Image Generation and Transformation with GAN Project Report

PROFESSOR: Kunpeng Zhang
BUDT 758B Spring 2019

Team Member:
Yefei Fan
Jiahui Meng
Bing Wu

2019/5/15

Outline

1.Introduction

The recent unprecedented advances in computer vision and machine learning are mainly due to the development of deep neural architectures, existence of abundant labeled data and more powerful computer. On the one hand, deep convolutional neural networks trained on large numbers of labeled images provide researchers powerful image representation that can be used for a wide variety of tasks including recognition, detection, and segmentation. On the other hand, there is still an emerging need to generate images using computer vision technology for game production and obtaining abundant annotated data for image to image transformation remains a cumbersome and expensive process in the majority of application such as autonomous driving, where a semantic segmentation network is required to be trained to perform roads, cars, pedestrians detection.

Generative Adversarial Network, called GAN, is a method of unsupervised machine learning algorithms. Two neural networks contest with each other in a zero-sum game framework. This technique can generate photographs that look at least superficially authentic to human observers, having many realistic characteristics.

Our objective is to use GAN method to generate digits and then use conditional-GAN method to perform image to image translation. We use the traditional GAN method to perform digits generation using the data from MNIST digit dataset. In an unconditioned generative model, there is no control on models of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process. Therefore, we explore the performance of conditional-GAN in image to image translation task using maps dataset.

2.Dataset

2.1. MNIST dataset

We use the digit-recognizer dataset obtained from the Kaggle website in previous phase. The data files, train.csv and test.csv, contain gray-scale images of hand-drawn digits, from zero to nine. The dataset contains 42000 digit images in the training set and 28000 digit images in the testing set.

Each image is a black-and-white picture and is with 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel values of the associated image.

2.2. Maps dataset

In phase two, we use maps dataset from website. It includes three sub-datasets: train, validation and test. The data are in the format of paired aerial and map view of the same region scraped from Google Maps. The dataset consists of 1097 training image pairs, 1098 validation image paris, and 1098 test image pairs. Each image composed of RGB channels and of size 600 pixels * 600 pixels. For example:

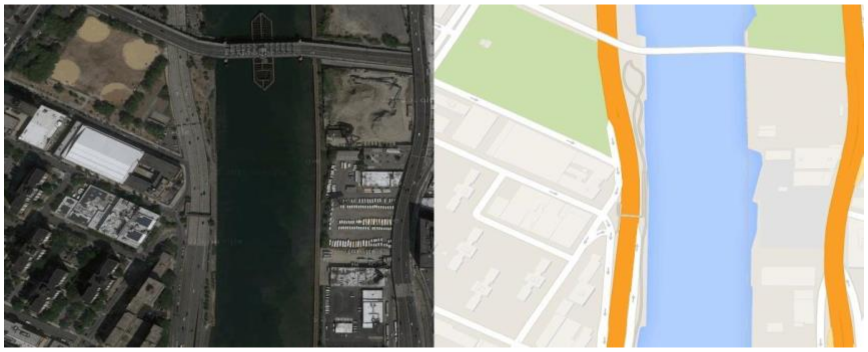

Figure 1: A sample of Maps training dataset

(source: https://people.eecs.berkeley.edu/~tinghuiz/projects/pix2pix/datasets)

Data Preprocessing: The original image size is 3 x 600 x 600. We first resize the input images to 3 x 256 x 256 to reduce computation cost.

3. Methodology

3.1. Highly Effective Platform

3.1.1. Instance for MNIST

We use EC2 from Amazon Web Service (AWS) to implement our code running for MNIST dataset, because it is more powerful and time-saving, compared to our local computer system with only 3 CPUs. The instance we created includes following details:

Table 1: Details of computational instance used

Name	GPUs	vCPUs	Memory(GiB)	GPU Memory(GiB)
g3.16xlarge	4	64	488	32

3.1.2. Instance for Maps

We also use EC2 from AWS to implement our code running for Maps dataset and created one instance with more CPUs to speed up image processing. The instance includes following details:

Table 2: Details of computational instance used

Name	vCPUs	Mem(GiB)	Storage(GB)	Network Performance(Gbps)
m5.24xlarge	96	384	EBS-only	25

3.2. Algorithm Explanation

We used generative adversarial networks (GAN) as our model structure for generating pictures. The GAN mainly consists of two parts: generator and discriminator.

