Image-to-image translation based on conditional Adversarial Networks and Cycle-consistent Adversarial Networks

Rui Shi
srui@kth.se
KTH Royal Institute of Technology

Yuchen Gao
yuchenga@kth.se
KTH Royal Institute of Technology

Yutong Jiang
yutongj@kth.se
KTH Royal Institute of Technology

Abstract

In this paper, we mainly focus on the implementation and possible improvement of image-to-image translation for paired images and unpaired images. To solve such problems, Conditional Adversarial Network(cGAN) and Cycle-Consistent Adversarial Network(CycleGAN) are implemented respectively. In order to further explore possible optimization for these networks, experiments on the architecture of generator and discriminator, different approaches of normalization, like batch normalization and instance normalization, and different loss functions have been compared. Experiments have shown that we successfully reproduced the results of cGAN and CycleGAN and managed to optimize the model.

1 Introduction

In recent years, Generative Adversarial Networks(GANs) has drawn great attention. GANs is a machine learning model in which two neural networks compete with each other to achieve higher accuracy on their predictions. More specifically, given a training dataset, this technique learns to generate new data with the same statistics as the training set.

Based on the unique architecture of GANs and the performance it could achieve, it has been applied in many fields, such as the domain of image-to-image translation. In such a context, the concept of image-to-image translation with Conditional Adversarial Networks has been raised.[1] A specific limitation of such networks is that it requires paired images as input. However, the paired pictures can be really hard to acquire in one domain, which might even not exist. Under such background, the concept of Cycle-Consistent Adversarial Networks is proposed which aims to learn a mapping $G : X -> Y$ and constrain this mapping with a reverse one $F : Y -> X$.[2]

Based on the background mentioned above, we managed to implement image-to-image translation in this project. The paper is organized as follows. Section 2 summarizes related work regarding image-to-image translation, Generative Adversarial Networks and instance normalization. Section 3 gives introduction on the dataset we used in our training process. Section 4 explains more details on the specific architecture, loss function of the networks. Section 5 shows the results we reproduced given the paper[1] [2] and results we achieved by optimizing the networks. Section 6 gives an overall summary of our work and advice for what could be done in the future.

2 Related work

Image-to-Image translation In the early stage, image-to-image translation has been mainly implemented by texture synthesize, in which hidden Markov Model has been used[3]. With the development of neural network, modern techniques, such as CNN, have been implemented and applied in the domain of image translation, which have made great breakthrough.[4]

GANs Generative Adversarial Networks were introduced as an alternative framework for training generative models to generate new data with the same statistics as the training set.[5] Based on the architecture of GANs, the idea of Conditional Adversarial Networks was proposed to guide the generator by adding conditional information.[1] In order to make the network more robust, which could be applied for the unpaired dataset, Cycle-Consistent Adversarial Network has been proposed.[2]

Instance normalization The normalization technique is used for making each layer affect the network performance proportionally, and there are two typical methods of normalization, which are batch normalization and instance normalization.[6] The difference between them is that batch normalization is used to average the result of a batch of training instances while the instance normalization is to calculate the average of a certain instance channel. In Cycle-Consistent Adversarial Networks, it has been proved that the quality of output images could be better when applying instance normalization.

3 Datasets

The dataset used for Conditional Adversarial Networks is originated from kaggle [1], which includes facades, cityscapes, maps and edges to shoes images.

The unpaired-images dataset used for Cycle-Consistent Adversarial Networks is originated from kaggle [2], including horse-to-zebra, apple-to-orange and so on.

4 Methods

4.1 Network architecture of Conditional GANs

The Conditional GANs we implemented consists of a generator and discriminator of which the basic modules is convolution-BatchNorm-ReLu.

Generator with skip connections For the generator, we use a typical "U-Net" architecture, which is shown in 1. [1]In the network we designed, input is passed through 7 two-dimensional down-sampling layers with stride 2 and kernel size 4, until it reaches a bottleneck layer, where the process is reversed to up-sampling. In order to ensure the network to learn more complex properties, we add some skip connections between each layer i and layer n-i, in which n is the total number of layers in the structure.

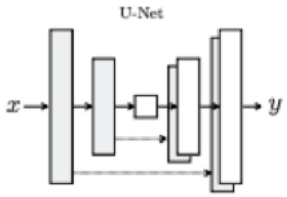

Figure 1: U-net architecture

Discriminator For the discriminator, it is used to distinguish the fake images and the real images. It extracts features by applying convolutional neural network with kernel size equals to 4.

[1]Source: `https://www.kaggle.com/datasets/vikramtiwari/pix2pix-dataset`
[2]Source: `https://www.kaggle.com/datasets/suyashdamle/cyclegan`

4.2 Loss function of conditional GANs

The final objective of conditional GANs could be expressed as the summation of adversarial loss and traditional loss, which is shown in equation 1.

$$G^* = argmin_G max_D L_{cGAN}(G, D) + \lambda L_{L1}(G) \tag{1}$$

The adversarial loss and L1 loss are explained in a more detailed way as follows.

