Multicyclic Loss for Multidomain Image-to-Image Translation

Ethan H. Schneider

Introduction

Introduction

- When dealing with Machine Learning Models for Image-to-Image Domain translation, not a lot of care is taken into determining if a generated image can be further translated by the generator.
- Additionally: some networks adopt a “cyclic” or “reconstructive” loss, that checks to make sure the image can be reconstructed from it’s translation.
- Theoretically, if you were to translate upon an image that’s already translated, you may create a “reconstructive gap,” or an error in reconstruction of the base image

Introduction

- To combat this, I present a Multicyclic Loss, where you translate prior generated images to improve both reconstruction of those images and to improve said generation upon those images.
- This loss was implemented using an existing model structure, StarGAN [2], to prove that it can improve said structure using this loss.
- As stated before, other models ignore this "reconstructive gap" and instead attempt to reduce the loss on just the real image.

Related Work

CycleGAN

CycleGAN, introduced by Zhu et al. [15], is a series of model used for domain-to-domain translations. It trains a pair of models for each domain translation you want to perform, a Generator and Discriminator. This is fine for systems where you have only two domains you want to translate, however if you have more than two domains, the number of pairs you have to translate essentially exponentially grows.

StarGAN

Introduced by Choi et al. [2], StarGAN seeks to solve the problem of of needing more and more Generators and Discriminators that occurs in CycleGAN. It does this by training a Generator to accept a style label to determine how to translate an image. In order to generate images correctly, it also trains the discriminator to detect the style label of the image

StarGAN v2

A further iteration of StarGAN, StarGAN v2, introduced by Choi et al. in 2020 [3], ditches style labels for style codes. This means instead of training to preexisting style labels determining things like hair color, gender, age, or what have you, they instead train these codes in an unsupervised way, which allows them to use unlabeled images. However, you lose the ability to change an image's style based purely on it's own, and instead you have to generate a style for an image based on another image.

Method

Background

Neural Networks

- Neural Networks are the basis for much of modern AI and Modeling.
- They work by having artificial neurons, which output a value given a series of inputs and a series of learned weights.
- If you have multiple of these neurons, you can construct what is considered a "Dense" or "Fully Connected" network layer, and you can construct a full network using one or more of these layers, as shown by Rosenblatt's Perceptron [12], a classic example of a traditional neural network.

Neural Networks -- Training

- In order to train a neural network, you need a series of original data points to “fit” it to.
- To fit it, you first calculate the loss or cost of the network, which represents some distance between the current output of the network and the expected output.
- You then calculate the the gradients of the loss using a mathematical optimizer, and then use these gradients to update the network’s weights in a way that reduces the gradients.

Convolutional Neural Networks

- Convolutional Neural Networks are an extension of traditional neural networks, where instead of training a series of neurons, you train a convolutional filter to apply to an input image (or image-like object)
- It comes in two forms, Convolutional Layers, which often are used for Downscaling Images, and Deconvolutional Layers, which are used for upscaling images.
- A traditional example of a Convolutional Neural Network is introduced by LeCun et al. [9], which is used to extract handwritten characters.

Generative Adversarial Networks

- Generative Adversarial Networks are networks that are trained traditionally to generate fake data from a latent space, however as shown in CycleGAN [15] and StarGAN[2], you can use them to translate images between two or more domains.
- Introduced by Goodfellow et al. in 2014 [4], you start by training two networks, a Generator which generates (or translates) an image, and a discriminator, which detects if an image is real.
- An improvement upon this called a WGAN, or Wasserstein GAN was introduced by Arjovsky et al. [1], where you train the discriminator to output a Wasserstein metric instead of a real or fake label.
- Another improvement as introduced by Gulrajani et al. [5], where instead of using weight clipping to enforce the wasserstein metric, you penalise the loss using a gradient penalty.

ResNet

- ResNet, introduced by He et al [6], is a modern Neural Network Layer where you use residual blocks.
- Co construct these residual blocks, you take an initial value and add it back to the result after a series of layers.
- This essentially makes those layers represent a transformation applied to x.

