Comparative Analysis of Generative Adversarial Networks for Converting Images to Easily Interpretable Sketches

– Honours Project –

submitted by

YanPeng Gao

Decemeber 18, 2020

YanPeng Gao

yanpeng.gao@carleton.ca

Student ID: 101090653

Supervisor

1st: Dr. Majid Komeili

Abstract

Generative Adversarial Networks have been shown to produce realistic and high quality images by leveraging adversarial training between a generator network and a classifier network. We implement and analyze two different image to image translation GANs, Pix2Pix and CycleGAN, on our datasets with the goal of producing highly interpretable black and white sketches. We analyze the performance of these GANs on different datasets and were able to assess some of the strengths and weaknesses of Pix2Pix and CycleGAN. For Pix2Pix, an improved generator architecture, data augmentation effect and training guidelines were also suggested.

Acknowledgments

Deepest thanks to my parents for believing in me, inspiring me, and supporting me.

Special thanks to my supervisor Dr.Majid Komeili. Firstly for being a fantastic professor, it was a joy to be a student in his Introduction to Machine Learning class. And secondly for being a great supervisor, providing me with guidance and resources to succeed.

Contents

List of Figures

List of Tables

1 Motivation

The motivation of this project is to build Generative Adversarial Networks that are skilled at the image to image translation task of converting photos to realistic, highly interpretable sketches. Such a system of converting complex photos to easy to understand sketches may have applications in aiding the visually impaired or helping create alternative styles in media entertainment. Furthermore, we are to do a comparative analysis of two GAN architectures Pix2Pix and CycleGAN to compare and contrast the advantages of either designs.

Generative Adversarial Networks have become a very popular research topic due it's power in generating very realistic and sharp images. For this reason it was the chosen approach for our project. However, the training of GANs is often very unstable so many experiments will be conducted to find ways to optimize training.

1.1 Related Work

This project builds upon the work of Pix2Pix[8] and CycleGAN [22] architectures. There have been many image to image projects of mapping sketches to images. Note that these implementations are often only bounded to 1 category, ie: sketches to cats, sketches to handbags, sketches to shoes [5]. The work of Li et al. [11] has demonstrated that Pix2Pix can generate good sketches from images though they did not explore CycleGAN. Our codebase contains some references from both Li et al. [11] and Zhu et al. [22].

2 Introduction to Generative Adversarial Networks

This chapter discusses the basics of Generative Adversarial Networks. Many variations of GANs exist but the following are a couple of the most foundational. Understanding these networks will make understanding Pix2Pix and CycleGAN much easier.

2.1 Original Generative Adversarial Network

GANs were first introduced in 2014 by Goodfellow et al. [3]. At the time, deep neural networks were most successful as discriminative models classifying high dimensional

data such as images into appropriate pre-labeled classes [10]. Models that were generative in nature did not yet have the same success as it struggled to approximate explicit representation of many intractable probabilistic estimations [6].

The framework for GANs was to have two multi layer perceptrons, a generator model **G** and a discriminator model **D** compete against each other in an adversarial manner to implicitly learn our data distribution. Our generator takes in noise z in the form $p_z(z)$ and has the mapping $G(z;\theta_g)$ that produces a data distribution of fake/generated images p_g. The discriminator $D(x;\theta_d)$ on the other hand is a classifier trained to produce a probability of whether the image belongs to the real distribution or the generated distribution. $D(x)$ sees samples that come from both the real p_{data} and the fake p_g. In the adversarial manner, the generator wins by producing fake images that the discriminator thinks is real while the discriminator wins by correctly classifying both real images as real and fake images as fake. It is worth noting that our generator only gets feedback of how well its training based on the results of the discriminator and not itself explicitly comparing its generated and real images.

The loss function, referred to as the value function in the paper is defined as a two player min-max game:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

2.1.1 Components of the GAN value function

First, note that the $\mathbb{E}(X)$ when X belongs to the a set of m samples is $\frac{1}{m}\sum_{x_i\in X} x_i$. This should clarify the practical meaning of the expected value in our value function.

The Discriminator's Role:

We have D aimed at maximizing our value function $V(D,G)$. Recall that $D(x)$ is the probability that an image is real. So for images in $x \sim p_{data}(x)$, D should label them all with 1, hence we should maximize the first part $\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)]$.

In the second portion, D is maximizing $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$. Because the value of D is $[0,1]$, the maximum is when $\log(1 - D(G(z))) = \log 1$ and $D(G(z)) = 0$. This happens when our discriminator classifies fake images as fake, which is our intention.

The Generator's Role:

The generator is trying to minimize $V(D, G)$. Notice however that the first portion of our value function does not contain the generator, so the generator is only applied to: $\min_G V(D, G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$.

The generator is trying to mimize this function because the minimum value of $\log(1 - D(G(z)))$ is when $D(G(z)) = 1$, and in this situation, the generator is winning the adversarial game by fooling the discriminator into classifying it's fake images as real.

2.1.2 Proof of the Global Optimality of $p_g = p_{data}$

Section 4.1 in Goodfellow et al. [3] shows that the generator succeeds in eventually producing a generated distribution that is identical to the real distribution. In this situation, the discriminator will assign a value of $\frac{1}{2}$ everywhere along the distribution of generated images. Their proof is very verbose so we shall provide a more detailed analysis.

Proposition 1: For a fixed G, the optimal discriminator is:

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

We first recall that $\mathbb{E}_{p(x)}[X] = \sum_{x \in X} x\, p(x)$ in the continous case becomes $\int_x xp(x)\, dx$.

So our value function can also be represented as, D trying to maximize for:

$$V(D, G) = \int_x p_{data}(x)\, \log D(x)\, dx + \int_z p_z(z) \log(1 - D(G(z)))\, dz$$

The next step which was skipped in the paper for perhaps being trivial was seeing that: $\mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] = \mathbb{E}_{x \sim p_g(x)}[\log(1 - D(x))]$. This is because x can be samples from either p_g or p_{data} and so $G(z)$ when sampled from p_z is equivalent to x when sampled from p_g.

Our value function is now:

$$\begin{aligned} V(D, G) &= \int_x p_{data}(x)\, \log D(x)\, dx + \int_x p_g(x) \log(1 - D(x))\, dx \\ &= \int_x p_{data}(x)\, \log D(x) + p_g(x) \log(1 - D(x))\, dx \end{aligned}$$

To maximize our value function, we are looking to maximize $p_{data}(x) \log D(x) + p_g(x) \log (1 - D(x))$. So we take the derivative to find the maximum.

$$\begin{aligned}\frac{d}{dx} &= p_{data}(x)\frac{d}{dx}\log D(x) + p_g(x)\frac{d}{dx}\log(1 - D(x)) \\ &= \frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)}\end{aligned}$$

Setting the derivative to 0 to find the the maximum, we get:

$$\begin{aligned}\frac{p_{data}(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} &= 0 \\ D(x) &= \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}\end{aligned}$$

We therefore have shown the first proposition.

Proposition 2: The best generator has the optimal value $p_g = p_{data}$ when given the optimal discriminator.

This proposition is crucial because it tells us that even when we optimize for the discriminator, we are still able to produce a generator that perfectly produces a distribution identical to our real distribution. We will proof this with a bit of algebra detailed in an article by Rome [14].

