Neural Style Transfer

Xiangtian Li
University of California, Berkeley
li.xiangtian@berkeley.edu

Abstract

The works of Gatys et al. demonstrated the capability of Convolutional Neural Networks (CNNs) in creating artistic style images. This process of utilizing CNNs to transfer content images in different styles referred to as Neural Style Transfer (NST). In this paper, we re-implement image-based NST, fast NST, arbitrary NST and human-to-anime face transfer. We also extend the algorithms to transfer daytime to night, mix different styles and create artistic style videos.[1]

1. Introduction

The first NST is proposed by Gatys et al. [2], [3], which is called image-based NST. Since their methods work on NST, there has been a wealth of research on improving their techniques. Johnson et al. [5] proposed an idea that is to pre-trained a feed-forward style-specific network and produce a stylized result with a single forward pass at testing stage. Huang and Belongie [4] proposed to modify conditional instance normalization [1] to adaptive instance normalization. Kim et al.[7] proposed an unsupervised method that can transfer human face into anime face. We base our work around the following four papers. Figure 6 shows some comparable results.

2. Image-Based Neural Style Transfer

We start by reimplementing the first NST algorithm proposed by Gatys et al. [2], [3]. Given a content image I_c and a style image I_s, the algorithm in [2] seeks a stylized image I that minimizes the following loss:

$$\arg\min_I \mathcal{L}(I_c, I_s, I) = \arg\min_I \ \alpha\mathcal{L}_c(I_c, I) + \beta\mathcal{L}_s(I_s, I)$$

where $\mathcal{L}_c$ and $\mathcal{L}_s$ represent the content loss and style loss respectively, both are computed based the layers of the VGG-19 [10].

[1]Project homepage is: https://inst.eecs.berkeley.edu/~cs194-26/sp20/upload/files/projFinalProposed/cs194-26-agr/

2.1. VGG-19

By reconstructing the middle layers of the VGG-19 network, Gatys et al. find that a deep convolutional neural network is capable of extracting image content and some appearance from well-known artwork. Fig. 1 shows the architecture of VGG-19. Designed for image classification, we are now more interested in the structure of the network.

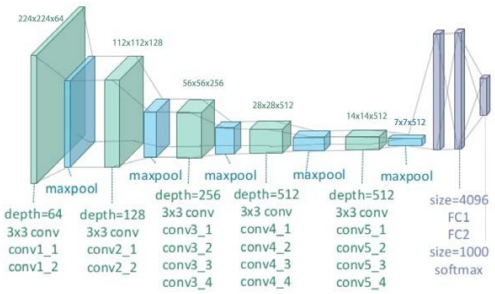


Figure 1. VGG-19

What the paper proposed to do is to get the convolutional filters outputs from a set of convolutional layers. We will get different types of information about the image from different layers.

2.2. Content Loss

The content loss $\mathcal{L}_c$ is defined by the squared Euclidean distance between the feature representations $\mathcal{F}^l$ of the content image I_c and the stylized image I in layer l, where I is initialized to the content image I_c:

$$\mathcal{L}_c(I_c, I) = \|\mathcal{F}^l(I_c) - \mathcal{F}^l(I)\|^2$$

In our experiment, we use the *conv*4_1 layer's filter outputs. This is because *conv*4_2 preserves the content while losing the high frequency details.

2.3. Style Loss

For style loss $\mathcal{L}_s$, the paper exploits Gram matrix to model the style. Assume that the feature map of a style image I_s at layer l in a pre-trained neural network is $\mathcal{F}^l(I_s) \in \mathbb{R}^{C\times H\times W}$, where C is the number of channels, and H and

W represent the height and width of the feature map $\mathcal{F}(I_s)$. Then the Gram matrix $\mathcal{G}(\mathcal{F}^l(I_s)') \in \mathbb{R}^{C\times C}$ can be computed over the feature map $\mathcal{F}(I_s)' \in \mathbb{R}^{C\times(HW)}$ (a reshaped version of $\mathcal{F}(I_s)$):

$$\mathcal{G}(\mathcal{F}^l(I_s)') = [\mathcal{F}^l(I_s)'][\mathcal{F}^l(I_s)']^T.$$

The style loss is defined by the square Euclidean distance between the Gram matrix of I_s and I:

$$\mathcal{L}_s(I_s, I) = \sum_{l\in\{l_s\}} w_l \|\mathcal{G}(\mathcal{F}^l(I_s)') - \mathcal{G}(\mathcal{F}^l(I)')\|^2$$

where $\{l_s\}$ represents some layers in VGG-19, i.e. *conv*1_1, *conv*2_1,*conv*3_1,*conv*4_1,*conv*5_1, and w_l are weighting factors of the contribution of each layer to the total loss.

