Image-to-Image Translation with Conditional Adversarial Networks

Abhishek Bhave
UBitName: abhave Person Number: 50289049
Department of Computer Science
University of Buffalo
Buffalo, NY 14214
abhave@buffalo.edu

Abstract

The goal of this project is to develop a conditional adversarial network system that learns a loss that tries to classify if the output image is real or not while simultaneously training a generative model to minimize the loss. Our basic aim behind this project is to look at Generative Adversarial Networks and their variants like Conditional Adversarial Networks and demonstrate that conditional GANs can produce good results on a wide variety of problems.

We are looking into conditional GANs as a generalized form of network for image to image translation which can learn a mapping from an input image to an output image as well as learn a structured loss function to this mapping which makes it possible to generalize our task of image to image translation.

1 Introduction

Image-to-image translation is a computer vision and machine learning problem where the goal is to learn the mapping between an input image and an output image. Image to image translation has various applications like collection style transfer, object transfiguration, season transfer and photo enhancement. As observed from Figure 1 below, we can see that GANs can be used to translate images of horses to zebra or translate an image of winter to summer.

Figure 1 Example of Image to Image Translation

In a simpler terminology we can say that image-to-image translation is the task of translating one possible representation of a scene/image into another representation of the same scene/image, given sufficient training data. Our task in this project is to try and apply image to image translation to conditional generative adversarial networks to convert images containing edges to images containing scenery.

2 Dataset

For the purpose of this project we are going to be using the Computational Visual Cognition Laboratory dataset called as LabelMe. It is an open source dataset available here - http://cvcl.mit.edu/database.htm. This dataset consists of Urban and Natural Scene Categories of which we are going to be working with the following:

- Tall Building dataset – 356 images
- Mountain dataset – 374 images
- Open Country dataset – 410 images
- Highway dataset – 260 images

Figure 2 Dataset Image

I tried merging all the above 4 datasets to form a single dataset and tried working with that but even after 4 hours of training the images were of very poor quality and after about 6 hours of training I faced the Google colab runtime disconnected error because of full usage of RAM.

I have split each of these datasets into a 80-10-10 split for training, testing and validation respectively. The model I am using required the input to be the original images as concatenated to the image containing edges of the original image.

3 Data Preprocessing

To feed the data to my model I had to do some data processing I had to concatenate the original image to the image containing edges of the original image. For this purpose, I first tried out Canny edge detection to get the edges of a particular image. With Canny edge detection the model, the output images were not of great quality. This maybe because of Hysteresis thresholding values that work well for one image may not work well for another. Also, Canny edge detector often requires a number of preprocessing steps in order to obtain a good edge map.

So, we use Holistically-Nested Edge Detection (HED) which attempts to address the limitations of the Canny edge detector through an end-to-end deep neural network. HED describes a deep neural network capable of automatically learning rich hierarchical edge maps that are capable of determining the edge/object boundary of objects in images.

Holistically-nested edge detection (HED), performs image-to-image prediction by means of a deep learning model that leverages fully convolutional neural networks and deeply-supervised nets. HED automatically learns rich hierarchical representations that are important in order to resolve the challenging ambiguity in edge and object boundary detection

Figure 3 Original Image on left and HED image on right

4 GANs

For the purpose of this project we will look at the basic implementation which is GANs and then try to fit our data to the conditional GANs implementation.

GANs are Generative adversarial networks which are deep neural net architectures comprised of two networks, Generator and Discriminator, pitting one against the other. GANs' can be used to mimic any distribution of data. That is, GANs can be taught to create something which is similar to the original in any domain: images, music, speech, prose. For the purpose of our project we will focus on the image domain.