The task of generator is to produce a fake image which is as much indistinguishable from a real image as possible and confuse the discriminator. The generator takes a random noise as input and tries to produce samples in such a way that Discriminator is unable to determine if they come from training data or Generator.

The task of discriminator is to distinguish between real image and fake image from the generator, given the reference input image. It learns in a supervised manner by looking at real and generated samples and labels telling where those samples came from. In some sense, Discriminator component replaces the fixed loss function and tries to learn one that is relevant for the distribution from which the training data comes. Its input parameter is X, which represents a picture, and its output is D(X), which represents the probability that X is the real picture. If the output is 1, then the probability that x is the real picture is 100%. If the output is 0, then the probability that x is the real picture is 0%.

For image translation task, we used conditional GAN to approach this task. Conditional GAN is an extension of original GAN if both generator and discriminator are conditioned on additional information Y. Y is supplementary information for input data in any types. In conditional GAN, Y is regarded as additional input fed into both the discriminator and generator.

In the generator, the noise Z and extra information Y are combined in joint hidden representation, and the network framework allows for huge flexibility in hidden representations. In the discriminator, output of the generator and target output are regarded as input of discriminative function.

The objective function of the conditional GAN is:

$$(G^*,D^*) = \mathit{arg}\ \min \max(L(G, D) + \lambda L(G))$$

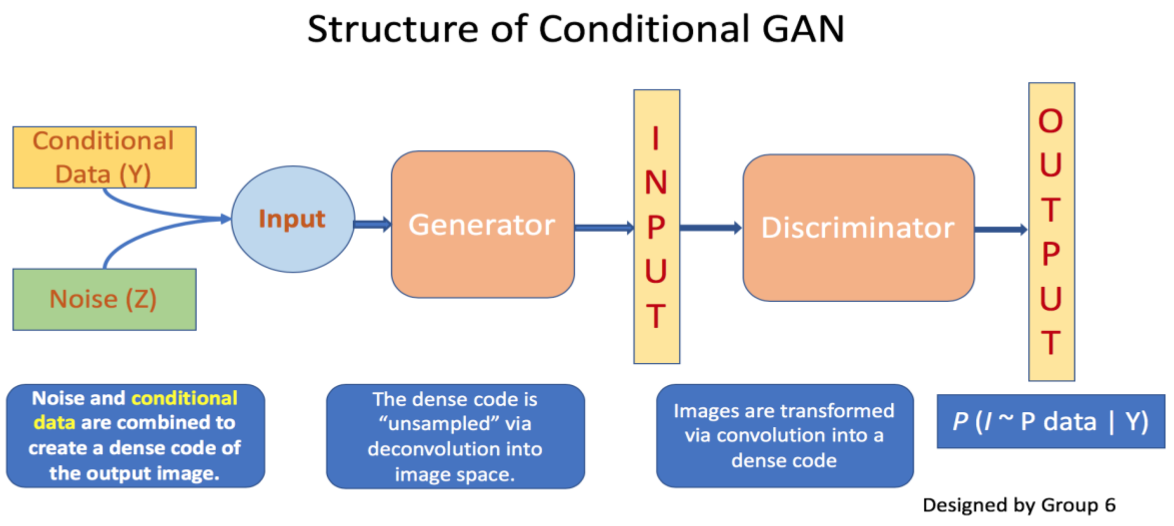
Structure of Conditional GAN
Conditional Data (Y)
Noise (Z)
Input
Generator
INPUT
Discriminator
OUTPUT
Noise and conditional data are combined to create a dense code of the output image.
The dense code is "unsampled" via deconvolution into image space.
Images are transformed via convolution into a dense code
P (I ~ P data | Y)
Designed by Group 6

Figure 2: Structure of conditional GAN

4. Network Structure

We develop our training framework using Keras in Python 3.

4.1. Structure for generating MNIST digits

4.1.1. Generator

The input of generator is a list consisting of an input image sample of size (28,28,1) and a latent variable (noise), and the output of the generator is an image with the same size of (28,28,1). Weights are initialized from the normal distribution with the mean of 0 and a standard deviation of 0.02. The activation functions of the generator are LeakyReLU for hidden layers and tanh for the output layer.

