Images generation for face verification

Chow Kwun Tat
Lui Man Hon

The Hong Kong University of Science and Technology
in Mathematics
Supervised by Prof. Shingyu Leung

December 6, 2017

Abstract

Generative adversarial network (GAN) is a powerful tool to generate a large face image dataset. Together with sufficient classified face image dataset, we hope that the accuracy of the face verification will be improved by providing a large photorealistic face dataset, which is generated by a trained ‘generating-photorealistic-face-image’ machine.

Introduction

Recently, face verification technology has widely applied to the mobile device securities replacing the traditional alphanumeric character password or fingerprint identification. Face verification aims to identify two face images – an original image and a captured image, then compare the differences between two face images. Although the verification has achieved over 99% accuracy by the deep learning algorithm, there exists errors because of the lighting condition, environmental or inter-personal variation, etc. Therefore, our approach is to develop a ‘generating-photorealistic-face-image’ machine which synthesizes one face image with various style images. After inputting a large amount of photorealistic face images (Fig. 1) to the face verification program, it increased the probability of successful face verification. Thus, the performance of face verification will be enhanced.

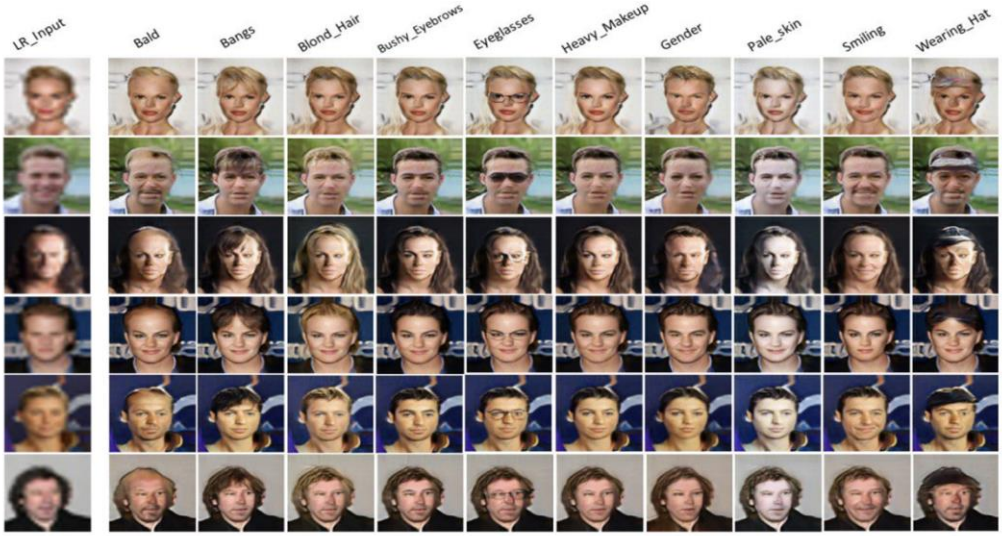

Figure 1: Photorealistic face image sample

Preparation of datasets

To develop a 'generating-photorealistic-face-image' machine, it is necessary to prepare sufficient classified datasets. For each training epochs, the organisation of the two different style face images, which downloaded from CelebA free source dataset[1], is very important. For instance, input 2000 male and female face images to dataset A and B respectively. Then, we separated two style of face image datasets into 4 files – trainA, trainB, testA and testB. The trainA/B file is used to develop machine, whereas the testA/B file is used to generate the results. After that, we randomly input 80% of the male face images file to trainA and 20% to testA separately. Similarly, with the same amount of ratio, we randomly input the female face images files to trainB and testB. In order to maximize the efficiency of the training process, it is vital to delete all repetitive images within train and test file.

