DOM-E5129 Intelligent Computational Media, Learning Outcome

Peizhe Hou, Student number: 765772

February 6, 2020

1 Basics of Machine Learning

To be honest the hardest part of my learning process is the basics of machine learning. I spent around 60 hours to go through the lectures, slides and assignments of Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition. It is really useful to understand neural networks, especially convolutional neural networks and their application.

2 Style Transfer

2.1 Model Set-up

A pre-trained feature extractor is used to extract feature maps from both the content image and the style reference in order to compare their styles. Instead of using just the output of the last CNN layer like image classifiers, we use several feature maps together to represent the input image according to our selection. Then we will define the loss function that allows us to perform gradient descent on the pixel values of input image in order to transfer the input image to our desired style while preserve the original content by minimizing the loss function. Details are shown below.

- Computing Loss

  The loss function is a weighted sum of three components: content loss, style loss and total variation loss. The idea is that we can generate an image that reflects the content of one image and the style of another by penalizing deviations from the content of the content image and deviations from the style of the style reference. And then we can then use this hybrid loss function to perform gradient descent as discussed before, not on the parameters of the model, but instead on the pixel values of our original image, to generate the image that could minimize the loss.

- Content loss

  Content loss measures how much the feature maps of the generated image differ from the feature map of the source image. Say we now concentrate on layer ℓ, it has feature map $A^\ell \in \mathbb{R}^{1\times C_\ell \times H_\ell \times W_\ell}$, where C_ℓ is the number of filters/channels in layer ℓ, H_ℓ and W_ℓ are the height and width. We can reshape it to $A^\ell \in \mathbb{R}^{C_\ell \times M_\ell}$ where $M_\ell = H_\ell \times W_\ell$ so that each row of A^ℓ represents the activations of a particular filter. Then, let $F^\ell \in \mathbb{R}^{C_\ell \times M_\ell}$ be the feature map for the current image and $P^\ell \in \mathbb{R}^{C_\ell \times M_\ell}$ be the feature map for the content source image, we can finally define the content loss as follows:

  $$L_c = w_c \times \sum_{i,j} (F^\ell_{ij} - P^\ell_{ij})^2$$

  where w_c is the weight of the content loss term in the loss function.

- Style loss

  In terms of style loss, we can compute the Gram matrix G which is an approximation to the covariance matrix and use it to compare and match the statistics of styles in generated image and style reference image because we want the activation statistics of the generated image to match the activation statistics of the style reference image.

  Given a feature map F^ℓ of shape (C_ℓ, M_ℓ), the Gram matrix has shape (C_ℓ, C_ℓ) and its elements are given by:

  $$G^\ell_{ij} = \sum_k F^\ell_{ik} F^\ell_{jk}$$

  Assuming G^ℓ is the Gram matrix computed from the feature map of the current image, A^ℓ is the Gram Matrix computed from the feature map of the source style image, and w_ℓ the weight of the style loss, then the style loss for the layer ℓ is just the weighted Euclidean distance between the two Gram matrices:

  $$L^\ell_s = w_\ell \sum_{i,j} \left(G^\ell_{ij} - A^\ell_{ij}\right)^2$$

  Normally in real case we compute the style loss of multiple layers $\mathcal{L}$ instead of just a single layer ℓ; then the total style loss is the sum of style losses at each layer:

  $$L_s = \sum_{\ell \in \mathcal{L}} L^\ell_s$$

- Total-variation regularization

  It's also helpful to encourage smoothness and penalize for wiggles or total variation in pixel values in the generated images. It can be computed as the sum of the squared differences in the pixel values for all pairs of pixels horizontally or vertically next to each other. Thus, total-variation regularization is defined as follows, where the the total-variation are summed for each RGB channels, and the weight is w_t:

  $$L_{tv} = w_t \times \left(\sum_{c=1}^{3} \sum_{i=1}^{H-1} \sum_{j=1}^{W} (x_{i+1,j,c} - x_{i,j,c})^2 + \sum_{c=1}^{3} \sum_{i=1}^{H} \sum_{j=1}^{W-1} (x_{i,j+1,c} - x_{i,j,c})^2 \right)$$

3 CycleGAN

3.1 Why use CycleGAN?

Style Transfer is good if we only have one single style reference image. But in my case actually the problem is more similar to what-so-called image-to-image translation, where we would like to learn a translation from one domain to another, for instance, translating summer landscapes to winter landscapes (or the reverse) or translating paintings to photographs (or the reverse). In addition, we have a bunch of images available both from domain A (sketches) and domain B (colored sketches) but most of them are not paired, which is exactly what CycleGAN is trying to solve. Therefore using CycleGAN may be more reasonable to color sketches for industrial designers.

3.2 What is a GAN?

Before going further to CycleGAN, I learned the basic knowledge of GAN to build a solid foundation for understanding CycleGAN.