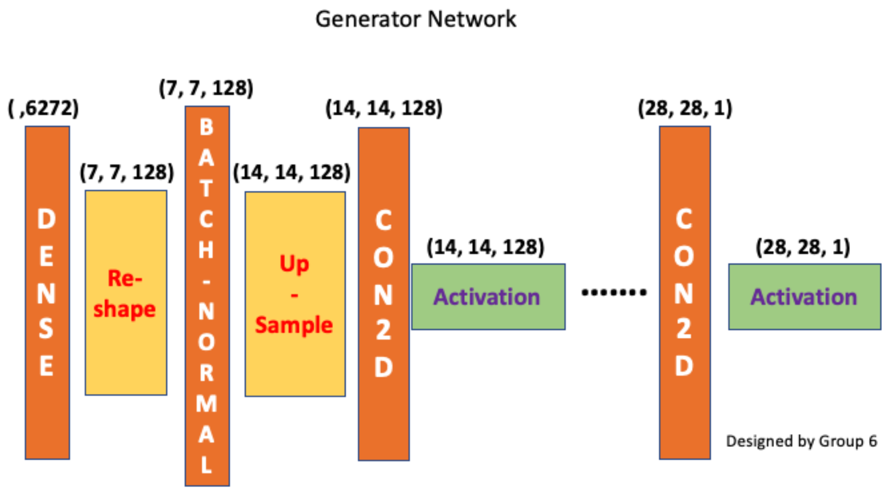

Figure 3: Generator network

4.1.2. Discriminator

We use convolutional neural network as the architecture of the discriminator. Weights are initialized from the normal distribution with the mean of 0 and a standard deviation of 0.02. The input dimension is 28x28x1 and the output is a list of a vector of 1 (probability of fakeness) and vector of 10 (digit classification). We use dropout with a rate of 0.3 to avoid overfitting. The activation functions of discriminator are LeakyReLU for hidden layers, linear and Softmax for output layer.

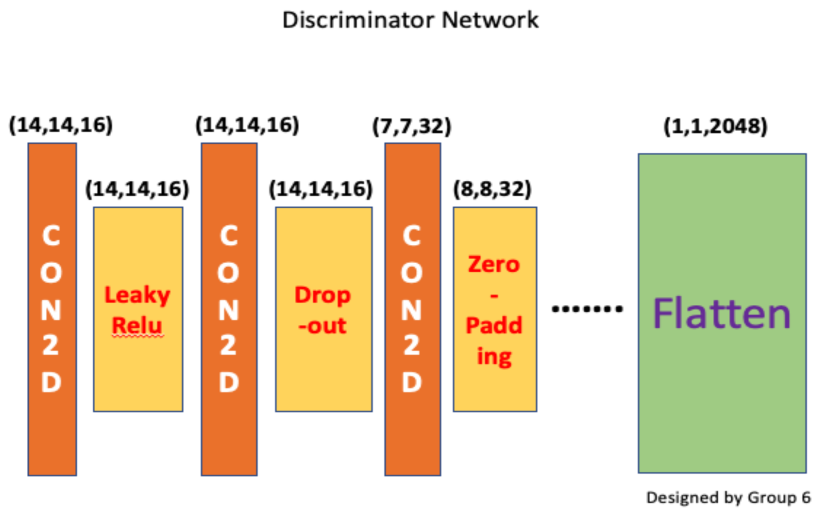

Figure 4: Discriminator network

4.1.3. Compiling Model

Then, we use optimizer *'RMSprop'* with a learning rate 0.00005 and loss function *'sparse categorical cross-entropy'* to compile the generator model and discriminator model together.

4.2. Structure of Conditional-GAN

We use U-Net structure to for the generator in conditional-GAN. The U-Net generator is an encoder-decoder network with symmetrical long skip connections. The network consists of 7 encoding layers and 7 decoding layers. Each encoding and decoding block follows the form of convolution/deconvolution-BatchNorm-LeakyReLU. Figure 5 illustrates the U-Net architecture.

