CMSC 25040: Final Project Report

Kameel Khabaz and Christian Urbanek

March 10, 2023

1 Project Objective

The objective of this project was to utilize a CycleGAN architecture in order to translate between images of selfies and anime. CycleGAN (Cycle-Consistnet Generative Adversarial Network) is a neural network architecture for image-to-image translation between unpaired groups of image classes [1]. Generative Adversarial Networks (GANs) utilize an adversarial loss to train a generative model G and a discriminative model D in an adversarial process to create images that look realistic in relation to the training data [2]. A CycleGAN applies this framework to convert between two groups of images from different domains (e.g. photographs and works of art) without any paired images in the training data. It learns a mapping $G : A \longrightarrow B$ from source domain A to target domain B such that $G(A)$ is regarded as being from domain B, as well as an inverse mapping $F : B \longrightarrow A$ such that $F(B)$ is regarded as being from domain A. In addition, a cycle consistency criterion $F(G(A)) \approx A$ and $G(F(B)) \approx B$ and an identity criterion $F(A) \approx A$ and $G(B) \approx B$ are enforced [1].

In this project, we created a CycleGAN to translate between images of selfies and anime. Unlike neural style transfer, which translates the semantic content of an image in different styles by learning from paired images, we use the CycleGAN framework to learn the mapping between two image collections (selfies and anime) [1, 3]. The anime pictures were generated to look like they were inspired by the original selfies in the characteristic anime design, replicating idiosyncratic elements of anime style (e.g. unusually large eyes, a two-dimensional appearance, high contrast, and vibrant stylization). At the same time, we aimed to achieve the inverse transformation from artificial anime art into realistic images of human selfies.

2 Methods

The methods involved with our project surrounded adapting the general CycleGAN architecture to our unique domain problem and training data through experimenting with a number of network components and implementation decisions.

We began by implementing the initial CycleGAN architecture for 256×256-sized images [1]. The architecture's generator network consists of three convolutions, nine residual blocks, two

fractionally-strided convolutions, and one final convolution to map features to a 3 color channels. The discriminator network uses a PatchGAN architecture in which instead of outputting a single binary value (real/fake), a $N \times N$ output vector representing patches of the image is produced (classifying if each patch is real or fake). By penalizing at the scale of patches, this architecture models high-frequency structure and utilizes a per-pixel loss to force low-frequency correctness [4].

After a series of trials (to be described later in the report), we converged on four additional modifications to this structure that produced higher quality results than the published CycleGAN architecture for our training data: spectral normalization in the discriminator, a leaky rectified linear unit (Leaky ReLU) activation layer in the generators, Xavier initialization for the convolutional layer weights, and a larger weight for the cycle consistency loss. Spectral normalization is a novel weight normalization method that has been found to stabilize the training of discriminator networks [5]. Leaky ReLU is a modification of the rectified linear unit (ReLU) activation function that has a non-zero gradient over the entire domain [6]. Xavier initialization defines the initial weights of convolutional layers such that the scale of the weights is preserved throughout the network (with the variance remaining the same), preventing the weights for deeper layers from exploding to large values or vanishing to 0 [7]. The weighting of the cycle consistency loss λ affects the degree to which this loss is enforced for the generator versus the GAN losses and the identity loss.

Our finalized model architecture is described in full. First, each of the two generator models is composed of a series of encoder blocks, residual blocks, and decoder blocks. Each encoder block was designed to decrease the image size by a factor of two while changing the number of channels as desired. It consists of the following layers:

1. 2D convolution with a kernel of 3, a stride of 2, and a padding of 1
2. 2D instance normalization
3. Leaky ReLU activation (default negative slope of 0.2 was used for all leaky ReLUs)

Each residual block was designed to preserve the image size and adopt a residual framework in which the output of the series of layers is summed with the input. Such a residual architecture was initially developed to improve deep convolutional neural networks for image classification [8]. It consists of the following:

1. 2D convolution with a kernel of 3, a stride of 1, and a padding of 1 (image reflection for the padding)
2. 2D instance normalization
3. Leaky ReLU activation
4. 2D convolution with a kernel of 3, a stride of 1, and a padding of 1 (image reflection for the padding)
5. 2D instance normalization

Decoder blocks were designed to increase the image size by a factor of two while changing the number of channels as desired. It consists of the following layers:

1. Fractionally-strided convolution (ConvTranspose2D) with a kernel size of 3, a stride of 2, a padding of 1, and an output padding of 1
2. 2D instance normalization
3. Leaky ReLU activation

The generator model consisted of the following layers:

1. An initial convolutional block consisting of 2D convolution (kernel of size 7, stride of 1, padding of 3 (using reflection)), 2D instance normalization, and Leaky ReLU activation. This block increases the number of channels in the image from 3 to 64.
2. 2 encoder blocks, each of which doubles the number of channels (so the result has 256 channels)
3. 9 residual blocks
4. 2 decoder blocks, each of which halves the number of channels (so the result has 64 channels)
5. A final convolutional block consisting of a 2D convolution (kernel size 7, stride 1, padding 3 (using reflection)), followed by sigmoid activation to map to the color range for images. The number of channels mapped from 64 to 3.