2.4. Experiment

We trained on a Linux (Ubuntu 16.04), with a Tesla P40 and 24GB memory. Our implementation is based on Python and Pytorch, and Pillow for image processing. We tried two optimizers, Adam and L-BFGS, and L-BFGS produced a better stylized result. However, L-BFGS consumes more memory and takes longer time to get the results. Our actual training was done in 400 iterations for artistic style transfer, with learning rate $= 1$.

As for the results, we generate artistic style transfer results (A.1) with $\alpha = 1$ and $\beta = 1000$, daytime transfer results (A.2) with $\alpha = 1000$ and $\beta = 10$ and style mixture results (A.3) with $\alpha = 1$, $\beta = 1$ and the weight of the first layer $w_{conv1\_1} = 5$.

3. Fast Neural Style Transfer

Although image-based NST is able to yield an impressive stylized images, there are still some limitations. The most concerned problem is the efficiency issue. Usually, image-based NST needs to trained for hundreds of iterations to get a result image, which is a large consume of time.

The first model-based NST algorithm is proposed by Johnson et al. [5].The idea of the algorithm is to pre-train a feed-forward style-specific network and produce a stylized result with a single forward pass at test stage. The algorithm of Johnson et al. achieves a real-time style transfer. Its architecture is demonstrated in Figure 2, which consists of image transformation network and loss network.

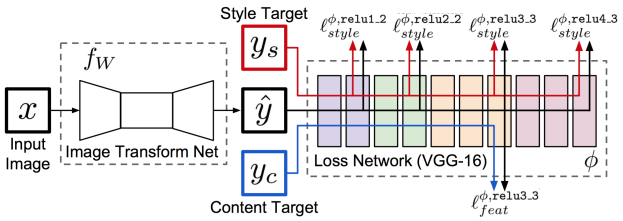


Figure 2. System Overview

3.1. Image Transformation Network

In the training stage, the image transformation network takes in a color image of shape $3 \times 256 \times 256$. Since the image transformation networks are fully convolutional, at testing stage it can be applied to images of any resolution. Table 1 includes a detailed architecture of the transformation network.

3.2. Perceptual Loss

We define two perceptual loss functions that make use of a loss network ϕ. In our experiments, the loss network ϕ is the VGG-16 [10] pretrained on ImageNet.

Feature Reconstruction Loss. The feature reconstruction loss is the squared Euclidean distance between feature representations:

$$\ell^{\phi,j}_{feat}(\hat{y}, y) = \frac{1}{C_j H_j W_j} \|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

where y and $\hat{y}$ is the target image and output image respectively, ϕ_j is the feature map in j-th layers of the VGG-16 network and $C_j \times H_j \times W_j$ is the shape of ϕ_j.

Style Reconstruction Loss. To penalizes the differences in style between $\hat{y}$ and y, we utilize the style reconstruction loss proposed by Gatys et al. [3]. Define the Gram matrix $G^\phi_j(x)$ whose elements are given by

$$G^\phi_j(x)_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}$$

The style reconstruction loss is then the squared Frobenius norm of the difference between the Gram matrices of the output and target iages:

$$\ell^{\phi,j}_{\text{style}}(\hat{y}, y) = \left\| G^\phi_j(\hat{y}) - G^\phi_j(y) \right\|^2_F$$

We define $\ell^{\phi,J}_{\text{style}}$ to be the sum of losses for a set of layers J and perform style reconstruction on J.

3.3. Experiments

We trained on a Linux (Ubuntu 16.04), with a Tesla P40 and 24GB memory. Our implementation is based on Python and Pytorch, and Pillow and OpenCV for image and video processing. We train style transfer networks on the MS-COCO dataset [8]. We resize each of the 80k training images to 256×256, and train with a batch size of 4 for 40k iterations. We have trained four models using four style images shown in Fig. 13 and apply the models on two content images (B.2).

We also utilize the model to produce a stylized video (B.3). What we have done is simply stylizing each frame in the video and combining the frames together. As we stylize each frame independently, the video is not really smooth.