We take the same value function as before, but this time we are conditioned on the optimal discriminator which we know is $D^*(x) = \frac{p_{data}}{p_{data}+p_g}$.

$$\begin{aligned}V(D, G) &= \int_x p_{data}(x)\ \log D^*(x)\, dx + \int_x p_g(x) \log (1 - D^*(x))\, dx \\ &= \int_x p_{data}(x)\ \log \frac{p_{data}}{p_{data} + p_g} + p_g(x) \log (1 - \frac{p_{data}}{p_{data} + p_g})\, dx\end{aligned}$$

Next, we add $(\log 2 - \log 2)p_{data}(x)$ and $(\log 2 - \log 2)p_g(x)$ as our algebra trick. Since $(\log 2 - \log 2) = 0$, nothing is changed in our value function.

$$\begin{aligned}V(D, G) = \int_x &(\log 2 - \log 2)p_{data}(x) + p_{data}(x)\ \log \frac{p_{data}}{p_{data} + p_g} \\ &+(\log 2 - \log 2)p_g(x) + p_g(x) \log (1 - \frac{p_{data}}{p_{data} + p_g})\, dx\end{aligned}$$

After factoring and simplyfing this expression, we are left with:

$$V(D,G) = -log2 \int_x p_g(x) + -log2 \int xp_{data}(x) + \int xp_{data}(x) \log \frac{2\, p_{data}(x)}{p_{data}(x) + p_g(x)} dx + \int xp_g(x) \log \frac{2\, p_g(x)}{p_{data}(x) + p_g(x)} dx$$

Following this, we recognize that the integral of p_g and p_{data} are both 1 as they are probability distributions and the last 2 integrals forms twice the Jensen–Shannon divergence:

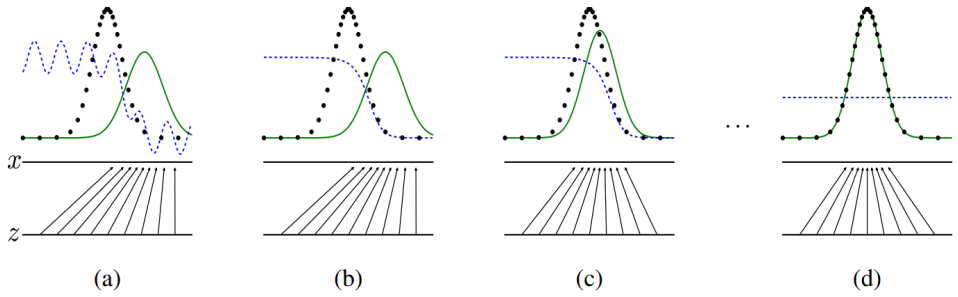
$$V(D,G) = -2\, \log 2 + 2\, JSD(P_{data}(x)||P_g(x))$$

Finally, when we minimize this, we are trying to minimize the Jensen-Shannon divergence. The $min(JSD) = 0$, and that happen's when $p_g = p_{data}$. So we proved the best optimal value of the generator is when it creates a generated distribution the same as the real distribution.

2.1.3 GANs in practice

The original loss has G minimizing $\log(1 - D(G(z)))$. However, when the discriminator is very good at identifying fakes, the gradients are close to 0 and our generator struggles to learn. We can see this by taking the derivative of $\log(1 - y) = -\frac{1}{1-y}$. When y is close to 1, our gradient has a large magnitude, but when it's close to 0, our gradient is small. To fix this, we don't change our discriminator's objective but for our generator, it is now trained to **maximize** $\log D(x)$. The purpose is still the same, to trick the Discriminator to classify generated images as real, but now when the discriminator is good at detecting fakes, the magnitude of the gradients is high allowing the generator to learn quicker. This method

Figure 1 shows how GANs iteratively learn. The dotted black line is the real distribution and the green line is the generated distribution. The dotted blue line is the value of the discriminator assessing if an image is real or fake. We see iteratively that the discriminator improves which then helps the generator approximate the real distribution more closely. Eventually, the generated distribution is identical to the real distribution and the discriminator's value is $\frac{1}{2}$ everywhere.



Source: Goodfellow et al. [3]

Figure 1: Gan Iteratively Training

2.2 Deep Convolutional GANs

The GAN models built by Goodfellow et al. [3] previously used multi layer perceptrons. The work of Radford, Metz, and Chintala [13], known as Deep Convolutional Generative Adversarial Networks, outlined a family of architectures that provides stable training across different types of datasets. This foundation is still used today for various GAN models, including Pix2Pix and CycleGan.

The general architecture guidelines for stable DCGANs are:

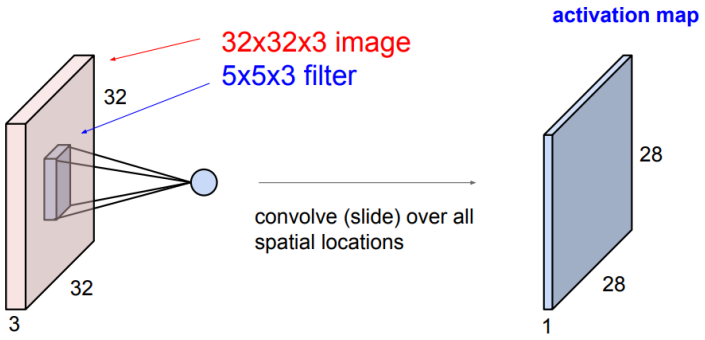
- Replace any pooling layers with strided convolutions and any upsampling layers with transposed convolutions.
- Use batch normalization in both the generator and discriminator.
- Fully connected layers are not needed.
- Use ReLU activation in the generator and LeakyReLu activation in the discriminator
- Use Adam optimization instead of stochastic gradient descent, with a learning rate of 0.0002 and beta hyperparemets $\beta_1 = 0.5$ and $\beta_2 = 0.999$

More details of some of these guidelines is as follows.

Strided and Tranposed Convolutions

Recall that a convolutional layer takes in an input image of $height \times width \times depth$. Kernels (also referred to as filters) of size $k \times k \times depth$ are slid over this image one stride at a time. At each pixel position, we apply the dot product of the kernel's

weights with the underlying values of the input image to generate a scalar. Sliding the kernel along every position in the image generates a field of outputs known as an activation map. The weights in these kernels are learnt with back propagation just like the weights in a multi layer perceptron.



Source: Stanford CS231n Lectures[2]

Figure 2: Diagram of a CNN Layer Filter

For an output of *depth* m, we would need m filters in this layer. The height or width can be formulated as:

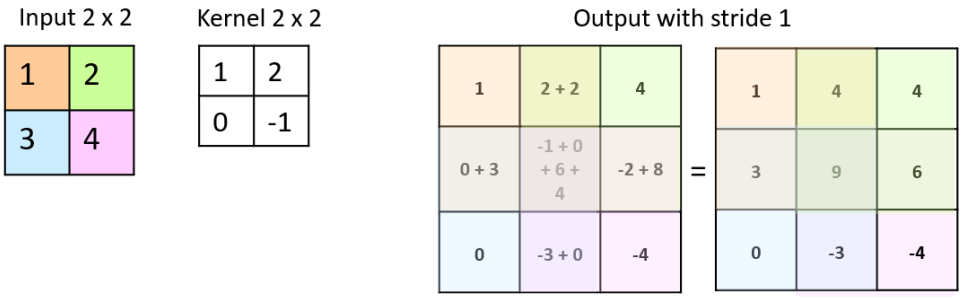
$$\text{Output size after convolution } = \frac{W - K + 2P}{S} + 1$$

Where W is our starting width/height, K is the width/height of our kernel, P is the the amount of padding we give the image and S is the stride length.