Note that instance normalization is used instead of batch normalization because we used a batch size of 1 to train the model (larger batch sizes seemed to produce inferior results, and much larger batch sizes caused memory issues for our machines). Furthermore, reflection-based padding is used instead of padding with zeros for several layers to reduce boundary effects. Finally, note how the generator decreases the image size from 256×256 to 64×64 in the encoder section before increasing it back to the original size in the decoder section.

The discriminator architecture consisted of a series of discriminator blocks, each of which halves the size of the image and maps the desired number of input/output channels. Each block consists of the following layers:

1. 2D convolution with a kernel of 4, a stride of 2, and a padding of 1
2. Spectral normalization
3. Leaky ReLU activation

The discriminator model consisted of the following blocks and layers:

1. A discriminator block mapping from 3 to 64 channels (without spectral normalization).
2. A discriminator block mapping from 64 to 128 channels

3. A discriminator block mapping from 128 to 256 channels
4. A discriminator block mapping from 256 to 512 channels
5. Padding the image height and width by 1 pixel each (to get the final height/width of the image to a power of 2).
6. A 2D convolution from 512 channels to 1 channel (each pixel outputs a real/fake discriminator for that patch) with a kernel size of 1, stride of 2, and padding of 0.

Thus, the discriminator changes the image dimensions from $256 \times 256 \times 3$ to $16 \times 16 \times 1$, classifying each patch as real or fake. As we will describe later, we found that this PatchGAN-based architecture performed much better than having the discriminator produce a single 1×1 output.

We utilized an L2 loss function for the discriminator outputs to penalize discrepancies between the patch-level classifications. While training the generators, the discriminator loss seeks to enforce that $D_B(G(A))$ and $D_A(F(B))$ are regarded as true images. While training the discriminators, the loss seeks to enforce that $D_A(A)$ is regarded as real and that $D_A(F(B))$ is regarded as fake (and vice versa).

For the generator outputs, we used an L1 loss to penalize discrepancies in the cycle loss and the identity loss. The cycle consistency loss enforces that $F(G(A)) \approx A$ and $G(F(B)) \approx B$ (preservation of an image after a cycle), while the identity loss preserves the identity mapping: $F(A) \approx A$ and $G(B) \approx B$.

We used Xavier initialization for the weights of the convolutional layers and the default initialization for all other layers [7]. We defined three optimizers: one for the two generator networks, one for the discriminator for the first class D_A, and one for the discriminator for the second class D_B. This allowed us to train the generators together before training each discriminator on its own. The optimizers were Adam optimizers with an initial learning rate of $\lambda = 0.0002$, $\beta_1 = 0.5$, and $\beta_2 = 0.999$ (the default value). To allow for fine-tuning of the model later in the training period, we defined schedulers for learning date decay in which the learning rate decays by a factor of 2 at epoch 5. The model was trained for a total of 10 epochs (due to constraints on time and computational resources).

For each iteration of training, we trained the generators together before training each discriminator on its own. To train the generators, we defined a total generator loss of $\mathcal{L}_G = \mathcal{L}_{GAN} + 20\mathcal{L}_C + 5\mathcal{L}_I$. $\mathcal{L}_{GAN} = (\mathcal{L}_G + \mathcal{L}_F)/2$ defines the loss of the two generator networks, in which we enforce that $G(A)$ is regarded as being from domain B and that $F(B)$ is regarded as being from domain A, respectively. This is tested by running each generator output through the discriminator for the intended class: $D_B(G(A))$ and $D_A(F(B))$. Notice the equation's larger weight on the cycle consistency loss $\mathcal{L}_C = (\mathcal{L}_{C_A} + \mathcal{L}_{C_B})/2$. Here, $\mathcal{L}_{C_A}$ enforces that $F(G(A)) \approx A$ and $\mathcal{L}_{C_B}$ enforces that $G(F(B)) \approx B$. Finally, $\mathcal{L}_I = \mathcal{L}_{I_A} + \mathcal{L}_{I_B}$ defines the identity loss, which enforces that $F(A) \approx A$ and $G(B) \approx B$, respectively.

The discriminator for the first domain A is trained using the loss function $\mathcal{L}_{D_A} = (\mathcal{L}_{real_A} + \mathcal{L}_{fake_A})/2$, in which $\mathcal{L}_{real_A}$ enforces that an image from domain A is classified as such and $\mathcal{L}_{fake_A}$ enforces that a generated image $F(B)$ is classified as fake. Similarly, the discriminator for the second domain B is trained using the loss function $\mathcal{L}_{D_B} = (\mathcal{L}_{real_B} + \mathcal{L}_{fake_B})/2$, in which $\mathcal{L}_{real_B}$ enforces that an image from domain B is classified as real and $\mathcal{L}_{fake_B}$ enforces that a generated image $G(A)$ is classified as fake.

