The University of Manchester
Department of Computer Science
Project Report 2020

Image-to-Image Translation Using Generative Adversarial Networks

Author: Wenchang Liu

Supervisor: Prof. Angelo Cangelosi

Abstract

Image-to-Image Translation
Using Generative Adversarial Networks

Author: Wenchang Liu

Automatically translating one possible representation of a scene into another given sufficient training data is one of the popular applications of recent generative models. The work of Isola et al.[IZZE16] has proved that Generative Adversarial Network(GAN) is an effective end-to-end process for generating images with rich details such as photorealistic images from sketches such as edge maps or segmentation maps. Following this work, more models have been proposed to improve the results. This report starts with a gentle introduction to these topics and discuss the achievements of the existent state-of-the-art models which translate segmentation maps into photorealistic images. Moreover, I manage to implement two recent high computational resources demanded state-of-the-art models with limited computational resources on a smaller dataset. The implementation details and results are shown and the advantages and disadvantages of the models are analyzed in this report.

Supervisor: Prof. Angelo Cangelosi

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Angelo Cangelosi for all his patience, guidance and constructive suggestions during the research and development of my final year project.

I also want to thank my school teachers, my friends, my family for their encouragement and support throughout my studies.

Without your help, none of this would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Background

This chapter contains a brief introduction to the problem of image-to-image translation, deep learning and Generative Adversarial Networks. General ideas of these subjects are needed for the reader to understand the more complicated concepts introduced in the later chapters.

1.1 Image-to-Image Translation

1.1.1 Definition

Many challenges in computer vision and machine learning can be regarded as "translating" an input image into a corresponding output image. Just as a concept may be expressed in either English or Chinese, a scene may be rendered as an RGB image, a gradient field, an edge map, a segmentation map, etc. In analogy to automatic language translation, we cite the definition of automatic image translation from Isola et al. [IZZE16]: tasks of translating one possible representation of a scene into another. Researchers have solved some kinds of image translation using separate, special-purpose machinery(e.g. style transfer[GEB15]), even if the settings of these problems are always the same: predict pixels from pixels. The models this report discuss (originated from [IZZE16]) make using one common framework for all these problems possible. In this project, we will focus on translating semantic segmentation maps into photorealistic images but you can apply these approaches on other image translation problems.

1.1.2 Segmentation Maps

In computer vision, image segmentation is often needed in order to simplify or change the representation of an image into something that is more meaningful and easier to analyze, and a lot of deep learning algorithms have been invented to do semantic labelling. Therefore, the data this project needs is easy to acquire.

Segmentation maps are also known as semantic label maps, which is a set segments that collectively cover the entire image. Each of the pixels in a region are similar with respect to the semantic of the image and each region is assigned with a different color and a semantic label.

Figure 1.1: A Segmentation Map

The following is an example segmentation map image, where each color represent one type of object:

1.1.3 Applications

The translation of photorealistic images from sketches is very useful once the technology is mature enough for commercial applications. Designers could get a fast preview of their work not by imagination, but with vivid photorealistic images. For example, game designers can preview the scene, items, or characters they design vividly by drawing just sets of color blocks or edges. Besides, this technology provides opportunities for people who are not good at art to create their own masterpieces.

1.2 Deep Learning

Deep learning is part of the broader family of machine learning algorithms, based on artificial neural networks and representation learning(i.e. automatically discover representations needed for feature detection or classification from raw data). Learning can be supervised, semi-supervised, or unsupervised. Deep learning approaches have widely been utilized in fields including computer vision, natural language processing, audio recognition, text filtering, machine translation, image analysis, drug design, etc., where they perform comparably to and in some cases better than human experts.

1.2.1 Neural Networks

Artificial neural networks(ANN) are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Neural networks used in deep learning can be regarded as a parametric approximation function that can map the input A into output B with parameters θ i.e. $f_\theta : A \to B$. The mapping function can gradually get optimized by updating its parameters through raw data and back propagation algorithms.

The multi-layer architecture of neural networks can achieve complex mappings by composing multiple but simple non-linear functions together. In neural network implementations, the input "signal" will pass into each neuron through connection edges, the output of each neuron is computed by some non-linear function of the sum of its inputs, typically, neurons are aggregated into layers and the "signals" travel from the first input layer to the last output layer to produce the final results.

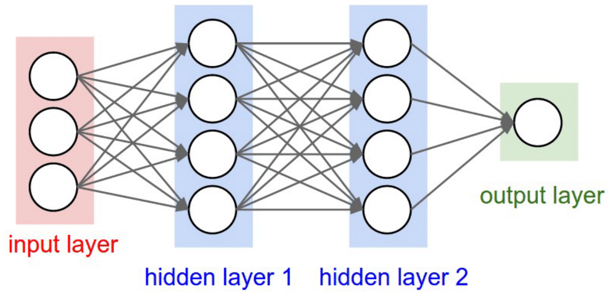

Figure 1.2: Neural Network Structure

1.2.2 Activation Functions

The activation function of a node(i.e. neuron) in neural networks defines the output of that node given an input or set of inputs. Note that only non-linear activation functions allow such networks to compute complex mappings, if we do not use activation function or use linear activation function, no matter how many layers we have, we can only result in getting linear mapping functions.

The activation functions this project uses are the following:

- Rectified Linear Unit(ReLU)

  The Rectified Linear Unit is one of the simplest and most commonly used activation functions in the last few years, it computes the function: $f(x) = \max(0, x)$. This activation function is just threshold at zero, which is much simpler than tanh or sigmoid, and it was found to greatly accelerate the convergence of gradient descent compared to other activation functions including tanh or sigmoid. However, it does has a disadvantage of being fragile during training, i.e. the ReLU units can irreversibly die and forever be zero during training since they can get knocked off the data manifold.