4. Arbitrary Style Transfer

Based on the previous methods that solve the efficiency issue, we now aim at one-model-for-all, i.e., one single model to transfer arbitrary styles. The algorithm of Huang and Belongie [4] is the first algorithm that achieves a real-time stylization.

4.1. Adaptive Instance Normalization

Instead of training a parameter prediction network, Huang and Belongie propose to modify conditional instance normalization (CIN) to adaptive instance normalization (AdaIN):

$$\text{AdaIN}\left(\mathcal{F}\left(I_c\right), \mathcal{F}\left(I_s\right)\right) = \sigma\left(\mathcal{F}\left(I_s\right)\right)\left(\frac{\mathcal{F}\left(I_c\right) - \mu\left(\mathcal{F}\left(I_c\right)\right)}{\sigma\left(\mathcal{F}\left(I_c\right)\right)}\right) + \mu\left(\mathcal{F}\left(I_s\right)\right)$$

where I_c and I_s is the content and style image respectively, and $\mathcal{F}$ is the feature representation.

AdaIN transfers the channel-wise mean and variance feature between content and style feature representations.

4.2. Network Architecture

Fig. 3 shows an overview of the style transfer network based on the AdaIN layer.

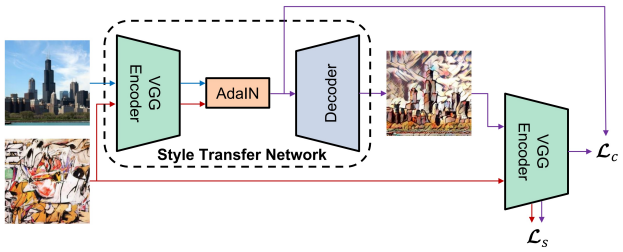


Figure 3. Style Transfer Algorithm Overview

The encoder f is the first few layers of a pretrained VGG-19 [10]. After the encoder, we feed both content and style feature maps to an AdaIN layer that produces the target feature maps t:

$$t = \text{AdaIN}(f(I_c), f(I_s))$$

A randomly initialized decode g, symmetric to the encoder f, is trained to map t back to the image space and get the output image $g(t)$.

4.3. Loss Functions

We use pre-trained VGG-19 [10] f to compute the loss to train the encoder. Define the content loss $\mathcal{L}_c$ to be the Euclidean distance between the target features and features of the output image t:

$$\mathcal{L}_c = \|f(g(t)) - t\|_2$$

The style loss matches the mean and standard deviation of the style features:

$$\mathcal{L}_s = \sum_{i=1}^{L} \|\mu\left(\phi_i(g(t))\right) - \mu\left(\phi_i(s)\right)\|_2 + \sum_{i=1}^{L} \|\sigma\left(\phi_i(g(t))\right) - \sigma\left(\phi_i(s)\right)\|_2$$

where each ϕ_i represents a layer in VGG-19. In our experiment we use *relu*1_1,*relu*2_1,*relu*3_1,*relu*4_1 layers with equal weights.

By introducing a style loss weight λ we have our final loss:

$$\mathcal{L} = \mathcal{L}_c + \lambda\mathcal{L}_s$$

4.4. Experiments

We trained on a Linux (Ubuntu 16.04), with a Tesla P40 and 24GB memory. Our implementation is based on Python and Pytorch, and Pillow for image processing. We train style transfer models using MS-COCO dataset [8] as content images and a dataset of paintings collected from WikiArt [9]. We resize each of the 80k training images to 512×512 and random crop down images with size 256×256 and train with a batch size 8 for 20 epochs. Fig. 17, 18 show the stylized images produced by our models.

Content-style trade-off. In addition to adjust the weight λ, we can also interpolate at testing stage between feature maps that are fed to the decoder:

$$T(c, s, \alpha) = g((1-\alpha)f(c) + \alpha\,\text{AdaIN}(f(c), f(s)))$$

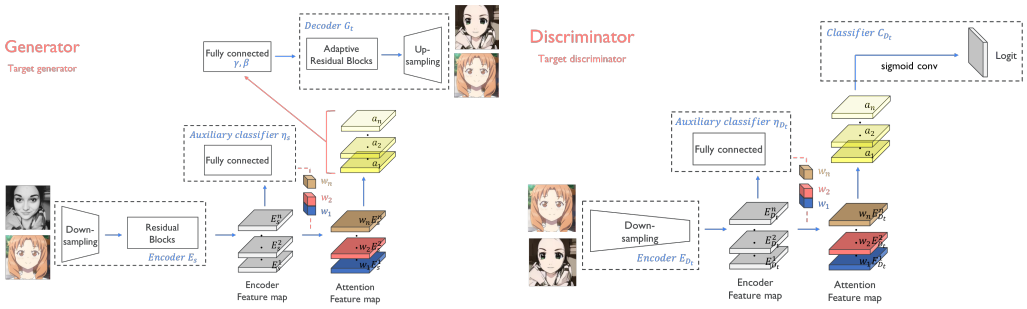
When $\alpha = 0$ the model tries to reconstruct the content image and when $\alpha = 1$ it stylizes the images most. As shown in Fig. 19, a transition between content and style consistency can be observed by changing α from 0 to 1.

