Cloud MoNet: An Adversarial Approach to Robust Unobserved Neural Style Transfer

Akhil Jalan
University of California, Berkeley
akhiljalan@berkeley.edu

Noah Wu
University of California, Berkeley
noahywu@berkeley.edu

Abstract

Neural style transfer seeks to create a realistic imitation of an image's style, but applied to another source (content) image. Previous work on style transfer indicates that the style and content portions of an image may be treated as separate components, and rearranging these portions is what drives neural style transfer's success. Given recent work on Cyclic Generative Adversarial Networks (CycleGAN), we seek to study neural style transfer as an instance of a CycleGAN. We find that it is difficult to balance the need to fool a discriminator with the original goal of high-quality style transfer, indicating the limits of both current style transfer approaches and the CycleGAN framework.

1 Introduction

Neural style transfer was first introduced by *Gatys et al.* in 2015 [3]. By observing that the representations of style and content in a Convolutional Neural Network (in particular, using VGG [6]) are separable, they used a convolutional neural network to produce realistic adaptations of images in the style of some pre-selected painting.

The quality of a style transfer is subjective and difficult to quantify. Measuring differences in the *content representations* and *style representations* of VGG was thus fundamental, as it formed a good proxy for the similarity in style/content of two images, which a network could train to optimize.

This breakthrough resulted in an explosion of work on neural style transfer in both industry and research [8]. Industrial applications such as Prisma, Ostagram, and Deep Forger allow users to stylize personal pictures. In addition to extensive study on its own, researchers have considered using neural style trasnfer in other machine learning applications, such as domain adaptation in reinforcement learning [8].

The initial version of style transfer repeatedly updated an image to minimize its content and style loss [3]. Subsequent work introduced a *style transfer network* which could transfer a learned style to an image in a single forward pass[10]. Such networks could learn only one style, however, and would have to be trained separately to transfer a different style.

Ulyanov et al. introduced *conditional instance normalization*, a surprisingly simple way for a single network to transfer a variety of styles[11]. This surprisingly simple technique simply replaced the scale and shift parameters of batch normalization with learned parameters γ_s, β_s that were different for each style s. Explicitly, if x is a unit in an activation map, with spatial (channel) mean μ and zariance σ^2, we transform it as

$$\gamma_s(\frac{x-\mu}{\sigma^2}) + \beta_s$$

CS 194-129: *Designing, Visualizing and Understanding Deep Neural Networks*. Spring 2018. Code available at: https://github.com/akhiljalan/cloud-monet

Conditional instance normalization was a significant step towards generalizing style transfer to many styles. With this simple method, others were able to use one network to transfer on 32 styles, by simply pre-setting the style-normalization parameters to those of a particular style [12].

Ghiasi et al. built on this work by introducing a *style prediction network* which maps style images to normalization parameters γ_s, β_s [1]. By training on over 8 million images, this model was able to achieve high-quality style transfer on over $80,000$ styles, and also transfer styles it had never before seen - hence the term *unobserved* neural style transfer.

Ghiasi et al is the starting point for our work. Our main contribution is to recognize that style transfer is an image-to-image translation task which exhibits the *cycle consistency* property. With this framework in mind, we formulate style transfer in a cyclic-consistent GAN framework, training a GAN end-to-end to improve the quality of style transfer. We find that it is not possible to simultaneously achieve high quality style transfer and fool a discriminator, although more work is needed.

2 Methods

2.1 Data and Pre-Processing

For content images, we used Microsoft COCO (Common Objects in Context) [13]. For style images, we used Kaggle's PBN (Painters by Numbers) dataset [14].

Our only pre-processing step was center-cropping all images to be $256 \times 256 \times 3$. This was to speed up training and evaluation.

Notably, we had more images than training steps for each model, so each step the model saw a completely new set of inputs.

2.2 Cycle Consistency and CycleGAN

Cycle-consistent adversarial networks ("CycleGAN") have shown promising results in a variety of image-to-image translation tasks, such as image colorization [5]. The key to these methods is the hypothesis that these tasks have the *cycle consistency* property. This is the hypothesis that if we are interested in some mapping $G : A \to B$, that applying the "inverse" mapping $F : B \to A$ should recover the original image. That is, for $a \in A$,