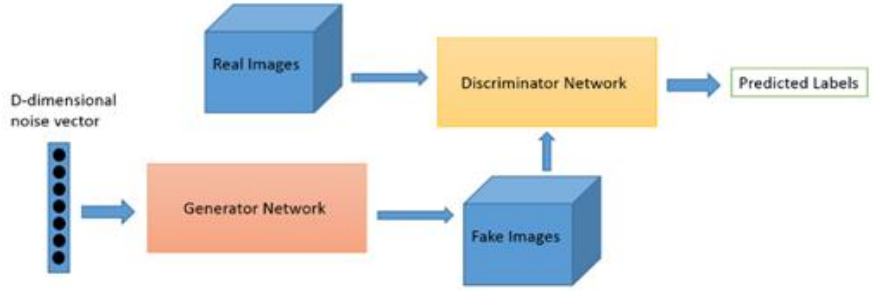

Figure 4 GAN architecture

4.1 Generator

A generator is a network which take the input as noise vector and generates an image from that noise distribution. The generator is trained on the input images and once trained, the generator can produce images from noise.
The goal of the generator is to create images with D(x) = 1 (matching the real image). So, we can train the generator by backpropagating the target value all the way back to the generator.

The job of the generator in simple words is to take in a noise distribution as the input and produce an image based on that noise distribution which should look real to the discriminator.

Generally, the generator learns to map from a random noise to a image of our domain The basic objective of the generative network's training is to increase the error rate of the discriminative network i.e. to basically fool the discriminator. The generator loss function changes based on whether it succeeds in fooling the discriminator. Typically, the generator is seeded with randomized input that is sampled from a noise distribution.

The goal of the generator is to minimize the loss as compared to the discriminator whose goal is to maximize this objective function.

This is the way the generator in the GANs work. As compared to GANs, in conditional generative adversarial networks, the generators learn a mapping from observed image x and random noise vector z, to y, G: $\{x, z\} \rightarrow y$.

4.2 Discriminator

The purpose of the discriminator is to basically classify whether the output of the generator is real or not. If the input is real, we want D(x)=1. If it is generated, it should be zero. By the above-mentioned procedure, the discriminator identifies features that contribute to real images.

While the discriminative network distinguishes candidates produced by the generator in terms of a sigmoid distribution, we use a known dataset serves as the initial training data for the discriminator. We train the discriminator by presenting it with samples from the training dataset.

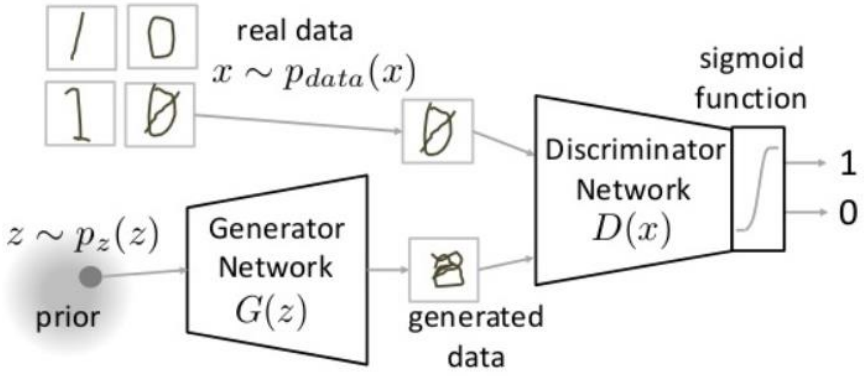

Figure 5 Example of GAN

In figure 5 we can see a sample diagram of a GAN which has a generator which has input as z noise distribution and emits an image. The discriminator D is trained with real images and then output of generator is fed to discriminator which provides a sigmoid result.