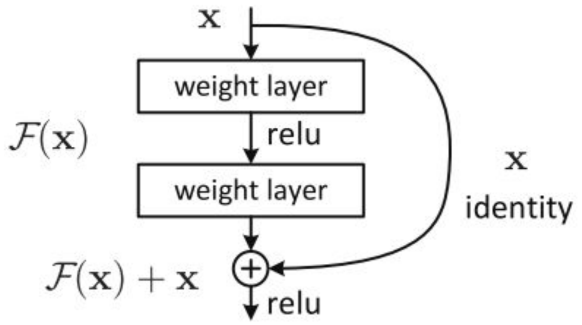


Figure: An Example of a Residual Block from [6]

Model Architecture

Losses Defined

In order to implement optimize the model, as stated before, losses are needed. For this model structure, several losses were defined, including:

- An Adversarial Loss to determine the “realness” of an input image,
- A Domain Classification Loss to train the Discriminator and Generator,
- A Reconstructive Loss, to preserve original Image,
- and A Multicyclic Loss, to make sure generated images can be further generated upon.

Losses - Adversarial Loss

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

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

- A WGAN-Style Loss is Implemented
- Discriminator by maximizing trains itself to detect images better
- Generator by maximizing trains itself to Generate images that look more fake.

Losses - Domain Classification Loss

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

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

- You train Discriminator to correctly predict a image’s style label
- You train the Generator to correctly generate images of a correct label.

Losses - Reconstructive Loss

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

- This loss trains the generator to preserve the content of the input image.
- It does this by generating on that image twice, once to an intermediary domain, and once to its original domain.
- Then, it tries to reduce the difference between the original image and the reconstructed image.

Losses - Multicyclic Loss

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

$$G_0(x, C) = G(x, c_0)$$

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

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

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

- To implement the multicyclic calls, we define a recursive series of Generator calls across a number of input labels
- Then, we reimplement the prior losses using these calls.

Multicyclic Loss -- Total

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

- You then calculate the total Multicyclic Loss in a similar fashion to how you would calculate the full loss
- This is because you essentially want further generated images to satisfy the same conditions that the original generated image would satisfy.

Losses - Full Objective Loss

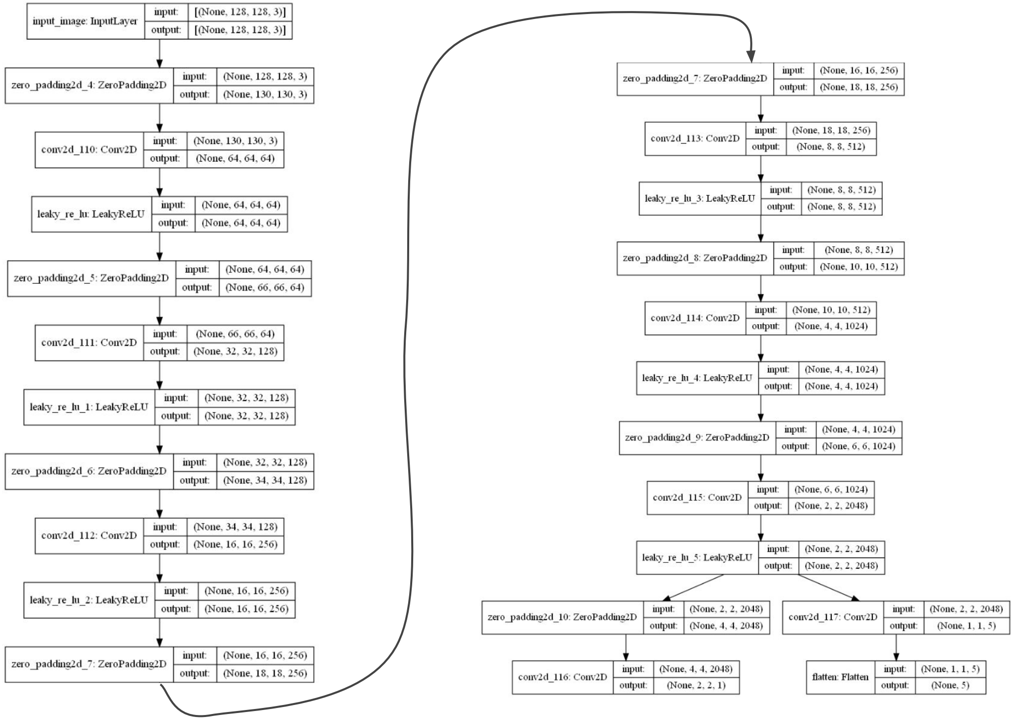
$$\mathcal{L}_D = -\mathcal{L}_{adv}^{D} + \lambda cls \mathcal{L}_{cls}^{r}$$