After organising the datasets, the next step is to pair up trainA and trainB datasets so that the training machine will be able to investigate the similarity part of the image, such as face. Traditionally, people would buy the paired dataset and extract them in pix2pix[2] but the cost is extremely high. Therefore, we planned to use CycleGAN[3]which is able to pair up trainA and train B dataset by undergoing inverse function. There are two steps on pairing two datasets. (Fig. 2) Firstly, we inserted file *input_A* to the forward GenertorA2B forming file *Generated_B*. Then, we inserted file *Generated_B* to the backward GeneratorB2A forming a new file *Cyclic_A* which is exactly same as *input_A*. Finally, we paired two datasets by repeating the process mentioned above for all image files. (Fig. 3)

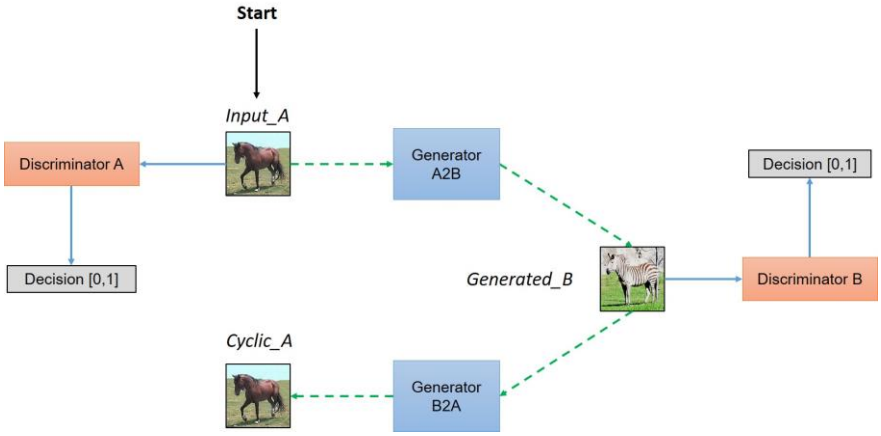

Figure 2: The process of CycleGAN

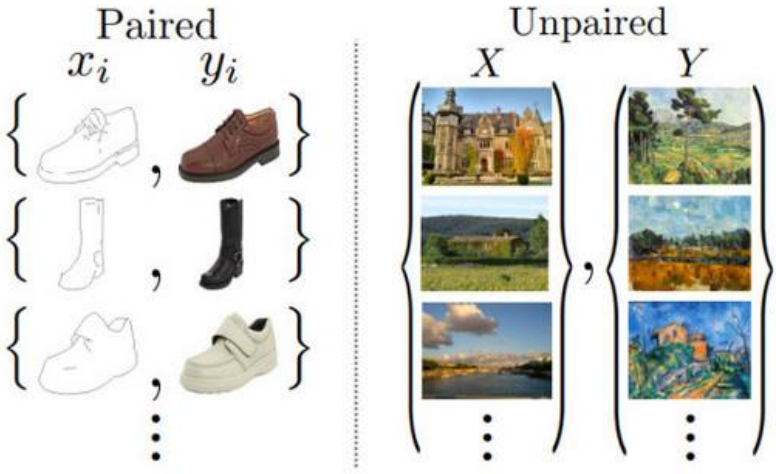

Figure 3: The difference between paired and unpaired datasets

Method to build a GAN

To generate photorealistic face images, it is necessary to build a generative adversarial network (GAN). There are two components within GAN – Generator and Discriminator (Fig. 4). Analogically, generator acts as a cheater, generating fake images from generator distribution $P_G(x)$ to fool the discriminator. To generate fake images, the generator transforms a noise variable $z \sim P_{noise}(z)$ into a sample G(z) from the real data distribution $P_{data}(x)$. On the other hand, the discriminator acts as a judger, eliminating fake images from the sample distribution G(z). The discriminator aims to distinguish between the all the sample images from $P_G(x)$ and $P_{data}(x)$. Therefore, after the confrontation between the generator and the discriminator, an equilibrium has reached. Hence, the discriminator will no longer be fooled by the generator. Finally, an optimal discriminator $D(x) = P_{data}(x)/(P_{data}(x) + P_G(x))$ is trained. The mathematical expression is given by the following expression:

$$\min_G \max_D V(D,G) = \min_G \max_D \left(\mathbb{E}_{x\sim p_{data}}[logD(x)] + \mathbb{E}_{z\sim p_z}[log(1-D(G(z)))]\right) \quad (1)$$