We can see that instead of pooling to reduce the size of our image, we can slide our kernel but skip every second pixel location reducing our scalar outputs by half both heightwise and lengthwise. This method means our stride has value 2. For example, for an input image of width and height 256×256, a CNN layer with kernels of size 4×4, padding the perimeter of the image with 1 pixel, and a stride of 2 units gives an output dimension of 128×128.

Transposed convolutions can be seen as the opposite of regular convolutions. Given a kernel of size $k \times k$, each pixel of the input is mapped to a $k \times k$ region in the output through a single multiplication of the input pixel by one of the k^2 weights in the kernel. When regions of the output have overlapping mappings, we add the output scalars together. The size of an output is: $S \times (W - 1) + k - 2p$.

Figure 3 shows a transposed convolution diagram with a 2×2 input and 2×2 kernel. Pixel 1 of the input maps to the 4 upper left pixels of the output. Pixel 2 maps to



Source: Original

Figure 3: A transposed convolution

4 upper right pixels of the output. Pixel 3 maps to 4 lower left pixels and Pixel 4 maps to the 4 lower right pixels.

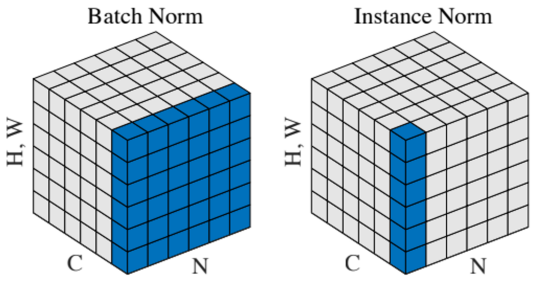
Batch Normalization

Batch normalization since its introduction by Ioffe and Szegedy [7] has helped to stabilize and accelerate the training of deep neural networks. The problem of internal covariate shift arises when the distribution of the data reaching the deeper hidden layers drastically changes each iteration as we update the weights of the beginning layers. To resolve this, we standard normalize after each hidden layer output to maintain a relatively stable distribution.

So for a given batch of m samples $z^{(1)}, ..., z^{(m)}$ we compute the mean $\mu = \frac{1}{m}\sum z^{(i)}$ and variance $\sigma^2 = \frac{1}{m}\sum(z^{(i)} - \mu)^2$ and normalize each sample to be: $z^i_{norm} = \frac{z^{(i)}-\mu}{\sqrt{\sigma^2}}$. Finally, we have the additional option of transforming our normalized inputs by $\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$ where γ and β are learnable parameters. Note that if we do apply this transformation, a bias in our convolutional layer is redundant.

Batch normalization works slightly differently when we are dealing with images. Consider a batch of N samples where the input has C channels as shown in the figure above. In this situation, we apply separate batch norms to every channel in our batch.

For image style transfer tasks, Ulyanov, Vedaldi, and Lempitsky [18] have found that instance normalization leads to better results as it is contrast invariant. Note that instance normalization is equivalent to batch normalization when our batch size is 1.



Source: From Wu and He [19]

Figure 4: Illustration of the pixels that get normalized together shown in blue

3 Pix2Pix

Pix2Pix is one of the two conditional GANs implemented in this project. It was introduced in the paper *Image-to-Image Translation with Conditional Adversarial Networks* by Isola et al. [8]. As the title suggests, it is used for image to image translation tasks where we want to map images from one representation to another. Pix2Pix requires paired data in the form $\langle x, y \rangle$ where we attempt to translate image $x^{(i)} \rightarrow y^{(i)}$. Paired images should be related in information but different in representation.

Source: From Isola et al. [8]

Figure 5: Example of a paired data sample. From segments to a realistic street

3.1 Loss Function

The loss function (previously referred to as the value function by Goodfellow et al. [3]) is made up of two parts: a adversarial loss and a pixel distance loss.

The adversarial loss is defined as:

$$L_{GAN}(D, G) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log (1 - D(x, G(x, z)))]$$

We play the same min-max game as before where the discriminator tries to maximize the Adversarial loss and the generator's objective is to minimize the loss. The difference is now the generator takes in both the noise z and images from the set x to produce the mapping $G(x, z) \rightarrow y_{fake}$. Likewise, for a given pair (x, y), the discriminator sees both $D(x, y_{real})$ and $D(x, y_{fake})$ and it's job is to classify them correctly.

The second part of the loss is a pixel distance L1 loss defined as:

$$L_1(G) = \mathbb{E}_{x,y,z}[||y - G(x, z)||_1]$$

This L_1 loss help enforce rough approximations of feature locations in the generated images by comparing pixels to it's real target. A similar feature matching approach was first suggested in Salimans et al. [16] as a way to stabilize the training of GAN's where the generator tries to minimize the distance of, real images' latent vectors and fake images' latent vectors present inside the discriminator.

Together, the loss function becomes:

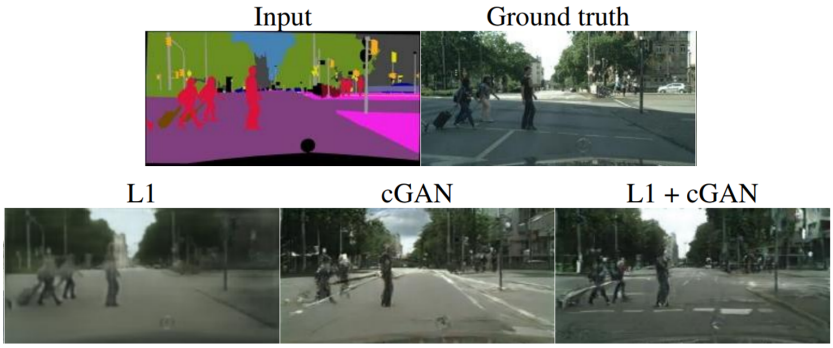
$$\min_G \max_D L(D, G) = L_{GAN}(D, G) + \lambda L_1(G)$$

The two losses work well in harmony. The adversarial loss generates photo realistic samples while the L_1 loss helps with feature location matching. Below is an ablation study done in the Pix2Pix paper. We see that when training the GAN with just an L_1 loss the image is very blurry. With just the adversarial loss, the generated image is very crisp but some feature are missing or displaced. Finally, using both the losses together give the best results.