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

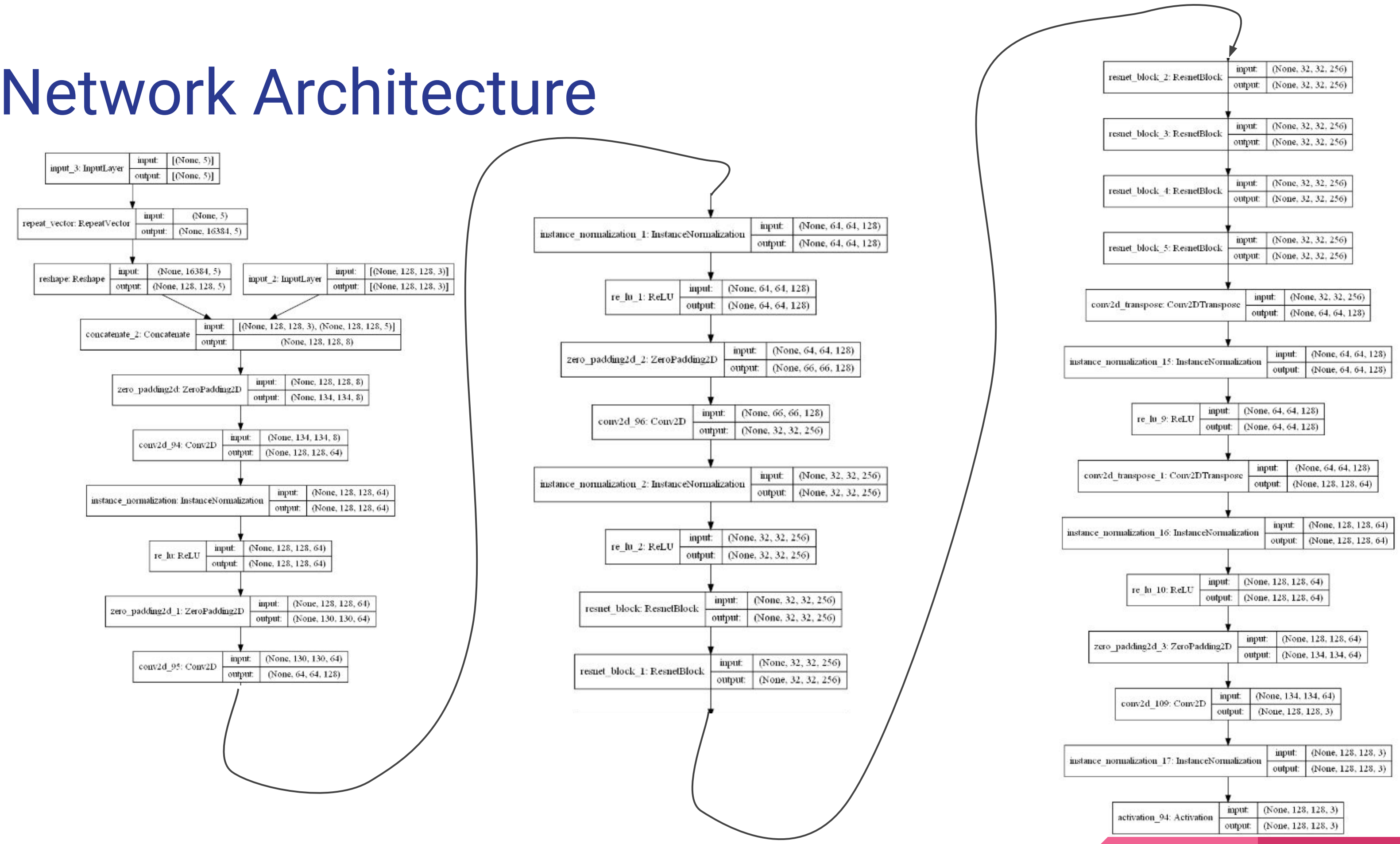
- Full Objective Loss, by reducing this, you train the generator and discriminator.
- For training, λ_{cls} = 1, λ_{cls} = 10, λ_{gp} = 10, λ_{rec} = 10, and λ_{multi} = [1.0, 0.25, 0.125, 0.0625]