2.1 Model Architecture Modifications

The following subsections describe several modifications that we attempted to make to the underlying model architectures to try to improve the results from the original CycleGAN model.

2.1.1 Upsampling Instead of Fractionally-Strided Convolution

We wanted to compare the effectiveness of using fractionally-strided convolution in the decoder portion of the generator as opposed to a simple combination of nearest-neighbor upsampling followed by convolution. As such, we modified the decoder block to contain the following layers:

1. Upsampling by a factor of 2
2. 2D convolution with a kernel size of 3, stride of 1, and padding of 1
3. 2D instance normalization
4. Leaky ReLU activation

2.1.2 Deeper Discriminator with a Binary Output

Since the authors of the CycleGAN paper intentionally described using a PatchGAN architecture for the discriminator to output a $N \times N$ output vector representing patches of the image, we examined the impact of using a more traditional discriminator that outputs a binary real/fake classification for each image [1]. To obtain this result, we modified the discriminator architecture to the following structure:

1. A discriminator block mapping from 3 to 8 channels (without normalization).
2. A discriminator block mapping from 8 to 16 channels
3. A discriminator block mapping from 16 to 32 channels
4. A discriminator block mapping from 32 to 64 channels
5. A discriminator block mapping from 64 to 128 channels
6. A discriminator block mapping from 128 to 256 channels
7. A discriminator block mapping from 256 to 512 channels

8. Padding the image height and width by 1 pixel each (to get the final height/width of the image to a power of 2).

9. A 2D convolution from 512 channels to 256 channels (each pixel outputs a real/fake discriminator for that patch) with a kernel size of 1, stride of 2, and padding of 0.

10. A 2D convolution from 256 channels to 1 channel (each pixel outputs a real/fake discriminator for that patch) with a kernel size of 1, stride of 2, and padding of 0.

This architecture outputs a 1×1-sized image that serves as a binary classification for that image. Since the discriminator now outputs a binary classification, we changes the loss function for the discriminator from a L2 loss function to a binary cross-entropy loss function.

2.1.3 Modified Generator Kernel Size

We attempted to modify the generator by changing the kernel size for the encoder and decoder from 3 (the original value) to 4 (an even number). Furthermore, the generator architecture was slightly modified to replace the initial and final convolutional blocks (which had a large kernel size of 7) to encoder and decoder blocks. The modified structure is as follows:

1. Initial encoder block that increases the channel number from 3 to 64

2. 2 encoder blocks, each of which doubles the number of channels (so the result has 256 channels)

3. 9 residual blocks

4. 2 decoder blocks, each of which halves the number of channels (so the result has 64 channels)

5. A final decoder block that changes the channel number from 64 to 3

2.1.4 Complex Generator Architecture

We attempted to modify the generator to a more complex architecture that may have a higher capacity for learning how to create realistic-looking selfie and anime images. The first modification was to include an extra encoder block and an extra decoder block in the generator (so 3 instead of 2). Furthermore, we borrowed ideas from the U-Net architecture for image segmentation to implement a broader residual framework for the generator architecture (in addition to the nine residual blocks). Specifically, we reintroduced the output from each encoder block to the model by adding it to the input of the corresponding decoder block. This modification was accomplished using the following modification to the `forward` function for the `Generator` class:

```
    def forward(self, x):
        x = self.initial_conv(x)
        xs = [x]
        for layer in self.downsample:
```

```
            x = layer(x)
            xs.append(x)
        xs = xs[::-1]
        x = self.residual(x)
        for i, layer in enumerate(self.upsample):
            x = layer(xs[i] + x)
        return self.output(x + xs[-1])
```

2.1.5 Increased Number of Residual Blocks in the Generator

To try to increase the capacity of the generator networks to create images that realistically look like they belong to the target domain, we attempted to increase the number of residual blocks in the generator from 9 to 11.

3 Results

Figures 1, 2, and 3 illustrate example outputs for our best model on the testing set data (50 pairs of images from each domain are shown). The generator/discriminator losses throughout the training epochs are shown in Figure 4. The image results are promising not only because of the stylized differences between selfie and anime images, but also (more importantly) because of how the model produces larger transformations between real image space and anime image space. If we compare the real selfie images in the top row and the fake anime images in the second row, we see that the three-dimensional settings of images has been flattened to a two-dimensional cartoon-like feel (characteristic of anime), that the eyes of the people have gotten larger and colored with more vibrancy (as in anime), and that the facial features have more striking contours that are characteristic of anime images. We also see that the real-life backgrounds of the selfie images have been stylized to look like cartoonish drawings, and the natural skin tones from the selfies have transformed to appear characteristic of anime drawings.

If we compare the third row (real anime) and the fourth row (fake selfies), we see that the 2D cartoonish feel of anime images has been transformed to a setting that looks more like real life (more 3D as well) and that the strong contrasts characteristic of anime images have softened. Furthermore, the skin tones in the fake selfie images are more realistic than the highly stylized skin tones in anime, and the backgrounds of the fake selfies look more like real backgrounds instead of cartoonish drawings.

