CMSC 25040 Final Project: Using CycleGAN to Generate Anime Faces from Selfies

By: Carlos Cardona, Taeang Moon

Objective:
There is a class of computer vision problems dedicated to image-to-image translations where the goal is to learn a mapping between an input image and output image using aligned image pairs. The difficulty with this, however, is that paired training data is not always available. CycleGAN serves as a method to handle this objective while keeping this limitation in mind. We attempt to adopt this approach into the task of translating anime faces onto female selfie images to effectively convert the selfie into an anime character.

Background:
CycleGAN is an extension of a normal generative adversarial network (GAN). A typical GAN uses both a generator to generate images and a discriminator to evaluate the generated image. It is trained in such a way where the generator is updated to fool the discriminator, and the discriminator is updated to be better at detecting generated images leading to more accurate images. CycleGAN extends this approach and aims to create an image-to-image translation by training an additional generator and discriminator model. One generator uses the first domain images to output second domain outputs, and another uses second domain images to generate images for the first domain. The discriminators attempt to read these generated images and update the generators to improve the performance and accuracy of the generated images. In order to generate translations, however, there is an addition of cycle consistency.

Cycle consistency can be thought of similarly to language translation tasks, where for example, a sentence can be translated from English to another language, and then from that language back to English and remain the same sentence. This is applied to vision where a mapping is allowed by ensuring that the first generator’s output can be used as input to the second generator, and the second generator’s output can match the original image. Likewise, the output of the second generator can be used as input to the first, and the output of the first can match the original input to the second. In our application, we use cycle consistency in CycleGAN to go from a selfie image to an anime image, and then back to the original selfie while training.

The dataset used is composed of female selfie pictures and pictures of female anime characters, which are both split into their respective training and testing sets. Each of the training sets have 3400 pictures and the test sets have 100 pictures. We extracted these pictures from the dataset and worked with their tensor representation of size [3, 256, 256]. We also modified the folder of the dataset a little bit by putting each subfolder into a class folder (for example train A was put into another folder called Atrain) so that we can more easily use the built in pytorch ImageFolder.

Method:
Our implementation of CycleGAN includes two generators (one for selfie to anime and another for anime to selfie) and two discriminators (one to find selfies and another to find anime pics). However, both generators and discriminators use the same architecture. For our Generators, we used a U-NET like structure composed of an encoder, a residual network, and a decoder. Each batch was composed of a single picture as implemented in the original paper. In the encoder block of the generator, we go from the original batch image of size [1, 3, 256, 256] to a size of [1, 256, 64, 64]. We do this through multiple convolutional, normalization, and ReLU layers. More specifically, in the encoder, the input (after first passing through a single reflection pad to keep sizes even) goes through three loops of a convolutional layer to an instance normalization layer to a ReLU layer. The first convolutional layer has a kernel of size 7, stride of size 1, and padding of 0 transforming the padded input to a tensor of size [1, 64, 256, 256]. The second and third layers have kernels of size 3, stride of 2, and padding of 1, and transforms the [1, 64, 256, 256] tensor first to a [1, 128, 128, 128] size tensor and ultimately to a [1, 256, 64, 64] size tensor, that is then inputted into the residual network layers. In the original CycleGAN paper, they use 9 residual blocks for a 256 x 256 image, but we use 6 for runtime efficiency. In each of these blocks, the input goes through a convolutional layer, a normalization layer, a ReLU layer, a dropout layer, another convolutional layer, and a final normalization layer in that order. These blocks keep the size of the input tensor constant. After these residual networks, the resultant image tensor is brought to the decoding part and gets upsampled back to its original shape through convolutional transpose, normalization, and ReLU layers. This part of the generator basically does the encoding part in reverse so it up-samples the image from [1, 256, 64, 64] to first a [1, 128, 128, 128] and then a [1, 64, 256, 256] tensor. Then it is mirror padded again and sent to a convolutional layer that changes it back to a [1, 3, 256, 256] image which is finally sent through a tanh activation layer to get the final output.

Our discriminator is a little simpler than the generator structure. Namely, the discriminator simply receives a [1, 3, 256, 256] batch image input and down-samples it to a [1, 3, 30, 30] decision tensor. It does so through multiple convolutional, normalization, and Leaky ReLU layers. Specifically, the input first goes through a convolutional and a Leaky ReLU layer. Each Leaky ReLU in this discriminator architecture has negative slope 0.2 and each convolutional layer has kernel size of 4. This first convolutional layer also has a stride of 2 and a padding of 1 and transforms the input to a [1, 64, 128, 128] tensor. This tensor is then inputted into 3 loops of a convolutional, followed by a normalization, and a Leaky ReLU layer. The first two convolutional networks in this loop have stride 2 and padding 1 while the last one has a stride 1 and padding 1. This loop transforms the [1, 64, 128, 128] tensor to a [1, 128, 64, 64] tensor then a [1, 256, 32, 32] size tensor, and finally a [1, 512, 31, 31] tensor. This [1, 512, 31, 31] tensor is ultimately inputted into a final convolutional layer of stride 1 and padding 1 that converts it into a [1, 3, 30, 30] tensor that will be used to determine whether the image is an anime face or not.