$$a = F(G(a))$$

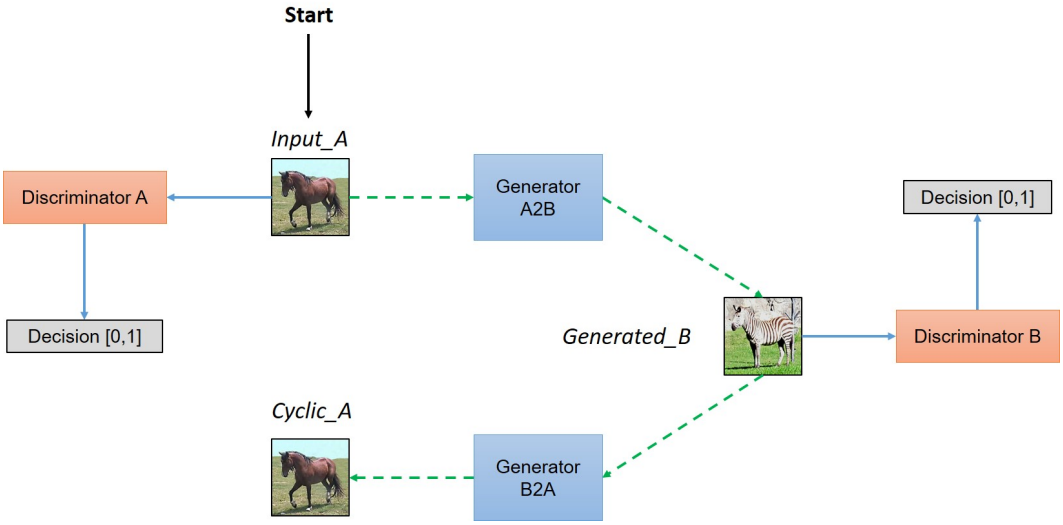

Figure 1: Example of CycleGAN. Generator A2B wants to "stripe" images of horses to make them resemble zebras. Conversely, Generator B2A wants to "unstripe" images. Discriminators A, B attempt to identify actual images of horses/zebras from generated ones. Image source: [15]

The CycleGAN model has two discriminators G_{AB}, G_{BA} which attempt to learn the mappings $G : A \to B$, $F : B \to A$. It also has two discriminators D_A, B_B which attempt to distinguish genuine instances of images from the domains A, B and generated ones. See below.

Previous work on neural style transfer has indicated that the style and content of an image are separable [3], [12], [11], [1]. This motivates our notion of cycle-consistency in neural style transfer.

Our hypothesis is that a perfect neural style transfer model should be able to not only stylize an image, but also reverse its stylization. Suppose C, S are content and style images respectively, and f maps this pair to a stylized image. If content and style are separable, then the stylized $X = f(C, S)$ should still contain the content of C, but the style of S. Therefore, applying stylization again should recover the content or style image. See figure 2 below.

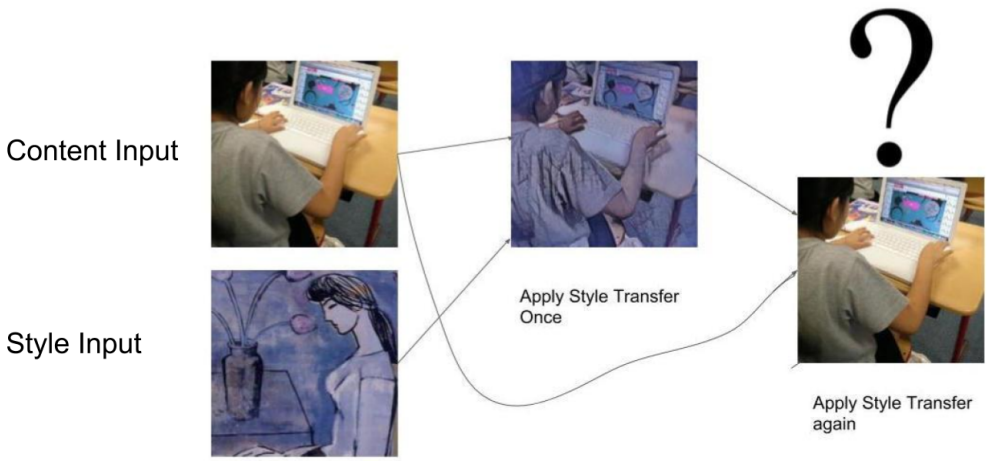

Figure 2: Example of our cycle-consistency hypothesis in the context of neural style transfer. If a style transfer network is perfect, it should be able to recover the original content image after another round of style transfer.

2.3 Baseline Model

At a high level, our baseline model has two components: A style transfer network, and a content-discriminator. The style transfer network maps pairs of content and style images to stylized images. See figure 3.

Our discriminator network is a very simple convolutional network, which assigns each input a probability of being a real content image or a generated one (see figure 4). Its loss is the cross-entropy loss between its assigned probabilities of an image being real or fake, and the "true" probabilities, which are $[1 \quad 0]$ for a real image, and $[0 \quad 1]$

Our generator is the style transfer network from *Ghiasi et al.* [1]. We use a pre-trained version of their model (available at [2]), which we then fine-tune on our COCO and PBN images.

Ghiasi et al. uses three Convolutional Neural Networks for style transfer. See Figure 5.

First, the *style prediction network* computes a set of conditional instance normalization parameters $\vec{S} = [\gamma_{s_1} \quad \beta_{s_1} \quad \cdots \quad \gamma_{s_n} \quad \beta_{s_n}]$ for an input style image S. Notably, this image might have been previously unseen.