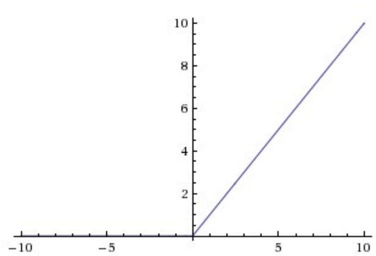

Figure 1.3: Rectified Linear Unit(ReLU) Activation Function

- Leaky Rectified Linear Unit(Leaky ReLU)

  Leaky ReLUs are one type of approach trying to fix the "dying ReLU" problem. Instead of just threshold at zero, it computes: $f(x) = 1(x < 0)(\alpha x) + 1(x >= 0)(x)$, where α is a small constant. Some report leaky ReLUs are effective but the reuslts are necessarily consistent.

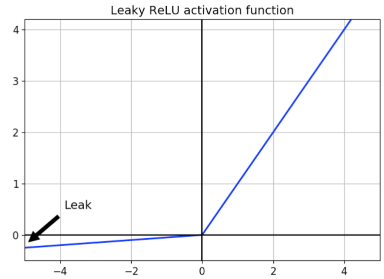

Figure 1.4: Leaky ReLU Activation Function

- Hyperbolic Tangent(tanh)

  The tanh activation function squashes a real-valued number to the range of [-1, 1], this activation saturates but is zero-centered so that it can be regarded as a scaled and more desirable sigmoid activation function in practice.

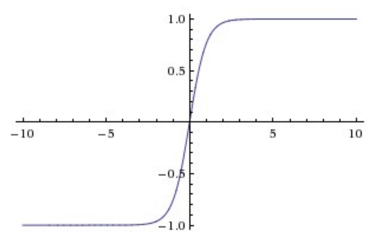

Figure 1.5: Hyperbolic Tangent Activation Function

1.2.3 Backpropagation

Backpropagation(BP) is a widely used algorithm for training neural networks for supervised learning. While training a neural network, there will be a loss function L describing how well the current approximation f_θ approximates the correct mapping function by calculate the differences between the output from the neural network and the output from the training dataset. Then backpropagation algorithm will try to approximate the correct mapping function by continuing minimizing the loss function, this can be done by computing the gradients of the loss function with respect to each weight from each pair of input-output data by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to the first(in order to avoid redundant computation) and update every parameter θ_i using $\theta_i \leftarrow \theta_i - \alpha \frac{\partial L}{\partial \theta_i}$ where $\alpha (> 0)$ is the learning rate and $\frac{\partial L}{\partial \theta_i}$ is the partial derivative(i.e. gradient). Theoretically, the gradient has the direction away from the minimal point, so each time of these updates will make the neural network approximate better by taking a little step in the opposite direction of the gradient, this idea of minimizing the loss function is called gradient descent.

1.2.4 Convolutional Neural Network

Convolutional Neural Network(CNN) is a popular type of deep neural networks which commonly applied to computer vision related tasks. A typical Convolution block consists of a convolution layer, a pooling layer and a fully-connected layer(exactly the same as regular neural network). A simple pipeline could be: [INPUT-CONV2D-ACTIVATION-POOLING-FC], In more detail:

- INPUT [width, height, channels] will hold the raw pixel values of an input image, for example, for MNIST would be [28, 28, 1], i.e. 28x28 resolution images with only 1 channel for black and white colors.
- CONV2D is the key of convolutional neural network, the convolution layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. The shape of the output tensor will be [width, height, filters], where number of filters is a hyperparameter for our CNN layer. The convolution layer can extract related feature maps from the original images(e.g. edges, corners, etc.) with appropriately optimized parameters, which is very useful for further analysis such as classification or generation.
- ACTIVATION is easy to understand, we can simply use RELU(or leaky RELU, tanh), this will not change the shape of the output tensor.
- POOLING: in most cases, we will use max pooling, which is typically a downsampling operation along the spatial dimensions(width, height), and change the output tensor shape to [width/n, height/n, filters].
- FC, fully-connected layer, each neuron in this kind of layer will be connected to all the numbers in the previous volume. For instance, in classification task, fully-connected layer will compute the class scores resulting in the shape of [1, 1, classes], where there will be a score for each class representing how likely the image is that class.

In this way, CNN transforms the original image from the pixel values to encoded feature maps or classification class scores. Note that the reverse version of CNN called CNN transpose can decode feature maps back to images, so in our image translation task, we will first use CNN to get the segmentation map into features maps, and then use CNN transpose to get the feature maps to photorealistic images.

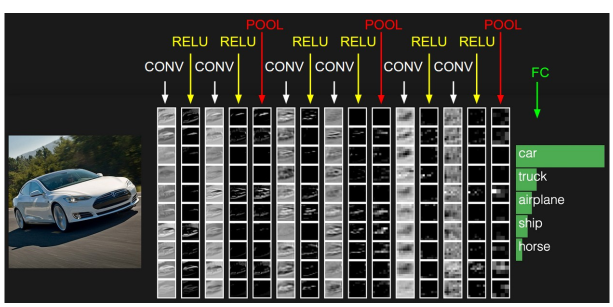

Figure 1.6: Example CNN Task

1.2.5 Residual Blocks

A traditional view of deep learning is that using more layers not necessarily results in better performance, in fact, simply stacking too many CNN blocks has been shown to cause a negative effect since the gradient can easily shrink to zero. However, the ResNet with residual blocks brought by He et al. [HZRS15] has eliminated this problem. The residual block skip connects between layers which adds the output from previous layers x to the output of stacked layers $F(x)$, in this way, even if something wrong happens to the stacked deeper layers output(e.g. gradient vanishing), the network is still able to learn the identity output from the previous output. Therefore, residual blocks guarantee us to get results no worse than a shallow network, and when this apply to CNN, a even deeper CNN can be more powerful for computer vision tasks.