Network Architecture

The network architecture, which is taken much from StarGAN is as follows:


input_image: InputLayer | input: [(None, 128, 128, 3)] | output: [(None, 128, 128, 3)]
zero_padding2d_4: ZeroPadding2D | input: (None, 128, 128, 3) | output: (None, 130, 130, 3)
conv2d_110: Conv2D | input: (None, 130, 130, 3) | output: (None, 64, 64, 64)
leaky_re_lu: LeakyReLU | input: (None, 64, 64, 64) | output: (None, 64, 64, 64)
zero_padding2d_5: ZeroPadding2D | input: (None, 64, 64, 64) | output: (None, 66, 66, 64)
conv2d_111: Conv2D | input: (None, 66, 66, 64) | output: (None, 32, 32, 128)
leaky_re_lu_1: LeakyReLU | input: (None, 32, 32, 128) | output: (None, 32, 32, 128)
zero_padding2d_6: ZeroPadding2D | input: (None, 32, 32, 128) | output: (None, 34, 34, 128)
conv2d_112: Conv2D | input: (None, 34, 34, 128) | output: (None, 16, 16, 256)
leaky_re_lu_2: LeakyReLU | input: (None, 16, 16, 256) | output: (None, 16, 16, 256)
zero_padding2d_7: ZeroPadding2D | input: (None, 16, 16, 256) | output: (None, 18, 18, 256)
zero_padding2d_7: ZeroPadding2D | input: (None, 16, 16, 256) | output: (None, 18, 18, 256)
conv2d_113: Conv2D | input: (None, 18, 18, 256) | output: (None, 8, 8, 512)
leaky_re_lu_3: LeakyReLU | input: (None, 8, 8, 512) | output: (None, 8, 8, 512)
zero_padding2d_8: ZeroPadding2D | input: (None, 8, 8, 512) | output: (None, 10, 10, 512)
conv2d_114: Conv2D | input: (None, 10, 10, 512) | output: (None, 4, 4, 1024)
leaky_re_lu_4: LeakyReLU | input: (None, 4, 4, 1024) | output: (None, 4, 4, 1024)
zero_padding2d_9: ZeroPadding2D | input: (None, 4, 4, 1024) | output: (None, 6, 6, 1024)
conv2d_115: Conv2D | input: (None, 6, 6, 1024) | output: (None, 2, 2, 2048)
leaky_re_lu_5: LeakyReLU | input: (None, 2, 2, 2048) | output: (None, 2, 2, 2048)
zero_padding2d_10: ZeroPadding2D | input: (None, 2, 2, 2048) | output: (None, 4, 4, 2048)
conv2d_117: Conv2D | input: (None, 2, 2, 2048) | output: (None, 1, 1, 5)
conv2d_116: Conv2D | input: (None, 4, 4, 2048) | output: (None, 2, 2, 1)
flatten: Flatten | input: (None, 1, 1, 5) | output: (None, 5)