5. Transfer Human Face into Anime Face

When it comes to style transfer for human face, we are interested in transferring our human face into anime face. The algorithm for image-to-image translation proposed by Kim et al. [7] helps to implement this. It incorporates a new attention module and a new learnable normalization function in an end-to-end manner.

5.1. Network Architecture

Fig. 4 shows the model architecture of U-GAT-IT. The Generator consists of a down-sampling encoder, a ResBlock bottleneck, a CAM generator [12], a AdaResBlock bottleneck and a up-sampling decoder, where the ResBlock applies Instance Normalization [11] and AdaResBlock applies Adaptive Layer-Instance Normalization. Table 2 includes a detailed description of the generator architecture.



(a) Generator (b) Discriminator

Figure 4. The model architecture of U-GAT-IT.

The discriminator consists a local discriminator (Table 3) and a global discriminator (Table 4). The major difference between them is that the size of the feature maps in the last layer in global discriminator achieves at $\frac{h}{32} \times \frac{w}{32}$ while that of the local one only achieves at $\frac{h}{8} \times \frac{w}{8}$.

5.2. Adaptive Layer-Instance Normalization

The Adaptive Layer-Instance Normalization (AdaIN) is applied in the residual blocks in the decoder bottleneck of the generator.

$$\begin{gathered}\mathrm{AdaLIN}(a, \gamma, \beta) = \gamma \cdot (\rho \cdot \hat{a}_I + (1-\rho) \cdot \hat{a_L}) + \beta \\ \hat{a_I} = \frac{a-\mu_I}{\sqrt{\sigma_I^2+\epsilon}}, \; \hat{a_L} = \frac{a-\mu_L}{\sqrt{\sigma_L^2+\epsilon}}, \\ \rho \leftarrow \mathrm{clip}_{[0,1]}(\rho - \tau \Delta\rho)\end{gathered}$$

where μ_I, μ_L and σ_I, σ_L are channel-wise, layer-wise mean and standard deviation respectively, τ is the learning rate, ρ is a learnable parameter and $\Delta\rho$ is the gradient determined by the optimizer, and γ and β are parameters generated by a fully connected layer.

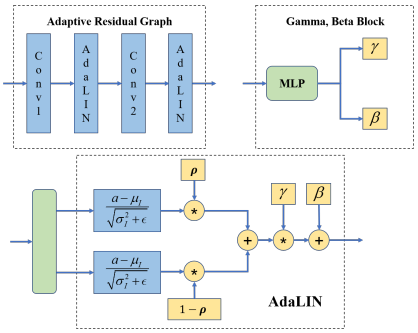


Figure 5. AdaLIN Operations

5.3. Loss Functions

The model comprises four loss functions. Here $G_{s\to t}$ maps images from a source domain X_s to a target domain X_t while $G_{t\to s}$ maps in a reverse direction, D_t is the discriminator.

Adversarial loss.

$$L_{lsgan}^{s\to t} = \mathbb{E}_{x\sim X_t}\left[D_t(x)^2\right] + \mathbb{E}_{x\sim X_s}\left[\left(1 - D_t\left(G_{s\to t}(x)\right)\right)^2\right]$$

Cycle loss.

$$L_{cycle}^{s\to t} = \mathbb{E}_{x\sim X_s}\left[|x - G_{t\to s}(G_{s\to t}(x))|_1\right]$$

Identity loss.

$$L_{identity}^{s\to t} = \mathbb{E}_{x\sim X_t}\left[|x - G_{s\to t}(x)|_1\right]$$