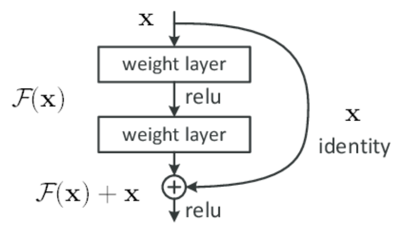

Figure 1.7: Structure of Residual Layer

1.2.6 Batch Normalization

Batch Normalization(BN) is a popular technique proposed by Ioffe and Szegedy [IS15] recently which alleviates a lot of headaches with properly initializing neural networks by forcing the activations throughout a network to take on a unit gaussian distribution at the beginning of the training. In deep learning, a layer could supply the next layer inputs with a high varience and a mean value far from zero, to fix this problem, BN process every data with equation:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} \text{ and } y_i = \gamma \hat{x}_i + \beta$$

Where x_i is an activation for the i^{th} example in the minibatch and $\hat{x}_i$ is the output after the process, μ and σ^2 are the mean vlaue and variance of the activation over the batch, and γ and β are trainable parameters.

In practice, we usually insert the BN layer between FC and non-linearities. It has been shown that BN can make networks more robust to bad initialization. In addition, BN can be interpreted as doing preprocessing at every layer of a network, but integrated into the network itself in a differentiable manner, which is why BN is widely used nowadays. For more details, please check the referenced paper [IS15].

1.3 Generative Adversarial Network

Generative Adversarial Network(GAN) is one kind of deep learning approach originated from Goodfellow et al. [GPAM+14]. The idea of GAN is inspired from game theory: two neural networks contest against each other in a game(i.e. the training process of deep learning), where one generator network tries to generate fake images while the other discriminator network tries to identify whether an image is real or fake. GAN models can learn a loss that tries to classify if the output image is real or fake, while simultaneously training a generative model to minimize this loss. One advantage that GAN is more powerful than traditional CNN approach on image translation tasks is that GAN can produce clearer results for blurry images look obviously fake. Furthermore, we need expert knowledge and carefully designed loss function for traditional CNN models, while we only need to specify a high-level objective for GAN models: make the output looks like real, and then automatically learn a loss function for satisfying this goal, which is much more desirable.

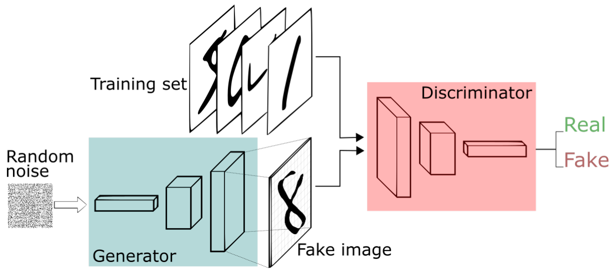

Figure 1.8: Structure of GANs

1.3.1 Conditional Generative Adversarial Network

Conditional Generative Adversarial Network(cGAN) is a special kind of GAN whose input of the generator is not random noises, but send in a condition image instead, the networks will learn to adapt and adjust their parameters to these additional inputs. For conventional GAN models, only the input noise can influence the output, however, for cGAN models, the conditional image can also influence the results. In image translation tasks, the encoded segmentation map is the condition we apply to the GAN model. Both pix2pix[IZZE16] and pix2pixHD[WLZ+18] use this kind of GAN model.

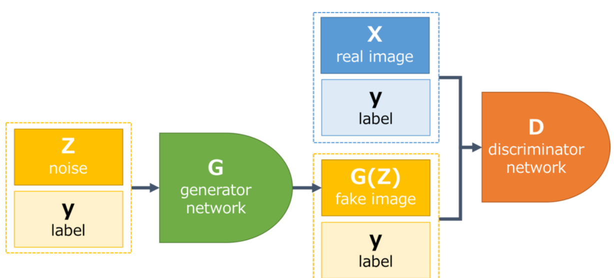

Figure 1.9: Structure of conditional GANs

1.4 Impact of COVID-19

I decided to go back to Beijing after my presentation of 3rd year project on 16th March, my flight arrived at Xi'an, China at 23rd March, I had a low fever and was sent to a hospital for further checks, and fortunately, I am OK. After that, I went to quarantine for 14 days at a hotel in Xi'an, and then traveled back to Beijing and self-isolate for another 14 days at home according to the policies from the government. Generally speaking, I had quarantine for a month before I can work on my work wholeheartedly. Fortunately, I finished my demonstartion before I left and the deadlines of 3rd year project and other courseworks has been extended.

Chapter 2

Literature Review

In this chapter, I will introduce how solving these kinds of image-to-image translation tasks with GANs are originated, how the following models trying to improve the results by discussing three representative models. For the two state-of-the-art models(pix2pixHD and SPADE), I will only discuss the theory here, please check chapter 4 for the implementation details and results evaluation.

2.1 Image-to-Image Translation with cGAN

Dealing Image-to-Image translation tasks with GANs originated from Isola et al.[IZZE16] which tries to develop a common framework for all “predict pixels from pixels” problems with conditional GANs, which shows the ability of generating decent 256×256 resolution images.

2.1.1 Objective Function

The objective of our model is to produce photorealistic images that is indistinguishable from the real images given input segmentation maps, the cGAN objective function is:

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

where the generator G tries to minimize the cGAN loss against an adversarial discriminator D that tries to maximize it, i.e. $G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D)$. The paper also claims that it is beneficial to mix the GAN objective with a conventional loss such as L1 loss: $\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1]$, so the final loss function of our GAN model is:

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

Note in this model, we do not provide noise z like the traditional cGAN does, for we only need deterministic results and the random noise is not found helpful in terms of the final outputs.