Second, the *transformer network* stylizes a content image C using these normalized parameters $\vec{S}$. It performs batch normalization using the instance normalization parameters $\vec{S}$.

Third, the VGG16 network (from [6]) computes the content and style loss of the resulting stylized image. This is exactly the same technique as *Gatys et al.*, which uses differences in VGG's hidden layer activations as proxies for differences in the style and content of two images. We introduce the following notation:

$$C \in \mathbb{R}^{H \times W \times 3} \leftarrow \text{Content Image} \tag{1}$$
$$S \in \mathbb{R}^{H \times W \times 3} \leftarrow \text{Style Image} \tag{2}$$
$$X \in \mathbb{R}^{H \times W \times 3} \leftarrow \text{Stylized Image} \tag{3}$$
$$\lambda_s \in \mathbb{R}^{+}, \lambda_v \in \mathbb{R}^{+} \leftarrow \text{Hyperparameters} \tag{4}$$
$$\tag{5}$$

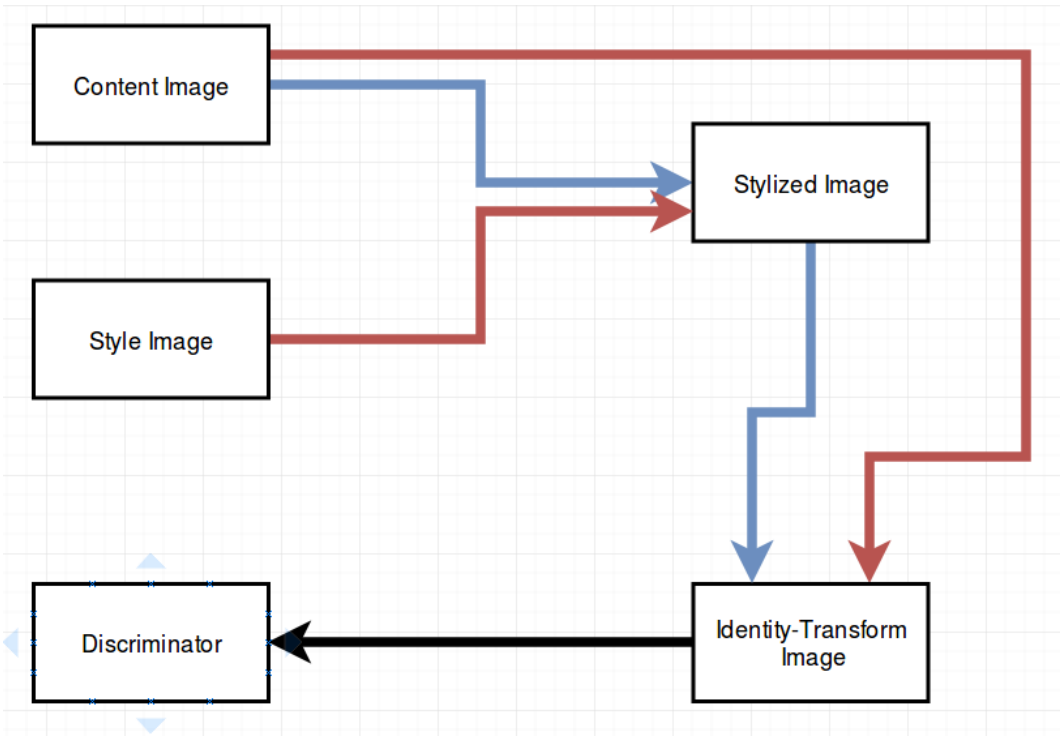

Figure 3: Schematic for our model architecture, which is similar to CycleGAN [5]. Style transfer is applied once to produced a stylized image. Then, the same style transfer network is applied again, but with the stylized image as "content", and the original content image as "style". The output, which we call the *Identity-Transform Image*, should ideally be identical to the content image. This should fool the discriminator, which seeks to distinguish Identity-Transform Images from true Content Images.

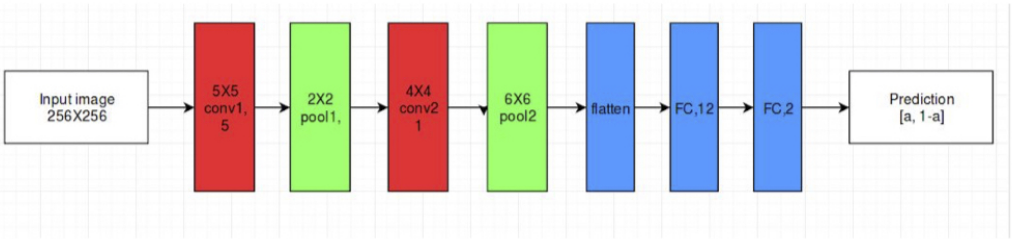

Figure 4: Model architecture for our discriminator network. It consists of two convolution, max-pool layers, followed by two fully-connected layers. All activation functions are ReLU, except the final fully-connected layer, which uses softmax.