Nonetheless, some limitations of our model include that the sizes of the eyes of the anime characters have not fully decreased to realistic eye sizes in the anime-to-selfie transition, and the hair remains very sharp in nature (unlike the selfie images). Furthermore, the selfie-to-anime transition can be improved because the mouths retain their natural shape in the generated anime images, even though (as shown by the true anime images) mouths in anime are usually depicted as lines.

While we attempted a number of modifications to try to mitigate these limitations, none of the experiments produced positive results in this regard.

Our loss results show the continued adversarial nature expected of Cycle-GAN training, in which discriminator and generator losses are in continual flux as they go up and down. The generator loss also undergoes a gradual overall decrease, indicating that the model is in fact learning over time. This is further illustrated by Figure 5, which shows the evolution of the generated images throughout the training span. We see that there is more noticeable improvement in the images near the beginning of training (such as between epochs 1 and 2) and more fine-tuned improvement later in training.

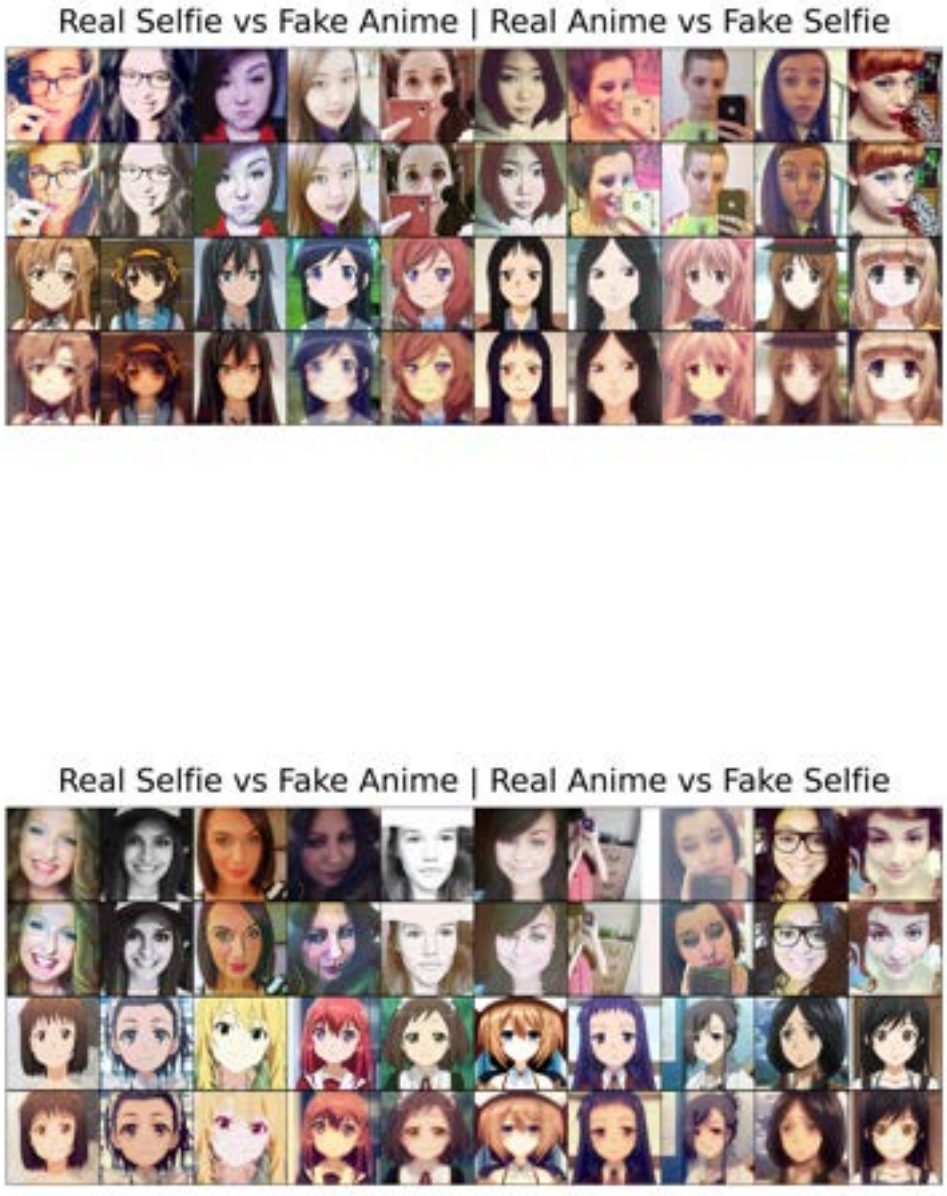


Figure 1: Real and generated images from testing set data for the best CycleGAN model.

Figure 2: Real and generated images from testing set data for the best CycleGAN model (part 2).

Figure 3: Real and generated images from testing set data for the best CycleGAN model (part 3).