3.2 U-Net Generator

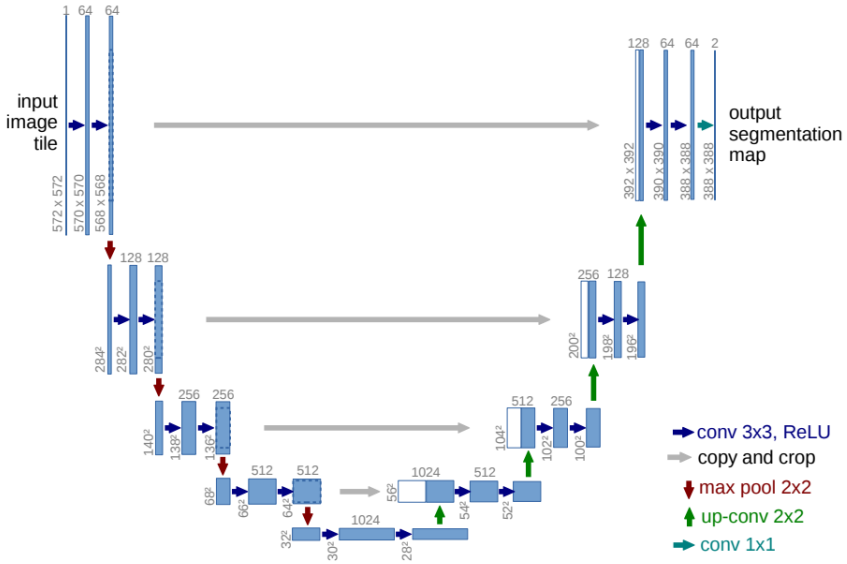
The generator proposed here is a encoder-decoder architecture with skip connections from encoder layers to decoder layers, known as a U-Net Ronneberger, Fischer, and Brox [15].

The U-Net architecture consists of successive pairs of convolutional layers followed by pooling layers to reduce image size until a bottle neck is reached. Then images are up-sampled while going through more convolutional layers. Though the original U-Net has pooling and upsampling layers, we will be implementing the model with strided convolutions instead.



Source: From Isola et al. [8]

Figure 6: Ablation study of the Pix2Pix Loss Function



Source: From Ronneberger, Fischer, and Brox [15]

Figure 7: U-Net generator

UNet is theorized to work well because it is able to pass finer details of the beginning input layers directly to the output layers through the skip connections. This prevents losing low level information when the image gets downsampled and bottle necked in our network. Another possible explanation for the success of UNets is the skip connections help gradient flow and avoids vanishing gradients similar to ResNets [4].

3.3 PatchGAN Discriminator

The discriminator in Pix2Pix contains layers of strided convolutions similar to the proposal in DCGAN. However, instead of the discriminator classifying if the entire picture is real or fake, it classifies $N \times N$ patches to be real or fake. Because we are already using the L_1 loss term from the generator to enforce feature locations, the discriminator in turn focuses more on classifying the finer details of the image as real or fake. For images of size 256×256, the typical patch size used is 70×70.

4 CycleGAN

The second image to image translation GAN implemented is CycleGAN Zhu et al. [22]. CycleGAN aims to translate images from one domain X to another domain Y without needing paired training data. Though a specific image from one domain does not need to have a direct pairing in the second domain, the individual domains themselves need to have a common characteristic. In our work, we are translating from a domain of pictures to a domain of sketches.

4.1 CycleGAN Loss Function

The components of CycleGAN is as follows:

- A generator G that maps images from set $X \rightarrow Y$
- A generator F that maps images from set $Y \rightarrow X$
- A discriminator D_X which distinguishes between images X and translated images $F(Y)$
- A discriminator D_Y which distinguishes between images Y and translated images $F(X)$

We have generator G and discriminator D_Y play the adversarial game since G maps from $X \rightarrow Y$ and the discriminator D_Y distinguishes fake or real $Y's$.

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

Similarily, we have F and discriminator D_X play the adversarial game.

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

Lastly, we implement a cycle consistency lost. We want $x \to G(x) \to F(G(x)) \approx x$ and $y \to F(y) \to G(F(y)) \approx y$. So we minimize:

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

The cycle consistency loss is important for a couple reasons. First, we are preventing mode collapse. We want to make sure that when given an x, $G(x)$ isn't mapping to one specific instance of y_{fake}. The cycle loss forces $G(x)$ to be unique since it needs to find a way back to the unique x through $F(G(x))$. Secondly, the cycle loss encourages G to capture important features of the image x because those are going to be the key features used in mapping back to x.

The **full loss term** is finally:

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + L_{cycle}(G, F)$$

4.1.1 Least Squares GAN

Although not initially defined in the paper, CycleGAN adapts it's adversarial loss function to a least squares method instead of binary cross entropy. This architecture was introduced in the paper *Least Squares Generative Adversarial Networks* (Mao et al. [12]). Recall the least squares method in machine learning algorithms minimizes the squared difference between our target and our prediction: $\sum(y_i - \hat{y}_i)^2$.

So for a $L_{GAN}(G, D, X, Y)$, the adversarial loss objective when using Least Squares become:

$$\min_D \mathbb{E}_{y \sim p_{data}(y)}[(D(y) - 1)^2] + \mathbb{E}_{x \sim p_{data}(x)}[(D(G(x)) - 0)^2]$$
$$\min_G \mathbb{E}_{x \sim p_{data}(x)}[(D(G(x)) - 1)^2]$$

This setup is still adversarial in nature. The discriminator is trying to minimize the sum of squares difference between real images and the label 1 and also minimize the sum of square difference between fake images and the label 0. Conversely, the generator is trying to trick the discriminator into classifying fake images as real by

minimizing the squared difference of the generators classification of fake images and the true label 1.

The authors of LSGAN proposed this method as a way of fighting mode collapse, a common problem in GAN training where for the current iteration the discriminator is fooled by a specific subset of generated images and continuously for the future iterations, the generator only produces this small subset of images. This is because in regular GAN's with a sigmoid decision boundary, fake images that are mistakenly classified as real, but are far away from the actual distribution of real images do not provide good gradients for updating the generator. The generator starts producing more of these fake images and so gradients start to vanish, resulting in mode collapse. But in LSGAN's, the adversarial objective of the generator is a least squares loss that continues to penalize fake images that are different than the real distribution even if they managed to fool the discriminator.

For example, due to the objective, most of $x \sim p_{data}$ has a $D(x) \approx 1$. If the generator produces an x_{fake} that has the discriminator value of $D(x_{fake}) = 2$, it technically has fooled the discriminator but due to least squares loss, the error and gradients is just as large as if $D(x_{fake}) = 0$.

4.1.2 CycleGAN Generator

The generator in CycleGAN uses two strided convolutional layers to downsample the image followed by 9 residual block layers with skip connections in the bottle neck before using transposed convolutions again to upsample [9].

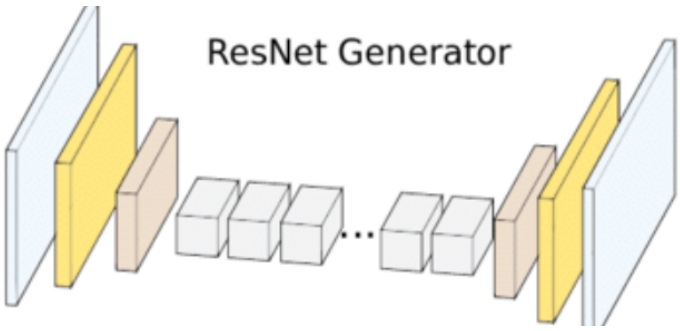


Figure 8: Resnet generator

5 Datasets

We will be using two datasets in this project. The first being referred to as *Photo-Sketch* [11] contains 1000 photos and that are each paired with 5 sketches. This dataset is of very high quality as sketches were gathered by asking humans to trace the outlines and contours over an underlain photo in a web app. Therefore the locations of the elements in both the photo and sketches are highly aligned. The

Source: From Li et al. [11]

Figure 9: Sketch Collections Through Outlining Underlain Images

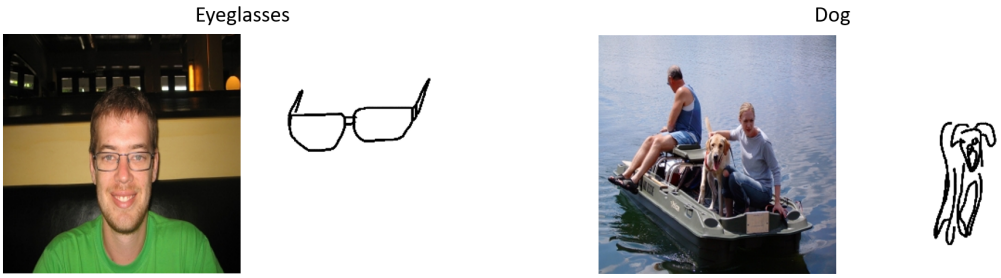
environment of photos in *Photo-Sketch* are all outdoor shots consisting of some combination of humans, dogs, frisbees and frisbee posts.