Network Architecture

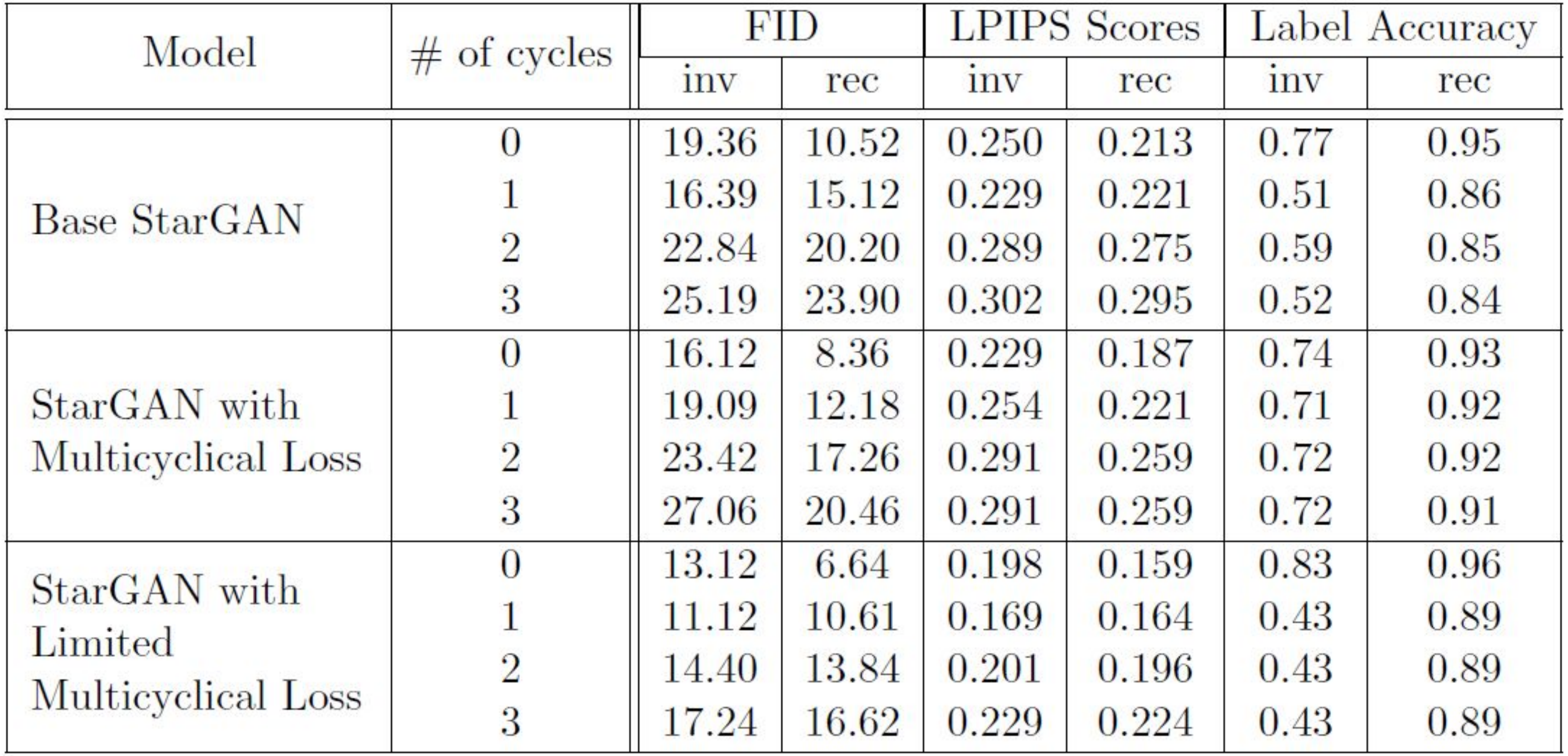

input_3: InputLayer | input: [(None, 5)] | output: [(None, 5)]
repeat_vector: RepeatVector | input: (None, 5) | output: (None, 16384, 5)
reshape: Reshape | input: (None, 16384, 5) | output: (None, 128, 128, 5)
input_2: InputLayer | input: [(None, 128, 128, 3)] | output: [(None, 128, 128, 3)]
concatenate_2: Concatenate | input: [(None, 128, 128, 3), (None, 128, 128, 5)] | output: (None, 128, 128, 8)
zero_padding2d: ZeroPadding2D | input: (None, 128, 128, 8) | output: (None, 134, 134, 8)
conv2d_94: Conv2D | input: (None, 134, 134, 8) | output: (None, 128, 128, 64)
instance_normalization: InstanceNormalization | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
re_lu: ReLU | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
zero_padding2d_1: ZeroPadding2D | input: (None, 128, 128, 64) | output: (None, 130, 130, 64)
conv2d_95: Conv2D | input: (None, 130, 130, 64) | output: (None, 64, 64, 128)
instance_normalization_1: InstanceNormalization | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
re_lu_1: ReLU | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
zero_padding2d_2: ZeroPadding2D | input: (None, 64, 64, 128) | output: (None, 66, 66, 128)
conv2d_96: Conv2D | input: (None, 66, 66, 128) | output: (None, 32, 32, 256)
instance_normalization_2: InstanceNormalization | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
re_lu_2: ReLU | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
resnet_block: ResnetBlock | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
resnet_block_1: ResnetBlock | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
resnet_block_2: ResnetBlock | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
resnet_block_3: ResnetBlock | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
resnet_block_4: ResnetBlock | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
