Multicyclic Loss for Multidomain Image-to-Image Translation

Except where reference is made to the work of others, the work described in this project is my own or was done in collaboration with my advisory committee. Further, the content of this project is truthful in regards to my own work and the portrayal of others' work. This project does not include proprietary or classified information.

Ethan H. Schneider

Certificate of Approval:

Michael J. Reale, Chair
Associate Professor
Department of Computer & Information Science

Amos Confer
Associate Professor
Department of Computer & Information Science

Bruno Andriamanalimanana
Associate Professor
Department of Computer & Information Science

Multicyclic Loss for Multidomain Image-to-Image Translation

Ethan H. Schneider

A Project

Submitted to the
Graduate Faculty of the
State University of New York Polytechnic Institute
in Partial Fulfillment of the
Requirements for the
Degree of

Master of Science

Utica, New York
May 9, 2021

Multicyclic Loss for Multidomain Image-to-Image Translation

Ethan H. Schneider

_____
Signature of Author

_____
Date of Graduation

Project Abstract

Multicyclic Loss for Multidomain Image-to-Image Translation

Ethan H. Schneider

Master of Science, May 9, 2021
(B.S., SUNY Polytechnic Institute, 2020)

39 Typed Pages

Directed by Michael J. Reale

GANs developed to Translate an Image's style between different domains often only care about the initial translation, and not the ability to further translate upon an image This can cause issues where, if one would want to generate upon an image and then further on, change that image even more that person may come into issues. This creates a "gap" between the base images and the generated images, and in this paper a Multicyclic Loss is presented, where the Neural Network also trains on further translations to images that were already translated.

Acknowledgments

I would like to acknowledge my full family, as they helped me immensely emotionally get through college, and they always helped me maintain my interest in the sciences. I would specifically like to thank my grandfather, Dr. Donald Gaylord Sr., who sadly passed away over the summer before this project. He did a lot to both spark a love of the sciences, and serve as a model of hard work to live up to.

I would also like to acknowledge the contributions of my project advisor, Dr. Micheal Reale, who both advised me for this project and has given me many opportunities to work hands-on with Neural Networks and GANs specifically. Without his help, I would not be the Computer Scientist I am today.

As a whole, I would also like to thank my Project Committee as well, which includes Dr. Micheal Reale, Dr. Bruno Andriamanalimanana, and Dr. William Confer. They all helped me make this paper into the best paper it could be.

Finally, I would like to thank more generally all the professors at SUNY Polytechnic, who all provided a place where I could ignite the spake I have for Computer Science and find the field that I have a passion in.

Style manual or journal used Journal of Approximation Theory (together with the style known as "sunypolyms"). Bibliography follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TEX (specifically LATEX2e) together with the style-file `sunypolyms.sty`.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Often when developing Generative Adversarial Networks, or GANs, for image-to-image translation, images are only run through the network for one "cycle," where the image is translated to the target domain once and losses are calculated. These losses often include reconstruction of the base image by translating it back to its original domain, and testing if the generated image is real. This is generally fine for translational GANs with only two domains, or in places where you only want to augment an image by translating it once. However, if you want to augment an image multiple times via this translation, this "reconstructive gap" grows, where further generations on augmented images seem less real and are less able to be reconstructed to its original domain.

However, the question arises, why would someone want to translate images in the first place, let alone translate images multiple times? As an example, say you have a set of stylized art pieces, with each style classified. You could use a translational GAN to translate between the styles represented by the pieces, and even translate a real image to a stylized image. This causes some issues, where you would have to keep the original image if you want to further change the style of that image. This novel multicyclic loss we introduce in this paper seeks to prevent that by making sure translated images are still translatable.

This loss works by taking the fake image generated during a generator's training step and generating upon that image additional times using the prior generated image as the new base to generate upon. This is to make sure that future images are indeed as translatable as the base set of images. This loss was implemented in two separate ways, a limited version which calculates only the ability to reconstruct the base image using this newly generated image, and one calculating the full training loss on the generated image. In either case, this loss is reduced by some factor to prevent it from dominating the primary loss during

training. This loss was then tested using an existing model to try and improve training results.

This approach is novel because, as stated before, it tries to reduce this "reconstructive gap" that can appear between real images and augmented images, which other GANs ignore.

Chapter 2

Related Work

2.1 CycleGAN

Zhu et al.[15] introduced CycleGAN in 2017, which is a structure for unpaired image to image domain translation. CycleGAN achieves this by training two Generator and Discriminator pairs, one pair generates and discriminates on one domain, and the other pair generates and discriminates on the second. In addition, a cycle-consistent loss was implemented that essentially makes sure an object returned to its original domain by a generator remains the same. The basic structure of cycleGAN is shown in Figure 2.1

2.2 StarGAN

CycleGAN works well for binary domain translation, however, you need to train exponentially more generators the more input domains are present. Choi et al.[2] introduced StarGAN in 2018 to combat these issues. Instead of each domain having a Generator and Discriminator pair, StarGAN uses one Generator and one Discriminator for all domains. It does this by training the Generator to accept a style label to generate an image translated

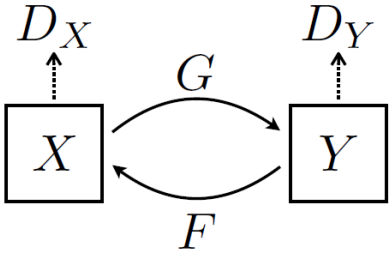

Figure 2.1: CycleGAN's basic structure, as taken from [15], showing a Generator G translatinf an image of set X to set Y, and a Generator F doing the inverse

to the domain defined in the style label, and training the Discriminator to predict the style label of a given image.

2.3 StarGAN v2

Choi et al.[3] introduced an iteration of StarGAN in 2020 where, instead of generating on a predefined style label, a separate network was developed to extract a style vector from an image. This allows for style extraction from unlabeled images, however, you lose the ability to change the image's style using a binary vector, you can only generate with a randomly generated style vector or a style vector extracted an image.