5 Conditional GANs

Conditional GANs are a basic solution to image-to-image translation problems. These types of networks not only learn the mapping from input image to output image, but also learn a loss function to train this mapping

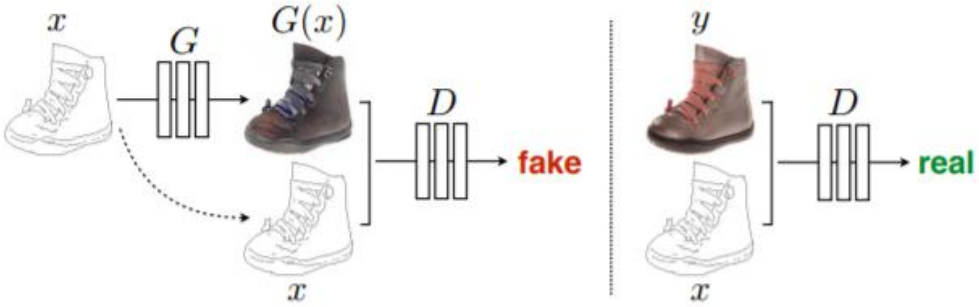

Figure 6 Example of conditional GAN

What is different in conditional GANs is the input that we provide to the generator. In GANs the input is random noise vector z to output image y, G: z → y, whereas in conditional GANs we give the input as observed image x and random noise vector z, to y, G: {x, z} → y.

As we can observe from the above figure 6, the input to the generator G is random noise as well as an image x. The input and output to the discriminator D remains the same. The function of the discriminator also remains the same.

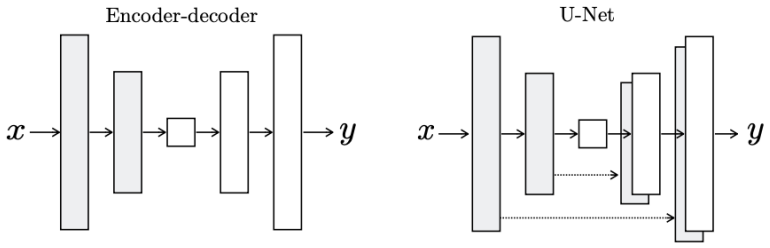

Figure 7a 7b Architecture of generator in conditional GAN

In our implementation of conditional GANs we give random noise as well as observed image as the input to the generator. But in our implementation, we use a modified generator which is generator with skips or a U-Net.

In simple generator architectures like in figure 7a we first reduce the size of the input and then to a single vector and then expand it again to the original size to get a generated image. This causes a bottleneck layer. Due to this bottleneck layer sometimes, we may lose out desirable information and very low-quality images will be generated.

To circumvent this problem, we use a modified generator called as a U-Net as described in figure 7b. In a U-Net we connect the first layer to the last layer and the second layer to the second last layer and so on and so forth. This is basically a encoder decoder network with skip connections.

5.1 Loss Function

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

Figure 8 Loss Function of conditional GAN

The above figure 8 depicts the loss function of a conditional GAN where G tries to minimize this objective against an adversarial D that tries to maximize it, i.e. G∗ = arg minG maxD LcGAN (G, D).

The evaluation metric that we use for the conditional GAN is the L1 loss which can be described as follows:

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

Figure 9 L1 loss

The overall objective is given as follows:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G).$$

Figure 10 Overall loss function

Here we can say that our objective is to minimize the generator loss and maximize the discriminator loss as well as minimize the L1 loss.

6 Results

I tried merging all the 4 datasets to form a single dataset and tried working with that but even after 4 hours of training the images were of very poor quality and after about 6 hours of training I faced the Google colab runtime disconnected error because of full usage of RAM.

The results for all the 4 datasets taken individually are as follows:

Figure 11 Output for Mountain dataset

In the above output, the first row is the image containing edges of the original image. The second row is the original image and in the third row we have the predicted image.

