APS360 Final Report
Skyline Generation

Yu Wang 1003245990
Shuyang Luo 1002926033
Jiajie Xu 1002937842

Word Count: 1975
Penalty: 0

1. Introduction

Modern movie industry requires various kinds of city skylines as backgrounds. Movie makers spend plenty of time finding the view and doing post-processing to fit different scenarios. There is a need for a solution that enables them to create city views based on general shapes and layouts in mind. Besides, we hope by making small adjustments to our program, skyline images of different styles and patterns could be generated, so that it could also be used by artists to create artistic work.

The goal of our project is to input a sketch of buildings and generate a skyline image accordingly, as indicated in Figure 1.1 below. Since there are various styles of skyline images, we decided to limit the scope and focus only on daylight and front-view perspective images.

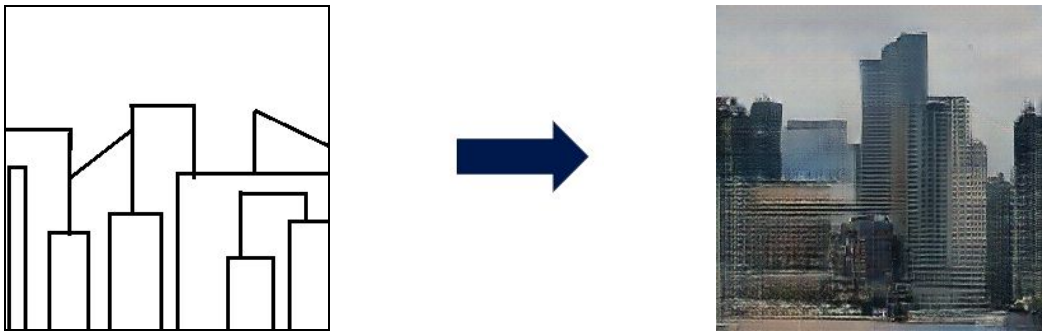

Figure 1.1 Sample input and generated image

2. Overall Software Structure

High Level Structure

At high level, the software consists of two main parts, training and application.

The training part is represented as the upper half of Figure 2.1. The software processes the original images through data cleaning steps to obtain building boundaries, then uses these boundaries as input to train the Generative Adversarial Network (GAN). The lower half of Figure 2.1 indicates the application part, which is a program that feeds input sketches into the pretrained model from training part, and then generates a corresponding skyline image as output.

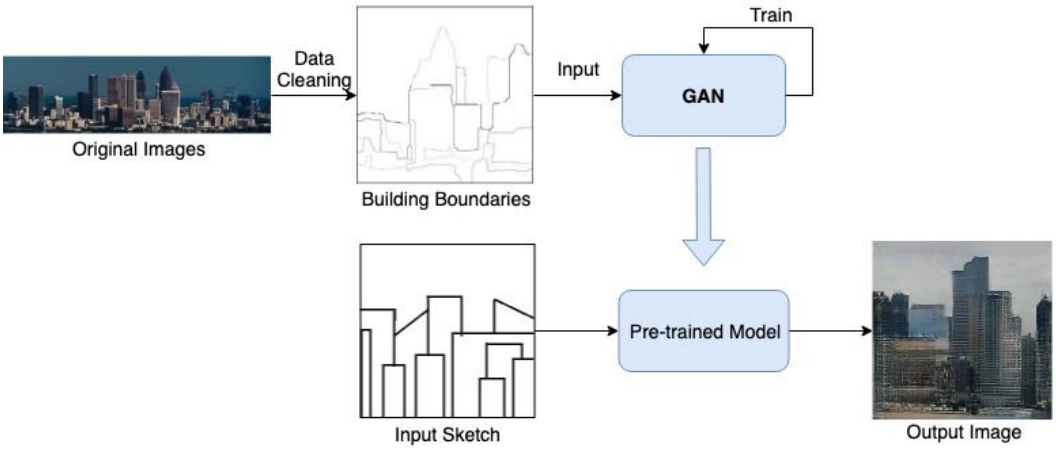

Figure 2.1 Overall software structure

Machine Learning Model

Overall Model

According to our researches online, the pix2pix model, as a general solution to image-to-image generation problems, matches our purpose of generating detailed city skylines from rough sketches [1]. Based on the pix2pix model, we designed our own model using conditional GAN, shown in Figure 2.2, where we feed building boundaries instead of random noise into the generator to gain control over the output images. Similar to regular GAN networks, the conditional GAN consists of two parts, discriminator and generator.