CAM loss

$$L_{cam}^{s\to t} = -\left(\mathbb{E}_{x\sim X_s}\left[\log\left(\eta_s(x)\right)\right] + \mathbb{E}_{x\sim X_t}\left[\log\left(1 - \eta_s(x)\right)\right]\right)$$

$$L_{cam}^{D_t} = \mathbb{E}_{x\sim X_t}\left[\eta_{D_t}(x)^2\right] + \mathbb{E}_{x\sim X_s}\left[\left(1 - \eta_{D_t}\left(G_{s\to t}(x)\right)\right)^2\right]$$

Full loss. We jointly train the encoders, decoders, discriminators, and auxiliary classifiers to optimize the final objective:

$$\min_{G_{s\to t}, G_{t\to a}} \max_{D_a, D_t} \lambda_1 L_{lsgan} + \lambda_2 L_{cycle} + \lambda_3 L_{identity} + \lambda_4 L_{cam}$$

where $\lambda_1 = 1, \lambda_2 = 10, \lambda_3 = 10, \lambda_4 = 1000$, $L_{lsgan} = L_{lsgan}^{s\to t} + L_{lsgan}^{t\to s}$. Other losses are defined in a similar way.

5.4. Experiments

We reconstruct the network structure and reuse the official data utils. We train our models on a Linux (Ubuntu 16.04), with a Tesla P40 and 24GB memory. Our implementation is based on Python, Pytorch and torchvision. We train style transfer models using the official released selfie2anime dataset [6]. We resize each of the images to 286×286 and randomly crop down images with size 256×256 and train with a batch size 1 for 100,000 epochs.

Fig. 20, 21 show some examples of good and bad results on the test image. The first row is the source faces, the third row is the reconstruction of the source faces, the 5th row is the anime faces generated by the source face and the last row is the generated faces from anime faces. The rows between them is the respective heat maps. As we can see from the examples, the model works well to transfer some faces into anime style, while the reconstruction of human faces from anime faces is not pretty good.

References

[1] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629*, 2016.

[2] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style, 2015.

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[4] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

[5] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[6] Junho Kim, Minjae Kim, Hyeonwoo Kang, and Kwanghee Lee. Selfie2anime dataset. https://drive.google.com/file/d/1xOWj1UVgp6NKMT3HbPhBbtq2A4EDkghF/view?usp=sharing, 2019.

[7] Junho Kim, Minjae Kim, Hyeonwoo Kang, and Kwanghee Lee. U-gat-it: unsupervised generative attentional networks with adaptive layer-instance normalization for image-to-image translation. *arXiv preprint arXiv:1907.10830*, 2019.

[8] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[9] K. Nichol. Painter by numbers, wikiart. https://www.kaggle.com/c/painter-by-numbers/, 2016.

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

[11] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[12] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

Appendices

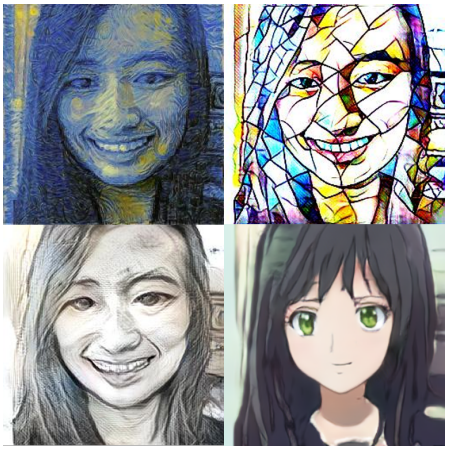


Figure 6. Results of different methods. The results are image-based, fast transfer, arbitrary transfer and U-GAT-IT in order.

A. Image-Based Neural Style Transfer

A.1. Artistic Style Transfer

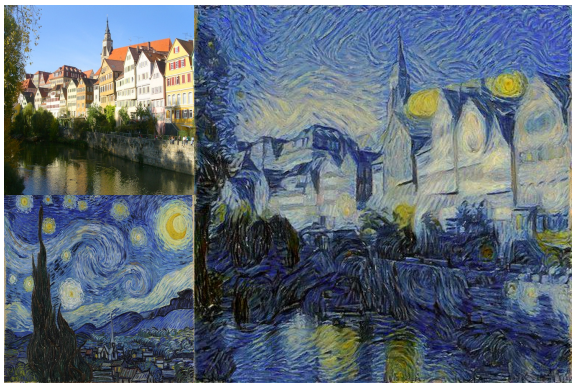


Figure 7. Starry Night

Figure 8. Starry Weeping

Figure 9. Berkeley Cubism

A.2. Daytime Transfer

Figure 10. Sunset to Nnight

A.3. Style Mixture

Here, we try to combine two style images together, i.e. Vangogh and Cubism.