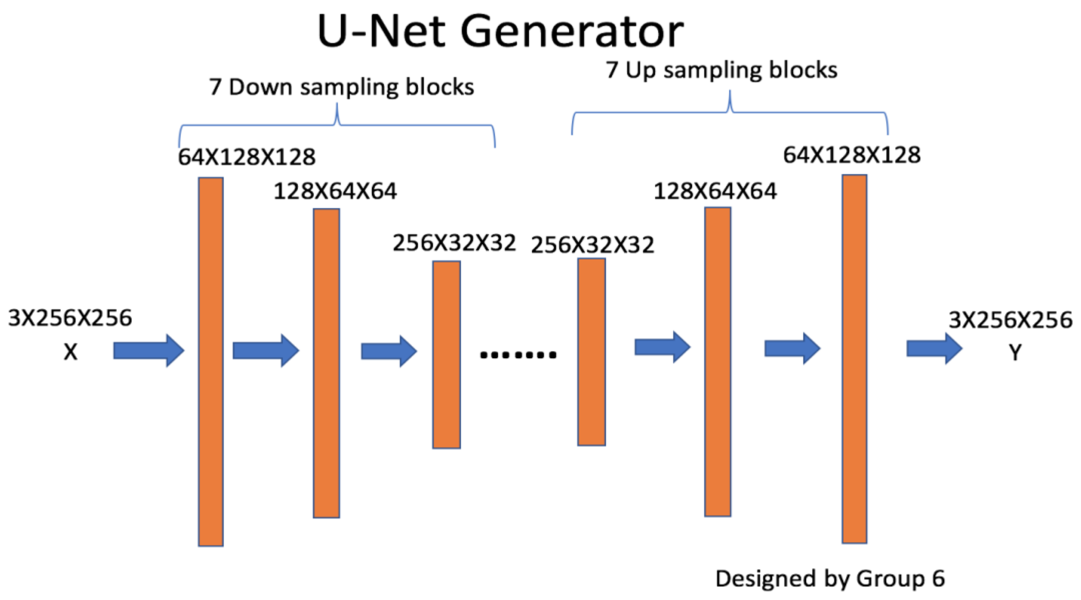

Figure 5: U-Net generator structure

5.Result

5.1. For digits image generation

5.1.1. Performance overview

We can see the result summary from the table and plot below. We will choose the three representative outputs for process demonstration.

Table 3: Mnist model performance for 14000th (last) result

Discriminator loss	Accuracy	Generator loss
0.819696	**95.31%**	**0.735431**

The first plot below is the result from the 200th (first) epoch, which is very messy and blurry so that we cannot find any information from generated images. The second plot is the result from the 2600th epoch. We can see the result is pretty good and the digits generated by our model is recognizable. The third plot is the result from the 8000th epoch, in which we get the highest testing accuracy. We can see the all digits very clearly.