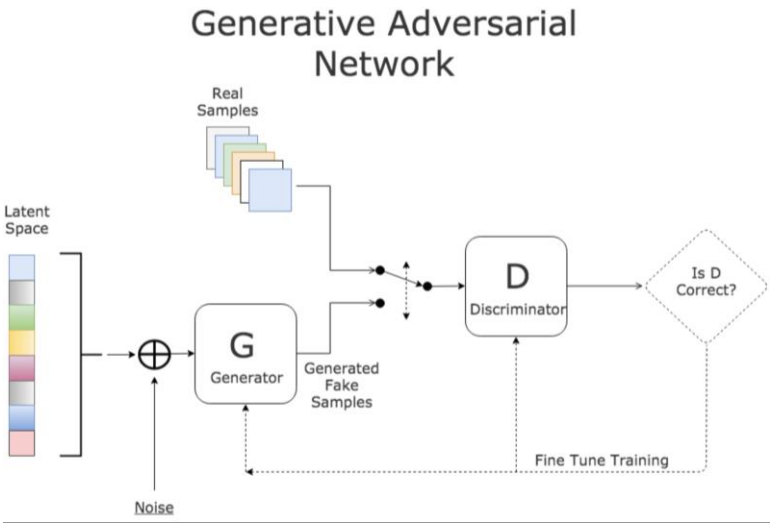

Figure 4: The process of generative adversarial network

Method to build a generator

The function of a generator is to transform styles to target image. For example, transform female's physical characteristics to a male image. There are three components needed to build a generator – encoder, transformer and decoder. (Fig. 5) In general, the encoder acts as a translator extracting the pixels within the image and transforms into a 1-by-3 matrix (which represents the colour of a pixel, such as [256,256,3]) in order to process the dataset by training machine. Then the encoder will send a large scale of encoded matrix to transformer for interpretation. The transformer investigates the similarities area among the encoded matrix and inject the style to the similarities area. After that, transformer will send the transformed matrix to decoder for transforming back to pixels (i.e. fake images). Finally, generator successfully generates a fake image and sends to the discriminator.

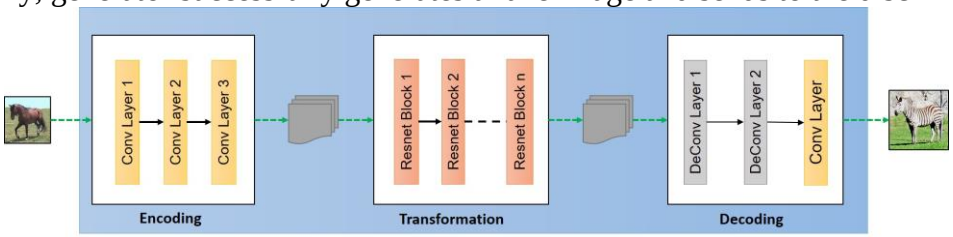

Figure 5: The process of the generator

Method to build a discriminator

The function of a discriminator is to distinguish the input image if it is an original image from real sample dataset $P_G(x)$ or it is a fake image from the generator. (Fig. 6) We set a 'model answer' list for the discriminator by inserting another real sample dataset $P_G(x)$ so that the discriminator is able to compare the input image with 'the model answer' list. At the beginning stage, the discriminator will extract the features among all the images in the dataset and make the decision to classify the feature. (Fig. 7) We let ndf be the number of features for the first layer of discriminator and we also added the final convolutional layer to from 1 dimensional output. The results will be represented by the output file *Dec_input*, that the original image or fake image will be recorded as 1 or 0 respectively. Finally, the discriminator will send the file *Dec_input* to generator in order to improve the image generation process.

