

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

<b>1</b>	<b>Background</b>	<b>6</b>
1.1	Image-to-Image Translation . . . . .	6
1.1.1	Definition . . . . .	6
1.1.2	Segmentation Maps . . . . .	6
1.1.3	Applications . . . . .	7
1.2	Deep Learning . . . . .	7
1.2.1	Neural Networks . . . . .	7
1.2.2	Activation Functions . . . . .	8
1.2.3	Backpropagation . . . . .	9
1.2.4	Convolutional Neural Network . . . . .	9
1.2.5	Residual Blocks . . . . .	10
1.2.6	Batch Normalization . . . . .	11
1.3	Generative Adversarial Network . . . . .	11
1.3.1	Conditional Generative Adversarial Network . . . . .	12
1.4	Impact of COVID-19 . . . . .	12
<b>2</b>	<b>Literature Review</b>	<b>13</b>
2.1	Image-to-Image Translation with cGAN . . . . .	13
2.1.1	Objective Function . . . . .	13
2.1.2	Architecture . . . . .	14
2.1.3	Summary . . . . .	14
2.2	State-of-the-art Model — Pix2pixHD . . . . .	15
2.2.1	Generator . . . . .	15
2.2.2	Discriminator . . . . .	16
2.2.3	Objective Function . . . . .	16
2.3	State-of-the-art Model — SPADE . . . . .	17
2.3.1	SPADE Block . . . . .	17
2.3.2	SPADE Residual Block . . . . .	18
2.3.3	Generator . . . . .	18
2.3.4	Other Features . . . . .	19
<b>3</b>	<b>Project Development</b>	<b>20</b>
3.1	Deep Learning Framework . . . . .	20
3.2	Computational Resources . . . . .	20
3.3	Graphical User Interface . . . . .	20
<b>4</b>	<b>Experiments and Evaluation</b>	<b>21</b>

4.1	Dataset . . . . .	21
4.1.1	Benchmark Datasets . . . . .	21
4.1.2	Used Dataset . . . . .	21
4.2	Pix2pixHD Implementation . . . . .	21
4.2.1	Training . . . . .	21
4.2.2	Results . . . . .	21
4.3	SPADE Implementation . . . . .	21
4.3.1	Simpler Structure of SPADE . . . . .	21
4.3.2	Training . . . . .	21
4.3.3	Results . . . . .	21
4.4	Comparison . . . . .	21
<b>5</b>	<b>Reflection and Conclusion</b>	<b>22</b>
	<b>Bibliography</b>	<b>23</b>
<b>A</b>	<b>Example of operation</b>	<b>24</b>
A.1	Example input and output . . . . .	24
A.1.1	Input . . . . .	24
A.1.2	Output . . . . .	24
A.1.3	Another way to include code . . . . .	24

# List of Figures

1.1	A Segmentation Map . . . . .	7
1.2	Neural Network Structure . . . . .	8
1.3	Rectified Linear Unit(ReLU) Activation Function . . . . .	8
1.4	Leaky ReLU Activation Function . . . . .	9
1.5	Hyperbolic Tangent Activation Function . . . . .	9
1.6	Example CNN Task . . . . .	10
1.7	Structure of Residual Layer . . . . .	11
1.8	Structure of GANs . . . . .	12
1.9	Structure of conditional GANs . . . . .	12
2.1	cGAN in pix2pix . . . . .	14
2.2	Encoder-decoder generator VS. Skip connections generator . . . . .	14
2.3	Example outputs from pix2pix model . . . . .	15
2.4	Pix2pixHD generator architecture . . . . .	16
2.5	Structure of a single SPADE Block . . . . .	18
2.6	Structure of a single SPADE Residual Block . . . . .	18
2.7	Architecture of SPADE generator . . . . .	19
2.8	Architecture of SPADE GAN model . . . . .	19

## 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  $\theta$  i.e.  $f_{\theta} : A \rightarrow B$ . 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.