Adversarial Loss is more efficient than cross entropy in training process of GANs, which is defined in equation 2.

$$L_{cGAN}(G, D) = E_{x,y}[logD(x, y)] + E_{x,z}[log(1 - D(x, G(x, z)))] \tag{2}$$

where G stands for the generator which tries to minimize the loss and D stands for the discriminator which tries to maximize the loss.

Traditional Loss is defined in equation 3, which is beneficial as it could not only ensure the generator fool the discriminator but also be close to the ground truth.

$$L_{L1}(G) = E_{x,y,z}[||y - G(x, z)||_1] \tag{3}$$

4.3 Network architecture of CycleGAN

The CycleGAN we implemented consists of two generators and two discriminators. The strcuture of discriminator is similar to that in cGAN. Other key features of the architecture are shown below.

Generator Generators in CycleGAN begins with three convolutional neural network with kernel size equals to 7 and stride equals to 2. Then the output is connected to 9 residual blocks, which consists of 2 3×3 convolutional blocks. Then the process is reversed to up-sampling to ensure the shape of output is the same as the input image.

4.4 Loss function of CycleGAN

The loss function of CycleGAN is composed of three parts, which are adversarial loss, cycle consistency and identity loss. The total loss is expressed in equation 4.

$$\begin{aligned} L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, Y, X) + \\ \lambda_{cyc} L_{cyc}(G, F) + \lambda_{Identity} L_{Identity}(G, F, X, Y) \end{aligned} \tag{4}$$

Adversarial loss is given in equation 5, where G tries to generate imgaes G(x) that looks similar to images from domain Y and D_Y aims to distinguish between translated samples G(x) and real samples y.

$$L_{GAN}(G, D_Y, X, Y) = E_{y-p_{data}(y)}[logD_Y(y)] + E_{x-p_{data}(x)}[log(1 - D_Y(G(x)))] \tag{5}$$

Cycle Consistency Loss is expressed in equation 6 and shown in Figure 2[2]. By considering the forward cycle consistency and backward cycle consistency, the consistency between the cycle image and input images is well kept.

$$L_{cyc}(G, F) = E_{x-p_{data}(x)}[||F(G(x)) - x||_1] + E_{y-p_{data}(y)}[||G(F(y)) - y||_1] \tag{6}$$

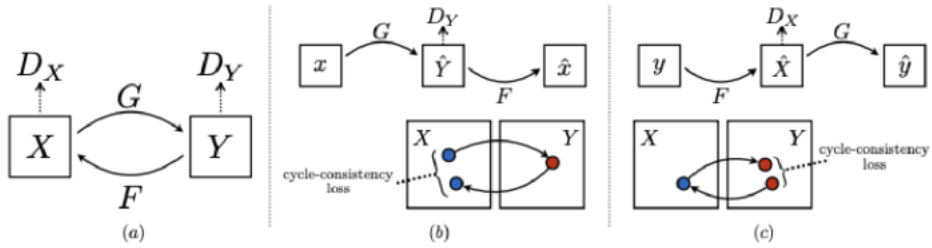

Figure 2: cycle-consistency loss

Identity loss is defined in equation 7. By applying identity loss, the color composition is well preserved.

$$L_{Identity}(G, F, X, Y) = E_{x-p_{data}(x)}[||F(x) - x||_1] + E_{y-p_{data}(y)}[||G(y) - y||_1] \quad (7)$$

4.5 Optimization method

In our experiments, Adam optimizer is used to train the discriminators and generators in both cGAN and CycleGAN. Coefficients used for computing running averages of gradients and its square is set to (0.5, 0.999) and learning rate is set to 1e-5.

5 Experiments

5.1 Implementation of conditional GAN

Based on the architecture of cGAN proposed above, we first try to implement the cGAN by applying the dataset "aerial-to-map". It turns out that our network could successfully work after training for 500 epochs in most of the cases and the results is shown in figure 3.

Figure 3: Top to bottom:(a)input aerial image (b)input target image (c)output fake image

However, given the output, the problem is that our model produces artifacts occasionally, which are highlighted by in Figure 3. Hence, further study will be launched to achieve better quality of the output. The details of optimization will be shown in the next subsection.

5.2 Further experiments on cGAN for optimization

5.2.1 Replace Batch normalization with Instance normalization

Batch normalization is a normalization that is done over batches. And instance normalization performs intensity normalization across the width and height of a single feature map of a single example, which is more commonly used in image stylization in CycleGAN as it could remove instance-specific contrast information from the content image. Hence, it could be interesting to explore the effects of instance normalization applied in cGAN.

The changes we made are to replace nn.batchnorm2d in both discriminator and generator with nn.instancenorm. The output fake image is shown in figure 4. By comparing the images with output with batch normalization, we could find the original artifacts has been greatly removed. However, instance normalization introduces some checkerboard artifacts. In general, the quality of images has been improved by applying instance normalization.