In all of these approaches, only the initially generated image is used when training. If you further generate on a generated image, you might get a drift between the image and the generated image. To combat this, my approach includes multicyclic loss which calculates loss values for further generated images.

Chapter 3

Method

3.1 Background

In this section, I clearly explain the background of modern Machine Learning, as it pertains to my project.

3.1.1 Neural Networks

Neural Networks or more specifically Artificial Neural Networks are one of the most important topics in modern approaches to AI and Modeling. They are made up of individual units called Artificial Neurons, each of which calculates an output for a series of given inputs.

Artificial Neurons and the Dense Layer

Artificial Neurons work by having a series of inputs with associated weights, which are multiplied by the inputs. Then, the neuron calculates the sum of the weighted inputs and gets the output by taking the result of that sum modified by some activation function. This activation function can be anything from a simple linear activation, which doesn't do anything, to a hyperbolic tangent, which essentially maps the output to a value between -1 and 1. Thus the full calculation for a single neuron output is:

$$y_n = act(\sum_i x_i * w_i) \tag{3.1}$$

Then, you can use multiple neurons to create an output vector. In this state, you then have what is called in modern parlance a "Dense" or "Fully Connected" layer in a neural network, as each input is fully connected to each output. To eventually get a

modern neural network, you stack multiple of these layers, connecting one's output to the next layer's input. One of the first implementations of a neural network was Rosenblatt's Perceptron [12], which was inspired by organic neurons.

Network Training

Now that we have the simple structure of a neural network, how do we get this to fit data? This is done via training. When you are training a neural network, you take a dataset that you want to train to and you run that data through the neural network to calculate an output. Then, you run that output through a loss or cost function, which represents the distance between your current output and your desired output. If this loss is calculated with the desired output you want the model to compute, the training is considered supervised. Otherwise, if the output loss takes no desired output into account, it is considered unsupervised.

The model then attempts to update its weights in a way to minimize this loss. This is done by using a mathematical optimizer to first calculate the gradients of the loss and update the model's weights in a fashion to minimize these gradients. This process doesn't fully fit the network to the dataset on its first full run-through of the dataset, so iterations of these training steps, epochs are needed to fully minimize the loss values. In addition, often computers don't have the full processing power needed to calculate the loss and update the gradients for a full dataset at once. In this case, we separate the data into "batches," or smaller subsets of the full dataset. Then, for each batch, the loss is calculated, optimized, and the model's weights are updated before moving on to the next batch.

3.1.2 Convolutional Neural Networks

Convolutional Neural Networks or CNNs are iterations upon traditional Neural Networks where instead of training a series of neurons, you train a series of convolutional filters that are run upon input images to extract features. These filters make up what is called a Convolutional layer. In addition to this, there is a concept of a Transposed Convolutional

Layer, which is trained to use a filter to perform an inverse of convolution on an input. The main difference between the two is that where a convolution decreases the size of a given input, a transposed convolution will increase the input's size. LeCun et al. introduced the Convolutional Neural Network in its modern form [9], using backpropagation along the loss's gradients to train the network to identify written numbers.

3.1.3 Generative Adversarial Networks

Generative Adversarial Networks, or GANs, are a comparatively modern development in Machine Learning. They are often trained to generate images based on random noise, but other iterations exist which instead train a network to augment an image in some form. To achieve this instead of training one network, two are trained instead: a Generator, which generates an image, and a Discriminator, which determines if an image is real or fake. Introduced by Goodfellow et al. in 2014 [4], these networks are trained in two separate steps, the first trains the Discriminator to detect if the Output is Real or Fake, traditionally classifying a 1 as real and a 0 as fake. Next, the second step trains the Generator to output images that look real to the discriminator. This essentially creates a condition where, as the discriminator tries to get better at detecting if an image is real, the generator gets better at creating fake images. This is often likened to a forger and an art critic collectively getting better over time, where the forger gets better at creating fake pieces and the critic gets better at realizing a piece is fake.

An iteration on this was introduced by Arjovsky et al. in 2017 [1], where instead of training the discriminator to output a real or fake label, you train the discriminator to output a Wasserstein metric or the "earth mover distance." This is achieved by training the generator to maximize the score of fake images while training the discriminator to maximize the score of the real images minus the fake images. However, to enforce the Wasserstein Metric, you need to clip the discriminator's weights to between a certain range and only train the generator a fraction of the time you train the discriminator. A further iteration upon this was introduced in 2017 by Gulrajani et al. [5], where instead of using weight

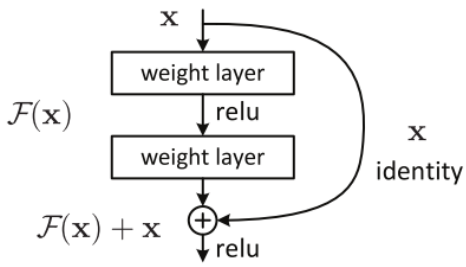

Figure 3.1: Example of A Residual Block [6]

clipping to enforce the Wasserstein distance, a gradient penalty is introduced. This penalty is the Mean Squared Error between the L2-norm of the gradients and 1, as given below:

$$\mathcal{L}_{gp} = \lambda_{gp}\mathbb{E}_{\hat{x}}[(\| \bigtriangledown_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{3.2}$$

3.1.4 ResNet

ResNet, or a Residual Network, is a neural network initially introduced by He et al. [6], and based on the idea of an "identity shortcut connection" where the initial value x is added back to a result $f(x)$ after a certain number of layers. This concept culminates in the form of a residual block, as shown in 3.1, which represents a single block in a residual network

3.2 Architecture/Structure

The structure of the model borrows much from the structure of StarGAN [2] this was chosen as it is a network structure that both implements multi-domain generation and supports target labels, while still being a well-defined model.

Although much of the structure of the model is implemented from StarGAN, starGAN was officially implemented in PyTorch, and I chose to implement my project in TensorFlow [11], and as such, I re-implemented everything for the model using the base TensorFlow and Keras libraries and API.

3.2.1 Loss

As stated before, the model needs a loss to optimize to train the model to the desired output. For our model, this loss takes several forms, which are all defined in the following sections below.