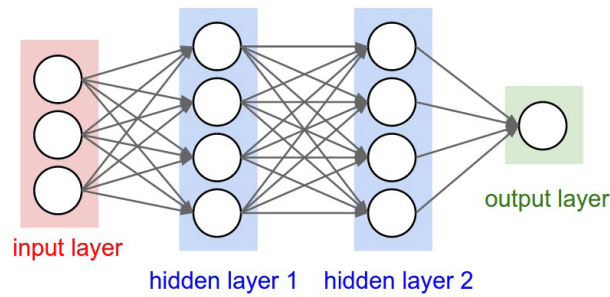


Figure 1.2: Neural Network Structure

### 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 have 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.

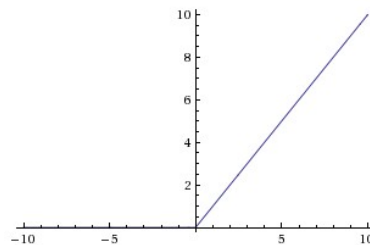


Figure 1.3: Rectified Linear Unit(ReLU) Activation Function

- 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:  $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$ , where  $\alpha$  is a small constant. Some report leaky ReLUs are effective but the results are necessarily consistent.

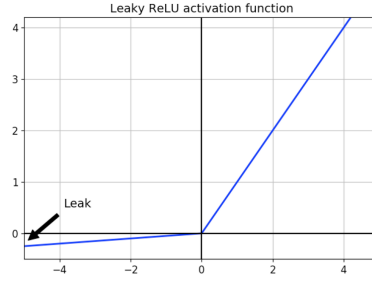


Figure 1.4: Leaky ReLU Activation Function

- 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.

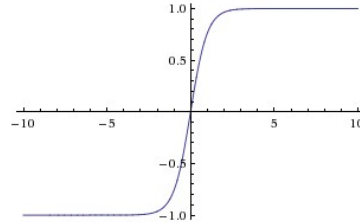


Figure 1.5: Hyperbolic Tangent Activation Function

### 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  $f_{\theta}$  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  $\theta_i$  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.

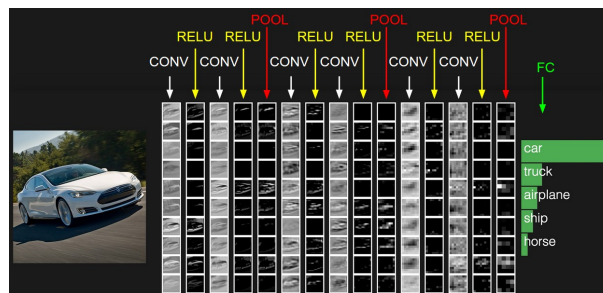


Figure 1.6: Example CNN Task

### 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.

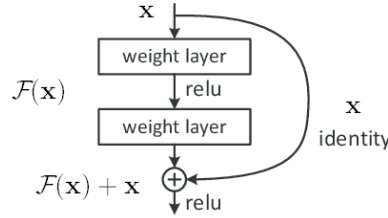


Figure 1.7: Structure of Residual Layer

### 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 variance and a mean value far from zero, to fix this problem, BN process every data with equation:

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

Where  $x_i$  is an activation for the  $i^{th}$  example in the minibatch and  $\hat{x}_i$  is the output after the process,  $\mu$  and  $\sigma^2$  are the mean vlaue and variance of the activation over the batch, and  $\gamma$  and  $\beta$  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<sup>+</sup>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.

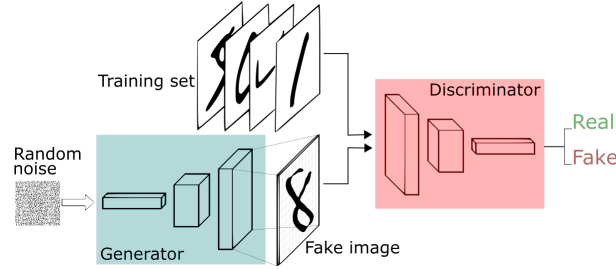


Figure 1.8: Structure of GANs

### 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[IZZ16] and pix2pixHD[WLZ<sup>+</sup>18] use this kind of GAN model.

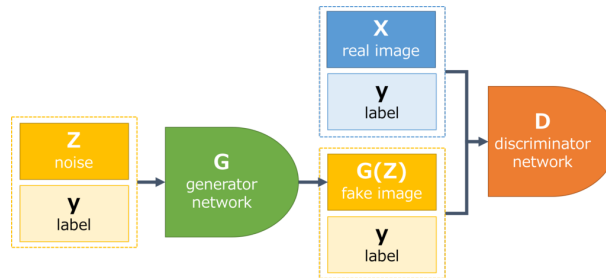


Figure 1.9: Structure of conditional GANs

## 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 demonstration 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  $256 \times 256$  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.