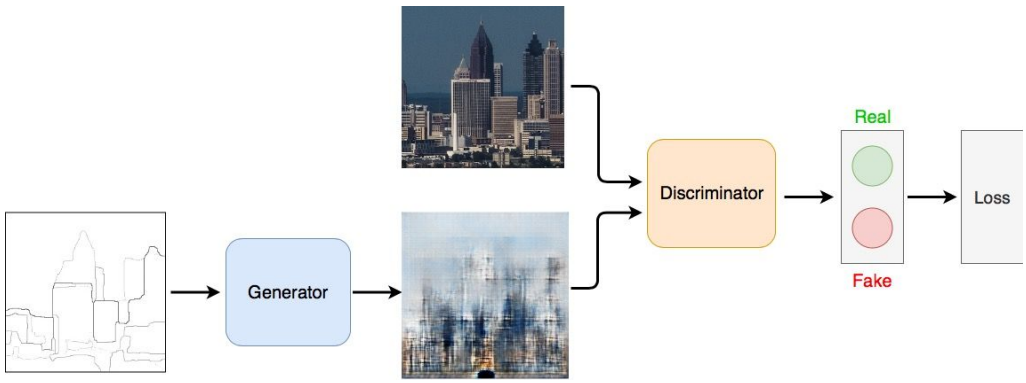

Figure 2.2 Overall model

Discriminator

The discriminator uses a fully convolutional network, and the input is a concatenation (along the "color" channel) of building boundaries and a corresponding real or fake image. The output size is 30×30, which is different from a regular discriminator with an output size of 1. Regular discriminator maps from a 256×256 image to a single scalar output, which signifies "real" or "fake" for the entire image. However, we use a so-called "PatchGAN" discriminator, which maps the image to 70×70 overlapping patches [1], runs a regular discriminator over each patch, and then averages the prediction results. In this case, the model doesn't just depend on a single prediction.

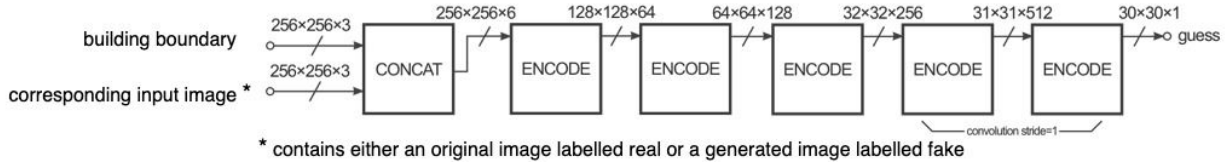

Figure 2.3 Discriminator structure [2]

Generator

Since we want to preserve the input dimension (256×256×3), our generator uses an autoencoder-like architecture, which includes an encoder and a symmetric decoder, and this is different from a regular generator which only has the decoder part. We also add skip connections to the autoencoder, and thus we have a "U-Net" structure [3], shown in Figure 2.4. These skip connections directly connect encoder layers to decoder layers so that information captured in encoder layers can be fed into decoder layers. In this way, low-level information such as the location of the boundaries can be shared between the input and the output [1].

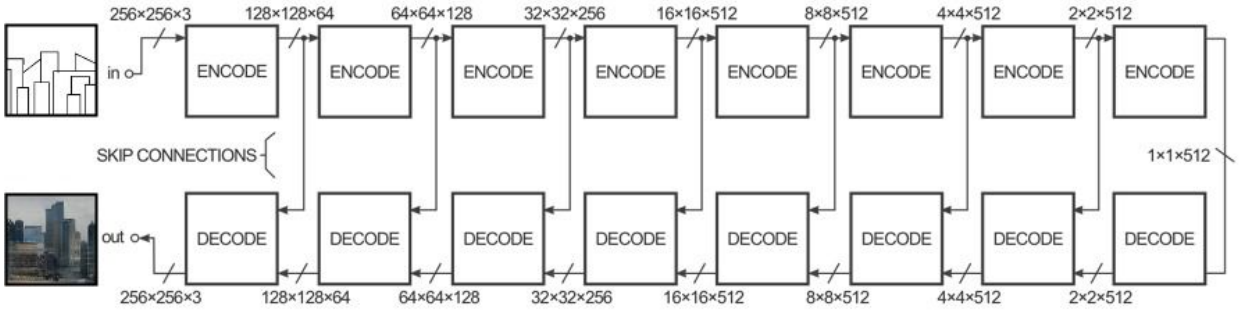

Figure 2.4 Generator structure [2]