Adversarial Loss

An adversarial loss was adopted to generate images indistinguishable from real images by the discriminator, the loss that was adopted was

$$\mathcal{L}_{adv}^{D} = \mathbb{E}_x[D_{src}(x)] - \mathbb{E}_{x,c}[D_{src}(G(x,c)] + \lambda_{gp}\mathbb{E}_{\hat{x}}[(\| \bigtriangledown_{\hat{x}} D(\hat{x})\|_2 - 1)^2] \tag{3.3}$$

$$\mathcal{L}_{adv}^{G} = \mathbb{E}_{x,c}[D_{src}(G(x,c))] \tag{3.4}$$

Where G generates an image, $G(x,c)$ on the input image x and a target domain label c, while the discriminator D tries to detect which is which by maximizing the output of a real image and minimizing the output of a fake image. The generator G however tries to maximize the output of a fake image.

Domain Classification Loss

For any generated image x generated using a target domain c, we want to make sure that x is translated to c. This is done by training the discriminator D to not only detect if images are real, but also classify the target domain label. To impose this, a loss is added to both the discriminator and the generator. The discriminator loss is as follows:

$$\mathcal{L}_{cls}^{r} = \mathbb{E}_{x,c'}[-\log D_{cls}(c'|x)] \tag{3.5}$$

Where $D_{cls}(c'|x)$ represents a probability distribution over domain labels classified by D, where x is a member of the domain class for each class in c'. By minimizing $\mathcal{L}_{cls}^{r}$, the

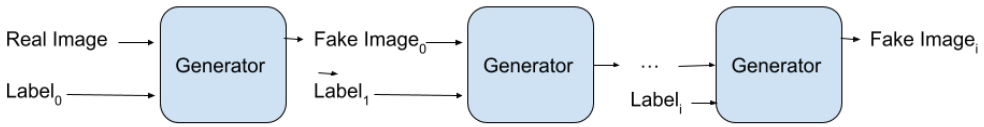

Figure 3.2: Multicylic Generator calls

discriminator learns to classify an image, x, to it's original domain, c'. On the other hand, the generator reduces the following loss:

$$\mathcal{L}_{cls}^{f} = \mathbb{E}_{x,c}[-\log D_{cls}(c|G(x,c))] \tag{3.6}$$

By reducing this, the generator tries to generate images in the class of c, where $D_{cls}(c|G(x,c))$ represents a probability distribution that a generated image, $G(x,c)$, is a member of the domain class for each class in c.

Reconstructive Loss

This loss is an important loss, as we want the image generated by the generator G to preserve the content of an input image, x. To guarantee this, we adopt a cycle consistent loss, as in CycleGAN [15].

$$\mathcal{L}_{rec} = \mathbb{E}_{x,c,c'}[\|x - G(G(x,c),c')\|_1] \tag{3.7}$$

This loss, as defined above, takes a translated image, $G(x,c)$ and translates it back to the original domain, c' through a second generator call. To achieve this, we adopt the L1 Norm of the difference between the original image and the cycled image as the loss.

Multicyclic Loss

This loss is to maintain consistency across generated images by making sure that those images can be generated again to produce images that are real, of the correct class, and reconstructable. We first begin by representing G_i as a series of generator calls on an input

image, x with different target labels across a set of target labels C, as follows. These multicyclic Generator calls are shown in Figure 3.2.

$$G_i(x, C) = G(G_{i-1}(x, C), c_i) \tag{3.8}$$

$$G_0(x, C) = G(x, c_0) \tag{3.9}$$

This set of iterative generator calls is important as it allows us to refer to the result of i iterative generator calls across a set of target labels on an original image, x. Then, we define several multicyclic iterations of previously defined losses, using these iterative generator calls.

$$\mathcal{L}_{multi\_adv}(i) = \mathbb{E}_{x,C}[D_{src}(G_i(x, C))] \tag{3.10}$$

$$\mathcal{L}_{multi\_cls}(i) = \mathbb{E}_{x,C}[-\log D_{cls}(c_i|G_i(x, C))] \tag{3.11}$$

$$\mathcal{L}_{multi\_rec}(i) = \mathbb{E}_{x,C,c'}[\|x - G(G_i(x, C), c')\|_1] \tag{3.12}$$

These losses mimic other predefined losses but instead using the iterative generator calls. Then, we can combine all these together to get the full multicyclic loss

$$\mathcal{L}_{multi}(i) = -\mathcal{L}_{multi\_adv}(i) + \lambda_{cls}\mathcal{L}_{multi\_cls}(i) + \lambda_{rec}\mathcal{L}_{multi\_rec}(i) \tag{3.13}$$

As you may notice, this loss is almost identical to the future full objective loss 3.15, this is because by reducing this loss, the generator is attempting to make sure further images are generated on G after an initial translation are still generated properly.

Full Objective Loss

And thus, the full loss to optimize G and D are written, respectively, as

$$\mathcal{L}_D = -\mathcal{L}^D_{adv} + \lambda cls\mathcal{L}^r_{cls} \tag{3.14}$$

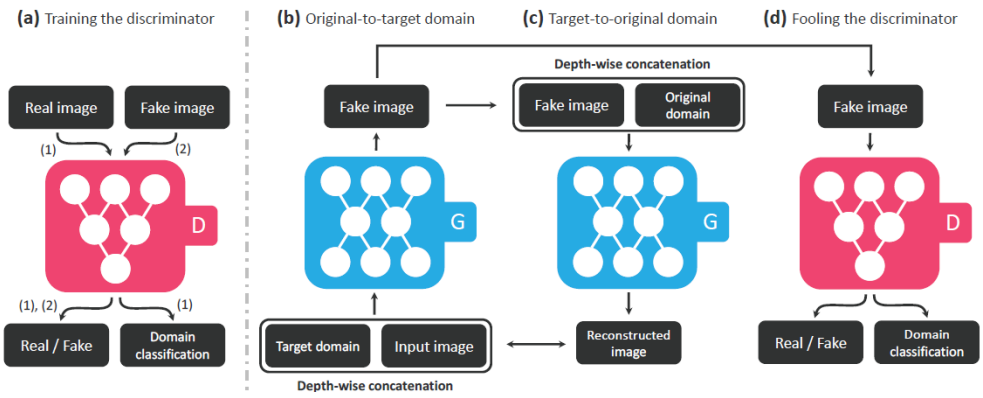

Figure 3.3: StarGAN's Architecture and training method, as taken from [2], showing a) discriminator training, b) target domain training, c) reconstructing to the original domain, and d) discriminator fooling