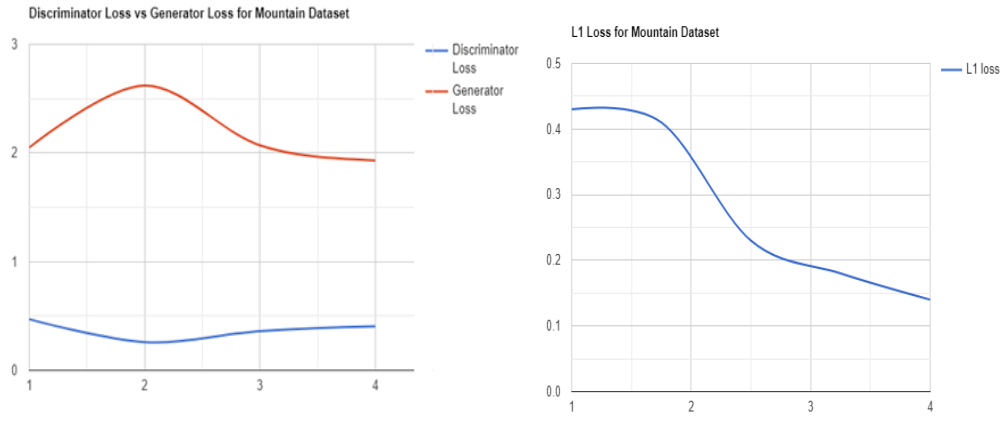

Figure 12 Discriminator vs Generator Loss and L1 loss for Mountain Dataset

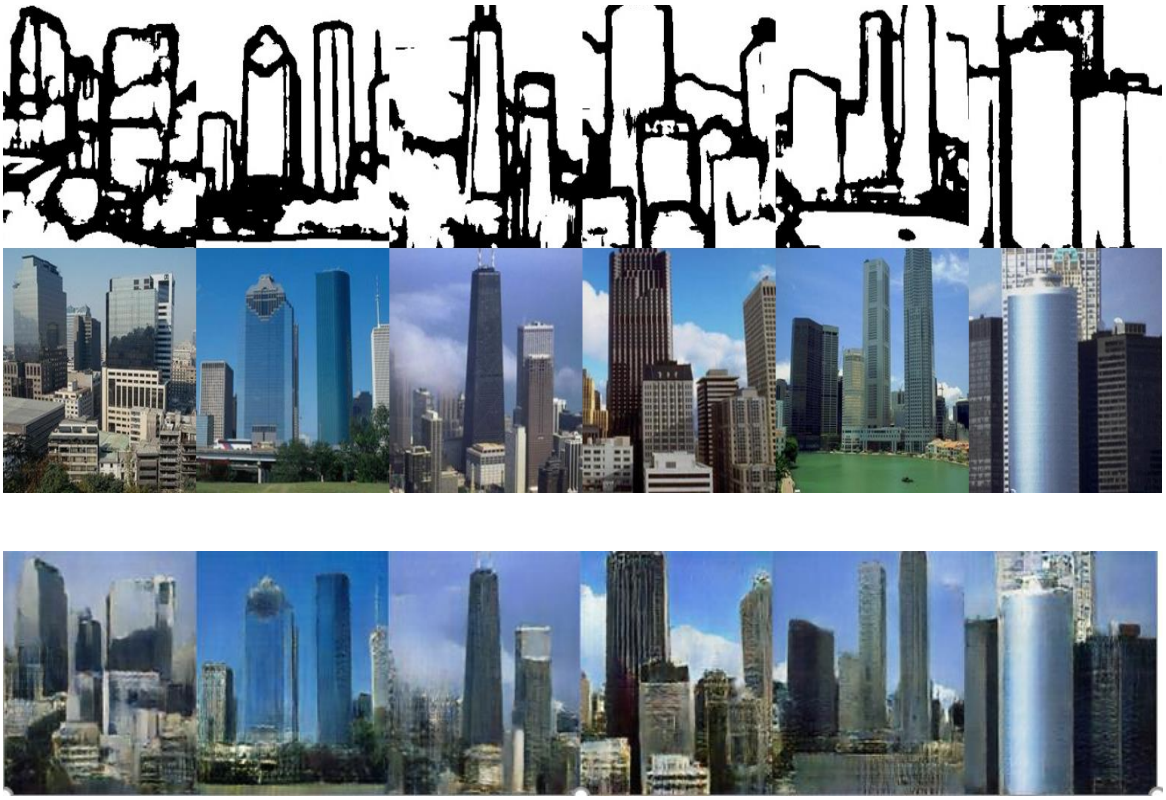

Figure 13 Output for Tall Building dataset

In the above output, the first row is the image containing edges of the original image. The second row is the original image and in the third row we have the predicted image.