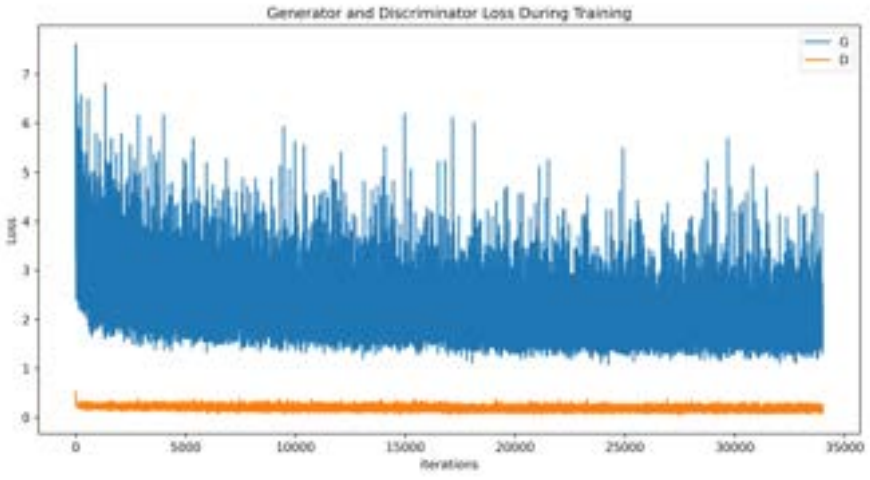


Figure 4: Generator and discriminator losses throughout training for the best CycleGAN model.

Figure 5: Real and generated images from testing set data for the best CycleGAN model throughout the training duration.

Figure 6 compares the image results for the original model and our best model with the best-performing modifications. Unlike our model, the original model did not use spectral normalization for the discriminator (instance normalization instead), leaky ReLU activations in the generator (regular ReLU instead), and Xavier weight initialization (convolutional layer weights were initialized to a Gaussian distribution with $\mu = 0$ and $\sigma = 0.02$ instead). It also used a smaller weight for the cycle consistency loss ($\lambda = 10$ in the original paper vs. $\lambda = 20$ in the our model). The unmodified model underperformed in comparison to our model, as can be seen by the lack of contrast in the right 3 generated anime images, the lack of clarity and artifacts on the faces of the generated anime images (especially the one on the right), and the worse quality of the generated selfie images (they had unrealistic facial expressions and dark contours on the face). The pattern in the losses for the two models are fairly similar (Figure 7).

Figure 6: Original CycleGAN results (left) and our modified CycleGAN results (right).

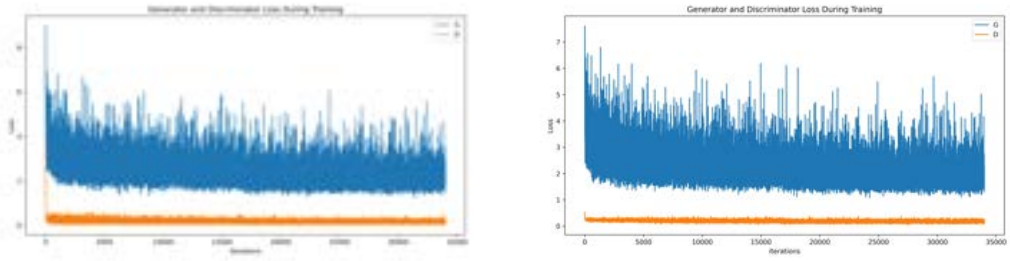
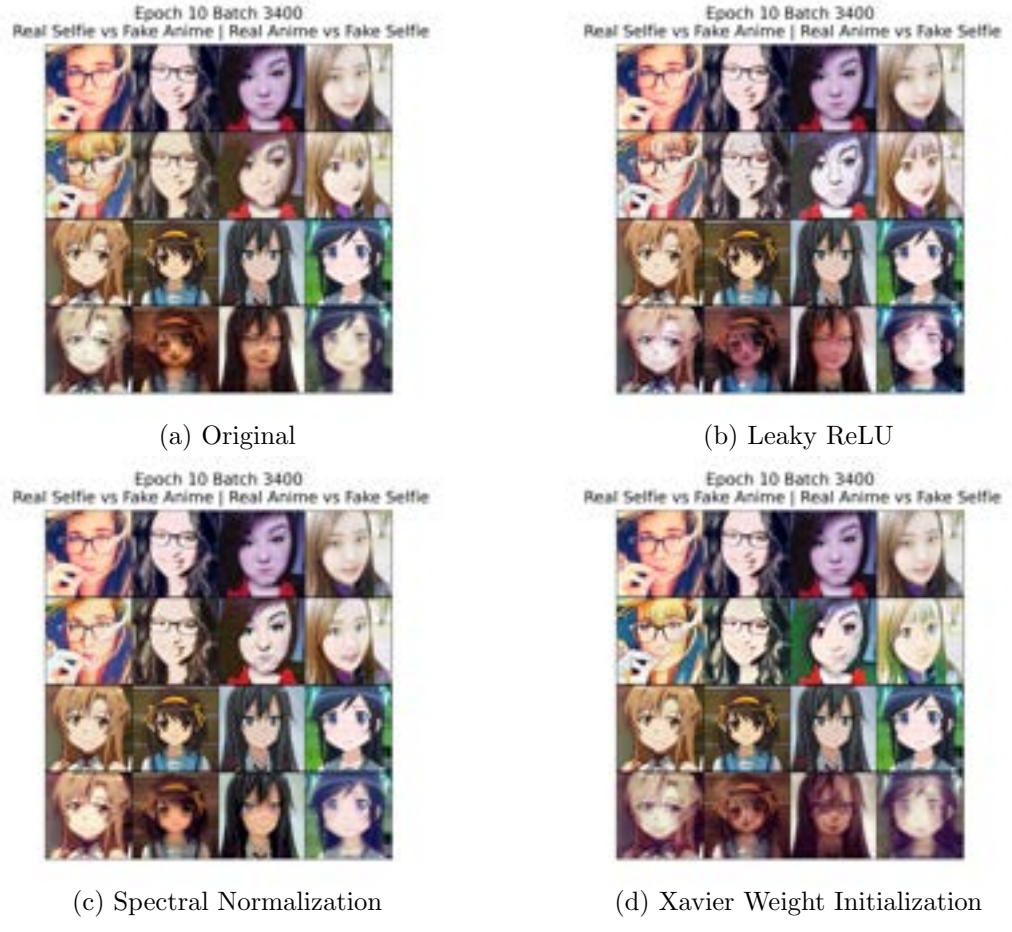