The definitions of style and content loss are then given below. Here, f_j denotes the hidden layer activation of the jth layer of VGG, and $\mathcal{G}$ denotes a Gram matrix [1].

$$\mathcal{L}_{total}(X, C, S) = \mathcal{L}_c(X, C) + \lambda_s \mathcal{L}_s(X, S) + \lambda_v \mathcal{L}_v(X) \leftarrow \text{Total Loss}$$

$$\mathcal{L}_c(X, C) = \frac{1}{n_c} \sum_{j \in N_C} \|f_j(X) - f_j(C)\|_2^2 \leftarrow \text{Content Loss}$$

$$\mathcal{L}_s(X, S) = \frac{1}{n_s} \sum_{i \in N_S} \|\mathcal{G}[f_i(X)] - \mathcal{G}[f_i(S)]\|_F^2 \leftarrow \text{Style Loss}$$

$$\mathcal{L}_v(X) = \frac{1}{3W(H-1)} [\sum_{i=1}^{H-1} \sum_{j=1}^{W} \sum_{k=1}^{3} (X_{i,j,k} - X_{i+1,j,k})^2]^{1/2} \leftarrow \text{Variation Loss}$$

$$+ \frac{1}{3(W-1)H} [\sum_{i=1}^{H} \sum_{j=1}^{W-1} \sum_{k=1}^{3} (X_{i,j,k} - X_{i,j+1,k})^2]^{1/2}$$

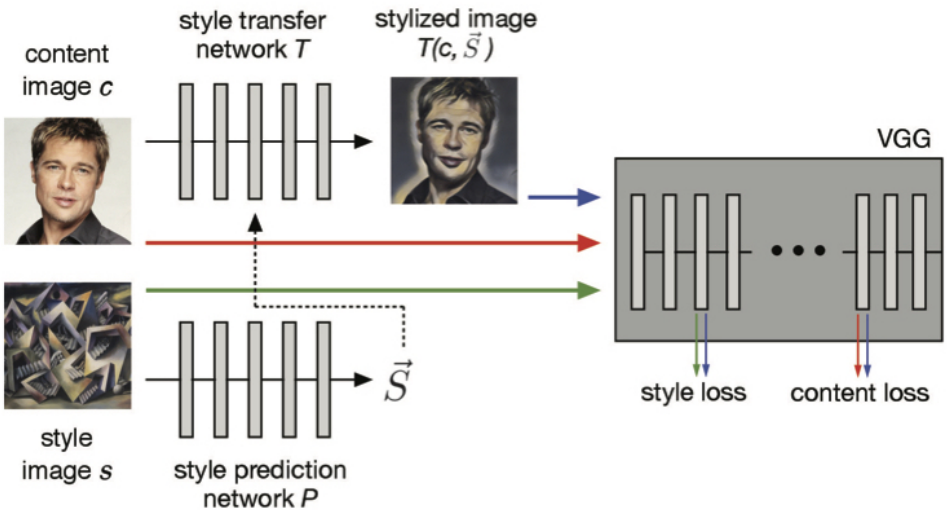

Figure 5: Style-Transfer Model from *Ghiasi et al.* [1]. By using a style-prediction network (bottom left), the model can compute instance normalization parameters for any style image, even an unobserved one. See sction 2.3 (Baseline Model) for more.

Because there are two passes of the style transfer network, we obtain a content, style, and variation loss twice - one for each pass. We denote these "Pass 1 Losses" with the superscript $\mathcal{L}^1$ and "Pass 2 Losses" with $\mathcal{L}^2$.

In our baseline model, the generator (that is, the neural style transfer network) is trained in a matter identical to [1]. While we perform the entire forwards pass of the model, in the backwards pass only the losses from the first style transfer are used for gradient updates on the style transfer network.

The discriminator trains on a 50-50 mix of true content images and generated ones. While it trains simultaneously with the generator, the training in our baseline model is not truly a "GAN" training regime, since there is no gradient flow from the discriminator to the generator.

[1]Notice that while content loss simply takes Eucliden distance of activations, style loss measures Frobenius difference of Gram matrices of the activations. This technique traces back to *Gatys et al.*, who found that for style differences the Gram matrix better captures variation in features of interest, such as texture [3].

The losses optimized by the baseline model are below. See our Github repository for a full list of hyperparameters and source code[2].

$$\mathcal{L}_G(X,C,S) = \mathcal{L}^1_{total}(X,C,S) = \mathcal{L}^1_c(X,C) + \lambda_s \mathcal{L}_s(X,S)^1 + \lambda_v \mathcal{L}_v(X)^1 \leftarrow \text{Generator Loss} \tag{6}$$

$$\mathcal{L}_D(Y) = \begin{cases} -log_e(p_{\hat{real}}) & \text{Y is a true content image} \\ -log_e(p_{\hat{fake}}) & \text{Y is an identity-transform image} \end{cases} \leftarrow \text{Discriminator Loss} \tag{7}$$

2.4 Final Model

Our final model has the same architecture as our baseline model, but trains end-to-end. Therefore, the style transfer network no longer optimizes just for style and content loss in the first pass, but for fooling the discriminator.