Figure 11. Vangogh and Cubism

Figure 12. Stylize Result

B. Fast Neural Style Transfer

B.1. Image transform net architecture

Table 1. Image transform net architecture.

Layer	Activation size
Input	$3 \times 256 \times 256$
Reflection Padding (40×40)	$3 \times 336 \times 336$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 336 \times 336$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 168 \times 168$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 84 \times 84$
Residual block, 128 filters	$128 \times 80 \times 80$
Residual block, 128 filters	$128 \times 76 \times 76$
Residual block, 128 filters	$128 \times 72 \times 72$
Residual block, 128 filters	$128 \times 68 \times 68$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride $1/2$	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride $1/2$	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

B.2. Image Style Transfer

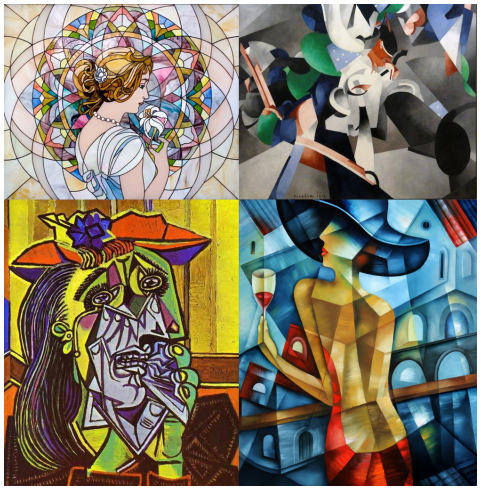


Figure 13. Style Images

Figure 14. Blond Hair Girl

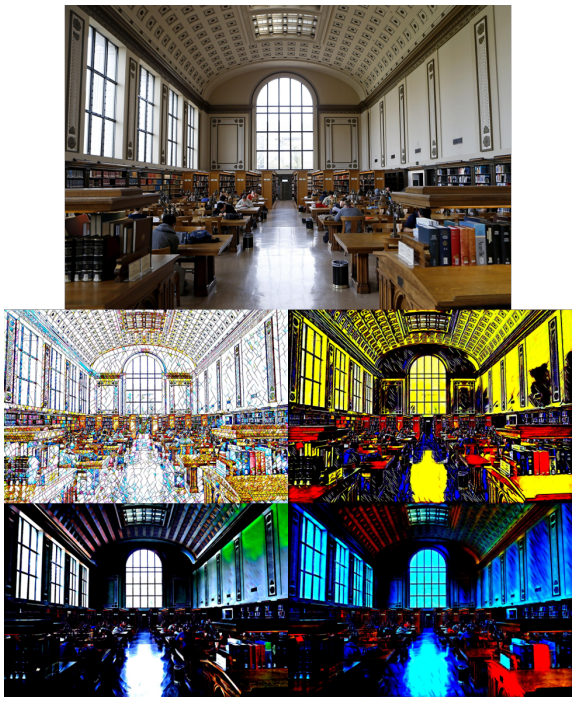


Figure 15. Berkeley Library

B.3. Video Style Transfer

The video can be found here.

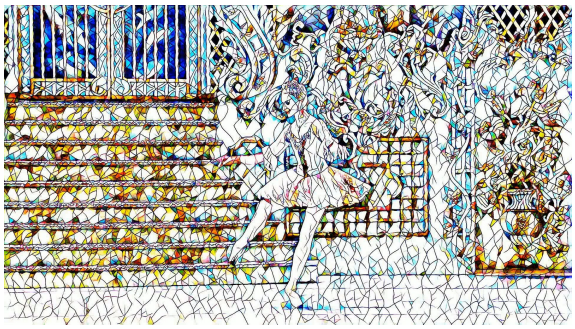


Figure 16. Ballet Video

Figure 17. Blond Hair Girl

Figure 18. Golden Gate

(a) $\alpha = 0$ (b) $\alpha = 0.25$ (c) $\alpha = 0.5$ (d) $\alpha = 0.75$ (e) $\alpha = 1$ (f) Style Image

Figure 19. Content-style trade-off.

Table 2. This detail of generator architecture.