$$\mathcal{L}_G = -\mathcal{L}_{adv}^G + \lambda_{cls}\mathcal{L}_{cls}^f + \lambda_{rec}\mathcal{L}_{rec} + \sum_{i=1}(\lambda_{multi}[i]\mathcal{L}_{multi}(i)) \tag{3.15}$$

with the λs representing hyperparameters that control the relative importance of the losses compared to the adversarial loss. For all of the experiments, $\lambda_{cls} = 1$, $\lambda_{rec} = 10$, $\lambda_{gp} = 10$, and $\lambda_{multi} = [0.0, 0.25, 0.125, 0.0625]$, however only the last three are used as the first item in λ_{multi} is skipped during training, as $\lambda_{multi}[i]$ essentially represents images that have been cycled through iterative generator calls 0 times, or the base generated images.

3.2.2 Network Architecture

For the network architecture, we take the architecture from StarGAN [2], which is itself adapted from CycleGAN [15]. For the generator network, you begin by first concatenating the label, c onto the input image, x depth-wise. Then, there is a convolutional layer to extract a feature vector, two convolutional layers used for downsampling, six residual blocks [6], two deconvolutional layers for upsampling the result, and a final convolutional layer to extract the color channels. This generator also utilizes instance normalization [13] to normalize input images to the style of the trained dataset. This output will have the same shape as the input image, x

Then, for the discriminator, we use 6 convolutional layers to continually downsample the input image, this leads to two separate output layers, one which downsamples the image a final time to get D_{src}, and one which downsamples the image even more to output D_{cls}, the class probabilities. These class probabilities are then flattened to get the output probabilities. The output shape of the D_{src} is $[h/64, w/64, 1]$ and the output shape of D_{cls} is $[1, 1, |c|]$ before flattening, or the same as c after flattening.

To get a closer look at the model, you can read the project's gitHub[1], or observe the full model architecture as implemented in TensorFlow in Appendix C. The full network training and architecture are also shown in Figure 3.3.

[1] `https://github.com/plaidScience/SchneiderMastersProject2021`

Chapter 4

Experiments and Results

In total, three experiments were run, a baseline experiment without the multicyclic loss, an experiment with the multicyclic loss, and the third experiment with a limited version of the multicyclic loss, where $\mathcal{L}_{multi}(i) = \lambda_{rec}\mathcal{L}_{multi_rec}(i)$ this was done to test if it is sufficient to simply test if the multicyclic loss sees improvement if the loss is simply reconstructive instead of being a full objective loss.

4.1 Data

For all of the experiments, the networks were trained on celebA [10], a dataset of celebrity faces annotated with 40 binary attributes. To process this data from its original form, we first cropped the data from its original shape of 178×218 to 178×178, and then resized those images to 128×128. In addition, from the binary attributes, we selected five, three which control hair color (black, blond, brown), one which controls gender, and one which controls age. In addition, the data were randomly flipped with a 50% probability during training. The testing and validation sets were selected using the dataset's splits.

4.2 Parameters

The networks were trained using the Adam optimizer with $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and a learning rate of 0.0001 for the first half of the epochs, after which the learning rate is linearly decayed to 0 over the next half. For all the experiments, the networks were trained for a total of 20 epochs. A batch size of 16 was used for the baseline experiment, but for the multicyclic experiments, that batch size was reduced to 8 to compensate for the additional load the additional generator calls put on the GPU's memory. This training took about 1

day on an NVIDIA Titan X Pascal for the Base StarGAN, and 2 days for both Multicyclic StarGANs.

4.3 Results

For my results, I analyzed both qualitative results and quantitative results. Quantitative results are results that include some form of measurement, such as accuracy or distance. Qualitative results, however, require some form of external observation. To explore these results, they have been separated into two separate sections below.

4.3.1 Quantitative Results

In each of the experiments, the quantitative results have been split into three separate measures. Each of these measures was calculated on a reconstructed image (rec), or the result of generating upon an image with its initial label, and an inverse image (inv), or the result of generating an image with the inverse of its initial label, as seen in the Table of Quantitative Results 4.1. For the inverse label, the age and gender domains were swapped, and a random hair color domain was chosen from all the hair colors. As well as this, all of the metrics were tested upon several cycles. This number represents the number of times the image was translated to a random domain from the set of domains in the batch before translating to the inverse or reconstructed image.

The first metric that was used is what is called an FID, or a Frechet Inception Distance, which was first introduced by Heusal et al. in 2018 [7]. This metric represents a distance between two sets of images, a set of real images and a set of generated images. As such, a lower score is preferable in our case, as all the images should appear to be part of the same set of faces.

The second metric is the Learned Perceptual Image Patch Similarity Score, or LPIPS score, which was first introduced by Zhang et al. in 2018 [14]. This metric attempts to calculate the distance between patches from two images. A lower score is generally better in this case as well, as we want the generated images to appear similar to their base images.

Model	# of cycles	FID		LPIPS Scores		Label Accuracy	
		inv	rec	inv	rec	inv	rec
Base StarGAN	0	19.36	10.52	0.250	0.213	0.77	0.95
	1	16.39	15.12	0.229	0.221	0.51	0.86
	2	22.84	20.20	0.289	0.275	0.59	0.85
	3	25.19	23.90	0.302	0.295	0.52	0.84
StarGAN with Multicyclical Loss	0	16.12	8.36	0.229	0.187	0.74	0.93
	1	19.09	12.18	0.254	0.221	0.71	0.92
	2	23.42	17.26	0.291	0.259	0.72	0.92
	3	27.06	20.46	0.291	0.259	0.72	0.91
StarGAN with Limited Multicyclical Loss	0	13.12	6.64	0.198	0.159	0.83	0.96
	1	11.12	10.61	0.169	0.164	0.43	0.89
	2	14.40	13.84	0.201	0.196	0.43	0.89
	3	17.24	16.62	0.229	0.224	0.43	0.89

Table 4.1: Table of Quantitative Metrics, for the FID and LPIPS scores, lower is better, for the Label Accuracy, Higher is better

The third metric used is the Label Accuracy, which represents the accuracy between the predicted label of the generated image using our Discriminator and the label used to generate that image. In this case, a higher score is better, as more of the input labels are accurate to the output.