resnet_block_5: ResnetBlock | input: (None, 32, 32, 256) | output: (None, 32, 32, 256)
conv2d_transpose: Conv2DTranspose | input: (None, 32, 32, 256) | output: (None, 64, 64, 128)
instance_normalization_15: InstanceNormalization | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
re_lu_9: ReLU | input: (None, 64, 64, 128) | output: (None, 64, 64, 128)
conv2d_transpose_1: Conv2DTranspose | input: (None, 64, 64, 128) | output: (None, 128, 128, 64)
instance_normalization_16: InstanceNormalization | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
re_lu_10: ReLU | input: (None, 128, 128, 64) | output: (None, 128, 128, 64)
zero_padding2d_3: ZeroPadding2D | input: (None, 128, 128, 64) | output: (None, 134, 134, 64)
conv2d_109: Conv2D | input: (None, 134, 134, 64) | output: (None, 128, 128, 3)
instance_normalization_17: InstanceNormalization | input: (None, 128, 128, 3) | output: (None, 128, 128, 3)
activation_94: Activation | input: (None, 128, 128, 3) | output: (None, 128, 128, 3)

Results

Data and Parameters

- 3 experiments were run, Base StarGAN, Multicyclic StarGAN, and a limited version of the Mullicyclic Loss where $L_{multi} = L_{multi\_rec}$
- Neural Network was trained on CelebA [10], a dataset of pictures with celebrity faces and over 40 domain labels, which were reduced to 5 for the experiments, Blond Hair, Brown Hair, Black Hair, Gender, and Old/Young
- The input images are of size 178x218, which were cropped to 178x178, and rescaled to 128x128. When used in the dataset, the images were also given a 50% probability for random flipping.
- The network was trained with the Adam Optimizer, with β_1=0.5 and β_2 = 0.999, and a learning rate of 0.0001 for the first half of the epochs. For the second half, it was linearly decayed to 0 over training.
- Network was trained for 20 epochs with a batch size of 8.