The generator fooling loss measures how successfully the style transfer network tricks the discriminator into believing the identity-transformed image is a genuine content image. Since we use cross-entropy loss for the discriminator, it is natural to use cross-entropy loss here as well, but with distributions reversed. That is, the generator wants the discriminator to assign $p_{\hat{real}} = 1$ and $p_{\hat{fake}} = 0$ to the identity-transformed image I. Therefore, it wants the distribution $[p_{\hat{real}} \quad p_{\hat{fake}}]$ to $[1 \quad 0]$, while the discriminator wants to match $[1 \quad 0]$.

$$I = G(G(C,S),C) \leftarrow \text{Identity-transformed image} \tag{8}$$

$$\mathcal{L}_{G,fooling}(I) = -[1 * log_e(p_{\hat{real}}) + 0 * log_e(p_{\hat{fake}})] \tag{9}$$

$$= -log_e(p_{\hat{real}}) \leftarrow \text{Generator Fooling Loss} \tag{10}$$

$$\tag{11}$$

The loss functions for the generator and discriminator are thus:

$$\mathcal{L}_G(X,C,S) = \mathcal{L}^1_{total}(X,C,S) + \lambda_f \mathcal{L}_{G,fooling}(I) \leftarrow \text{Generator Loss} \tag{12}$$

$$\mathcal{L}_D(Y) = \begin{cases} -log_e(p_{\hat{real}}) & \text{Y is a true content image} \\ -log_e(p_{\hat{fake}}) & \text{Y is an identity-transform image} \end{cases} \leftarrow \text{Discriminator Loss} \tag{13}$$

Notably, the generator loss does not include any loss terms for the second pass. We found that explicitly optimizing for this was unnecessary given that our model was already pre-trained to optimize for content and style loss.

Before deciding on this model architecture, we explored several others. We discuss notable alternatives below.

At first, we attempted to train a discriminator on stylized vs original content/style images, only performing one forward pass of style transfer. However, we found that this task was far too easy for the discriminator, rapidly achieving 100 percent accuracy. Furthermore, we reasoned that a generator which optimizes to fool the discriminator would learn to simply perform the identity function on its inputs, defeating the purpose of achieving good style transfer. Cyclic consistency was key to defeating this problem, since now the network is penalized for producing a "bad" stylized image.

Second, we considered optimizing explicitly for ℓ_1 difference between the identity-transformed image and the original content image, as was done for CycleGAN [5]. In the case of neural style transfer, however, we felt this was inappropriate because ℓ_1 difference measures pixel-wise differences, while content and style are global properties of an image. Further, this would class with our *variation loss* which smooths out the image by penalizing regions that are not locally self-similar [4]. This is needed to prevent images which look like noise, because they have been optimized pixel-wise as opposed to globally.

[2] `https://github.com/akhiljalan/cloud-monet`

Finally, we discussed performing un-stylization on style images rather than content. That is, in the second pass we would try to recover the original style image by treating the stylized image X as style, and the original style image S as content. We thought content recovery would be an easier task, given that computation of style differences is more subtle and difficult, requiring Gram Matrix differences rather than Euclidean distance [3]. However, this is a valuable direction for further research.

3 Results

We trained both our baseline model and final model from the pre-trained model checkpoint available at [2], and both times the discriminator was trained from scratch. Losses displayed below are all during training. Because we had enough data to introduce a new input for each step of training, this is equivalent to measuring test loss on a withheld test set.

Our main finding is that when our final model is trained end-to-end, there is a seemingly inescapable tradeoff between the quality of a style transfer, and the extent to which it can fool the discriminator. While our final model fools the discriminator well, it performs poorly on the task of style transfer itself.

In our baseline model, the generator optimizes only for style and content loss from the first pass. The discriminator is then able to quickly achieve 100% classification accuracy, easily distinguishing identity-transformed images from genuine content images.

In our final model, we include a weighted fooling loss in the generator's loss, as described in Methods. After training model is able to beat the discriminator 100% of the time, with all of the identity-transformed images being classified as real. However, it has considerably worse content and style losses in the first pass than the baseline model. See figure 6 below.