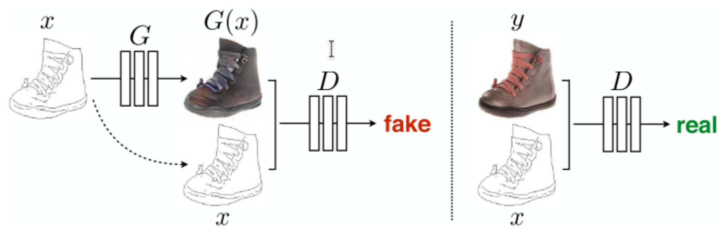

Figure 2.1: cGAN in pix2pix

2.1.2 Architecture

Both the generator and discriminator use modules of the form [CONV2d-BN-ReLU], the generator uses a classic encoder-decoder structure where the encoder extracts the feature maps from the input segmentation map, and the decoder filled the details to produce a photorealistic output from that feature maps. In addition, sharing low-level information between input and output is helpful for many image translation tasks, so in order to shuttle this information across the neural network, the author adds skip connections following the general structure of a "U-Net", specifically, we try to connect each layer i and corresponding layer $n-i$, where n is the total number of layers, and each skip connections simply concatenates all channels at layer i with those at layer $n-i$. For discriminator, the author introduces PatchGAN which tries to classify if each $N \times N$ patch in an image is real or fake, this discriminator has fewer parameters, runs faster, and can be applied to larger images.

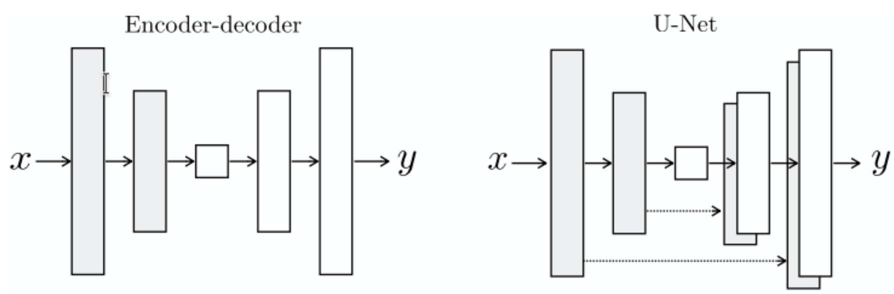

Figure 2.2: Encoder-decoder generator VS. Skip connections generator

2.1.3 Summary

The pix2pix model does not require a lot of computation resources, one can easily achieve similar results on Google Colab. I ran the official tutorial of image translation from Tensorflow2.0 web page, and here are some example outputs after 100 epochs of training on facade dataset[TŠ13]:

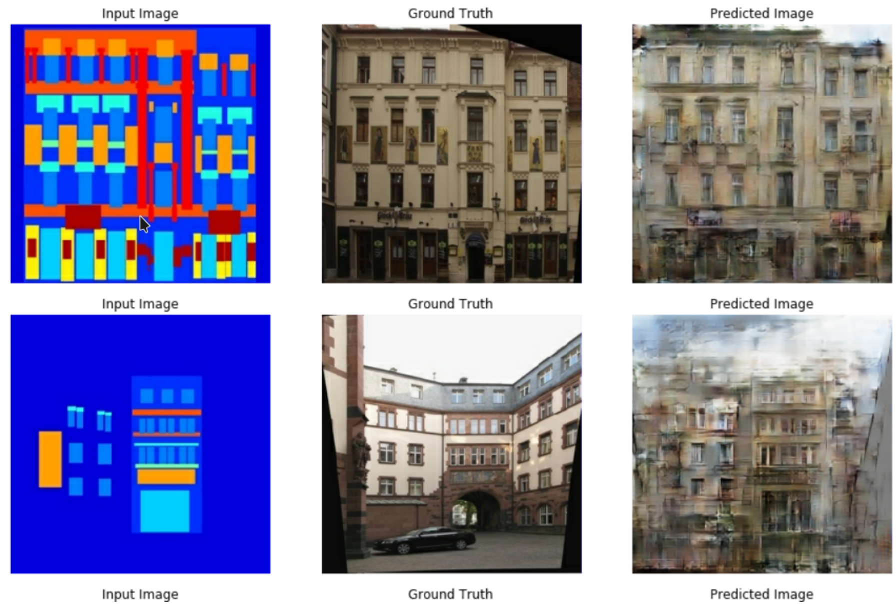

Figure 2.3: Example outputs from pix2pix model

The pix2pix model can achieve decent results with limited computation resources on small datasets such as the facade dataset. However, it still cannot produce very clear textures or high resolution photorealistic images.

2.2 State-of-the-art Model — Pix2pixHD

Pix2pixHD[WLZ+18] is the upgraded version of pix2pix brought by Nvidia which can generate high resolution images. In the paper they successfully generate 2048×1024 resolution images using the Cityscapes benchmark dataset [COR+16]. The pix2pixHD model is still based on conditional GAN, but it makes improvements on the generator, the discriminator, and the loss function.

2.2.1 Generator

Generator G consists of two generators, we term G1 as the global generator and G2 as the local enhancer. The global generator plays the similar role as the pix2pix model which is basically an encoder-decoder generator with residual blocks that can output lower resolution(1024×512) images, and the local enhancer will augment the output images to a higher resolution, e.g. 2048×1024. Both the global generator and local enhancer consist of convolution blocks, residual blocks, and transposed convolution blocks, we named them $G_1^{(c)}$, $G_1^{(r)}$, $G_1^{(t)}$ and $G_2^{(c)}$, $G_2^{(r)}$, $G_2^{(t)}$ respectively. In order to integrate the global generator and local enhancer, the input to the residual block of the local enhancer is the element-wise sum of two feature maps: the output feature map of $G_2^{(c)}$, and the last feature map of the global generator $G_1^{(t)}$. The architecture of the whole generator is shown as the following figure:

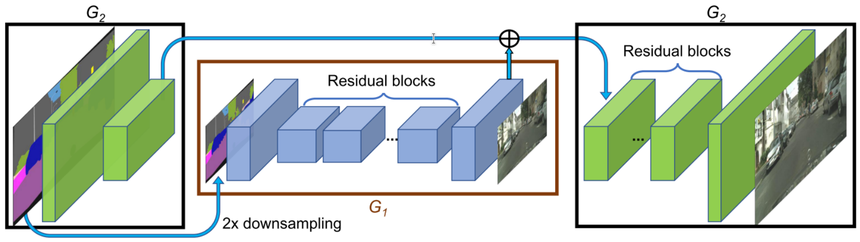

Figure 2.4: Pix2pixHD generator architecture

When training the generator, we need to train the global generator first and then train the local enhancer in the order of their resolutions.