Source: From Li et al. [11]

Figure 10: Sample Images From Photo-Sketch

The second dataset used is from *Sketchy* [17] and it contains 125 categories of objects where each object has 100 photos. Each photo is paired with at least 5 human drawn sketches. In total, there are 12,500 photos and 75,471 sketches. A problem arises with this dataset for our application due to the way it was gathered. Sketchers

were told a category of object and shown an image and were asked to sketch the object. Therefore, if the image had many objects, only the object that belonged to the specific category was drawn. Many times, the category being drawn was not the main focus of the photograph. Furthermore, the sketches did not have to be feature aligned, and the human drawers had more artistic freedom and were more flexible in interpretation compared to our first dataset *Photo-Sketch*.



Source: From Sangkloy et al. [17]

Figure 11: Sample photos and its paired sketches From *Sketchy*

From Sketchy, we sampled 10 categories: cabin, castle, church, skyscraper, car, pickup, duck, swan, rabbit and deer. This contained a good mixture of buildings, vehicles and animals. For consistency, these categories were chosen as the photos were mainly outdoor pictures, just like in Photo-Sketch.

Table 1: Comparison of datasets

Name	**Photo-Sketch**	**Sketchy**
Number of Categories of Elements	4 (humans, dogs, frisbees, frisbee goals)	10
Number of Photos	1000	1000
Number of Sketches	5000	6214
Data is paired?	Yes	Yes
256x256 Dimension?	No, but resizeable to 256x256	Yes
Paired photos and sketches are feature aligned?	Yes	No
Sketches contain human synthesized visual cues?	Yes	Yes
Multiple elements in one sketch?	Yes	No

6 Methodology

As from before, both datasets have 1000 photos with each photo being paired to 5 sketches. We forego a validation set and use 900 photos to train and the other 100 photos to test.

6.0.1 Evaluating Training

Finding a quantitative measurement to monitor image to image GAN training still is an unsolved problem. Because of the adversarial nature of the two networks, looking at their losses is rather inconclusive. For example, consider an adversarial loss where both the discriminator and generator are minimizing their objectives (ie least squares GAN). A low discriminator loss may mean that either the discriminator is strong or the generator is weak. Likewise, a low generator loss just means it's fooling the discriminator, but that could be because the discriminator is weak and then images are still not of good quality.

So we monitor training in two way:

1. Monitor the generated fake images of the training and test sets
2. Monitor the pixel distance L_1 loss. This is less important as L_1 losses consider each pixel value independent of it's neighbour and doesn't take into account the structure of features.

6.1 GAN training with Photo-Sketch

We first start with the Photo-Sketch dataset as the data seems to be of higher quality and easier to train. Pix2Pix was the first GAN used as it made sense to choose the GAN with paired training considering both our datasets were paired. General hyperparamter guidelines were established by Isola et al. [8] and so we started off with those settings. A minor change must be made because instead of a 1 to 1 pairing of photos and sketches, we have a 1 to 5 pairing.

- Instead of the discriminator only seeing one real sketch per epoch, it now sees 5 and so we just take the average of those losses

- The L_1 loss is computed with the generated sketch and the real sketch that is the closest pixel distance from the generated sketch. This is because all five real sketches are equally valid so we only need our generator to match to one of them.

6.1.1 Using Different Generator Architectures in Pix2Pix

Pix2Pix suggests using a U-Net generator but we also explored other generator architectures. Using the Resnet generator from CycleGAN seemed like an obvious choice. To study the effect of skip connections, we implement a generator exactly like U-Net but with the skip connections removed.

We first see that without skip connections, the GAN after just the first epoch has experienced mode collapse. The generated sketches appear to be just black dots. The encoder-decoder structure stills seems to be capturing the location of the targets but without skip connections passing finer details from the encoding layers to the decoding layers, the generator isn't able to produce images to fool the discriminator.

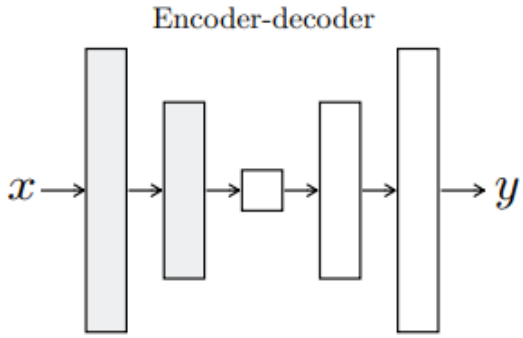


Figure 12: U-Net without skip connections

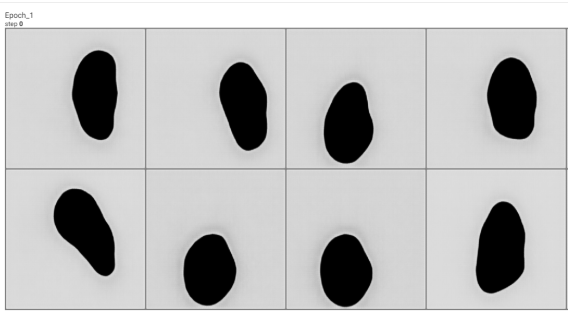


Figure 13: Mode collapse after 1 epoch

Meanwhile U-Net and Resnet seems to be doing equally well after 100 epochs though Resnet has a lower pixel distance loss

We see that U-Net and Resnet both utilize skip connections but in different ways. U-Net has skip connections from encoding layers to decoding layers while Resnet has skip connections from one block to the next inside the bottle neck layer. This demonstrates that both forms of skip connections are helpful in generating images.

Figure 14: U-Net

Figure 15: Resnet

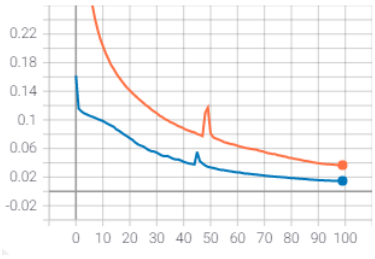


Figure 16: Pixel loss, blue is Resnet, orange is U-Net

So we implement U-Net++ Zhou et al. [21] which has skip connections both from encoding to decoding layers as well as to its neighbouring blocks.