As you can see in the table of quantitative metrics 4.1, Base StarGAN does worse for the FID and LPIPS Scores generally across the board, and in those same metrics, StarGAN with the limited multicyclic loss does better than StarGAN with the full multicyclic loss. However, the full multicyclic loss does better at correctly identifying images in further cycles than the other models. This seems to point towards the idea that the multicyclic loss will generate images that are more correct to the label in further image cycles than the Limited loss or the Base StarGAN. This improvement doesn't translate to the first cycle, however, images from the first cycle seem to be more part of the base image set than the Base StarGAN.

Figure 4.1: Two sets of Qualitative Results

4.3.2 Qualitative Results

For the Qualitative results, we took images from the test set and generated upon them. To do this, we took each label domain and set it to a value. For the domain of hair colors, it was set so that it would generate blond hair individuals. For the gender and age domains, they were set to the opposite values than that of what they were in the original. Again, like the quantitative results, these images were also cycled through several times to show the multicyclic loss in action, with a final generator call to the desired domain done at the end to correctly translate the image. As you can see from the results from the experiments, as shown in Figure 4.1 show that the multicyclic loss works, preserving facial attributes, such as blond hair, aged faces, and swapped gender, across multiple generational cycles on the data itself. There is an issue the full multicyclic loss presents, however In the full multicyclic loss on the second image, the non-facial features in the image do degrade over time, this may be due to multiple reasons, such as the need for stronger importance placed on the image reconstruction when using multicyclic loss, as an example. Additional Qualitative Results can be found in Appendix D.

Chapter 5

Conclusion

Implemented in this paper is a multicyclic loss, which looks at generating upon an already generated image an additional time to make sure that that image satisfies any requirement that you would want the original generated image to satisfy. This loss was implemented on top of a preexisting network structure named StarGAN [2], which is a network aimed at multidomain image to image translation.

This loss was created by redefining StarGAN's existing losses to work with multiple generator calls on a series of labels with a given input image. It was then tested with the CelebA dataset [10], a dataset of labeled pictures of celebrity faces. The results of these tests show that although this loss works, it does come at the cost of computational time. Although I feel this loss is useful and I have shown its benefits, this time cost is something to take into account. If you train a model that would usually take a week, for example, and you copy this loss directly as-is you would double that training time to two weeks. And, if you wanted to add more cycles onto the loss, you would see worse and worse computational times. It is also important to understand that while this does show improvements, you still see some image degradation over multiple cycles, so it is not a perfect solution. It does, however, show marked improvement over the base model.

Chapter 6

Future Work

One of the things I want to experiment upon more with is altering some of the hyperparameters to see if I can better compensate for the multicyclic loss compared to the base loss. This was tested a bit in initial testing with these hyperparameters, however, I feel that I could be more thorough with it in the future, especially with more time and more computational hours available to me.

Another point that I feel I failed to hit precisely is exploring this loss using different multidomain models, such as StarGAN V2 [3]. While I feel that the loss would work with other models, you can never really tell without testing.

I would also be interested in implementing this for different datasets, such as the Radboud Faces Database, as presented by Langner et al. [8], which is a dataset that was also used in the original StarGAN. This is a dataset of face images for each of the following emotions: anger, disgust, fear, happiness, sadness, surprise, contempt, and a neutral expression. I would also like to try StarGAN's method for joint training across multiple datasets with this loss. It would be especially interesting to implement this loss for cross-domain translation with both datasets so that you can freely generate with both data labels.

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv:1701.07875 [cs, stat]*, December 2017. arXiv: 1701.07875.

[2] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. *arXiv:1711.09020 [cs]*, September 2018. arXiv: 1711.09020.

[3] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. StarGAN v2: Diverse Image Synthesis for Multiple Domains. *arXiv:1912.01865 [cs]*, April 2020. arXiv: 1912.01865.

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

[5] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. *arXiv:1704.00028 [cs, stat]*, December 2017. arXiv: 1704.00028.

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

[7] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *arXiv:1706.08500 [cs, stat]*, January 2018. arXiv: 1706.08500.

[8] Oliver Langner, Ron Dotsch, Gijsbert Bijlstra, Daniel H. J. Wigboldus, Skyler T. Hawk, and Ad van Knippenberg. Presentation and validation of the Radboud Faces Database. *Cognition & Emotion*, 24(8):1377–1388, December 2010.

[9] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. Publisher: MIT Press.

[10] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat,

Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015.

[12] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[13] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv:1607.08022 [cs]*, November 2017. arXiv: 1607.08022.

[14] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *arXiv:1801.03924 [cs]*, April 2018. arXiv: 1801.03924.

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

Appendices

Appendix A
Notes from the Original Style Files

These style-files for use with LaTeX are maintained by Darrel Hankerson[1] and Ed Slaminka[2].

In 1990, department heads and other representatives met with Dean Doorenbos and Judy Bush-Crofton (then responsible for manuscript approval). This meeting was prompted by a memorandum[3] from members of the mathematics departments concerning the *Thesis and Dissertation Guide* and the approval process. There was wide agreement among the participants (including Dean Doorenbos) to support the basic recommendations outlined in the memorandum. The revised *Guide* reflected some (but not all) of the agreements of the meeting.

Ms Bush-Crofton was supportive of the plan to obtain "official approval" of these style files.[4] Unfortunately, Ms Bush-Crofton left the Graduate School before the process was completed. In 1994, we were revisiting some of the same problems which were resolved at the 1990 meeting.