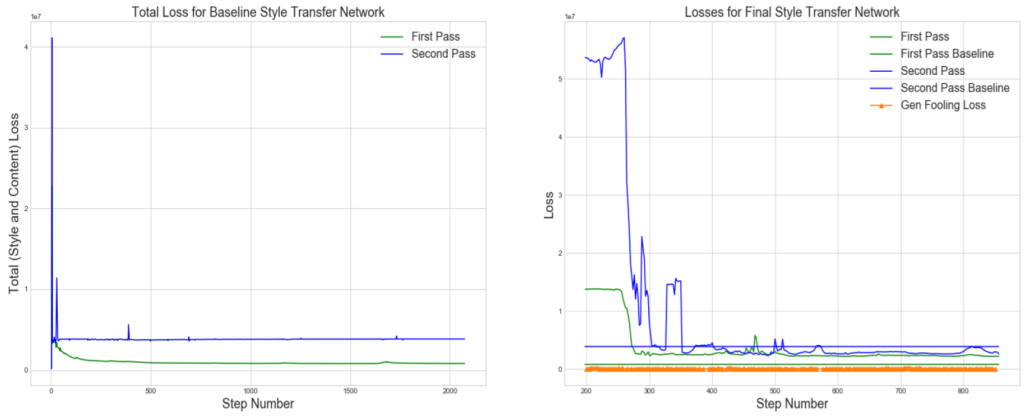

Figure 6: Total losses. Left: Baseline Model, with the generator only optimizing for first forward pass. Right: Final model, with the generator optimizing for fooling loss and loss from pass 1. Note that loss from pass 1 is worse than the baseline, but loss from pass 2 is better, despite not optimizing for it.

Notably, the network achieves better second-pass loss than the baseline model, despite not optimizing for it.

By prioritizing fooling loss heavily, the generator is able to compete with the discriminator in the final model during its training history. Even at the end of training, the generator and discriminator fooling losses are unstable.

In contrast, the baseline model, which does not include fooling loss, almost immediately stabilizes to a point where the discriminator achieves 100% accuracy, as is evident from the confusion matrices. See figures 7 and 8 below.

All three losses - pass 1, pass 2, and fooling - can be explained by the fact that the final model is facing mode collapse. While the stylized and un-stylized images in the baseline model appear reasonably

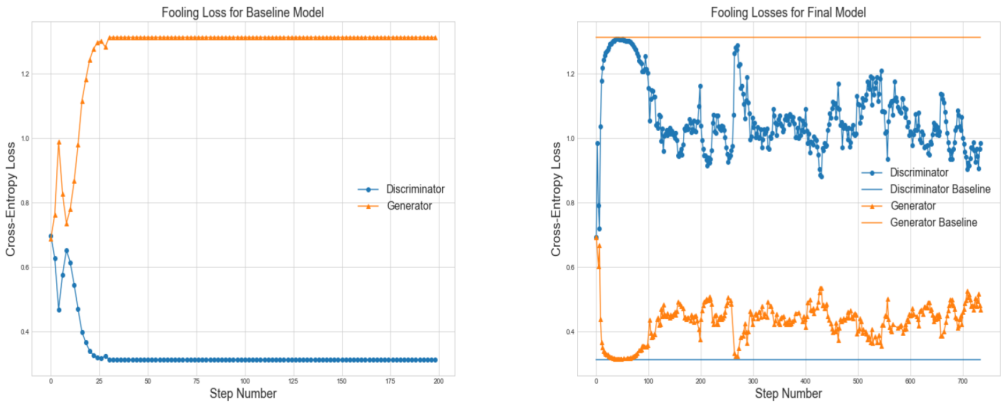

Figure 7: Fooling Losses (Measured by Cross-Entropy). Left: Baseline Model, with the generator only optimizing for first forward pass. Right: Final model, with the generator optimizing to fool the discriminator and minimize style and content losses.

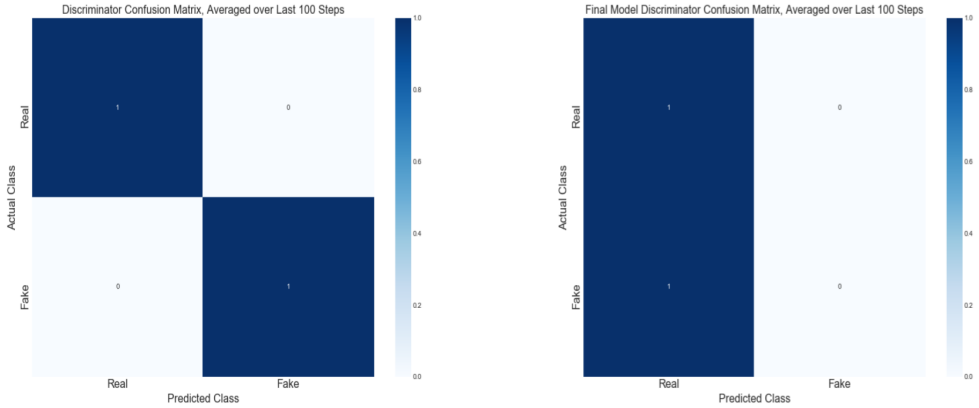

Figure 8: Confusion Matrices for Discriminator. Note that the discriminator produces probabilities of an image being real and fake. For classification purposes, we classify image as "real" if the discriminator has $\geq 50\%$ confidence that it is real. Left: Baseline Model, with the generator only optimizing for first forward pass. Right: Final model, with the generator optimizing to fool the discriminator and minimize style and content losses.

good, the final model produces utter rubbish which is designed to mimic a true content image by being a blank, black image. See figure 9.