2.2.2 Discriminator

Discriminator D is upgraded to Multi-scale discriminators, which alleviate the headache that a discriminator needs to have a larger receptive field, i.e. more memory is required, however, we all know that memory is already a scarce resource for deep learning models. Therefore, what a multi-scale discriminator does is to use 3 smaller discriminators that have identical network structure but operate at different scales, we named the discriminators as D_1, D_2, D_3 respectively. D_1, D_2 will deal with the real and synthesized images by factor 2 and 4, while D_3 will deal with the original images. The discriminator that operates at the finest scale will focus on the details of an image, while the discriminator that operates at the largest scale will have the largest receptive field, discriminating the image with a global view. In implementation, we can simply add up the three discriminators' objective functions, which makes the learning problem to be:

$$\min_{G} \max_{D_1,D_2,D_3} \sum_{k=1,2,3} \mathcal{L}_{\text{GAN}}(G, D_k)$$

2.2.3 Objective Function

The improved adversarial loss consists of:

- The GAN loss similar to the one in pix2pix model:

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

- The feature map loss, it is desirable if we can learn to match the intermediate representations from the real and the synthesized image, so in pix2pixHD we extract features from multiple layers of the discriminator and calculate the feature map loss of each layer and then add them up together:

$$\mathcal{L}_{\text{FM}}(G,D_k) = \mathbb{E}_{(\mathbf{s},\mathbf{x})} \sum_{i=1}^{T} \frac{1}{N_i} \left[\left\| D_k^{(i)}(\mathbf{s},\mathbf{x}) - D_k^{(i)}(\mathbf{s},G(\mathbf{s})) \right\|_1 \right]$$

  Where $D_k^{(i)}$ is the i-th layer feature extractor of discriminator D_k, T is the total number of layers and N_i is the number of elements in each layer.

- (Optional)VGG loss, also called perceptual loss, we calculate loss with the help of pretrained VGG network[SZ14]: $\lambda \sum_{i=1}^{N} \frac{1}{M_i} \left[\left\| F^{(i)}(\mathbf{x}) - F^{(i)}(G(\mathbf{s})) \right\|_1 \right]$, where we choose $\lambda = 10$, and F_i is the i-th layer with M_i elements of the VGG network.

The final objective without VGG could be:

$$\min_{G} \left(\left(\max_{D_1,D_2,D_3} \sum_{k=1,2,3} \mathcal{L}_{\text{GAN}}(G, D_k) \right) + \lambda \sum_{k=1,2,3} \mathcal{L}_{\text{FM}}(G, D_k) \right)$$

or including VGG loss:

$$\min_{G} \left(\left(\max_{D_1,D_2,D_3} \sum_{k=1,2,3} \mathcal{L}_{\text{GAN}}(G, D_k) \right) + \lambda \sum_{k=1,2,3} \mathcal{L}_{\text{FM}}(G, D_k) + \lambda \sum_{i=1}^{N} \frac{1}{M_i} \left[\left\| F^{(i)}(\mathbf{x}) - F^{(i)}(G(\mathbf{s})) \right\|_1 \right] \right)$$

2.3 State-of-the-art Model — SPADE

Spatially-adaptive (DE)normalization(SPADE)[PLWZ19] is another upgraded image-to-image translation model brought by Nvidia, previous designs including pix2pix [IZZE16] and pix2pixHD [WLZ+18] are based on cGAN structure, which directly feed the semantic layout as input to the generator network, then the segmentation map is processed through stacks of convolution, batch normalization, residual, non-linearity layers. The paper claims that some semantic information get removed by normalization layers, therefore, they proposed a new normalization layer which can integrate the information of semantic label maps instead of just two trainable parameters in the traditional batch normalization layer. The SPADE borrows the design of objective function and discriminator, so we will mainly discuss the generator upgrades next.

2.3.1 SPADE Block

The SPADE block is the key of SPADE generator, similar to batch normalization, the activation is normalized in the channel-wise manner by the following formula:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \varepsilon}} \text{ and } y_i = \gamma \hat{x}_i + \beta$$

Where x_i is an activation for the i-th example in the minibatch and $\hat{x}_i$ is the output after the process, μ and σ^2 are the mean vlaue and variance of the activation over the batch. However, in SPADE block, γ and β are not just trainable parameters anymore, they are determined by the input segmentation map, the SPADE block uses a two-layer convolution module that convert an image into values. Since the important parameters in this new normalization block are influenced by the input segmentation map, we do not need to worry the semantic information being “washed away” during normalization.