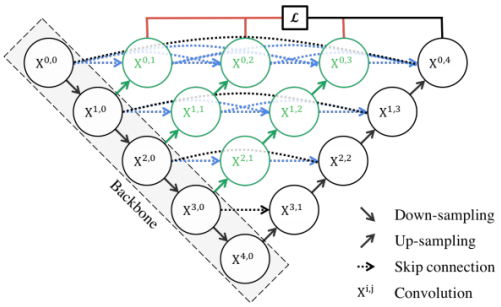


Figure 17: U-Net++

Figure 18: Test Image U-Net++

The authors Zhou et al. [21] offer the explanation that the regular U-Net passes semantically different images through the skip connection as a lot has changed from encoder to decoder, causing learning to be less efficient. In U-Net++, images flowing through the skip connections travel a shorter distance so they have more in common so the network learns quicker.

We conclude that for our generator, a U-Net++ architecture works the best.

6.1.2 Weighing the L_1 Loss

The default parameters suggest a $\lambda = 100$ for our loss function: $L(D, G) = L_{GAN}(D, G) + \lambda L_1(G)$. Here we experiment with different λ values. We try: $\lambda = [0, 10, 100, 200]$.

Running 200 epochs for each, see that the pixel distance loss is identical when λ is 100 or 200 but progressively worse for 10 and 0.

Secondly, when monitoring the test images, we find that the GAN fails to produce legible images when we set $\lambda = 0$.

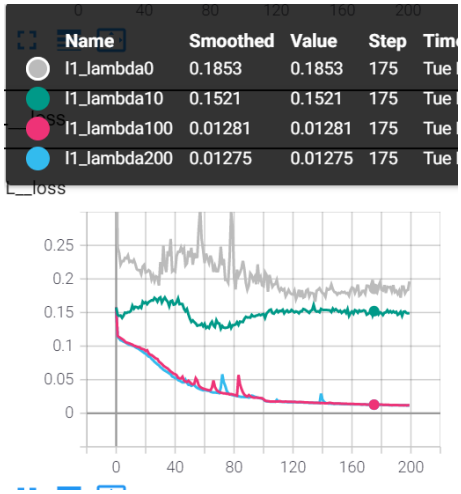


Figure 19: Lambda values for the L1 Loss

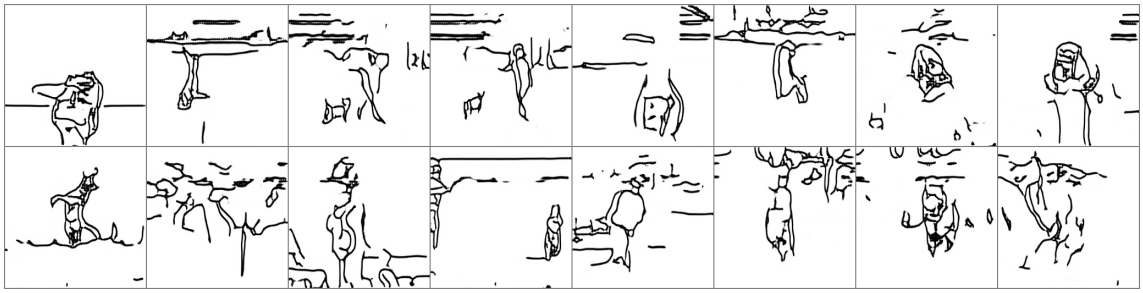


Figure 20: No L1 Loss term produces bad results

We see therefore that Pix2Pix depends quite a lot on the L_1 loss term to help enforce rough approximations of features and without it, Pix2Pix fails to perform well.

6.1.3 Experimenting with the Discriminator

The original Pix2Pix paper suggested that the discriminator should classify images using 70×70 patches. From our experiment, we found that a global discriminator worked better. A global discriminator can be thought of as a 256×256 patch that covers the whole image. The purpose of a PatchGAN is better detect the finer details of a image, but given the amount of white spaces in our sketches, the discriminator doesn't need to be looking at finer details when distinguishing between fakes and reals. Surprisingly, we found that the GAN produces more noisy sketches when using a PatchGAN.

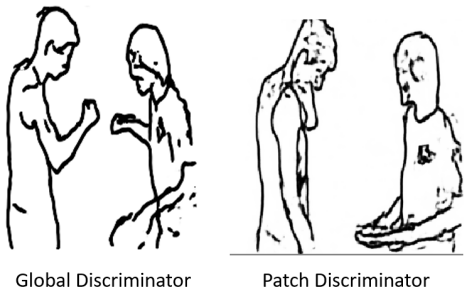


Figure 21: Noisier images when using PatchGAN

6.1.4 Learning Rate, Batch Size and Least Squares GAN

The Learning Rate that was proposed in DCGAN and subsequently used in Pix2Pix was 0.0002. However, Pix2Pix by default trains only 1 image at a time. Because we do training in batches, experiments were done on increasing the learning rate. No noticeably different results were seen when using different learning rates. For batch size, Brock, Donahue, and Simonyan [1] showed that increasing the batch size helped the generator learn more modes and improved image quality. However this was not the case in our situation and training with batch sizes of [4,16,32,64] provided no noticeable differences. In the end, a batch size of 16 was used as this trained the fastest. Finally, comparing Binary Cross Entropy and Least Squares losses also did not impact GAN pictures.

6.1.5 Denoising The Testing Set

One observation that was seen was during training was that the test set produced images with noise while the training set did not.

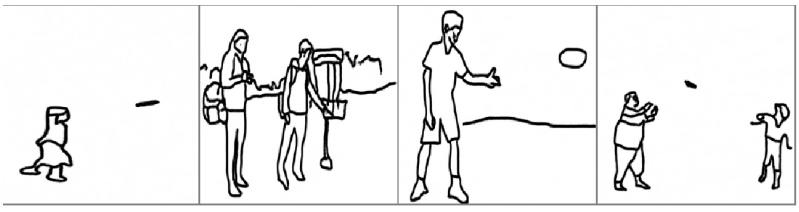


Figure 22: Images from the training set generated by the GAN has no noise

I suspect that this is the result of the generator slightly over-fitting the training set. So to combat this, regularization techniques were investigated. However, adding

dropout or random noise to the input layer of the GAN did not work. As documented in [8], "the generator simply learned to ignore the noise".

The methods that did work were one sided label smoothing and random erasing. One sided label smoothing as discussed in Salimans et al. [16] set real labels to be 0.9 instead of 1.0. This penalizes the discriminator for being too confident in a prediction.

Random erasing was first used in Zhong et al. [20] to increase image classification tasks. Random erasing can be informally thought as dropout specifically for images. This is because random erasing removes a region of the image so the image is partially obscured. The motivation for implementing random erasing was when I noticed the model provides worse sketches when the images had water splashes disrupting continuous contours of the objects. With random erasing, the generator learns how to better sketch images that have a disrupted contour.

Finally, we perform early stopping to prevent overfitting. We carefully monitor the training images and when the improvements each iteration are negligible, we stop the training.