Part	Input $\rightarrow$ Output Shape	Layer Information
Encoder Down-sampling	$(\mathrm{h}, \mathrm{w}, 3) \rightarrow (\mathrm{h}, \mathrm{w}, 64)$	$\mathrm{CONV}-(\mathrm{N64}, \mathrm{K7}, \mathrm{S1}, \mathrm{P3}), \mathrm{IN}, \mathrm{ReLU}$
	$(\mathrm{h}, \mathrm{w}, 64) \rightarrow \left(\frac{h}{2}, \frac{w}{2}, 128\right)$	$\mathrm{CONV}-(\mathrm{N128}, \mathrm{K3}, \mathrm{S2}, \mathrm{Pl}), \mathrm{IN}, \mathrm{ReLU}$
	$\left(\frac{h}{2}, \frac{w}{2}, 128\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	CONV-(N256, K3, S2, P1), IN, ReLU
Encoder Bottleneck	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	ResBlock-(N256, K3, S1, P1), IN, ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	ResBlock-(N256, K3, S1, P1), IN, ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	ResBlock-(N256, K3, S1, P1), IN, ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	ResBlock-(N256, K3, S1, P1), IN, ReLU
CAM of Generator	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 512\right)$ $\left(\frac{h}{4}, \frac{w}{4}, 512\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	Global Average & Max Pooling, MLP-(N1), Multiply the weights of MLP
γ, β	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow (1, 1, 256)$	MLP-(N256), ReLU
	$(1, 1, 256) \rightarrow (1, 1, 256)$	MLP-(N256), ReLU
	$(1, 1, 256) \rightarrow (1, 1, 256)$	MLP-(N256), ReLU
Decoder Bottleneck	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	AdaResBlock-(N256, K3, S1, P1), AdaILN, ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	AdaResBlock-(N256, K3, S1, P1), AdaILN, ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	AdaResBlock-(N256, K3, S1, P1), AdaILN, ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 256\right)$	AdaResBlock-(N256, K3, S1, P1), AdaILN, ReLU
Decoder Up-sampling	$\left(\frac{h}{4}, \frac{w}{4}, 256\right) \rightarrow \left(\frac{h}{2}, \frac{w}{2}, 128\right)$	Up-CONV-(N128, K3, S1, P1), LIN, ReLU
	$\left(\frac{h}{2}, \frac{w}{2}, 128\right) \rightarrow (\mathrm{h}, \mathrm{w}, 64)$	Up-CONV-(N64, K3, S1, P1), LIN, ReLU
	$(\mathrm{h}, \mathrm{w}, 64) \rightarrow (\mathrm{h}, \mathrm{w}, 3)$	CONV-(N3, K7, S1, P3), Tanh

Table 3. The detail of local discriminator.

Part	Input $\rightarrow$ Output Shape	Layer Information
Encoder Down-sampling	$(\mathrm{h}, \mathrm{w}, 3) \rightarrow \left(\frac{h}{2}, \frac{w}{2}, 64\right)$	CONV-(N64, K4, S2, P1), SN, Leaky-ReLU
	$\left(\frac{h}{2}, \frac{w}{2}, 64\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 128\right)$	CONV-(N128, K4, S2, P1), SN, Leaky-ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 128\right) \rightarrow \left(\frac{h}{8}, \frac{w}{8}, 256\right)$	CONV-(N256, K4, S2, P1), SN, Leaky-ReLU
	$\left(\frac{h}{8}, \frac{w}{8}, 256\right) \rightarrow \left(\frac{h}{8}, \frac{w}{8}, 512\right)$	CONV-(N512, K4, S1, P1), SN, Leaky-ReLU
CAM of Discriminator	$\left(\frac{h}{8}, \frac{w}{8}, 512\right) \rightarrow \left(\frac{h}{8}, \frac{w}{8}, 1024\right)$ $\left(\frac{h}{8}, \frac{w}{8}, 1024\right) \rightarrow \left(\frac{h}{8}, \frac{w}{8}, 512\right)$	Global Average & Max Pooling, MLP-(N1), Multiply the weights of MLP
Classifier	$\left(\frac{h}{8}, \frac{w}{8}, 512\right) \rightarrow \left(\frac{h}{8}, \frac{w}{8}, 1\right)$	CONV-(N512, K1, S1), Leaky-ReLU

Table 4. The detail of global discriminator.

