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

Introduction

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. $f_\theta : A \rightarrow B$. 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.

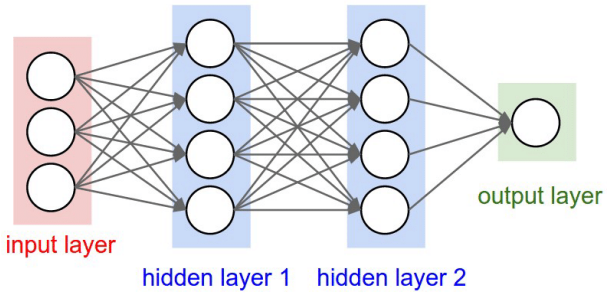

Figure 1.2: Neural Network Structure

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 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.

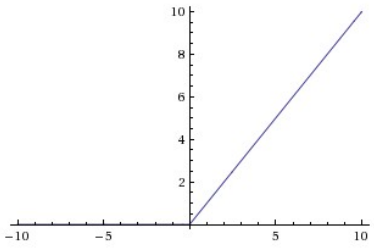

Figure 1.3: Rectified Linear Unit(ReLU) Activation Function

- 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.

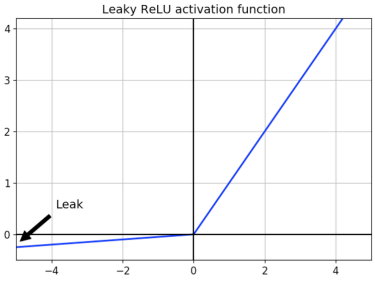

Figure 1.4: Leaky ReLU Activation Function

- 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.

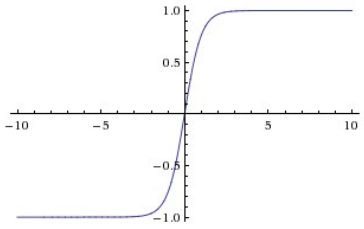

Figure 1.5: Hyperbolic Tangent Activation Function

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.

1.3 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.

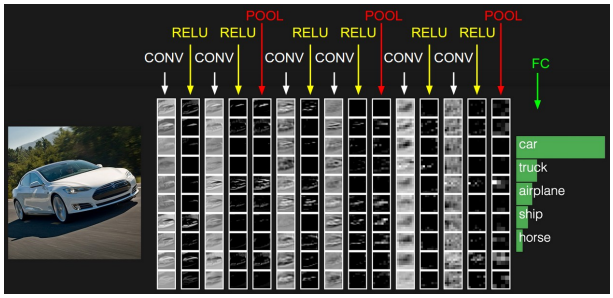

Figure 1.6: Example CNN Task

1.3.1 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.

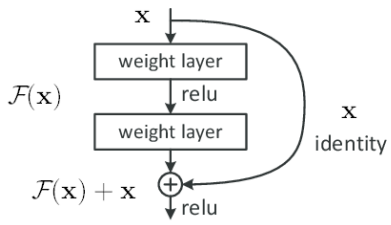

Figure 1.7: Structure of Residual Layer

1.4 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].

1.5 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.

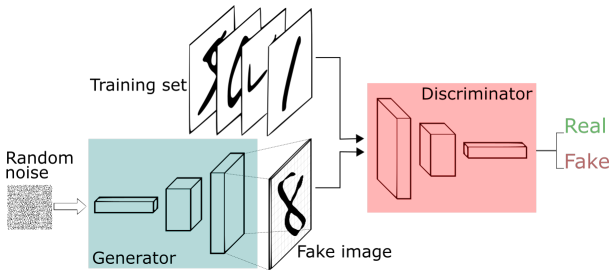

Figure 1.8: Structure of GANs

1.5.1 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.

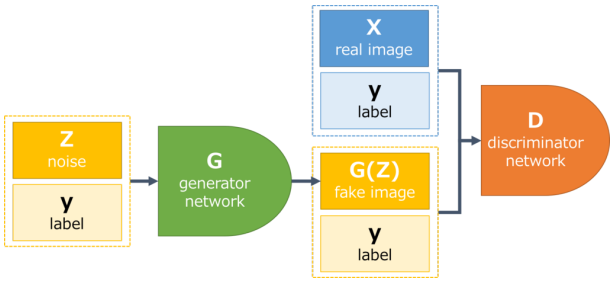

Figure 1.9: Structure of conditional GANs

1.6 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, as well as details of two state-of-the-art models.

2.1 Image-to-Image Translation with cGAN

Dealing Image-to-Image translation tasks with GANs originated from Isola et al.[IZZE16].

2.2 State-of-the-art Model — Pix2pixHD

2.3 State-of-the-art Model — SPADE

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

[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.

[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.

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 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.

```c
/* Hello world program */

#include <stdio.h>

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