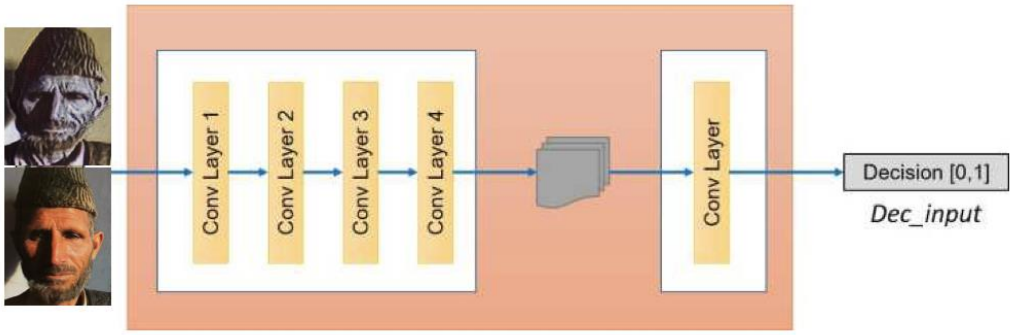

Figure 6: The process of the discriminator

```
o_c1 = general_conv2d(input_disc, ndf, f, f, 2, 2)
o_c2 = general_conv2d(o_c1, ndf*2, f, f, 2, 2)
o_enc_A = general_conv2d(o_c2, ndf*4, f, f, 2, 2)
o_c4 = general_conv2d(o_enc_A, ndf*8, f, f, 2, 2)
```

Figure 7: The code of feature extraction and classification

Method to connect the generator and discriminator and start training

In CycleGAN, in order to get the unpaired dataset to be paired, we set two generator and discriminator. To build a reversible model, we set the model workable in both direction which are AtoB and BtoA. Therefore, in total we have GeneratorAtoB, GeneratorBtoA, DiscriminatorAtoB and DiscriminatorBtoA.

We used the following code to get the input from the dataset.

```
input_A = tf.placeholder(tf.float32, [batch_size, img_width, img_height, img_layer], name="input_A")
input_B = tf.placeholder(tf.float32, [batch_size, img_width, img_height, img_layer], name="input_B")
```

As mentioned above, to set up the generator (gen) and discriminator (dec), we used the following code.

```
gen_B = build_generator(input_A, name="generator_AtoB")
gen_A = build_generator(input_B, name="generator_BtoA")
dec_A = build_discriminator(input_A, name="discriminator_A")
dec_B = build_discriminator(input_B, name="discriminator_B")

dec_gen_A = build_discriminator(gen_A, "discriminator_A")
dec_gen_B = build_discriminator(gen_B, "discriminator_B")
cyc_A = build_generator(gen_B, "generator_BtoA")
cyc_B = build_generator(gen_A, "generator_AtoB")
```

After setting up all the model, we start to train our model as the following.

```
for epoch in range(0,100):
    # Define the learning rate schedule. The learning rate is kept
    # constant upto 100 epochs and then slowly decayed
    if(epoch < 100) :
        curr_lr = 0.0002
    else:
        curr_lr = 0.0002 - 0.0002*(epoch-100)/100

    # Running the training loop for all batches
    for ptr in range(0,num_images):

        # Train generator G_A->B
        _, gen_B_temp = sess.run([g_A_trainer, gen_B],
                                 feed_dict={input_A:A_input[ptr], input_B:B_input[ptr], lr:curr_lr})

        # We need gen_B_temp because to calculate the error in training D_B
        _ = sess.run([d_B_trainer],
                     feed_dict={input_A:A_input[ptr], input_B:B_input[ptr], lr:curr_lr})

        # Same for G_B->A and D_A as follow
        _, gen_A_temp = sess.run([g_B_trainer, gen_A],
                                 feed_dict={input_A:A_input[ptr], input_B:B_input[ptr], lr:curr_lr})
        _ = sess.run([d_A_trainer],
                     feed_dict={input_A:A_input[ptr], input_B:B_input[ptr], lr:curr_lr})
```

Loss function

To generate a qualified image, the data loss should be minimized. Therefore, we have to conduct optimization of the process to improve photo quality and performance of the transformation process. Here we define the condition of the loss function.

1. Discriminator should approve all original image.
If the original image passed through the discriminator, it will return the output 1 by the discriminator. To minimize our loss, it is necessary to train Discriminator A to minimize |Discriminator A(a)-1|. Repeat the steps for Discriminator B.

2. Discriminator should reject all generated image.
When the discriminator rejects the image, it will return the output 0. To minimize our loss, it is necessary to train Discriminator A to minimize $\left|Discriminator\ A\left(Generator\ BtoA(b)\right)\right|$. Repeat the steps for Discriminator B.