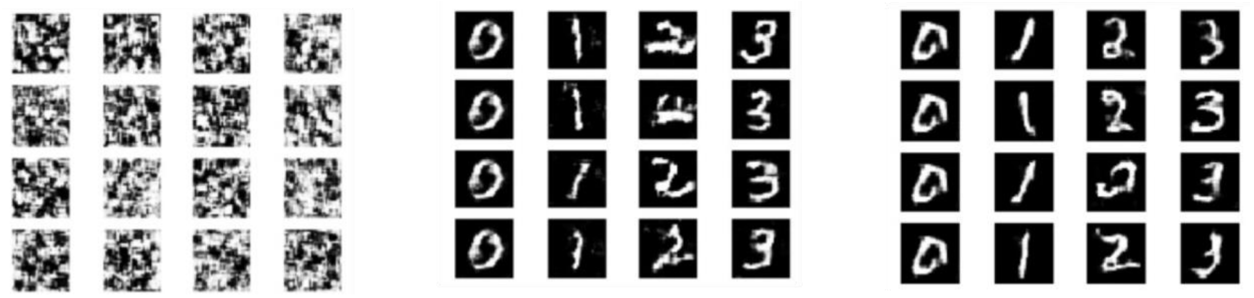

Figure 6: Main results from MINIST GAN model

5.1.2. Loss Plot

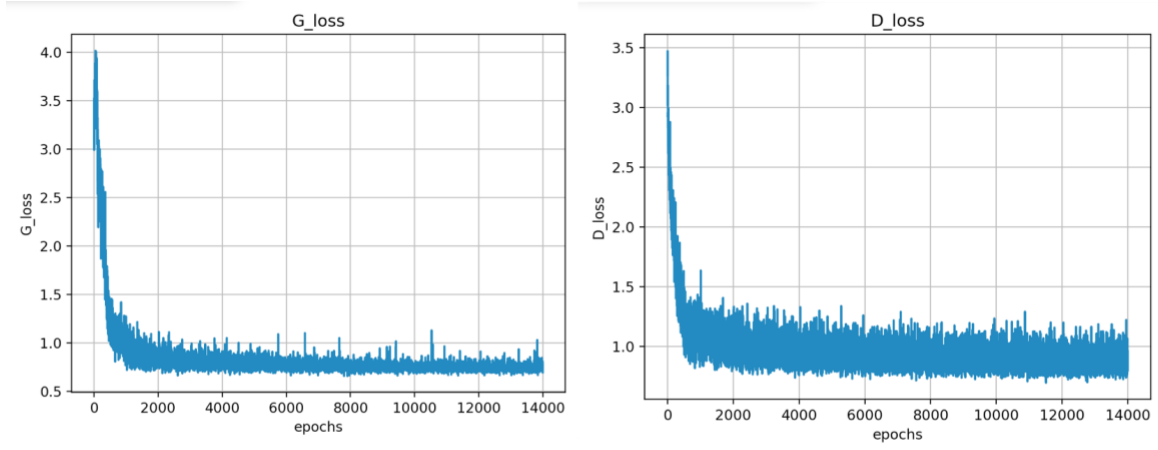

Figure 7: Loss plot from MINIST GAN model

The plot above is the loss change of Generator network (left) and Discriminator network (right). As we can see from the loss plot, after 8000 training iterations, loss becomes close to zero and the performance of our model is getting more stable.

5.1.3. Accuracy Plot

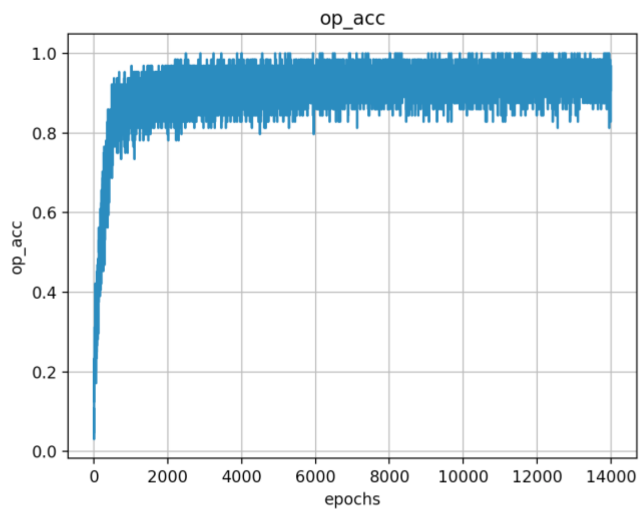

Figure 8: Accuracy plot of MINIST model

As we can see from the accuracy plot, the result is corresponding to the result we observed in loss plot. The generated digits image is performing well to fool the discriminator.

5.2. For maps image generation

5.2.1. Performance overview

We can see the result summary from the table and plot below. We run 23 epochs with 200 batches each. We will choose the three representative outputs for process demonstration.

Table 4: Maps model performance for 23th (last) epoch result

Discriminator loss	**Accuracy**	**Generator loss**
0.000405	**100%**	**0.026529**

Main results review:

1) The plot below is the result from the 0 (first) epoch, which is very blurry so that we cannot find any information from generated images.

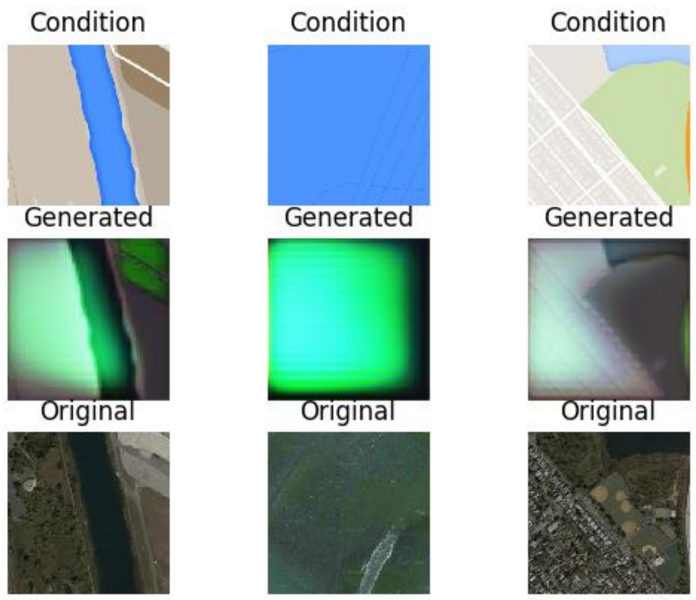

Figure 9: Output from 1st epoch

2) The plot below is the result from the 10th epoch. We can see the result is better and the maps generated in the middle is recognizable.