5.2.2 Apply L2 loss instead of L1 Loss in the network

Although the loss function where the paper [7] used is L1 loss, L2 Loss function is preferred in most of the cases. And by checking the dataset we used for training, it seems to contain few outliers. Hence, in this section, we intend to replace L1 loss with L2 loss.

The output images are shown in figure 4. By comparison, we could find the artifacts has been greatly removed except for the fourth image and there is an obvious improvement on the quality of fake images.

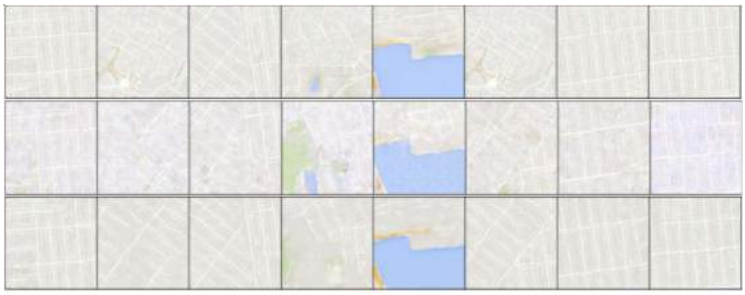

Figure 4: Top to bottom:(a)output with batch norm (b)output with instance norm (c)output with L2 Loss

5.3 Implementation of CycleGAN

Based on the architecture of cycleGAN proposed above, we try to implement the cycleGAN by applying the datasets "horse2zebra", "apple2orange" and "sum2win" respectively. The results could be shown as follows.

Figure 5: (a)input horse (b)fake zebra (c)cycle horse

Figure 6: (a)input zebra (b)fake horse (c)cycle zebra

Figure 7: (a)input orange (b)fake apple (c)cycle orange

Figure 8: (a)input apple (b)fake orange (c)cycle apple

Figure 9: (a)input winter (b)fake summer (c)cycle winter

Figure 10: (a)input summer (b)fake winter (c)cycle summer

From the results, we could find that our network could successfully work after training for 260 epochs in most of the cases. For the training process of horse to zebra, it is interesting to find that it is easier to translate horse to zebra when comparing with the inverse process(translate zebra to horse). And we think the reasons why this situation happens could be concluded as the zebra's pattern are more distinct and easy to learn.

However, the training process takes much more time(26 hours/260 epochs) when comparing with the cGAN(1.5 hours/ 500 epochs). More experiments on how to speed up the training process of CycleGAN and how to achieve better performance of the output images are shown in the next chapter.

5.4 Further experiments on CycleGAN

5.4.1 Change the loss function to speed up the training process

For cycleGAN, there is 3 loss functions in total, which are adversarial loss, cycle consistency Loss and identity loss. The speed depends on the calculation of gradients of loss functions, which indicates that changing/removing some loss functions could be an efficient way of speeding up the training process. Hence, in this section, we mainly concentrate on what performance we could achieve when ignoring identity loss. Figure 11 and 12 show the performance of network without identity loss.

Figure 11: (a)input apple (b)fake orange (c)cycle apple

Figure 12: (a)input orange (b)fake apple (c)cycle orange

We could find that the network without identity loss could still translate orange to apple. And the training process has been significantly speed up, which changes from 2.67 it/s to 4.02it/s. Hence, in the apple-to-orange translation, ignoring identity loss could be an efficient way for speeding up.

5.4.2 change the architecture of discriminator and generator

Initially, the form of modules we applied in both discriminator and generator is convolution-InstanceNorm-ReLu. However, we find the color of images has been greatly changes, which is shown in Figure 13.

Figure 13: IN is applied in initial layer

Figure 14: IN is not applied in initial layer

After changing the architecture of discriminator and generator, we find the performance will get better when remove the InstanceNorm in the initial layer of both generator and discriminator. We think the reason could be that when applying InstanceNorm in the initial layer, the color of input images will be normalized and get ignored.

6 Conclusion and Future work

In this report, we successfully implemented Conditional Adversarial Network and Cycle-Consistent Adversarial Network for paired image translation and unpaired image translation. After implementing the network, more experiments have been done to optimize the network. The experiments are (1)replace batch normalization with instance normalization (2) change/remove loss function to speed up training process/achieve better performance (3) change the architecture of discriminator and generator for better performance. Our results give quite impressive and reasonable support for the hypothesis we made.

For future work, we look forward to deal with the "checkerboard artifacts" which is induced by replacing batch normalization with instance normalization in cGAN. Further, we also intend to try different architecture to find the most splendid results in CycleGAN.

References

[1] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp1125-1134*, 2017.

[2] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[3] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

[4] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[6] Md Foysal Haque and Dae-Seong Kang. Ainn: Adversarial instance normalization network for image-to-image translation. In *Proceedings of KIIT Conference*, pages 117–120, 2020.

[7] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.