In 2014, Goodfellow et al. presented a method for training generative models called Generative Adversarial Networks (GANs). In a GAN, a traditional classification network called the discriminator and a generative network called the generator are built to do adversarial training. The discriminator will take images, and classify them as being real (from the training set) or fake (not from the training set). The generator will take random noise as input and transform it into produce images via neural networks. The aim of the generator is to learn to fool the discriminator into thinking the images it produced are real. The structure of GAN is shown in Figure 3.1.

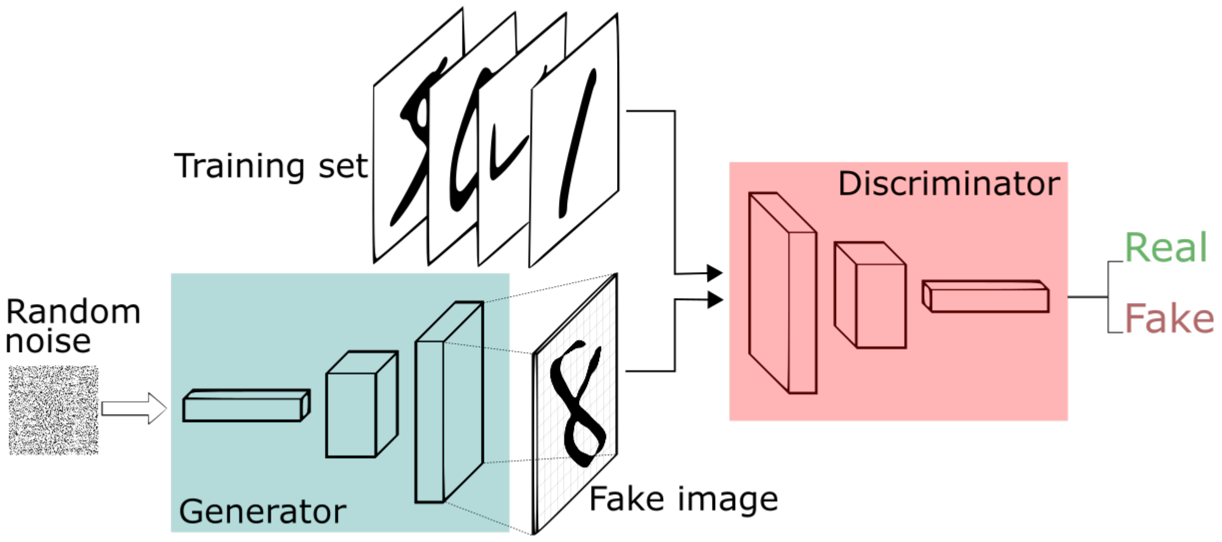

Figure 3.1: The structure of a standard GAN

We can view the whole adversarial training as a minimax game:

$$\underset{G}{\text{minimize}}\ \underset{D}{\text{maximize}}\ \mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

where $z \sim p(z)$ are the random noise samples, $G(z)$ are the generated images using the neural network generator G, and D is the output of the discriminator.

To optimize this minimax game, we have to alternate between taking gradient descent steps on the objective for G, and gradient ascent steps on the objective for D:

1. Update the generator (G) to maximize the probability of the discriminator making the incorrect choice on generated data:

$$\underset{G}{\text{maximize}}\ \mathbb{E}_{z\sim p(z)}\left[\log D(G(z))\right]$$

2. Update the discriminator (D), to maximize the probability of the discriminator making the correct choice on real and generated data:

$$\underset{D}{\text{maximize}}\ \mathbb{E}_{x\sim p_{\text{data}}}\left[\log D(x)\right] + \mathbb{E}_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

We can see that if we use GANs to do image-to-image translation, which means to use images from domain A as inputs to the generator and let discriminators to discriminate generated images from the images from domain B, we can have a huge problem. Since our images are not paired, our GAN can simply learn to remember one image from domain B and always use it as the output image, which can easily fool the discriminator but definitely not the model we would like to have. That's the reason why we would like to use CycleGAN.

3.3 Structure of CycleGAN

We are interested in translating images from sketches to colored sketches and colored sketches to sketches (not we really want but used as assistance for learning the correct translation). And we have two sets of images and they are unpaired, meaning they are images of different products; we don't have the exact same product in sketch and colored sketch.

Set 1: Images of sketches. Set 2: Images of colored sketches. To overcome the problem of having unpaired data, [3] proposed a special GAN structure called CycleGAN which consists of two complete GANs, both with a discriminator and a generator model, meaning there are four models in total in the architecture. The structure of CycleGAN is shown in Figure 3.2.