3. Generator should make the discriminator approve all the generated image.
If the generator successfully fools the discriminator by the generated image, the discriminator will return the output 1. To minimize our loss, it is necessary to train the GeneratorAtoB to minimize $\left|Discriminator\ B\left(Generator\ AtoB(a)\right)-1\right|$. Repeat the steps for GeneratorBtoA.

4. All generated image should keep the property of original image. For example, if we use the GeneratorAtoB to generate an image, we must be able to use the Generator BtoA to get back to a

cyclic image which is similar to the original image. That we call cyclic-consistency satisfaction. Therefore, we have to minimize the difference between the original image and the cyclic image. We denoted the loss as cyc_loss.

```
cyc_loss = tf.reduce_mean(tf.abs(input_A-cyc_A)) + tf.reduce_mean(tf.abs(input_B-cyc_B))
```

Finally, we set the 10 as the coefficient of cyc_loss in order to show the importance order of cyclic loss better than discriminator loss.

Results

In our research, we classified female, male and zebra datasets to train the 'generating-photorealistic-face-image' machine. There are two sets of results. The first result is a reversible transformation machine which generated a fake man image from an original female image or vice versa. (Fig. 8) It costed 30 days with 100 epochs. Another result is an irreversible transformation machine which merged zebra-stripe to human face. (Fig. 9) and it leaded to the corrugation effect of human face. It costed 3 days with 10 epochs.

Figure 8: Results from the female-to-male generation (Left) and male-to-female generation (Right)

Figure 9: Results from the zebra-to-human generation

Discussion

At the beginning stage of our research, we have used different approaches in order to generate a large photorealistic face images dataset.

Our first approach is to generate a three-dimension (3-D) face shape model from a two-dimension (2-D) input face image. The 3-D face reconstruction is mainly about recovering the 3-D facial geometry from some 2-D images. After successfully generating the 3-D face shape model, we can obtain numerous face images captured from the output 3-D face shape model by rotating it with different angles. The output result is like in (Fig. 10) below. However, by using the above method, there is a challenge that we do not manage to change the styles of the input face image, for example, changing the color of the skin or wearing the sunglasses on the face model, but only the angles. Also, if the person in that face image is wearing some eyeglasses, this 3-D face shape model cannot form the shape of the eyeglasses, so we have some retrictions with this approach.

Figure 10: The captured face images by rotating the 3-D face shape model

Then, we tried another approach that is using a pre-trained model VGG-16[5] in Tensorflow to change the style of an input image. VGG-16 required two sets of images – a content image and a style image to form an output – a mixed image. (Fig. 11) However, there exist two problems during the photorealistic face images generation. With using this approach, we are not able to change some parts of the input content image only, such as changing the style the face part. It means that this pre-trained model can only change the style of the whole input image, including the background like the mixed image in figure 2. Apart from this mentioned challenge, the another one is the problem of high time cost. Basically the time we need to use to generate one output mixture image is around 10 minutes by a normal laptop computer without GPU nowadays. With a view to dealing with the time consuming and the partly style changing problem, we try to use the another approach to train the model by ourselves in machine learning.

Figure 11: A result from pre-trained model VGG-16

Therefore, we decided to build a new machine model – CycleGAN which is able to accumulate the experience after every successful or failure image synthesis trial. Therefore, the generator is now able to selectively change the style of the face in an image. We also improved our training dataset by regulating same size among all the images in order to enhance the transformation precision.

Problem and limitation

One of the problems we faced in this project is the time consuming problem in the training process, not the running time, but this problem is not unsolvable. Since there is a limition that our computers and the free charge platform in google cloud tensorflow only offer the CPU, we basically need several days to train an acceptable model in google cloud tensorflow or our laptop computers, but this model is still not the ideal one. However, if our computers have GPU inside, the time of the training process should be within one day to obtain a wonderful model.