3. Sources of Data

Data Collection

Since there wasn't any existing dataset for city skylines, we collected a total of 282 unique images using Bulkr, an application to bulk download images from Flickr. In order to maintain consistency of our dataset, the images chosen are in daylight condition and front view perspective. Figure 3.1 shows an example of qualified and unqualified image.

Unqualified Image

Qualified Image

Figure 3.1 Example of qualified and unqualified image

Data Cleaning

There are two main purposes for data cleaning. Data augmentation is essential because the amount of data collected for training our network is limited. Obtaining building boundaries through data cleaning is also important since these boundaries are used as input to train our network.

As described in Figure 3.2, our data cleaning program first crops images into squares, which are favorable for most image-related machine learning tasks. In addition, this step obtains multiple pictures from original images whose width are longer than height, thus increases the amount of our data. After cropping, we resize the images into 256×256 dimension, and then flip them horizontally to double the size of our dataset. After the above augmentations, the final dataset has 1536 images.

After cropping, resizing, and flipping, we applied global probability of boundaries (gPb) from existing research to extract the building boundaries [4]. Finally, we denoised the excessive details to obtain the final boundaries, by zeroing out pixels with relatively small intensities [5].

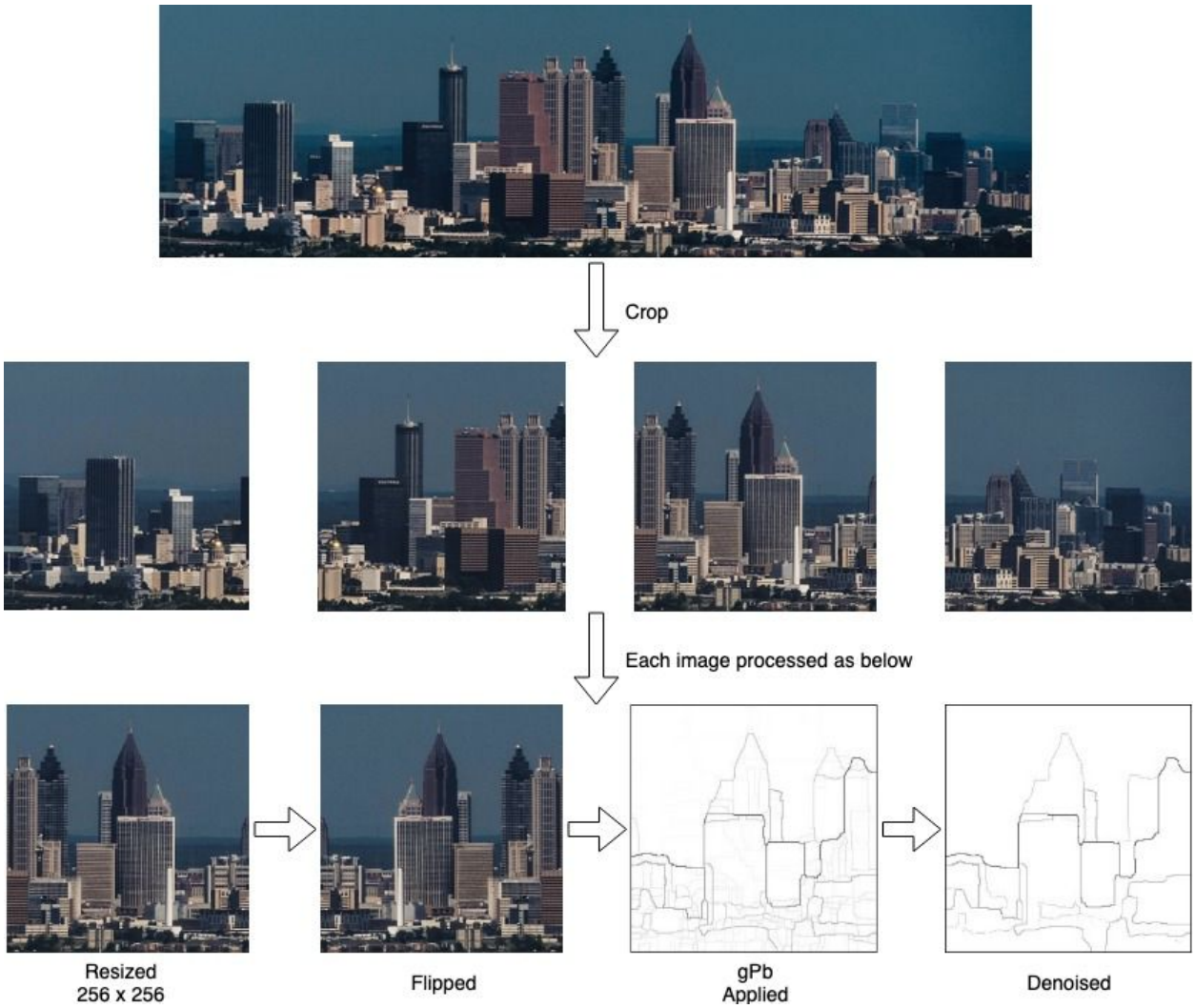

Figure 3.2 Data cleaning processes

4. Training, Validation and Test

Training

To train our GAN network, we alternate the training process between the generator and the discriminator. In each training iteration, we first feed building boundaries into the generator to generate fake skyline images.

In order to train our discriminator, we set up two Binary Cross Entropy (BCE) losses for identifying both real and fake images. We feed the concatenation of building boundaries and corresponding real images into the discriminator, and label them as “real”. Discriminator is therefore, trying to identify correctly whether these images are real, as shown in Figure 4.1. Similarly, we label and feed fake images and let the discriminator identify them. In each iteration, we add up and minimize these two losses.

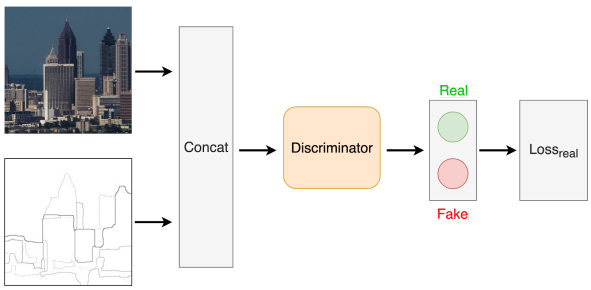

Figure 4.1 BCE Loss for real images

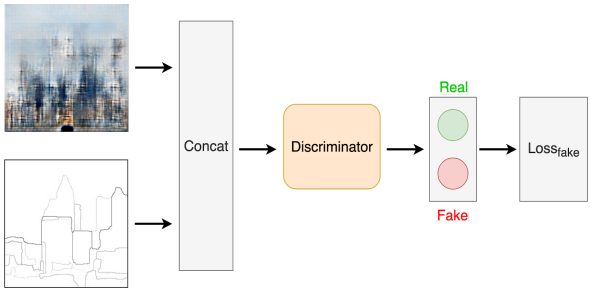

Figure 4.2 BCE Loss for fake images

To train the generator, we are doing the opposite compared to training the discriminator, where we feed the concatenation of boundaries and fake images into the discriminator, but label them as “real”. In this way, we are minimizing the probability of discriminator recognizing fake images. Moreover, to match the generated and real image, we add an L1 loss, which is the absolute difference between these two images. A lambda term is also used to control the contribution of this loss. These two losses are indicated below in Figure 4.3 and Figure 4.4.