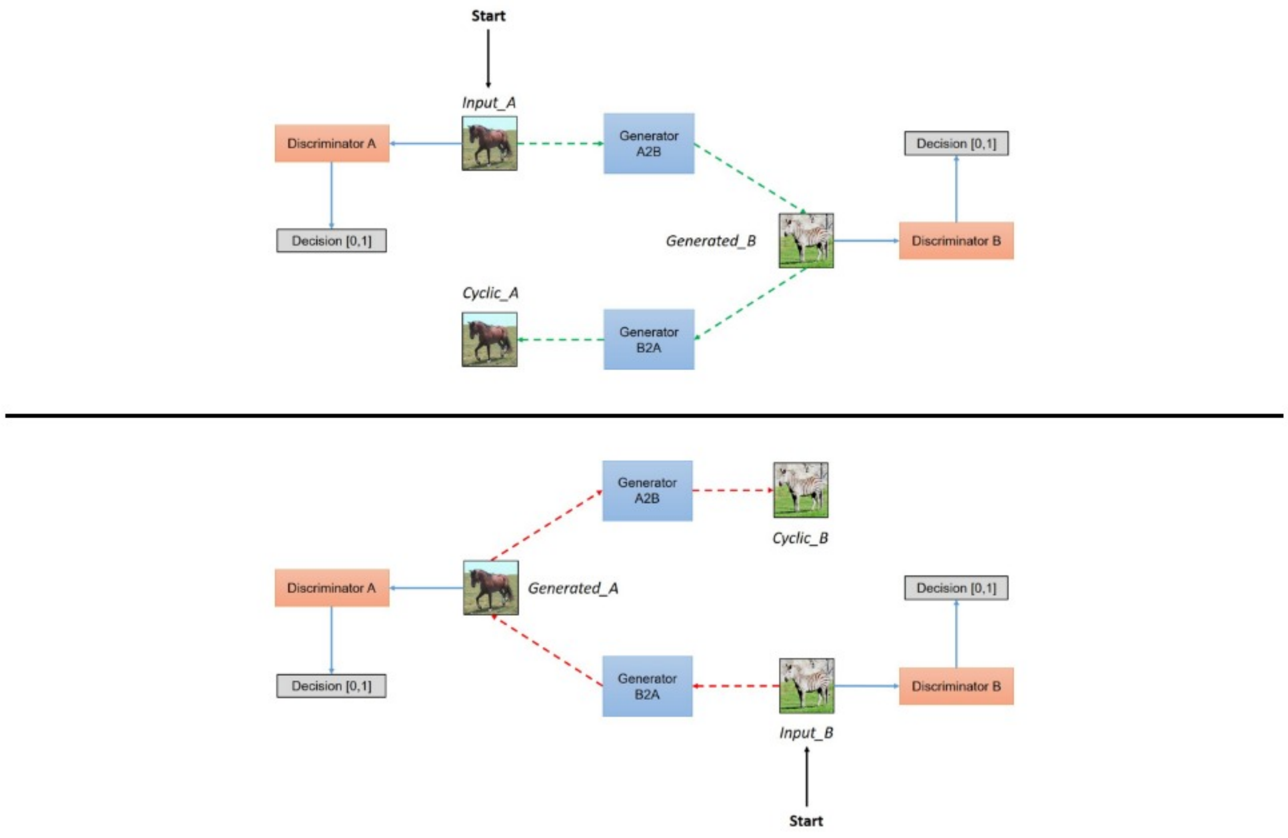

Figure 3.2: The structure of CycleGAN model

The first GAN (Generator model 1 and Discriminator model 1) will generate photos of colored sketches given images of sketches, and the second GAN (Generator model 2 and Discriminator model 2) will generate photos of sketches given images of colored sketches. The discriminator and generator models for each GAN are trained under normal adversarial loss like a standard GAN model. This is sufficient for generating plausible images in the target domain but it's not sufficient for learning correct translations of the input image. Consequently, we must incorporate other sources of loss.

In summary, we have 3 sources of loss. First one is the normal GAN loss. Second one is so-called cycle consistency loss, which is designed to encourage the synthesized images in the target domain that are translations of the input image. It compares an input photo to the CycleGAN to the generated image and calculates the summed absolute difference between these two in pixel values. In addition, we have identity loss that ensure that the output of Generator model 2 should equal the sketch if real sketch is fed and the output of Generator model 1 should equal the colored sketch if real colored sketch is fed. The final loss function is the weighted sum of these three losses.

References

[1] Peizhe Hou. Project of Intelligent Computational Media. Github Repository. https://github.com/peizhehou/Project-of-Intelligent-Computational-Media

[2] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *CVPR*. 2016.

[3] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In *ICCV*. 2017.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair†, Aaron Courville, Yoshua Bengio. Generative Adversarial Nets. arXiv preprint arXiv:1406.2661, 2014.

[5] Aitor Ruano. Pytorch-CycleGAN. Github Repository. 2017. https://github.com/aitorzip/PyTorch-CycleGAN

[6] Chris Nicholson. A Beginner's Guide to Generative Adversarial Networks (GANs). https://pathmind.com/wiki/generative-adversarial-network-gan

[7] Taebum Kim. Anime Sketch Colorization Pair. Kaggle Dataset. https://www.kaggle.com/ktaebum/anime-sketch-colorization-pair

[8] Winnie Lin. Notes and assignments for Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition. Github Repository. https://github.com/cs231n/cs231n.github.io