Although we are satisfied with the results generated by CycleGAN, there still remained challenges, such as the long training time and the quality limitation during the face generation process. (Fig. 11) According to Figure 11, the quality of a generative image is directly proportional to the epoch which is the training time. Thus, it is necessary to proceed 100 epochs (30days) for the sake of generating satisfied photorealistic face images. Hence, we hope that we will have more resources to train our machine in more epochs by building a GPU/TPU SSH computers with at least 16GB ram on the Google Cloud Platform.

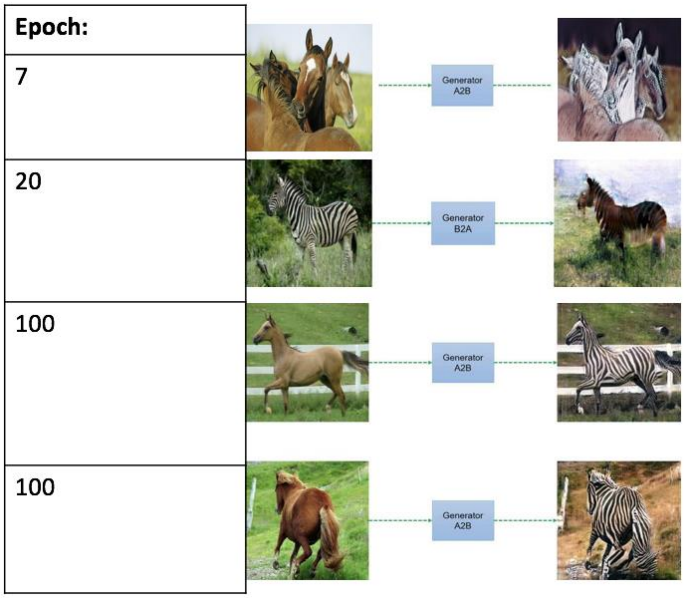

Figure 11: The relation between image quality and epoch

In addition to the resource problem, in the absence of the perfect classified datasets, this causes that our results may not be optimal solutions. We hope to collect better dataset with classified style. For example, a dataset with 2000 human wearing hat images, a dataset with 2000 Asian or White. Since we collected the image data from the free resources website, some of the image has become unreachable (Fig. 12) that deteriorate the training process. Having some errors in the datasets downloaded from the Internet, such as mixing with some unrelated images in the downloaded dataset folders, it may enable our model to have some deviation during the training process, but there is still a suggestion to minimise this error, which is using the trained translation model from images to sentences. Therefore, we hope we will have more resources to get access to better classified data sources so that we will reach an optimal result. With using this translation model like in (Fig. 13), we can simply loop for each image in the datasets to check whether those images is related to what we want or not. Then, we can get the perfect classified datasets to train the model.

Figure 12: The un-linkable and unclassified dataset

Figure 13: The example of the ideal translation model

Conclusion

With the CycleGAN trained machine, we are able to generate a large photorealistic face images dataset with various style of the same person. Then, we could input the dataset to the face verification program to enhance the performance and increase the probability of successful face recognition.

Reference

[1] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. CoRR, abs/1411.7766, 2014.

[2] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. CoRR, abs/1611.07004, 2016.

[3] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. arXiv preprint arXiv:1703.10593, 2017.

[4] Ahyoung Shin, Seong-Whan Lee, Heinrich Bülthoff, Christian Wallraven, "A morphable 3D-model of Korean faces", Systems Man and Cybernetics (SMC) 2012 IEEE International Conference on, pp. 2283-2288, 2012.

[5] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photorealistic single image super-resolution using a generative adversarial network. CoRR, abs/1609.04802, 2016.

Relative articles

1. J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. arXiv preprint arXiv:1703.10593, 2017.

2. C. Dong, C. C. Loy, K. He, and X. Tang. Learning a Deep Convolutional Network for Image SuperResolution, pages 184–199. Springer International Publishing, Cham, 2014.

3. F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In CVPR, June 2015.

4. Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." ICLR 2015

5. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Identity Mappings in Deep Residual Networks." ECCV 2016.

6. Hardik Bansal , Archit Rathore. Understanding and Implementing CycleGAN in TensorFlow [Blog post]. Retrieved from https://hardikbansal.github.io/CycleGANBlog/