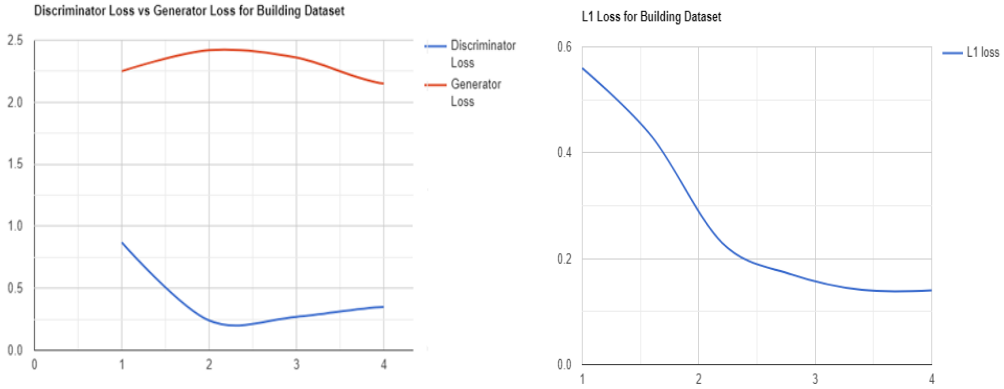

Figure 14 Discriminator vs Generator Loss and L1 loss for Tall Building Dataset

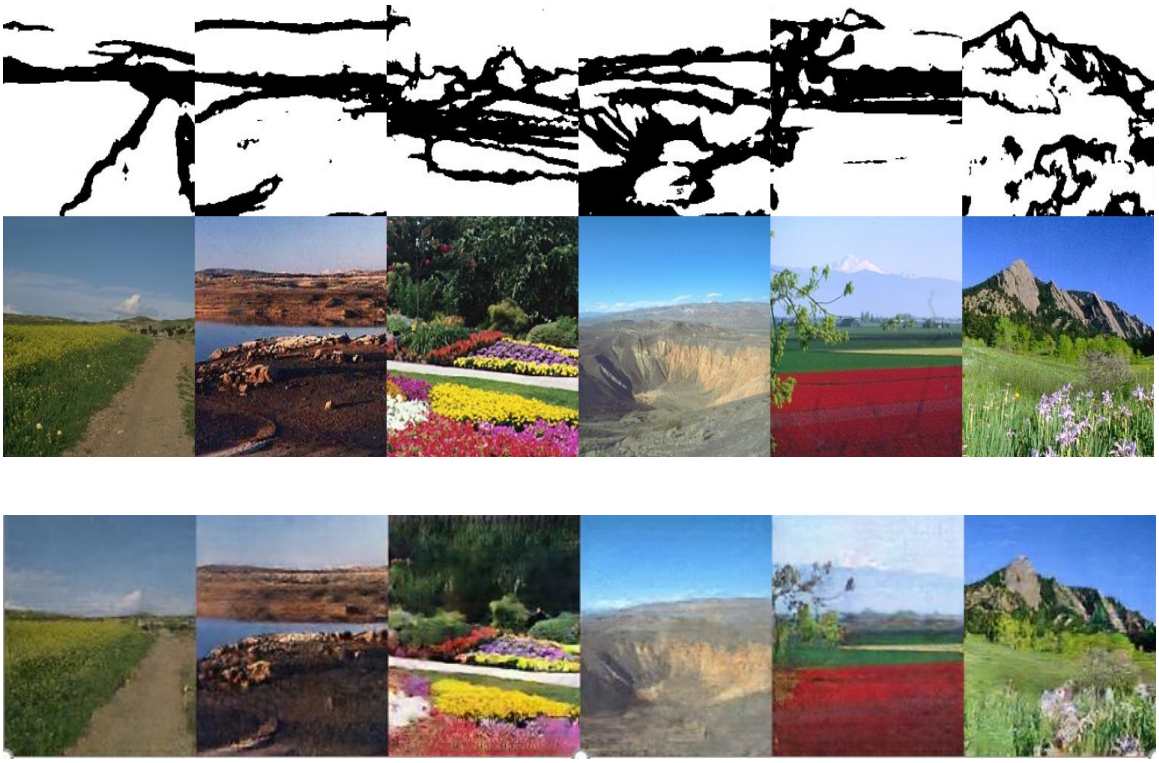

Figure 15 Output for Open country dataset

In the above output, the first row is the image containing edges of the original image. The second row is the original image and in the third row we have the predicted image.