4 Discussion

We contend that properties of the optimization landscape make it difficult to simultaneously achieve our goals of fooling a discriminator and producing high-quality style-transfer.

Earlier versions of our final model used identical architecture and hyperparameters, but a much lower fooling loss weight. We found that this resulted in no noticeable difference in any metrics, nor in images. While we did not train these as long, we believe further training with such low λ_f values would not have helped. With $\lambda_f = 1$, for example, the discriminator easily achieved 100% accuracy easily, and there were no noticeable differences in images. See figure 10 below.

Therefore, it is necessary to have a relatively high λ_f. However, as we observed, this ruins the quality of original style transfer, with much higher content and style losses.

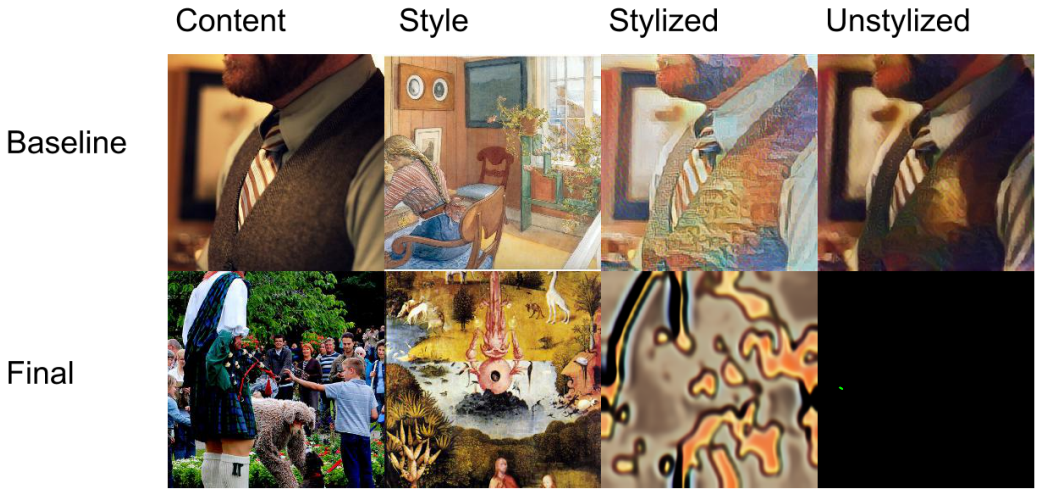

Figure 9: Images produced by baseline and final models. The style transfer network degenerates, producing strange results for the first pass and an almost blank image in the second pass.

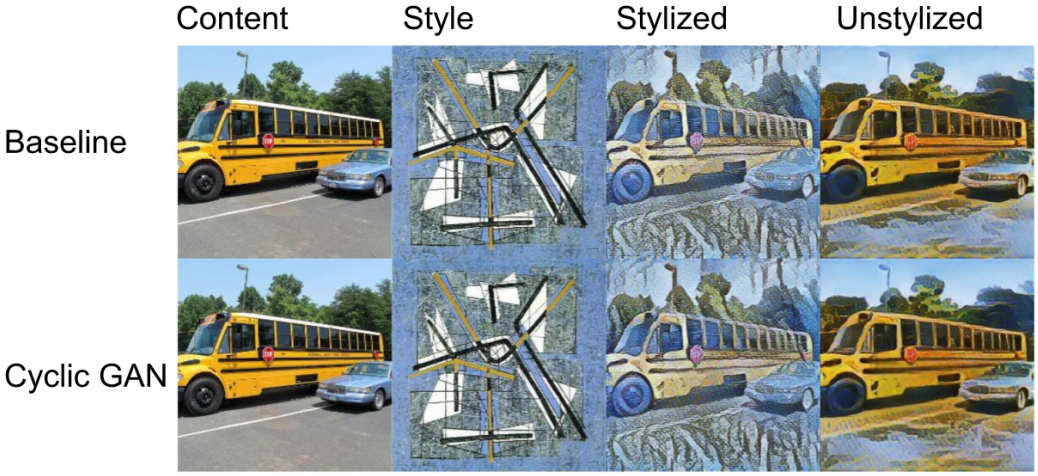

Figure 10: Images produced by baseline and an older version of our final model, after 100 training steps. Given $\lambda_f = 1$, fooling loss has virtually no contribution to the generator loss.

Is it possible to have it both ways? Our results would indicate not, but given the hundreds of permutations of loss functions and training regimes we did not explore, we are not confident in saying so.

We speculate the task of our generator (style transfer) is much more difficult than the task of our discriminator, any training regime must heavily favor the generator.