Part	Input $\rightarrow$ Output Shape	Layer Information
Encoder Down-sampling	$(\mathrm{h}, \mathrm{w}, 3) \rightarrow \left(\frac{h}{2}, \frac{w}{2}, 64\right)$	CONV- $(\mathrm{N64}, \mathrm{K4}, \mathrm{S2}, \mathrm{Pl})$, SN, Leaky-ReLU
	$\left(\frac{h}{2}, \frac{w}{2}, 64\right) \rightarrow \left(\frac{h}{4}, \frac{w}{4}, 128\right)$	CONV- $(\mathrm{N128}, \mathrm{K4}, \mathrm{S2}, \mathrm{P1})$, SN, Leaky-ReLU
	$\left(\frac{h}{4}, \frac{w}{4}, 128\right) \rightarrow \left(\frac{h}{8}, \frac{w}{8}, 256\right)$	CONV- $(\mathrm{N256}, \mathrm{K4}, \mathrm{S2}, \mathrm{P1})$, SN, Leaky-ReLU
	$\left(\frac{h}{8}, \frac{w}{8}, 256\right) \rightarrow \left(\frac{h}{16}, \frac{w}{16}, 512\right)$	CONV- $(\mathrm{N512}, \mathrm{K4}, \mathrm{S2}, \mathrm{P1})$, SN, Leaky-ReLU
	$\left(\frac{h}{16}, \frac{w}{16}, 512\right) \rightarrow \left(\frac{h}{32}, \frac{w}{32}, 1024\right)$	CONV-(N1024, K4, S2, P1), SN, Leaky-ReLU
	$\left(\frac{h}{32}, \frac{w}{32}, 1024\right) \rightarrow \left(\frac{h}{32}, \frac{w}{32}, 2048\right)$	CONV-(N2048, K4, S1, P1), SN, Leaky-ReLU
CAM of Discriminator	$\left(\frac{h}{32}, \frac{w}{32}, 2048\right) \rightarrow \left(\frac{h}{32}, \frac{w}{32}, 4096\right)$ $\left(\frac{h}{32}, \frac{w}{32}, 4096\right) \rightarrow \left(\frac{h}{32}, \frac{w}{32}, 2048\right)$	Global Average & Max Pooling, MLP-(N1), Multiply the weights of MLP
Classifier	$\left(\frac{h}{32}, \frac{w}{32}, 2048\right) \rightarrow \left(\frac{h}{32}, \frac{w}{32}, 1\right)$	CONV-(N2048, K1, S1), Leaky-ReLU

Figure 20. Good examples from selfie to anime

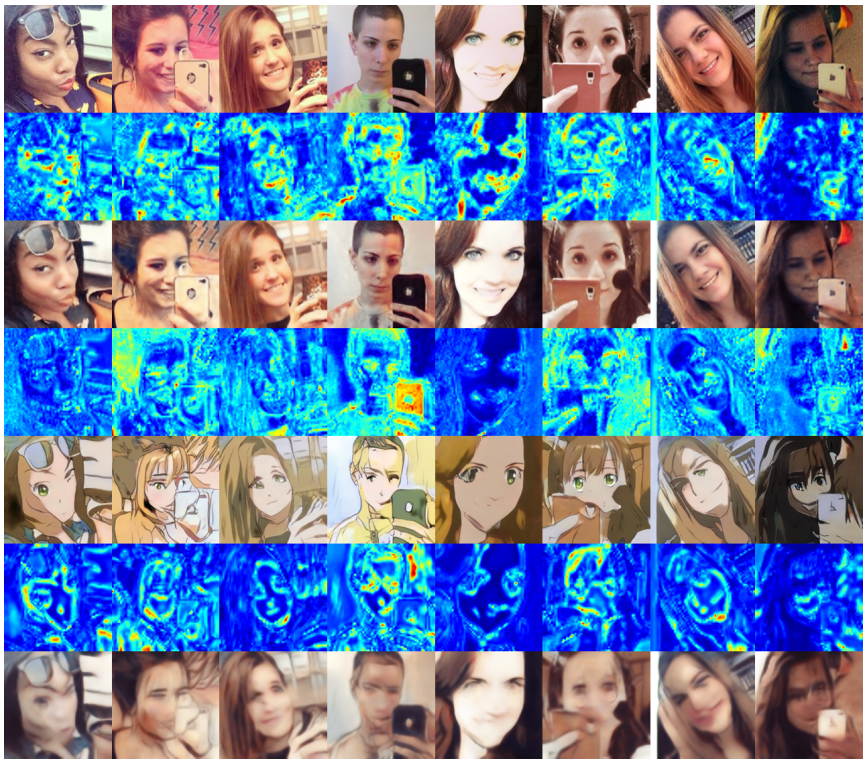


Figure 21. Bad examples from selfie to anime