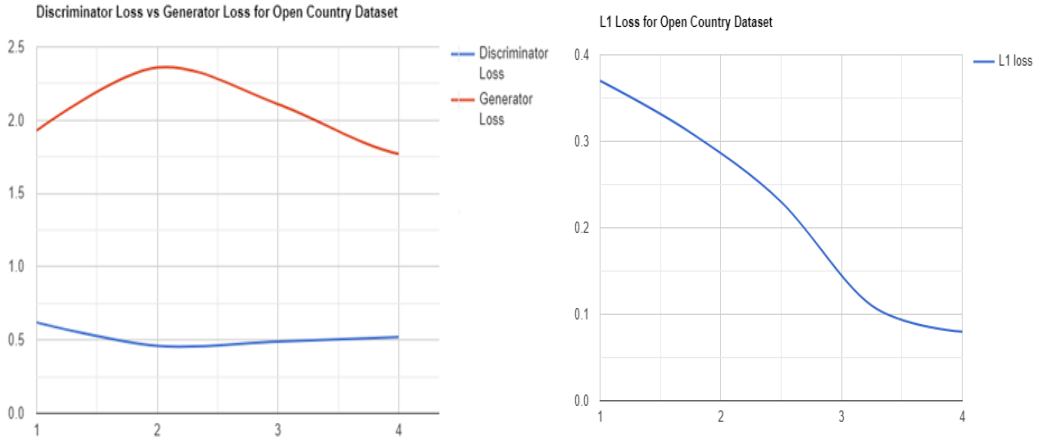

Figure 16 Discriminator vs Generator Loss and L1 loss for Open Country Dataset

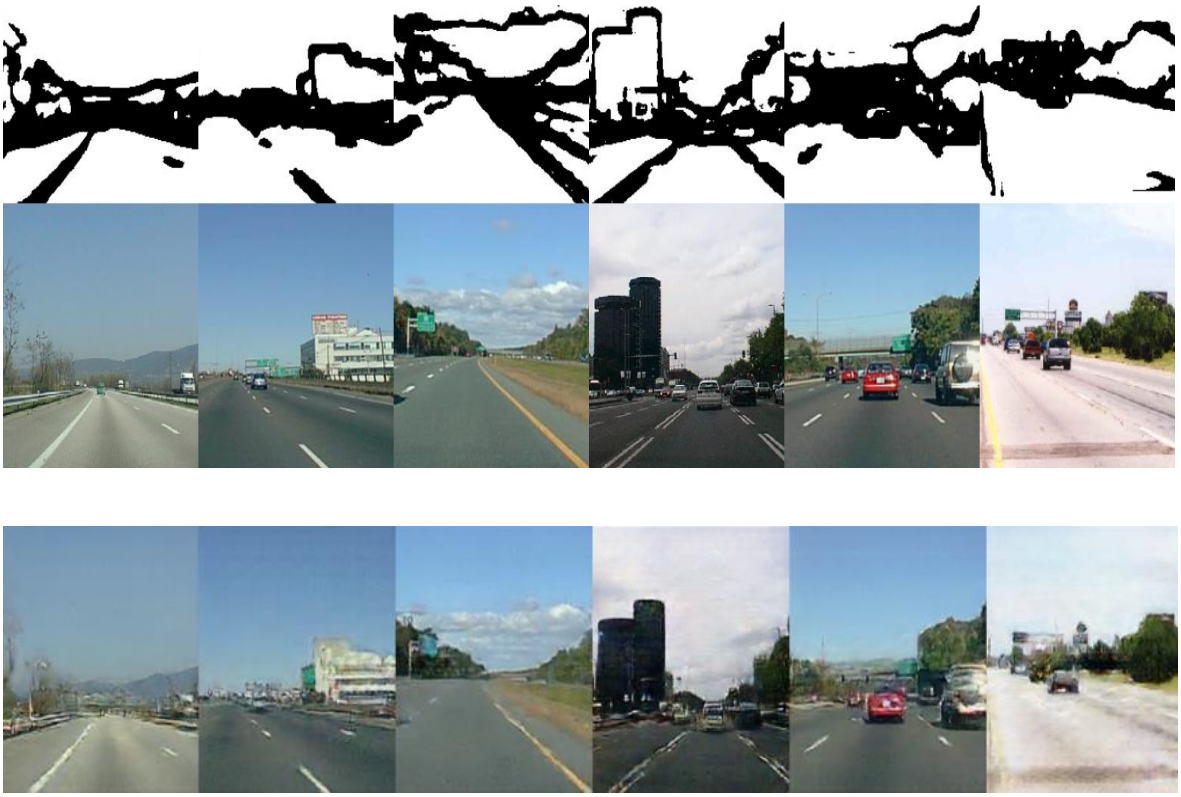

Figure 17 Output for Highway dataset

In the above output, the first row is the image containing edges of the original image. The second row is the original image and in the third row we have the predicted image.

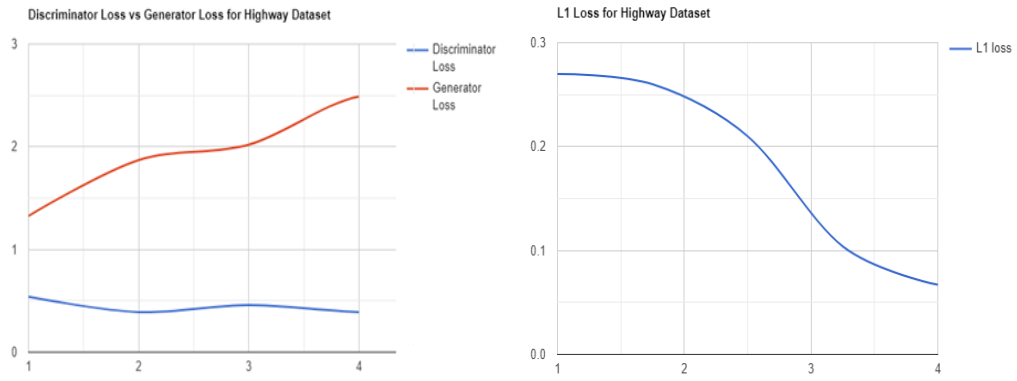

Figure 18 Discriminator vs Generator Loss and L1 loss for Highway Dataset

References

[1] - https://towardsdatascience.com/image-to-image-translation-69c10c18f6ff

[2] - https://skymind.ai/wiki/generative-adversarial-network-gan

[3] - https://github.com/s9xie/hed/blob/master/README.md

[4] - https://en.wikipedia.org/wiki/Generative_adversarial_network

[5] -https://arxiv.org/pdf/1611.07004.pdf Image-to-Image Translation with Conditional Adversarial Networks