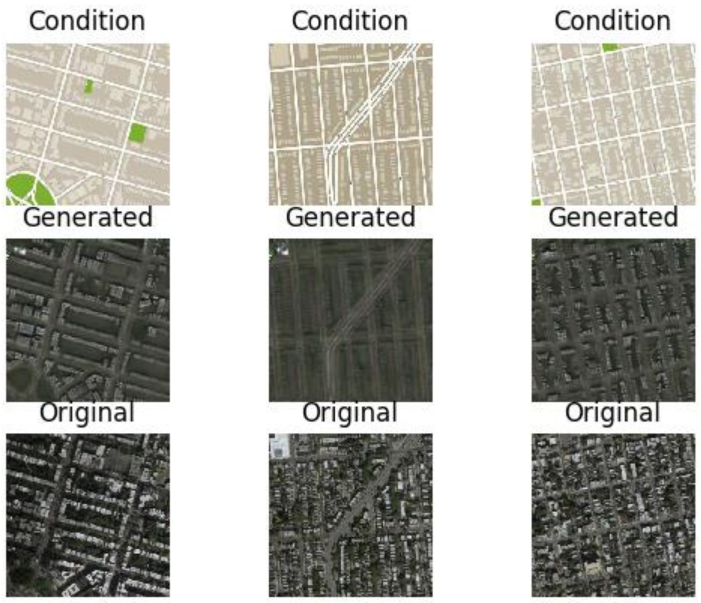

Figure 10: Output from 10th epoch

3) The plot below is the result from the 23rd epoch, in which we get the highest accuracy. The generated maps are quite desired and we can see the all maps very clearly.

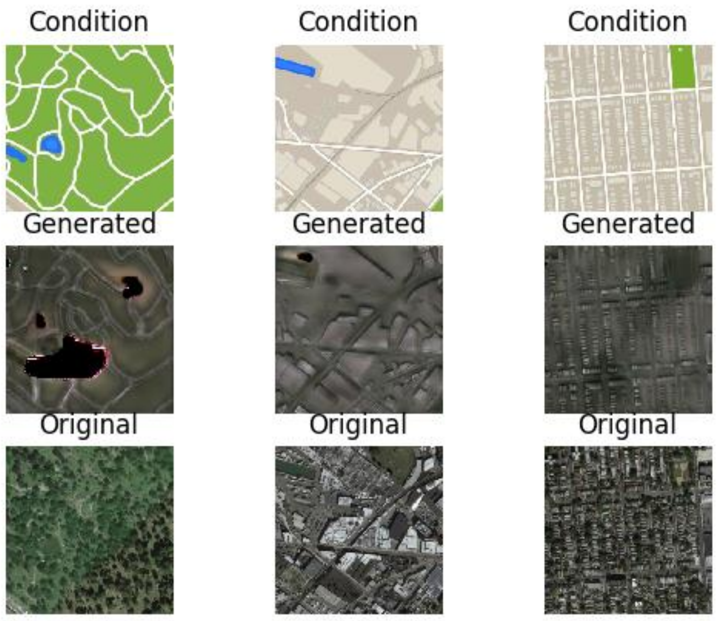

Figure 11: Output from 23th epoch

5.2.2. Loss Plot

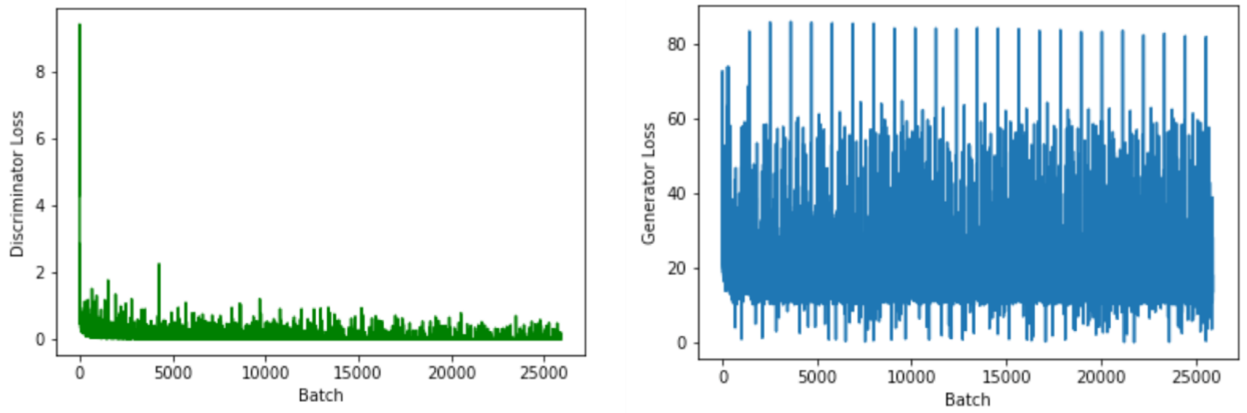

Figure 12: Loss plot from Maps model

As we can see, Discriminator loss decreases as batches are processed. When the batch reaches 15000th, the loss tends to become very low and closer to 0. For Generator loss, during each epoch (1097 batches), the loss decreases dramatically.