In Summer 1994, I sent several memoranda to Ms Ilga Trend of the Graduate School, reminding her of the agreements made at the 1990 meeting. Professors A. Scottedward Hodel and Stan Reeves provided additional support. In short, it is essential that the Graduate School honor its commitments of the 1990 meeting. It should be emphasized that Dean Doorenbos is to thank for the success of that meeting.

Maintaining these LaTeX files has been more work than expected, in part due to continuing changes in requirement by the graduate school. The Graduate School occasionally has complete memory loss about the agreements of the 1990 meeting. If the Graduate School rejects your manuscript based on items controlled by the style-files, ask your advisor to contact the Graduate school (and copy to the chair) to urge cooperation.

Finally, there have been several requests for additions to the package (mostly formatting changes for figures, etc.). While such changes are not really part of the thesis-style package, it could be beneficial to collect these options and distribute with the package (making it easier on the next student). I'm especially interested in changes needed by various departments.

---

[1] Mathematics and Statistics, 221 Parker Hall, 844-3641, `hankedr@auburn.edu`

[2] Mathematics and Statistics, 218 Parker, `slamiee@auburn.edu`

[3] Originally, the memorandum was presented to Professor Larry Wit. A copy is available on request.

[4] Followup memoranda gave a definition of "official approval." Copies will be sent on request.

Appendix B
How to Run Code

The entirety of the project's code can be found on the Project's GitHub[1], with the final submitted project code found under the FinalProject branch. In order to run the code, both a python run environment is needed, alongside several python packages that need to be installed. These packages are NumPy, Matplotlib, Pandas, and TensorFlow. The TensorFlow package also needs to be version 2.5.X, as it uses several features that were experimental but were made into base features, and as such have changed location import-wise.

To simply run the code, use the main.py script, which will ask for ids of scripts to run. To run base StarGAN, use id 3, for Limited Multicylic StarGAN, use 4, and for Full Multicyclic StarGAN use id 5. For evaluation, just simply run the evaluate.py script and fill out the prompts as needed.

In order to evaluate an LPIPS score, an external package needs to be downloaded here[2]. This package's code needs to be placed in an external folder in the utils folder of this model, and renamed to lpips_tf2 to be imported correctly. However, evaluation is possible without this import, you just can't calculate the LPIPS score.

[1] `https://github.com/plaidScience/SchneiderMastersProject2021`
[2] https://github.com/moono/lpips-tf2.x