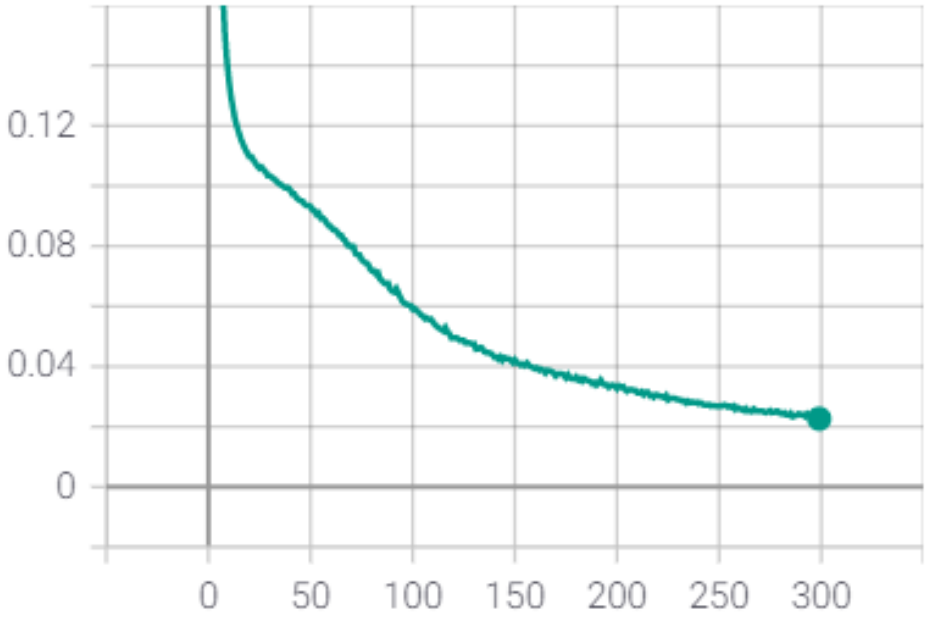


Figure 23: Early stopping, though the pixel loss is still decreasing, we stopped training at 300 epochs as the loss was flattening

Our final model that performed the best has a UNet++ generator, a global discriminator, and utilizes label smoothing, random erasing and early stopping.

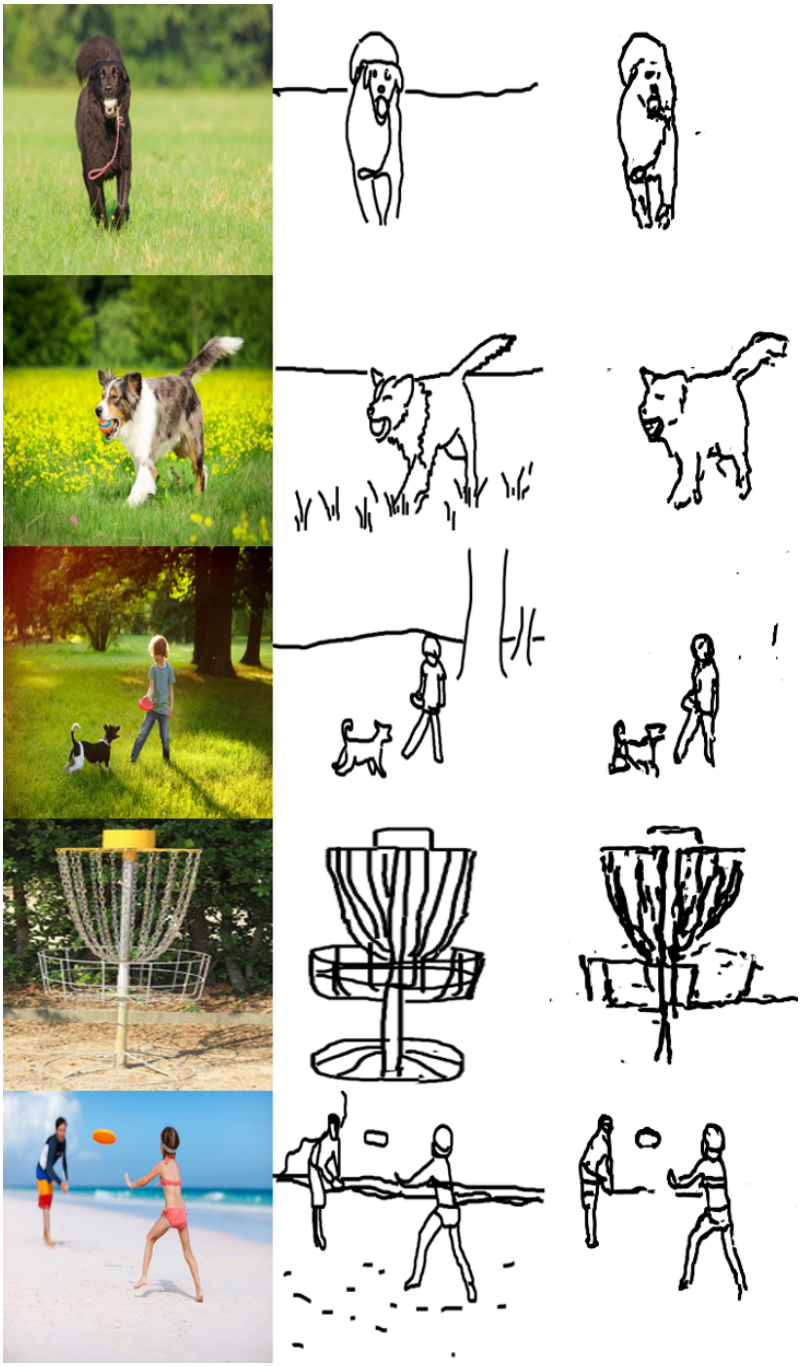


Figure 24: Some final generated images using Pix2Pix from the test set. Middle is original sketch, Right is generated sketch.

6.1.6 Photo-Sketch with CycleGAN

Although Photo-Sketch had great results with Pix2Pix, CycleGAN did not work.

In CycleGAN, although images are unpaired, the elements of set X must be similar to each other and likewise the elements of set Y must be similar with each other.

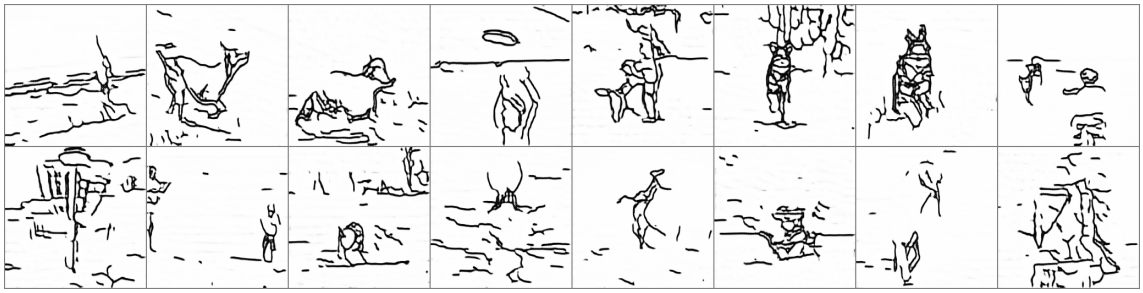


Figure 25: CycleGAN on Photo-Sketch after 50 epochs

If the distribution was diverse and the dataset is small, the discriminators struggles to learn what is real and what is fake and we get the failure shown here where there is a lot of random lines. Though it may seem like our distribution of images are similar and our distribution of sketches are similar, there are a lot of different objects and shapes in our sketches. For example, body parts of humans and animals are round and oblong, but when combined with waves in the water and more rigid frisbee posts, lead to a varied distribution.