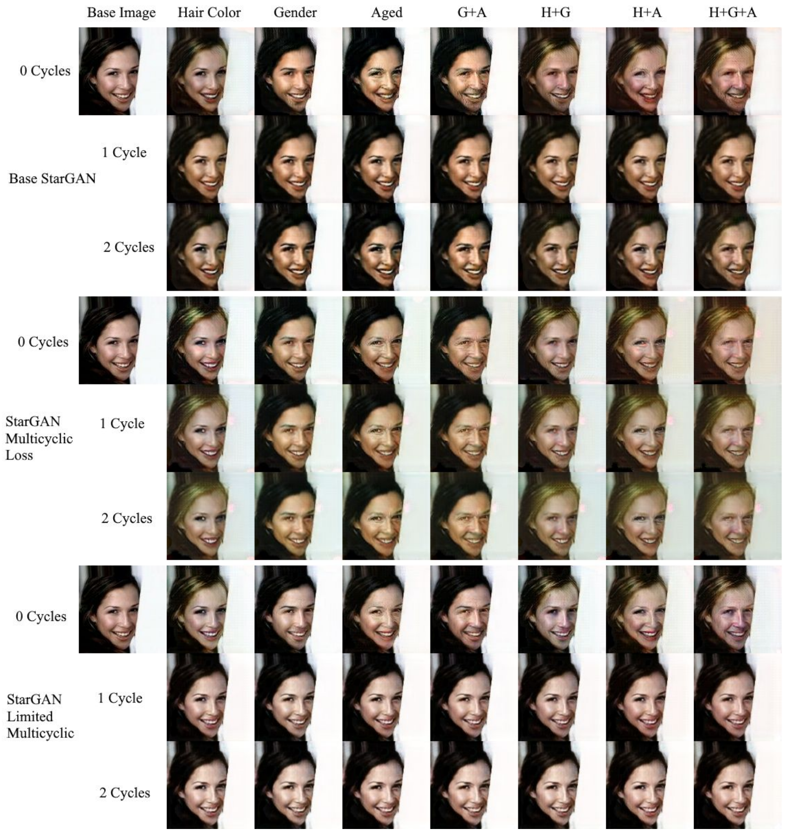
Quantitative Results

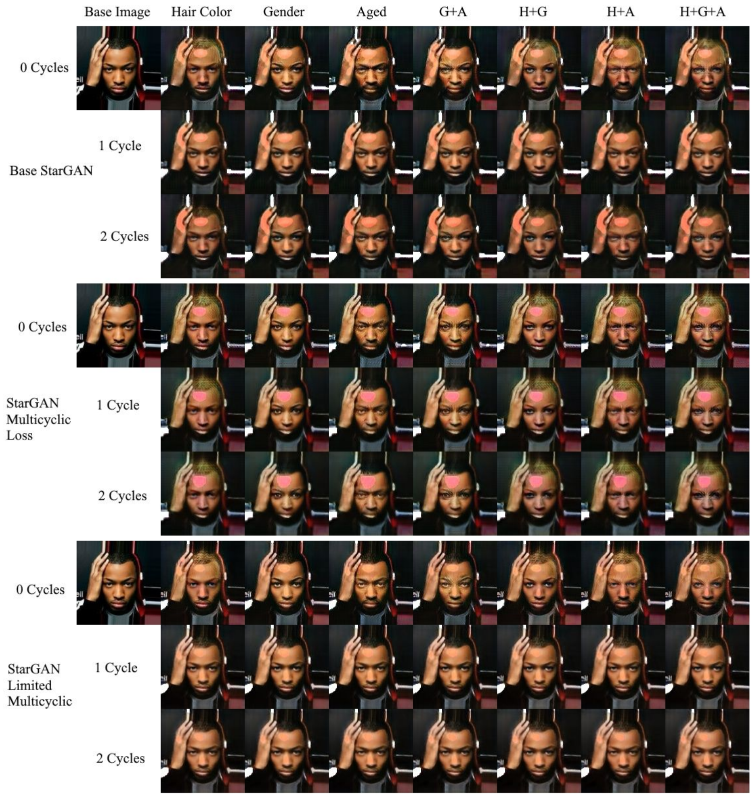


Model	# of cycles	FID		LPIPS Scores		Label Accuracy	
		inv	rec	inv	rec	inv	rec
Base StarGAN	0	19.36	10.52	0.250	0.213	0.77	0.95
	1	16.39	15.12	0.229	0.221	0.51	0.86
	2	22.84	20.20	0.289	0.275	0.59	0.85
	3	25.19	23.90	0.302	0.295	0.52	0.84
StarGAN with Multicyclical Loss	0	16.12	8.36	0.229	0.187	0.74	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

FID introduced by Heusal et al. [7], LPIPS introduced by Zhang et al. [14]

Qualitative Results





Conclusion

Conclusion

- This loss works, however it is at a cost of training time, as initial network training took 1 day on a NVIDIA Titan X Pascal, whereas both multicyclic training took 2 days in total.
- Although it does work, the data seems to both qualitatively and quantitatively degrade over time.
- There still may be room for improvement, however the new loss works.

Future Work

Future Work

- I want to work on determining the best hyperparameters to use with the model, as I did some work into that with testing, but I feel more work could be done there
- I want to try it with different models, to see if similar results are possible, particularly I want to try it with StarGAN v2, to see if the latent style codes make this loss infeasable.
- I want to try it with different datasets, including using StarGAN v1's method for training with more than one dataset using a mask vector.

Bibliography

Bibliography

[1] Martin Arjovsky, Soumith Chintala, and L´eon 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] Mart´ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, 18 Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Man´e, 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 Vi´egas, 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.