Figure 7: Original CycleGAN losses (left) and our modified CycleGAN losses (right) throughout training.

3.1 Model Modifications and Results

Figure 8 compares generated images for the original model, the original model with leaky ReLU activation layers implemented in the generator, the original model with spectral normalization implemented in the discriminator, and the original model with Xavier weight initialization for convolutional layers (instead of a Gaussian distribution). We see that the addition of the leaky ReLU improved the quality of the color contrast and contours in the generated anime images and removed the unnatural facial features in the generated selfie images. However, it also resulted in a degradation of the quality of the colors in the generated selfie images. On the other hand, the addition of the spectral normalization removed the facial artifacts in the generated anime images (especially the rightmost one) while also fixing the facial expressions in the generated selfie images and preserving the natural-looking color. The Xavier initialization also fixed the facial artifacts while also improving the color vibrancy and contrast in the generated anime images. However, it added a purple-ish hue to some of the generated selfie images. Since each modification caused resulted in generated images of improved quality, we chose to use all of them for our best model.



(a) Original

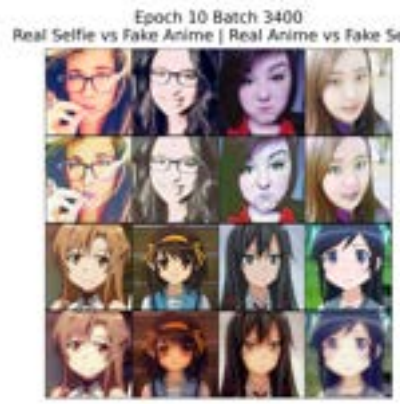
(b) Leaky ReLU

(c) Spectral Normalization

(d) Xavier Weight Initialization

Figure 8: CycleGAN results for the original model, model with leaky ReLU activations, model with spectral normalization, and model with Xavier weight initialization.

In Figure 9, we tried changing the weighting of the cycle loss for our best model. As compared to the cycle loss of $\lambda = 10$ that was used in the original CycleGAN paper, using a cycle loss of $\lambda = 20$ removed some excessive contouring from the anime images (rightmost anime image) and slightly improved the coloring of the anime and selfie images [1]. When we tried a smaller cycle loss of $\lambda = 5$, we obtained much worse coloring for the selfie images (the faces had a purple/pink hue), as well as a lack of contours in the anime images (such as the middle-right one).



(a) Best model with a cycle loss weight doul that of the original CycleGAN paper ($\lambda = 20$)

(b) Smaller cycle loss matching original CycleGAN cycle loss paper ($\lambda = 10$)

(c) Much smaller cycle loss weight half of the weight in the CycleGAN paper ($\lambda = 5$)

Figure 9: CycleGAN results for the best model with modifications to the weighting of the cycle loss in the total generator loss.

3.2 Loss Function Modification

In Figure 10, we tried changing the loss function for the discriminator from an L2 loss function to the hinge loss (using the `nn.HingeEmbeddingLoss()` function). For the generator loss criterion, we tried to equalize the scale of the GAN loss by scaling it down by a factor of 10, such that the generator loss becomes $\mathcal{L}_G = \frac{1}{10}\mathcal{L}_{GAN} + 20\mathcal{L}_C + 5\mathcal{L}_I$. The results using the hinge loss were much deteriorated in terms of quality, particularly in the lack of color saturation in the generated anime

images, the color artifacts in the generated selfie images, and the pixelated checkerboard pattern in all of the generated images. We can see from the plots of the loss functions that when the hinge loss is used, the discriminator loss continually decreases at a rate faster than the generator loss, so there is no adversarial training process, which explains the lack of learning in the model. We tried to fix this problem by changing the scaling factor for $\mathcal{L}_{GAN}$ in the generator loss criterion, but nothing was successful.