6.2 GAN training with Sketchy

6.2.1 Pix2Pix with Sketchy

Pix2Pix with the strategies used on Photo-Sketch lead to mode collapse right away. The reason for this mode collapse was interference from the L_1 term. Recall that

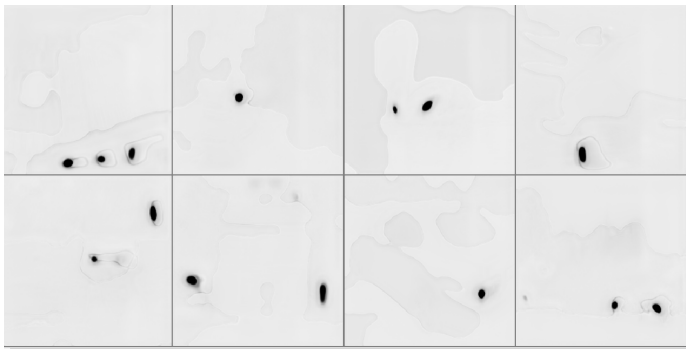


Figure 26: Mode collapse with Pix2Pix training on Sketchy 4 epochs

the Sketchy database did not have pairs of images that were feature aligned. The sketches were more so rough representation of the pictures. As such, when the generator generates a fake sketch, the features of the fake sketch have no way of

lining up with the features of the real sketch through the L1 loss, so the network fails.

So, we try removing the L_1 term, and although we no longer have this specific version of mode collapse, the generator still fails due to the fact that L1 loss is a crucial part of the optimization of Pix2Pix. Without the L1 term, images are purely random and

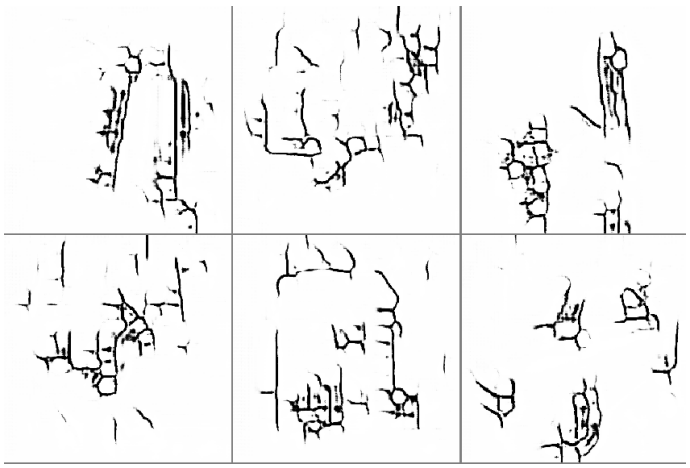


Figure 27: Mode collapse to a lesser extent after 20 epochs without the L1 term

still suffer from mode collapse in the form of pebble like textures. Wasserstein GAN was also attempted as a way to remove this mode collapse but Pix2Pix without L_1 loss no matter what couldn't get good results.

6.2.2 CycleGAN with Sketchy

Since paired training didn't work because our pairs weren't feature aligned, CycleGAN was attempted next. The results for CycleGAN, though not good are much better than that of Pix2Pix. Despite Sketchy having 10 categories of objects compared to 4 categories from Photo-Sketch, the CycleGAN results are similar. This is because most sketches from Sketchy are much more rudimentary compared with sketches from Photo-Sketch. The simpler sketches allows for easier classification by the discriminator and the generator is discouraged from creating complex sketches.

7 Discussion and Conclusion

In conclusion, Pix2Pix works best when the paired images are feature aligned. Without feature aligned pairs of images and sketches, the L_1 loss is high and the model

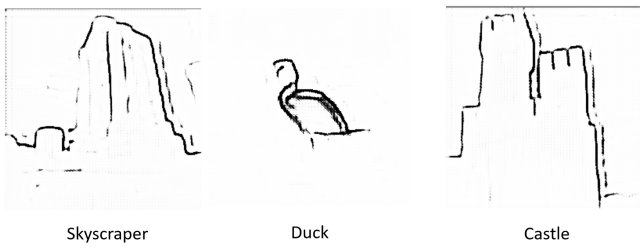


Figure 28: Few CycleGAN sketches with Sketchy dataset

suffers mode collapse. The alternative of removing this L_1 loss when pairs are not feature aligned also doesn't provide good results as $L_1 Loss$ is a key feature in Pix2Pix. For CycleGAN to work well, images do not have to be paired. Instead, you have to make sure that the images from both sets each share similar characteristics with the set.

Some improvements have been made in this project for training with Pix2Pix. UNet++ is a superior generator to both UNet and Resnet. Random erasing works well as a form of data augmentation as well as injecting randomness into the training set. The effect of random erasing helps the generator better overcome discontinued contours in photographs.

Training GANs is a very challenging task. Many solutions that are suggested to work from papers may not work at all for your dataset. It is very much a trial and error game of finding the best set of techniques for your individual task.

References

[1] A. Brock, J. Donahue, and K. Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. 2019. arXiv: 1809.11096 [cs.LG].

[2] Fei-Fei, Justin, and Serena. *Lecture 5: Convolutional Neural Networks*.

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[4] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[5] C. Hesse. *Image-to-Image Demo*. Feb. 2017. URL: https://affinelayer.com/pixsrv/.

[6] G. E. Hinton, S. Osindero, and Y. W. Teh. "A Fast Learning Algorithm for Deep Belief Nets." In: *Neural Computation* 18.7 (2006), pp. 1527–1554. URL: http://dblp.uni-trier.de/db/journals/neco/neco18.html#HintonOT06.

[7] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: ICML'15 (2015), pp. 448–456.

[8] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks". In: *CVPR* (2017).

[9] J. Johnson, A. Alahi, and L. Fei-Fei. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. 2016. arXiv: 1603.08155 [cs.CV].

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: (2012). Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[11] M. Li, Z. Lin, R. Měch, E. Yumer, and D. Ramanan. "Photo-Sketching: Inferring Contour Drawings from Images". In: *WACV*. 2019.

[12] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley. *Least Squares Generative Adversarial Networks*. 2017. arXiv: 1611.04076 [cs.CV].

[13] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: (Nov. 2015).

[14] S. Rome. *An Annotated Proof of Generative Adversarial Networks with Implementation Notes*. Aug. 2017. URL: https://srome.github.io/An-Annotated-Proof-of-Generative-Adversarial-Networks-with-Implementation-Notes/.

[15] O. Ronneberger, P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[17] P. Sangkloy, N. Burnell, C. Ham, and J. Hays. "The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies". In: *ACM Transactions on Graphics (proceedings of SIGGRAPH)* (2016).

[18] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. "Instance Normalization: The Missing Ingredient for Fast Stylization". In: *CoRR* abs/1607.08022 (2016). arXiv: 1607.08022. URL: http://arxiv.org/abs/1607.08022.

[19] Y. Wu and K. He. "Group Normalization". In: *CoRR* abs/1803.08494 (2018). arXiv: 1803.08494. URL: http://arxiv.org/abs/1803.08494.

[20] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. *Random Erasing Data Augmentation*. 2017. arXiv: 1708.04896 [cs.CV].

[21] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, and J. Liang. *UNet++: A Nested U-Net Architecture for Medical Image Segmentation*. 2018. arXiv: 1807.10165 [cs.CV].

[22] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. arXiv: 1703.10593 [cs.CV].