While we tried to make things relatively easy for the generator and hard for the discriminator, evidently this was not enough to balance the relative difficulties of their tasks. Notably, we started training the generator from a pre-trained checkpoint with over 8 million steps of training, while the discriminator was trained from scratch.

Despite this, the generator resorted to mode collapse when tempted with a high enough weighting towards fooling loss (that is, a high λ_f). This indicates the presence of spurious local minima in the optimization landscape of the generator, which result in better overall loss from its point of view (by fooling the discriminator), but defeat the purpose of a high-quality style transfer.

5 Conclusion

We conclude that the simultaneous task of producing good neural style transfer and fooling a content-reconstruction discriminator is too difficult for current models. When style transfer is good, a discriminator can easily discriminate real from fake content images. But when the style transfer begins to optimize for fooling the discriminator, mode collapse occurs, resulting in low-quality style transfer.

We found this project a valuable and interesting experience, and learned many valuable lessons. Despite being experienced Python programmers, we both struggled to write robust Tensorflow code and debug well. Further, performing literature search was valuable in and of itself, as neural style transfer is a relatively interdisciplinary subfield of machine learning. We found that via our research, we learned not just about style transfer, but also GANs, optimization, and neural networks in general.

There are several fruitful directions for future work. Due to the intense computational requirements of style transfer, we were unable to perform any hyperparameter searches which might have yielded better results. In particular, the relative weightings of style, content, total variation, and fooling loss are likely to have serious effects on the quality of the eventual style transfer.

As discussed in section 2.4 (final model), formulations besides our content-recovery GAN model merit attention, especially the analogous task of a style-recovery model. Further, because we ended up not optimizing for content and style loss in the second pass of style transfer, using a separate network for this transformation might result in better outcomes.

Finally, as we discuss in our results section, even when the above questions are answered, we are left with a difficult optimization problem. Exploration of different optimization algorithms[3].

Team Contributions

Both team members contributed 50% of the work for the project. We worked together most of the time, and both worked on writing code, reading papers, and creating the slides, poster, and final paper.

Tools

We wrote code in Python 2.7, using Tensorflow version 1.7 for our neural networks. This was mainly to maintain consistency with the source code for our baseline model, which was in the Magenta environment available at [2].

We used Tensorboard during model training to track progress. This was essential to finding bugs quickly and assessing model progress in real-time for runs that took several hours.

Acknowledgments

We are grateful to the instructors of CS 194-129 for their support and advice. Professor John Canny, as well as our Gradudate Student Instructors Erin Grant and Carlos Florensa, were very generous with their time and helped guide our thinking at numerous steps in the process.

We are also grateful to Magenta at Google for open-sourcing the code for our baseline model [2]. In particular, their release of pre-trained model weights made this project possible, since we did not have the computational resources to re-train their model from scratch.

We created this paper in LaTeXusing the NIPS 2017 paper template, availabe at: `https://nips.cc/Conferences/2017/PaperInformation/StyleFiles`.

References

[1] Ghiasi, Golnaz, et al. "Exploring the structure of a real-time, arbitrary neural artistic stylization network." arXiv preprint arXiv:1705.06830 (2017).

[2] `https://github.com/tensorflow/magenta/tree/master/magenta/models/arbitrary_image_stylization`

[3] We used Adam, or Adaptive Moment Estimation.[16]

[3] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." arXiv preprint arXiv:1508.06576 (2015).

[4] Chambolle, Antonin. “An algorithm for total variation minimization and applications." Journal of Mathematical imaging and vision 20.1-2 (2004): 89-97.

[5] Zhu, Jun-Yan, et al. "Unpaired image-to-image translation using cycle-consistent adversarial networks." arXiv preprint arXiv:1703.10593 (2017).

[6] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[7] Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[8] Jing, Yongcheng, et al. "Neural style transfer: A review." arXiv preprint arXiv:1705.04058 (2017).

[9] Marchant, Jo. "A Journey to the Oldest Cave Paintings in the World." Smithsonian Magazine, Jan. 2016.

[10] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." European Conference on Computer Vision. Springer, Cham, 2016.

[11] Ulyanov, D., Vedaldi, A., and Lempitsy, V. "Instance normalization: The missing ingredient for fast stylization." arXiv preprint arXiv:1607.08022 (2016).

[12] Dumoulin, Vincent, Jonathon Shlens, and Manjunath Kudlur. "A learned representation for artistic style." CoRR, abs/1610.07629 2.4 (2016): 5.

[13] Lin, Tsung-Yi, et al. "Microsoft COCO: Common objects in context." European conference on computer vision. Springer, Cham, 2014.

[14] `https://www.kaggle.com/c/painter-by-numbers`

[15] Bansal, Hardik and Archit Rathore. "Understanding and Implementing CycleGAN in TensorFlow." `https://hardikbansal.github.io/CycleGANBlog/`

[16] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[17] Gulrajani, Ishaan, et al. "Improved training of wasserstein gans." Advances in Neural Information Processing Systems. 2017.