results were better, but there is still noticeable blur in the sketches (see Fig. 5). Another challenge with the dataset was that the 1280 images from ImageNet for class `oak` were very diverse. The trees weren't always centered, or the image was a close-up of a tree, or the image was cluttered with other objects. Thus, a subset of 200 of the images were chosen in order to curate a better dataset.

Another major challenge we faced was the sheer time it takes to train GANs. For members of the team without GPUs, training the GANs is, for any practicality, impossible. To combat this, we used a cloud VM from Google Cloud Platform. This sped up the training from about 180 seconds per epoch on a 1080 TI to about 30 seconds per epoch running on 4 V100s.

4 Individual Contributions

4.1 Greg Lund

I focused mostly on Pix2Pix for this project, and did all the training for this model. I researched a decent bit about Pix2Pix's architecture and how it works, by reading the original paper and a few other resources. I took the QuickDraw dataset and processed it into the appropriate format for training. I trained Pix2Pix on the various classes on my personal computer, which ended up working fine (1050Ti). I also helped create the progress figures and error metrics.

4.2 Chris Beggs

I focused on the initial explorations portion of the project. For this, I created fully connected generator and discriminator networks from scratch, and spent time tweaking and adding tricks to make the training stable. After producing good reproducible results with that, I re-implemented DCGAN. For this I used *many* available online resources for the architectures, and modified the training data so that it would work with 2D convolution operations. Both networks were trained on a small laptop, so the number of params and training data had to be kept to a minimum. To solve both GAN's, a solver interface was created, and a small command line interface was created for users.

4.3 Connor Thompson

For this project, my scope was somewhat limited by the hardware I had available (a 13" Macbook Pro with an Intel Iris Plus 655 GPU) so I wasn't able to assist with any of the neural network training. However, I still familiarized myself with general GAN and specific Pix2Pix and CycleGAN theory and methodologies by reading various materials (including the original GAN, Pix2Pix, and CycleGAN papers). Beyond learning this information (which was really the goal of this project), I also developed the GUI used to input new data into our GANs and display the results (giving me hands on GAN experience) and provided a good amount of work on both the final project presentation and report.

4.4 Soroush Khadem

I was focused on the CycleGAN portion of the project. I spent time to understand how it worked as well as some of the theory behind cycle loss. I assembled the QuickDraw/ImageNet dataset and spent time curating it. I also trained all of the CycleGAN models using a combination of my local desktop and my Google Cloud Platform account which had a VM that was loaded with faster and bigger GPUs. At one point, I had a model training locally, and two training in the cloud at the same time using separate GPUs. Parallelism! In addition to that, I helped create the backend for the UI to be able to pass images back and forth and support the creation of various figures from our models.

5 Conclusion

We took on this project in order to learn more about the theory and applications of GANs and some of their specific implementations. We also had the goal of getting some hands-on experience working with GANs at both a high and low level. Throughout the duration of this project, we have achieved these goals in numerous ways.

Addressing the first goal, we have all further developed our knowledge of GANs in general and specifically our knowledge of conditional GANs and CycleGAN. This has given us not just a conceptual understanding of these generative models, but also insight into the mathematics required to get them to work. We have taken this knowledge and applied it in novel ways such as for our initial data exploration.

Furthermore, we have achieved our second goal by taking existing GAN implementations and adapting them for use on our own unique datasets and applications. This has given us practical hands-on experience working with GANs that we would not have received otherwise. Going further, this project has allowed us to analyze the differences between two unique implementations of GANs – namely that CycleGAN shows direct responses from our input in the targets style space and that differences in the two architectures leads to Pix2Pix being non-deterministic while the same cannot be said for CycleGAN.

This project has shown us that while GANs are super fun to work with and have many practical applications (such as design [11], art [12], and logistics [2]), they require a strong mathematical background and access to ubiquitous computing resources. Regardless, GANs are an amazing field of research to be involved in and we will be waiting with bated breath to see what progress is made next.

References

[1] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018.

[2] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks, 2018.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[5] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

[6] Patsorn Sangkloy, Nathan Burnell, Cusuh Ham, and James Hays. The sketchy database: Learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (proceedings of SIGGRAPH)*, 2016.

[7] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The quick, draw! - a.i. experiment. *https://quickdraw.withgoogle.com/*, 2016.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] Jun-Yan Zhu, Taesung Park, and Tongzhou Wang. Cyclegan and pix2pix in pytorch. https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix, 2020.

[10] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[11] Stanislas Chaillou. Archigan: a generative stack for apartment building design, Aug 2020.

[12] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization, 2019.