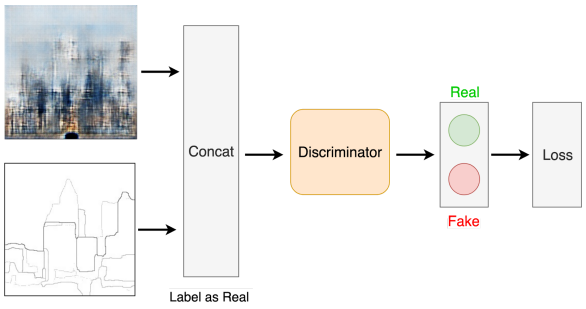

Figure 4.3 BCE Loss for fake images

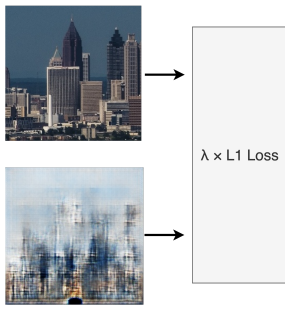

Figure 4.4 L1 Loss

Validation

Since the input for training is building boundaries, which is somewhat "different" from the input sketch for the purpose of our project, we want to validate how well our model performs on both types of input images, and tune hyperparameters based on the performances. We first collected some more skyline images and used the same data cleaning technique to obtain building boundaries. Moreover, we drew some sketches by ourselves. Examples of both types are shown in Figure 4.5 and Figure 4.6.

We observe the validation results for both types of validation images, and found that they both give outputs of similar style, so we don't differentiate hyperparameter tuning for each type.

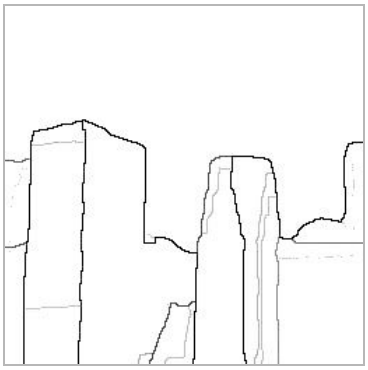

Figure 4.5 Building boundaries (validation)

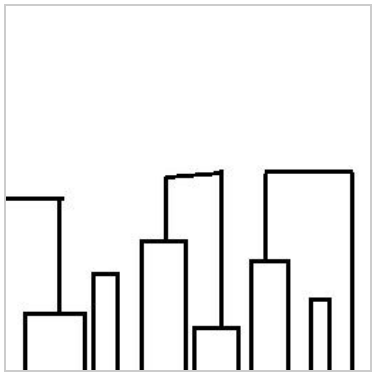

Figure 4.6 Hand-drawn sketches (validation)

As we tuned different hyperparameters, such as batch size, learning rate and the λ value, we found that λ affected the model output the most. Therefore, we focused on tuning the λ value. The output image with λ = 1, 10 and 100 are shown below respectively. With smaller lambda, the model tends to generate more realistic images to fool the discriminator, and with larger lambda, it will focus more on characteristics of each input and average them, so we get a style of sky and building mixing together. As a result, we use λ with a value of 1 to generate the most realistic images. Additionally, we use Adam Optimizer with a learning rate of 0.0001, and momentum parameters β1 = 0.5, β2 = 0.999. The batch size is 64.

Figure 4.7 λ = 1

Figure 4.8 λ = 50

Figure 4.9 λ = 100

Test

We use another 20 hand-drawn sketches as our test images. As mentioned above, when $\lambda = 1$, our model performs well and generates quite realistic images. However, there are two issues that come to our attention.

Mode collapse is one common issue that GAN network has [6], in which case the generator network only learns to produce low variety of samples to fool the discriminator. We can see from Figure 4.10 and Figure 4.11 that the input sketches are entirely different, but the generated images are quite similar. The generator tends to produce certain realistic buildings that could have a higher chance of fooling the discriminator.

Second issue is that the input sketch and generated image sometimes don't match well. This is also evident in Figure 4.11, where we have a small building on the left, but the generated image doesn't really indicate this.