Appendix C
Model Architecture

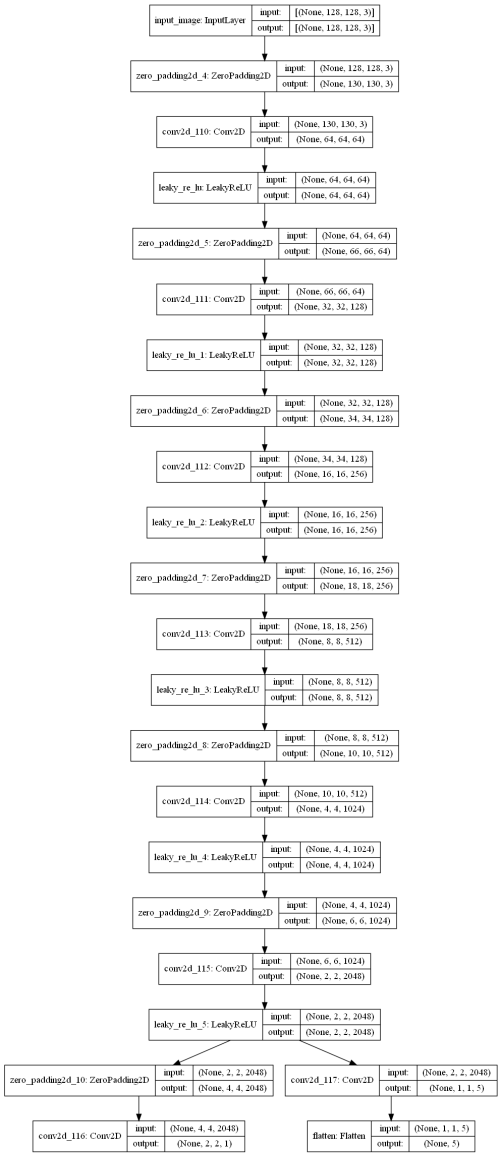

Figure C.1: Discriminator Model Structure

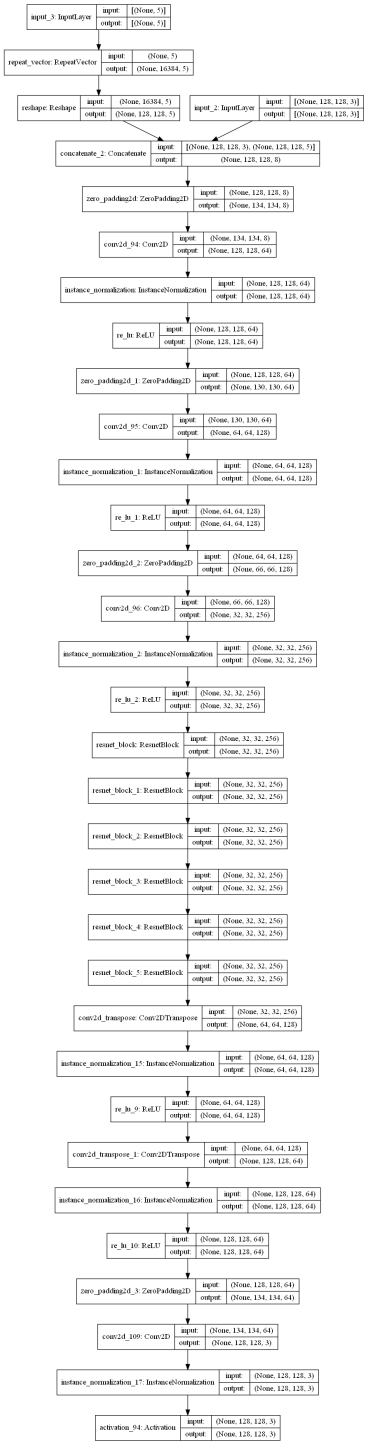

Figure C.2: Generator Model Structure

Appendix D
Additional Qualitative Results