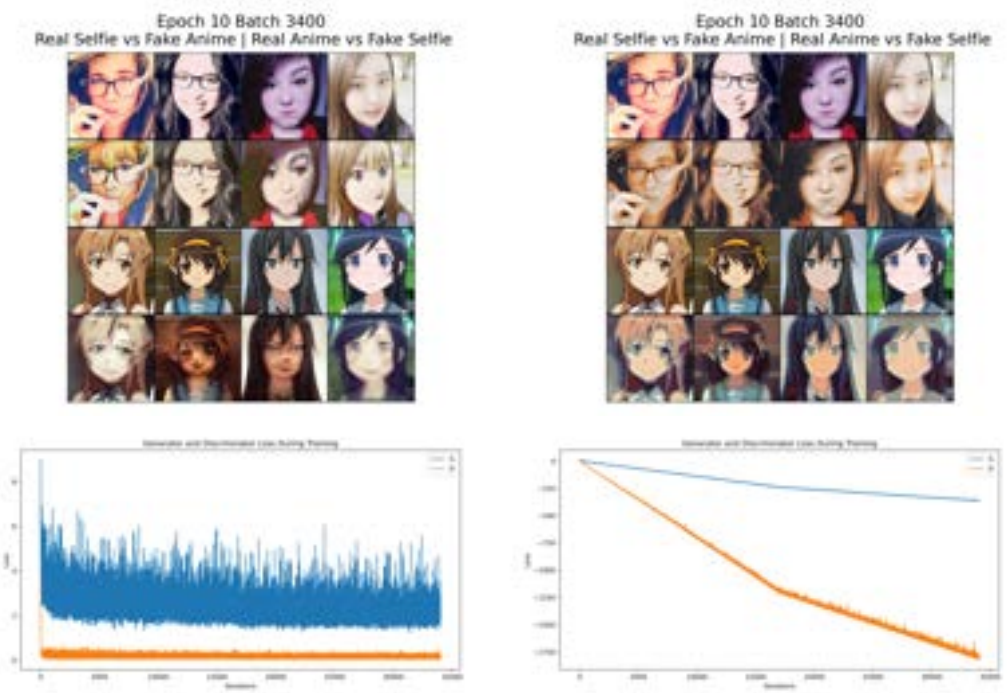


Figure 10: Original CycleGAN results (left) and results using a hinge loss function for the discriminator (right).

3.3 Model Architecture Modifications and Results

This section examines the results of the modifications to the model architectures described in Section 2.1. Each modification resulted in similar or decreased quality of the generated images, which is why we did not use any of these generated architectures in our final best model.

Figure 11 compares the image results for the original CycleGAN model and with a modified model that replaces the fractionally-strided convolutional layers in the generator with upsampling and convolutional layers. Even though upsampling followed by convolution should theoretically perform similarly to (or even better than) the fractionally-strided convolution, the results were much worse because the generated images had clear artifacts of extremely bright regions (looking like bleached circular regions in the image). Furthermore, the generated anime images had a dark gray shading in specific regions, and some generated selfies had image regions with an unnatural purple-ish hue.

Figure 12 compares the image results for the original CycleGAN model with a modified model

Figure 11: Original CycleGAN results (left) and results with upsampling instead of fractionally-strided convolution (right).

that changes the discriminator architecture to output a 1×1 binary classification. The results for the modified discriminator model exhibit a stark degradation in the generated image quality. This is a little unintuitive because the only change between the models is a discriminator with more layers such that it produces a 1×1 output instead of a 16×16 output. Even though we may intuitively expect that a larger model is more complex and thus has a larger capacity to learn from data, this may not be the case here. Alternatively, when compared to the patch-level classifications with the PatchGAN discriminator, a binary image-level classification from the discriminator may not be detailed enough to allow for effective training of the model.

Figure 12: Original CycleGAN results (left) and results with a modified discriminator that outputs a binary classification (right). Because of the stark drop in performance, the modified model was only run for less than an epoch instead of the full 10 epochs.

Figure 13 compares the original model to the model with the modified generator that used convolutional layers with even kernel sizes. The image outputs from the modified models are significantly worse because the generated images are much darker, have much less contrast, and are little changed in style from the image inputs. If one looks closely, the color channels (red, green, and blue) appear to clearly segregate among adjacent pixels and create a checkerboard pattern.

This indicates that there is likely an issue with the convolutions (either the 2D convolution in the encoder or the fractionally-strided convolution in the decoder) that is creating the color artifact and thus degrading the quality of the model output. The graphs of the losses are interesting because they show that the modified model had a discriminator loss of zero almost throughout the entire training period, which means that there was no adversarial learning process ocurring. This may be due to the fact that the color separation artifact make the generated images trivially distinguishable from the real ones.