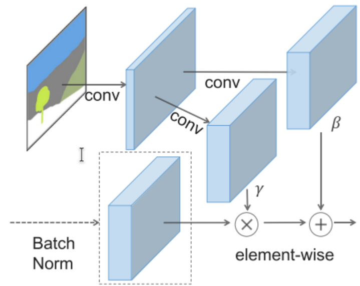

Figure 2.5: Structure of a single SPADE Block

2.3.2 SPADE Residual Block

The SPADE residual block is used to replace the conventional residual block, instead of simply using convolutions, it also uses SPADE Block, in this way, we can add the information of the input segmentation map into the residual blocks. The structure of a single SPADE residual block is as follows:

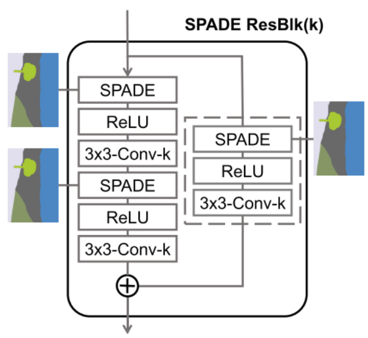

Figure 2.6: Structure of a single SPADE Residual Block

2.3.3 Generator

In order to build a SPADE generator, we need one more step which is to put SPADE residual blocks into the structure of a GAN generator. Because we have already put the information of the input segmentation map into the residual blocks, we do not need the encoder part of conditional GAN, we can just use a random noise as the input like the original GAN models and only keep the decoder part. So the structure of the SPADE generator is like the following:

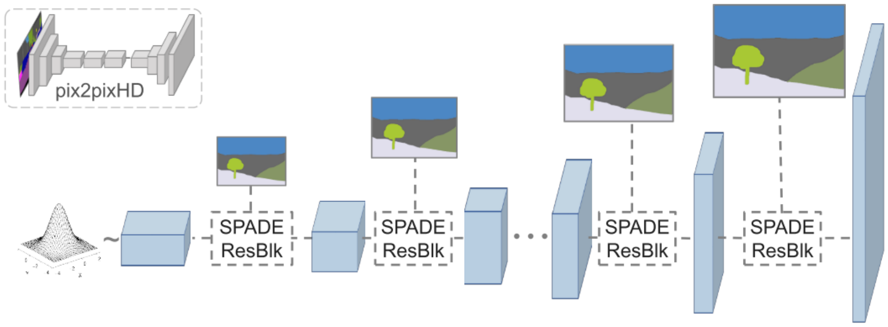

Figure 2.7: Architecture of SPADE generator

2.3.4 Other Features

Apart from the improvement on the generator, SPADE also make some other slight modifications including changing LSGAN loss in pix2pixHD with Hinge loss, applying spectral normalization to the convolution layers in the discriminator. In addition, since the encoder is no longer necessary to the image transaltion task, SPADE allows using a variational auto encoder(VAE) to achieve style transfer. The complete architecture of SPADE GAN model is like the following:

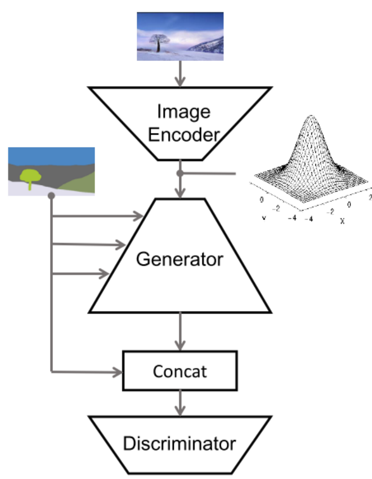

Figure 2.8: Architecture of SPADE GAN model

The VAE image encoder encodes a real image to generate a mean and variance which are used to further compute the noise that is input to the decoder. The discriminator takes the concatenation of the semantic label map and the output image from the generator as input and tries to identify whether each image is real or fake.

Chapter 3

Project Development

In this chapter, I will introduce how the project is developed, the development tools and computational resources that I use for development.

This project tries to implement the two state-of-the-art models and expand their scope of application from datasets to arbitrary drawing. Therefore, the whole project consists of two parts, training the deep learning models and building GUI to show the capability of the models. Since training deep learning networks requires high computation power especially Graphics Processing Units(GPUs), which is not available to my laptop, so the plan is to train the models on Google Colab with free a GPU to use and then download the models and build a web app locally.

3.1 Computational Resources

Colaboratory brought by Google, often refered as Google Colab, is a free online interactive environment as the form of a jupyter notebook, where we can write our python code and execute like our local jupyter notebook, furthermore, Google Colab has already configured most of the libraries for data science and machine learning, and it also provides a free GPU for us to use(K80, T4, P4, P100 will be allocated randomly). However, it does have some problems, for example, sessions can get disconnected and accounts can be banned for long time continuously using the GPU. So my solution is to save the model checkpoints after certain epochs of training, since Colab allows users to mount their google drive, I set the checkpoints saving directory in google drive so that I can continue training when I get disconnected. Despite these inconveniences, this is the best solution I can think of.

3.2 Deep Learning Framework

The deep learning framework I choose for this project is PyTorch from Facebook. Pytorch offers tensor computation with strong GPU acceleration and deep neural networks built on a tape-based autograd system and it has become more and more popular in academia recently. Pytorch is friendly to tyros for you can easily understand what the code is doing and use the predefined frequently used layers such as convolution layers, batch normalization layers, etc. Also, it allows researchers to build deep neural networks with complex architectures flexibly which is an advantage over popular frameworks like Keras. In addition, unlike Tensorflow, Pytorch uses dynamic computation graph instead of static computation graph which is beneficial for tweaking and debugging. And the most important advantage of Pytorch over Tensorflow for me is that Pytorch has better documentation than Tensorflow, for example, you may found several functions doing the thing under different packages in Tensorflow documentation, which is kind of chaos. For more information about Pytorch please visit https://pytorch.org/.

3.3 Graphical User Interface

I decided to use a web page as the graphical user interface(GUI) for my project. Unlike C++ or $C\sharp$, python does not have powerful desktop application GUI frameworks, however, web development tools like Flask and Django for python can easily integrate the Pytorch library and can also make use of the front-end technology to make decent-looking GUI. When the user performs an action on the GUI, the front-end will send a request to the back-end, and the Flask framework will handle the request and ask Pytorch model to do the image translation if necessary.