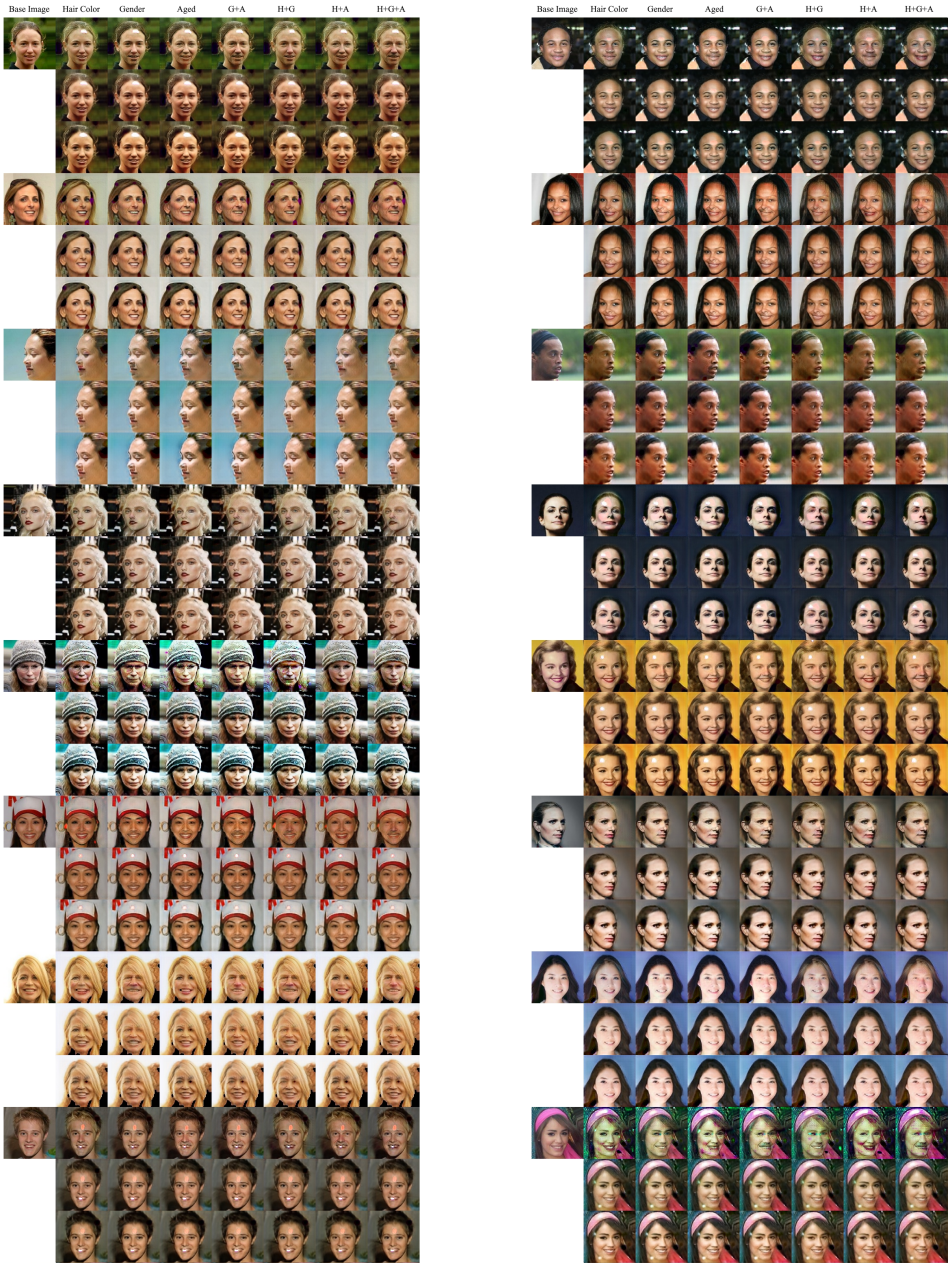

Figure D.1: Base StarGAN Results

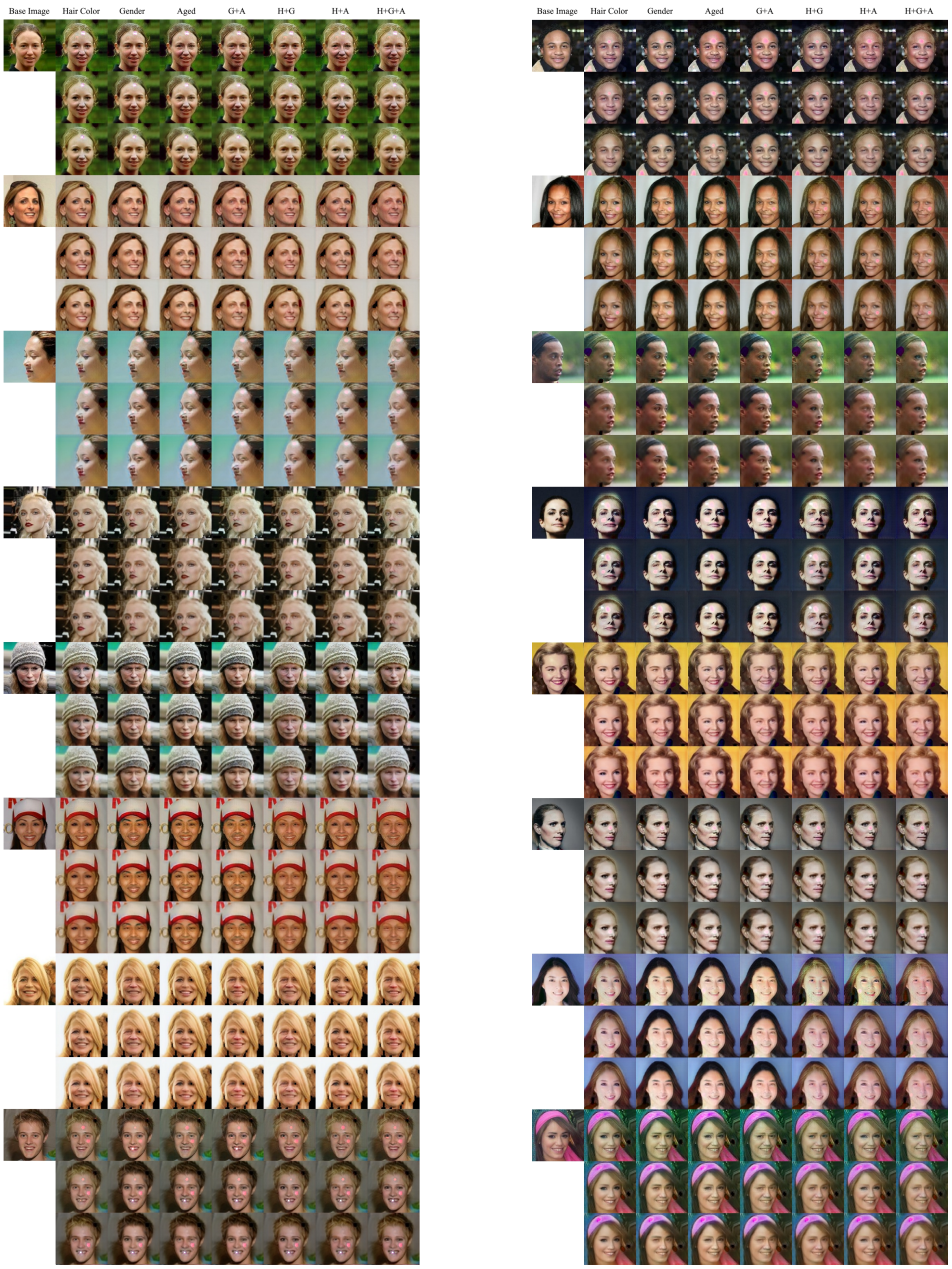

Figure D.2: Multicyclic Loss StarGAN Results

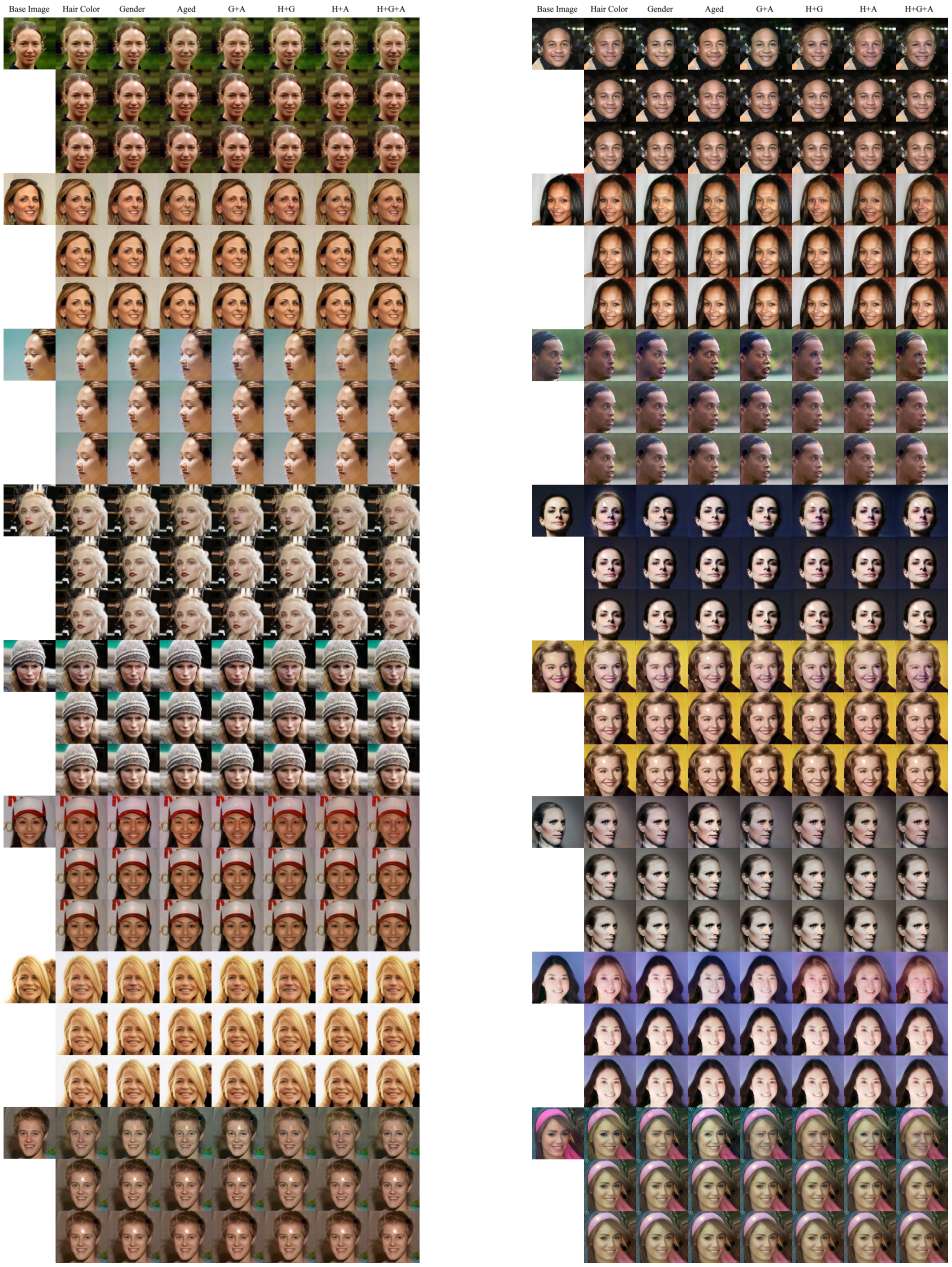

Figure D.3: Limited Multicyclic Loss StarGAN Results