Now that we have discussed the architecture of the generators and discriminators, we can discuss how we found the generator loss and the discriminator loss. To find the overall generator loss, we summed up the losses from each generator (selfie to anime and anime to selfie) and the cycle losses. To find the selfie to anime generator loss, we used an MSE loss between a label of ones and the output of the anime photo discriminator judging a fake anime photo. A label of ones implies the anime pic discriminator thinks that the photo is an anime pic. This MSE loss is useful, since the closer the discriminator's decision is to 1 the better the selfie to anime generator is at tricking the discriminator and thus the lower this MSE loss is the better the generator is. We followed a similar process with the anime to selfie generator, just flipping the inputs. Then we found the cycle loss which was a main component of the project. To find the cycle loss of the selfie to anime generator, we looked at the L1 Loss between a real selfie face and the generated selfie face inputted with the fake anime photo that the original selfie face generated. This loss was multiplied by a lambda factor of 10. Similarly to the MSE loss described above, we want this L1 Loss to be small so that the real selfie face is similar to the generated selfie face when going through the cycle. Again, to find the cycle loss of the anime to selfie generator we followed similar steps with the inputs flipped. Then we let the overall generator loss be the sum of all the losses mentioned above (the two individual generator losses and the two cycle losses).

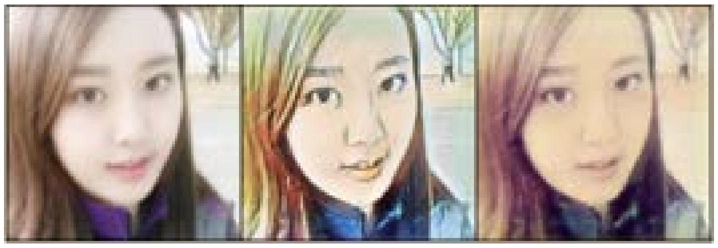
To find the loss of the discriminators we used MSE loss again, but this time we wanted the reading of the discriminator to be good. Thus, the discriminator loss measures a sum of two component losses. First, we look at the MSE loss between the discriminator's decision on a real selfie/anime pic and a label of 1's. We want this loss to be small so that the decision is as close to 1 as possible. Second, we look at the MSE loss between the discriminator's decision on a fake selfie/anime picture and a label of 0's. For the fake pictures, we sampled from a list of 20 of the past fake photos to reduce model oscillation as recommended in the original paper. Similarly, we would want this loss to be small so that the discriminator correctly distinguishes false selfies or anime pictures. Then we set each discriminator loss (one for the selfies and one for the anime pictures) to be the sum of those two MSE losses times a factor of ½. Since we want each individual component loss to be small, we would want to optimize so that the general sum of the two is small as well.

Finally, to optimize the model and the losses, we used the Adam Optimizer with a learning rate of 0.0002 and trained the model for 50 epochs. Our results for these transformation pictures are shown below.

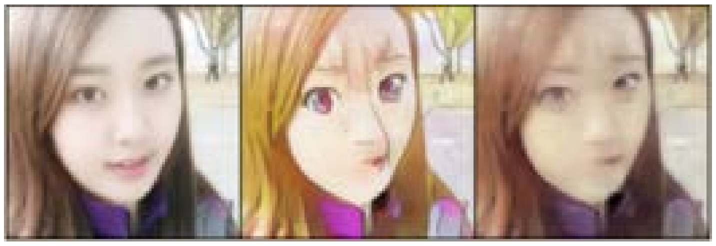
Results:
Following our training for 50 epochs, our observed generated images for 50 test pictures are shown in the appendix below, but we show an example of one test picture changing through multiple epochs here. In this series of pictures, the leftmost is the original selfie, the middle is the generated anime picture, and the last one is the reversion back to the selfie picture. As we can

see, the difference between the 1st epoch and the 50th show the improvement in quality amongst the mapping of anime to selfie. In the first epoch, the image is only sort of blurred and has just begun to change, and is easily revertible. By the 25th epoch, there are many distinguishing features of an anime face in the transformation, but it is still a little blurry, but by the 50th, we get a pretty clear picture of an anime generated face of the person. Our selfie generator also reverts the anime picture back to a pretty similar real selfie to the original, although some parts are blurrier, and some resemblance of the anime picture remains. Overall though, the picture reverts back nicely to the original and shows the cycle loss at work.

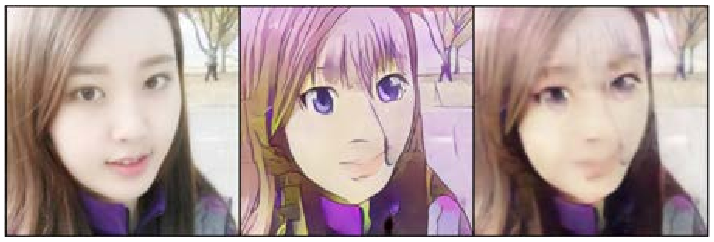
Epoch 1



Epoch 25



Epoch 50



Conclusion:
In conclusion, we saw how effective CycleGAN was in performing image-to-image translation. There are some limitations, however, as the period for training was long. This can be improved upon by using lower quality images (at the expense of quality) or by using stronger hardware. Not only that, but the original image faces some difficulty in conversion if it is not clear enough. For example, selfies without a clear shot of the face reduced the accuracy of the anime conversion. Also, sometimes there would be hair generated for the anime generation where the original selfie didn't have hair before. Finally, the cycle consistency wasn't always able to reliably return from anime to selfie and would retain some anime features like enlarged eyes. In future steps, we would like to see the results after adding modifications like an identity loss and different convolutional architectures but overall we saw some great performance on this task.

APPENDIX: OUTPUTS OF 50 TEST IMAGES