The front-end is basically developed with HTML, CSS, JavaScript, and libraries such as Bootstrap and Font Awesome. In terms of back-end, I chose is Flask, which is a lightweight WSGI web application framework for python. Flask projects start with quick and easy setup but can scale up to complex applications, it also wraps Jinja2 which is a full-featured template engine that allows developers to integrate with front-end.

The web app including 4 pages including a home page, an about page, two pages demonstrating the capability of the model for translating segmentation maps in the dataset and arbitrary drawing semantic label maps respectively.

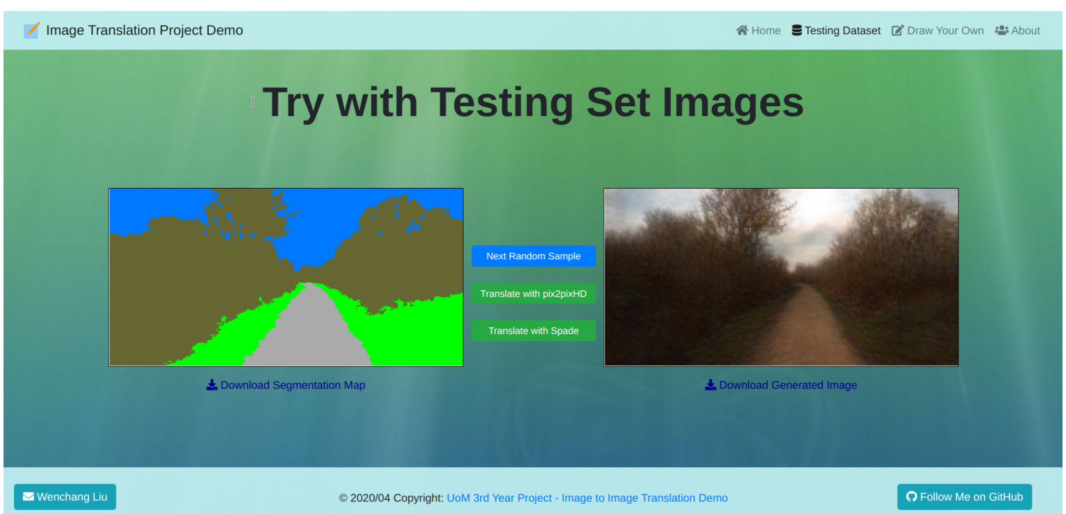

Figure 3.1: Screenshot of the web page that allow users to try segmentation maps from the test dataset

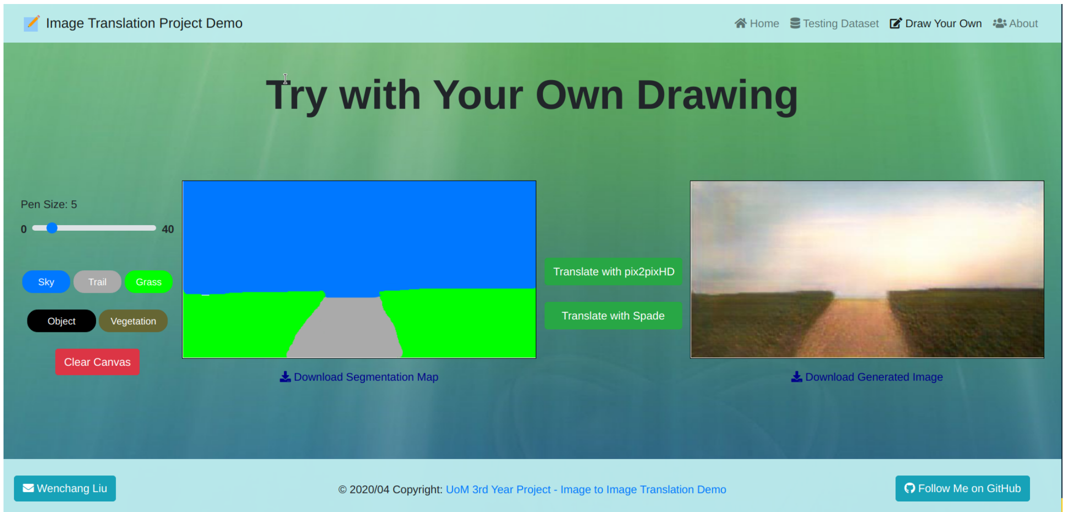

Figure 3.2: Screenshot of the web page that allow users to try segmentation maps drawn by themselves

The web app offers functionalities including getting next random sample segmentation map from the test dataset, simple drawing tool for users to draw their own style of segmentation map, downloading segmentation maps or generated photorealistic images, and translating the segmentation map into photorealistic with both pix2pixHD and SPADE model.

Chapter 4

Experiments and Evaluation

4.1 Dataset

4.1.1 Benchmark Datasets

4.1.2 Used Dataset

4.2 Pix2pixHD Implementation

4.2.1 Training

4.2.2 Results

4.3 SPADE Implementation

4.3.1 Simpler Structure of SPADE

4.3.2 Training

4.3.3 Results

4.4 Comparison

Chapter 5

Reflection and Conclusion

5.1 Planning and Management

The plan of my project is divided into four parts:

1. Read related research papers and prepare related knowledge and tools:

   Main research papers related to this project are [IZZE16], [WLZ+18], [PLWZ19], I also spend time learning knowledge related to deep learning and computer vision including CNN, ResNet, GAN, style transfer, etc. Learning frameworks including Pytorch and flask are also needed for implementation.

2. Implement image translation models and trained them:

   This is the most difficult and time-consuming part of the project, I decided to implement the two state-of-the-art models regarding image translation, however, it is not easy to train such models on Colab because I do not have enough computation power, for example, to duplicate the results mentioned in the paper of SPADE, the authors have trained the network using 8 Nvidia GPUs for 4 days which is obviously not practicable for me, not to mention the disconnecting issue. The solution here is to remove the unnecessary parts of the project and choose a relatively small but still effective dataset for my implementation.