5.2.3. Accuracy Plot

The plot in right is the accuracy performance. We can see the accuracy can quickly reach to 100%, during each epoch.

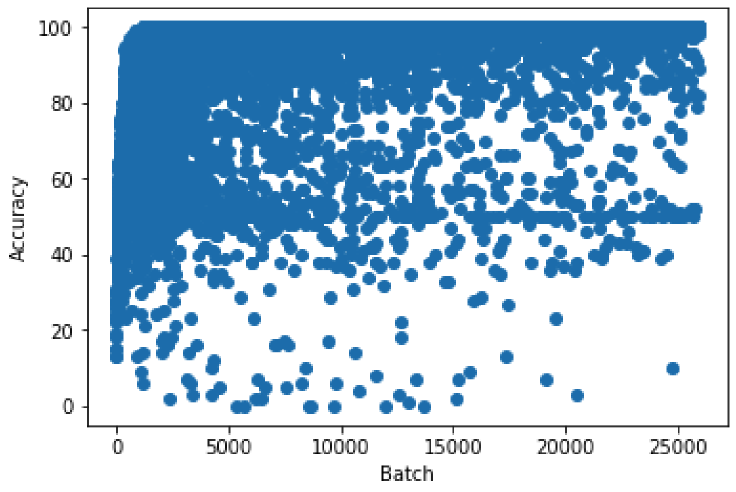

Figure 13: Accuracy plot from Maps model

6.Business value

6.1. Fashion Industry product image generation or style transformation

Our project can be implied in fashion industry. Pursuing the fashion, the high-quality life of having intension is the world trends. Apparel companies focus on designing unique and fashionable products for customers. From skirts to coats, hats and handbags and boots, the variety and style are dazzling. Our model can help generate unlimited apparel picture samples, including different shapes, colors, characters, textures, so designers can use them for inspirations. For example, they can choose specific raw materials according to their explanation of pictures. Especially, customized cloth industry is profitable, lots of customers, such as celebrities want to choose appropriate match in terms of color, cloth, and style for different events and purpose.

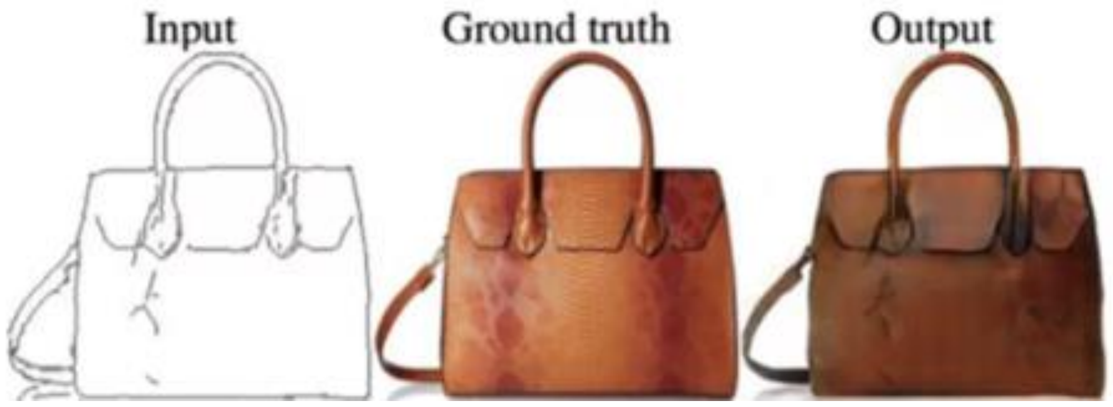

Figure 14: Implementation for fashion designing

6.2. Autonomous vehicles object detection

Image to image transformation could also be implied in autonomous vehicles. This could be applied to studying the road conditions and provide information for human drivers and self-drive cars. In addition, image to image translation can be used for generating realistic environments in simulation for training self-driving car policies.