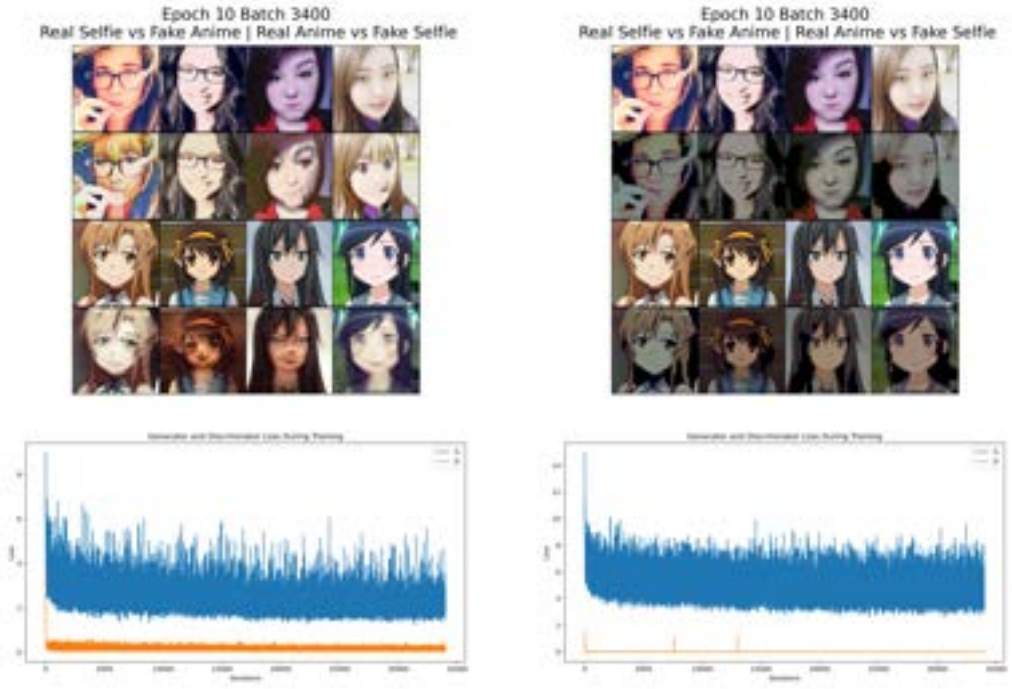


Figure 13: Original CycleGAN results (left) and results with a modified generator kernel size (right).

Figure 14 compares the original model to the model with the more complex generator architecture, in which an additional encoder and decoder layer were used in addition to a residual framework in which outputs from the encoder blocks were added to the same-sized inputs for the corresponding decoder blocks. While the generated anime images looked somewhat better due to the more vibrant colors and starker contrast, the generated selfie images looked worse due to a deep purple/pink hue for the middle two images in the row.

Due to the differing impacts of the new generator architecture on the generated anime images and selfies, we decided to investigate the effect of this alternative model architecture on our best model (which had the spectral normalization, leaky ReLU, Xavier weight initialization, and larger cycle loss). These results, illustrated in Figure 15, show that the generated images with the altered generator architecture were slightly worse in quality than the images with the base generator architecture for our best model. In particular, generated anime images had fewer distinct contours (see the right two images in the second row), and the two generated selfie images in the middle of the bottom row still had the unsual artifact of a pinkish/purple hue on the face. Because of this

Figure 14: Original CycleGAN results (left) and results with a more complex generator architecture (right).

deterioration in image quality, we decided not to include this modification in our final best model.

Figure 15: Our best CycleGAN results (left) and results with the more complex generator architecture (right).

Figure 16 shows results for the original model in comparison to a model with the generator architecture consisting of a larger number of residual blocks (11 blocks instead of 9). While we initially hypothesized that this deeper network may improve generated image quality, the results in the bottom row show a clear degradation in the quality of the generated selfie images due to an increasing amount of artifact, blurring, pixelation, and unrealistic colorization (the purple/red hue). The generated anime images in the second row show very slight improvement with the deeper generator architecture (e.g. darker contours in the left-most image), but the results are fairly comparable. We did not include this modification in our final model due to the lower image quality for generated selfie images.

Figure 16: Original CycleGAN results (left) and results with a generator architecture consisting of a larger number of residual blocks (right).

References

[1] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, August 2020. arXiv:1703.10593 [cs].

[2] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks, June 2014. arXiv:1406.2661 [cs, stat].

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016. ISSN: 1063-6919.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Networks, November 2018. arXiv:1611.07004 [cs].

[5] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks, February 2018. arXiv:1802.05957 [cs, stat].

[6] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Proceedings of the 30 th International Conference on Ma- chine Learning*, 28, 2013.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010. ISSN: 1938-7228.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. arXiv:1512.03385 [cs].