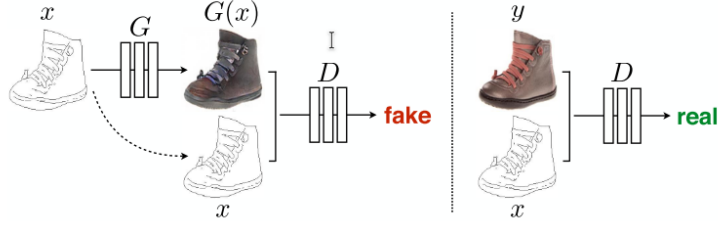


Figure 2.1: cGAN in pix2pix

### 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.

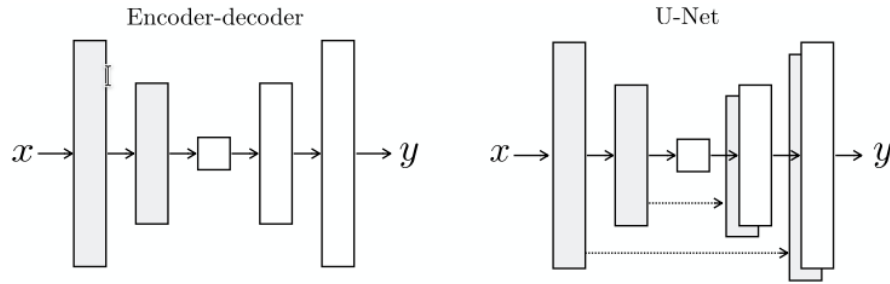


Figure 2.2: Encoder-decoder generator VS. Skip connections generator

### 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]:



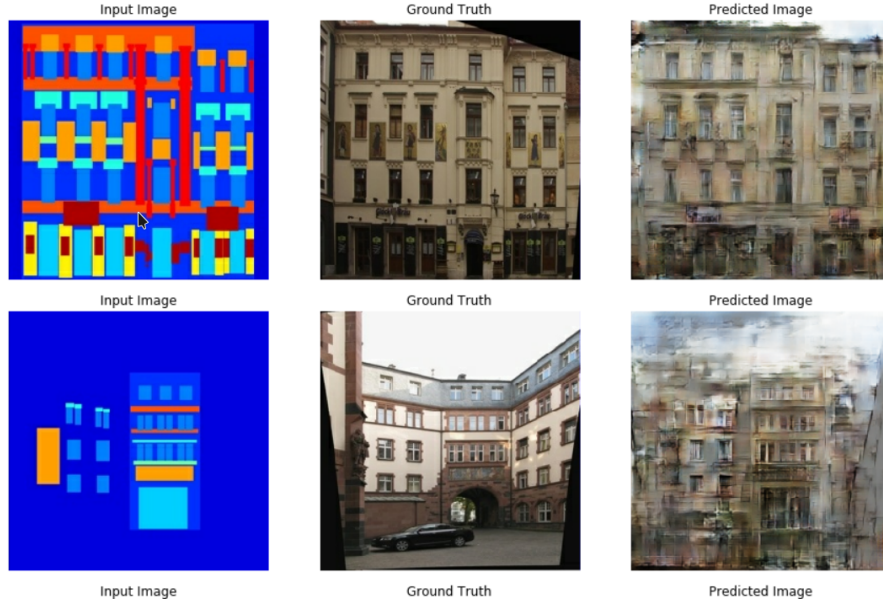


Figure 2.3: Example outputs from pix2pix model

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<sup>+</sup>18] is the upgraded version of pix2pix brought by Nvidia which can generate high resolution images. In the paper they successfully generate  $2048 \times 1024$  resolution images using the Cityscapes benchmark dataset [COR<sup>+</sup>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( $1024 \times 512$ ) images, and the local enhancer will augment the output images to a higher resolution, e.g.  $2048 \times 1024$ . 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:

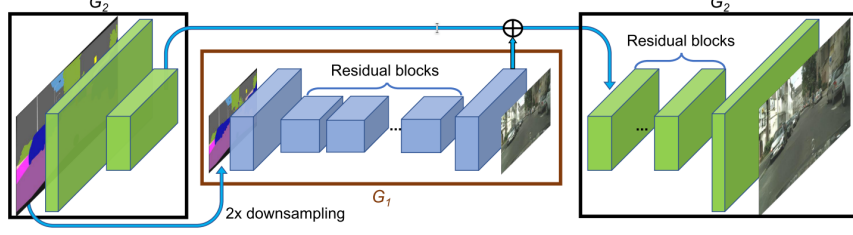


Figure 2.4: Pix2pixHD generator architecture

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:

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

### 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:

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

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<sup>+</sup>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:

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

Where  $x_i$  is an activation for the  $i$ -th example in the minibatch and  $\hat{x}_i$  is the output after the process,  $\mu$  and  $\sigma^2$  are the mean vlaue and variance of the activation over the batch. However, in SPADE block,  $\gamma$  and  $\beta$  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.