3. Develop GUI for the project:

   This is mainly a software engineering task, I have similar experience using Spring framework and Java for web development, so it is relatively an easy work for me to learn the similar framework Flask for this project.

4. Write report and record screencast:

   The original plan is to start writing report once the demonstartion(March 16th) is finished, nevertheless, due to COVID-19 issue, I decided to travel back home on March 22nd, it took me nearly a month before I can work on the report wholeheartedly.

Table 5.1 includes the milestones for the project together with the original planned timeline and the actual carrying out timeline. Several things were not carried out according to the original plan, the original plan was made before the arrangement of lightening talk and demonstartion, so I decided to start the training after I finished the lightening talk on November 7th. Apart

Milestones	Planned Weeks	Actual Weeks
Background Preparation	Oct 01 - Nov 04	Oct 01 - Nov 07
Train neural networks	Nov 04 - Feb 01	Nov 04 - Mar 16
Develop Application	Feb 01 - Mar 02	Nov 04 - Mar 16
Report and Screencast	Mar 16 - Apr 28	Apr 06 - May 05

Table 5.1: Milestones of 3rd Year Project

from that, Since I have a lot of time waiting the intermediate results during training, I decided to merge the model training phase with the GUI development phase so that I can make full use of my time. For the last part of the project, I could not follow my original plan due to COVID-19 outbreak, and I cannot focus on writing the report before I finishing my quarantine and go back home, and fortunately, the deadline has also postponed.

5.2 Future Works

Due to computational resources and development time limitations, I am not able to implement all the functionalities mentioned in the paper of pix2pixHD[WLZ+18] and SPADE[PLWZ19], including:

- Instance Maps in Pix2pixHD In segmentation maps, each pixel value represents a class of object, the maps will not differentiate objects of the same category, e.g. if two cars stitch together in a segmentation map, we are unable to know the boundaries of each car. On the other hand, an instance map(or boundary map) will track each object with a unique ID. With the help of instance maps, the model can deal the boundaries of objects better.

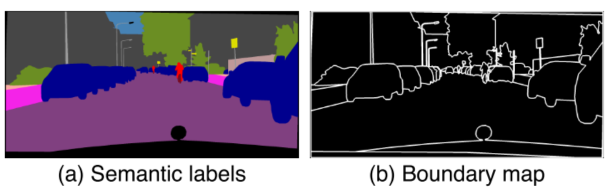

Figure 5.1: Segmentation Map(a) VS. Instance Map(b)

- Variational Auto Encoder in SPADE

  As introduced in chapter 2, SPADE allows using a variational auto encoder to achieve style transfer along with image translation. The encoder is consists of several convolution blocks and convert the image into a mean vector μ and a variance vector σ, which is later used to calculate the noise input into the generator. The VAE part needs separate training and tuning.

- Benchmark Datasets

  There are several benchmark datasets for image translation tasks including Cityscapes [COR+16], ADE20K [ZZP+17], etc. These benchmark datasets provide much more training data than the one I use for this project, which can check if the model can cope with complex scenes, i.e. test its ability of translating segmentation maps with various objects(e.g. there are 20 classes of objects in Cityscapes) or different kind of scenes(e.g. ADE20K contains houses scences, art galleries scenes, airports scenes, etc.). However, it requires more computational resources to train networks on such datasets.

- Hyperparameters Search

  It is desirable if we can search for the best hyperparameters for each model, this requires training the networks multiple times(each time may take days). We might need some ways to get some heuristics for hyperparameters combinations or it is not possible to iterate all the possible options.

5.3 Conclusion

This project provides literature reviews on image-to-image translation topic and experiments with two state-of-the-art image translation models with limited computational resources and compare their advantages and disadvantages. The experiments details have been written in jupyter notebooks and the implementation of the models are also wrapped up into a web app with a GUI so that people who are not familiar with this topic can check the experiments process and results, and people who have no ideas of coding and machine learning can also try image translation themselves.

During this research and development process, I not only acquired a broad range of knowledge about deep learning and image translation, but also enhanced my self-learning skills. At first, I only knew some basic concepts of neural networks, then I learned how convolutional neural network and its variations can solve those complex computer vision tasks, how to build a simple convolutional neural network myself with the help of deep learning frameworks. I read several papers regarding the image-to-image translation task, it is not easy to keep up with the novel ideas at first, but after searching for explanations from blogs or YouTube channels, I managed to get the ideas. Apart from that, I also practiced how to deploy the machine learning models into a functional application like a web app.

In my perspective, even if the current state-of-the-art models can achieve astonishing results and are good enough for working prototypes, the approaches are still far from commercial uses. First, the approaches still demand too many computation resources, for example, if I want to duplicate image translation for the landscape dataset that the researchers did, I would still need 8 Nvidia V100 GPUs, which is not available for most of the users who just want to train a model for their specific domain. Another problem is that we have to provide enough training data for the model to work, which is not necessarily as easy as it seems, for example, if we want to make a commercial application for designers to assist their designs, it is not easy to acquire novel designs as the training dataset.

Bibliography

[COR+16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[GEB15] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[GPAM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[IZZE16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[Li20] Fei-Fei Li. Stanford cs231n convolutional neural networks for visual recognition. http://cs231n.stanford.edu/, 2020.

[PLWZ19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[TŠ13] Radim Tyleček and Radim Šára. Spatial pattern templates for recognition of objects with regular structure. In *Proc. GCPR*, Saarbrucken, Germany, 2013.

[WLZ+18] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[ZZP+16] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *arXiv preprint arXiv:1608.05442*, 2016.

[ZZP+17] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.