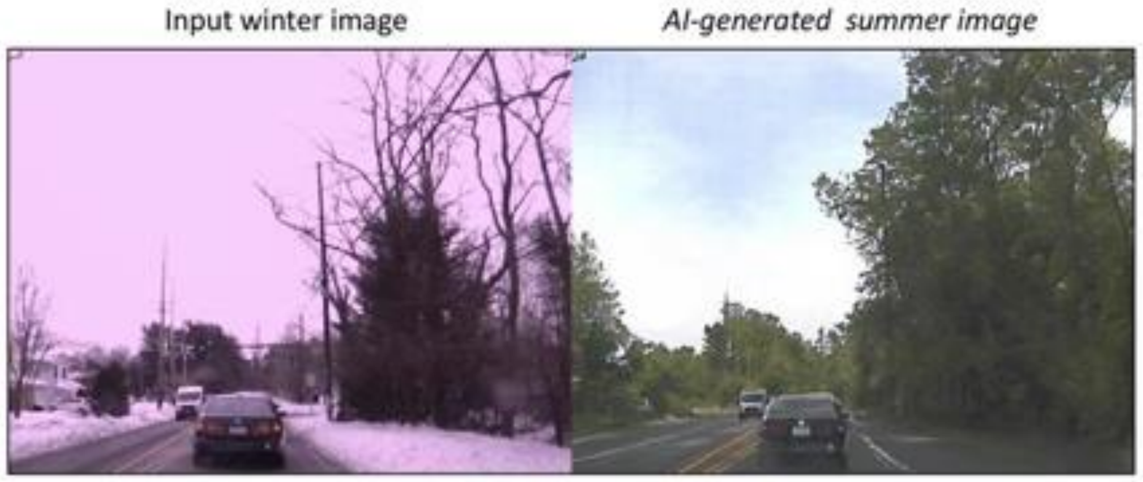

Figure 15: Implementation for auto-vehicle image detection

7.Challenges

7.1. Limited computational power and high time costs and risk

As we processed GAN on computers, we found that the speed of computation was very low. It usually operated several hours to finish one complete network. Especially for Maps model implementation. It would spend 1219 hours to run the whole epochs with 3 CPUs in our local computer system. We thought this is a problem because we might have to spend much long time in running models but the results might not be good. Thus, we first used a small subset of data to test whether the structure was correct and whether the outputs are what we expected. After we confirmed that our model was correct, we then applied the whole dataset via servers in AWS platform.

7.2. Dataset not large enough

We thought that our dataset is not very large. In general, the more data we have, the better for our model. As there are only about 128 MB of digit recognizer file, we concern that the data is not convictive that our model is good. Thus, we also train our networks by using a new dataset, 1.5 GB in map dataset.

7.3. Gap between research and real application

After we completed GAN network, we faced a problem that there is a gap between theory and practical application. GAN has the high potential commercial value and the worldwide market foreground. We do hope more researchers would make image to image more applicable to different users.

8.Future work

8.1. Try other improved GAN methods

There are many other GAN methods for different tasks. Learning new ideas from other's methods might help us to improve the performance of our model. For example, DiscoGAN is designed specifically for both customer and company by successfully transferring style from one domain to another while preserving key attributes such as orientation to recommend the perfect color match. Using this GAN could significantly improve the quality and efficiency of generating new images with different style.

8.2. Try other Generator network structures and training frequency

We would explore residual-based network for discriminator, and experiment with dynamic training frequency to allow generator train more often than discriminator in the beginning and gradually slow down.

Reference

[1] M. Arjovsky, and L. Bottou: Towards Principled Methods for Training Generative Adversarial Networks (2017)

[2] M. Arjovsky, S. Chintala, and L. Bottou: Wasserstein GAN (2017)

[3] I. Gulrajani , F. Ahmed, M. Arjovsky, V. Dumoulin, and Aaron Courville: Improved Training of Wasserstein GANs (2017)

[4] Ian Goodfellow: NIPS 2016 Tutorial: Generative Adversarial Networks (2016)

[5] Luke de Oliveira, Augustus Odena, Christopher Olah, Jonathon Shlens: Conditional Image Synthesis with Auxiliary Classifier GANs 5. Keras ACGAN implementation (2016)

[6] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets (2014)

[7] Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks (2016)

[8] Mehdi Mirza, Simon Osindero: Conditional Generative Adversarial Nets (2014)