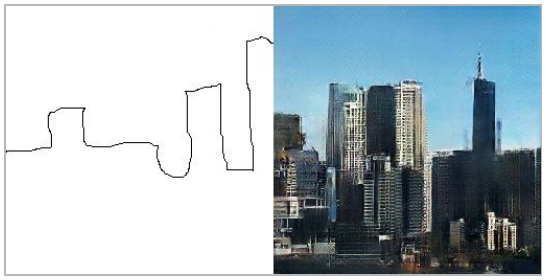

Figure 4.10 First sample of test output

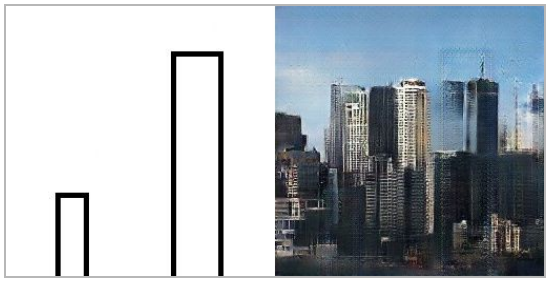

Figure 4.11 Second sample of test output

5. Ethical Issues

Our team evaluated four aspects of ethical concerns, including autonomy, beneficence, justice and non-maleficence, among which we consider non-maleficence as the major concern for our project. By definition, this means our program shall make the best effort to prevent harm.

Our program could potentially be utilized to generate misleading photos for fake news and cause loss for the society. Besides, metaphor and sarcasm to sensitive topics such as religion might be created through offensive images generated. These are unintended uses that must be prohibited. A potential solution would be adding mandatory watermark on every image that is generated from our released software.

Figure 5.1 Output with Watermark

In addition to this potential solution, we hope terms and conditions could be made to regulate the unintended uses in the designer community. We consider content creators using our program having the same responsibility as traditional designers. Everyone can use our program to design their work, but at the same time they are using it at their own risk and should be responsible for all the work released to public.

6. Key Learnings

Creating a rigorous dataset is much more time-consuming and complicated than expected. Based on our experience, roughly 1 out of 10 raw images downloaded meets our criteria, which is why we only have a limited number of images in our dataset. Additionally, many panoramic photos contain large portion of sky and lake, leaving the useful building patterns only a small portion in the image, therefore we need to crop them manually before inputting into our data cleaning algorithm. In spite of this tedious process, a clean and rigorous dataset is vastly important for training our machine learning model to produce generalized results.

In the early stage of this project, we spent lots of time on designing our own model and data processing algorithm before doing enough research. By investigating state of the art model from pix2pix paper and gPb boundary extraction algorithm [1][4], we got immediate improvements in the quality of the outputs. Researching through existing achievements could not only inspire us on the topic itself, but also save us plenty of time on developing ineffective models.

We also trained our models using our own laptops at the beginning. However, it took a tremendously long time, which delayed the process of improving our model. To solve this problem, we started to use computing resources from Google Cloud Computing Platform, by renting multiple cpus and gpus to accelerate the training process. Utilizing these cloud computing resources saves a significant amount of time in training and allows us to train the model enough to see the behavior of our model more clearly, thus it is worthwhile to learn and use them early on in the project.

7. References

[1] P. Isola, J. Zhu, T. Zhou and A. Efros, *Arxiv.org*, 2018. [Online]. Available: https://arxiv.org/pdf/1611.07004.pdf. [Accessed: 10- Apr- 2019].

[2] M. Chablani, "CycleGANS and Pix2Pix", *Medium*, 2017. [Online]. Available: https://medium.com/@ManishChablani/cyclegans-and-pix2pix-5e6a5f0159c4. [Accessed: 10- Apr- 2019].

[3] O. Ronneberger, P. Fischer and T. Brox, Arxiv.org, 2015. [Online]. Available: https://arxiv.org/pdf/1505.04597.pdf. [Accessed: 10- Apr- 2019].

[4] P. Arbela´ez and C. Fowlkes, *Www2.eecs.berkeley.edu*, 2011. [Online]. Available: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/papers/amfm_pami2010.pdf. [Accessed: 10- Apr- 2019].

[5] Appendix 8.1

[6] T. Dinh, K. Liu and V. Sah, *Pages.cs.wisc.edu*, 2017. [Online]. Available: http://pages.cs.wisc.edu/~tuandinh/GANs_Mode_Collapse_DinhLiuSah_Final.pdf. [Accessed: 10- Apr- 2019].

8. Appendices

8.1 Definition of pixel intensity

By intensity, we mean the intensity of black colour, i.e. 0 in grayscale is high intensity and 255 is low intensity.

8.2 Sample test outputs from our program