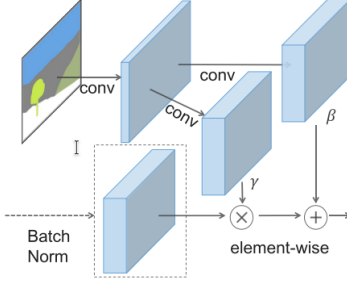


Figure 2.5: Structure of a single SPADE Block

### 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:

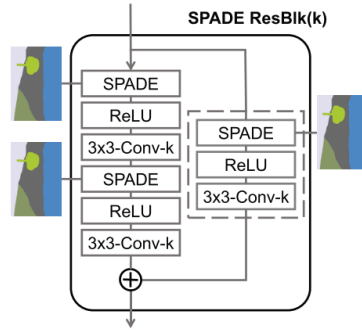


Figure 2.6: Structure of a single SPADE Residual Block

### 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:

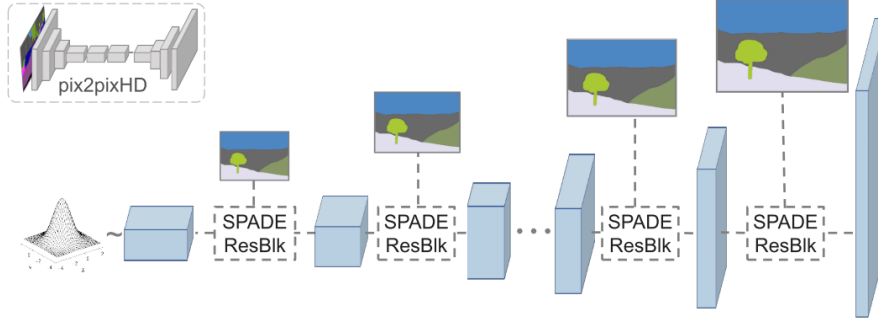


Figure 2.7: Architecture of SPADE generator

### 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:

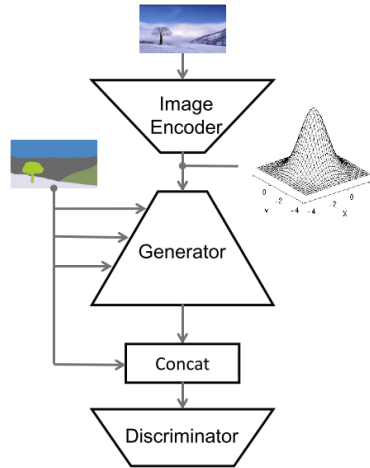


Figure 2.8: Architecture of SPADE GAN model

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.

### **3.1 Deep Learning Framework**

### **3.2 Computational Resources**

### **3.3 Graphical User Interface**

# **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**



# Bibliography

- [COR<sup>+</sup>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<sup>+</sup>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*, Saarbrücken, Germany, 2013.
- [WLZ<sup>+</sup>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.

# Appendix A

## Example of operation

An appendix is just like any other chapter, except that it comes after the appendix command in the master file.

One use of an appendix is to include an example of input to the system and the corresponding output.

One way to do this is to include, unformatted, an existing input file. You can do this using `\verbatiminput`. In this appendix we include a copy of the C file `hello.c` and its output file `hello.out`. If you use this facility you should make sure that the file which you input does not contain TAB characters, since  $\text{\LaTeX}$  treats each TAB as a single space; you can use the Unix command `expand` (see manual page) to expand tabs into the appropriate number of spaces.

### A.1 Example input and output

#### A.1.1 Input

(Actually, this isn't input, it's the source code, but it will do as an example)

```
/* Hello world program */

#include <stdio.h>

int main(void)
{
    printf("Hello World!\n") ;
    return 0 ;
}
```

#### A.1.2 Output

#### A.1.3 Another way to include code

You can also use the capabilities of the `listings` package to include sections of code, it does some keyword highlighting.

```
/* Hello world program */

#include <stdio.h>

int main(void)
{
    printf("Hello _World!\n") ;
    return 0 ;
}
```