A Literature Review of Neural Style Transfer

Haochen Li
Princeton Univeristy
Princeton University, Princeton NJ 08544
haochenl@princeton.edu

Abstract

Neural Style Transfer is the problem of taking a content image and a style image as input, and outputting an image that has the content of the content image and the style of the style image. The key technique that makes neural style transfer possible is convolutional neural network(CNN). This paper will first survey major techniques of doing neural style transfer on images, and then briefly examine one way of extending neural style transfer to videos.

1. Introduction

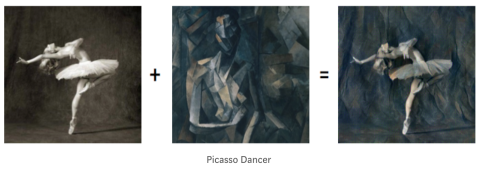


Figure 1: Example of Neural Style Transfer[3]

In order to combine the content of the content image and the style of the style image, we have to find ways to independently represent the semantic content of an image and the style in which the content is presented. Thanks to the recent advances in deep CNN [10], we are able to tackle this challenge with great success. By tackling this challenge, the field of neural style transfer "provide new insights into the deep image representations learned by Convolutional Neural Networks and demonstrate their potential for high level image synthesis and manipulation" [5].

In this paper, we will first survey several techniques for neural style transfer. We will then briefly look at one way to extend neural style transfer to videos. One major issue of doing neural style transfer on videos is the lack of temporal coherence between frames and we will give a high level overview of techniques from [6] to overcome this challenge.

We will also cover the datasets used to train deep CNNs to perform neural style transfer and evaluation metrics for this task.

2. Neural Style Transfer On Images

2.1. A Neural Algorithm of Artistic Style

Gatys et al proposed the first algorithm [5] that worked really well for the task of neural style transfer and we will look at this algorithm in detail in this section. In this algorithm, a VGG-16 architecture [15] pretrained on ImageNet [2] is used to extract the features that represent semantic content and style.

2.1.1 Content

A given image $\hat{x}$ is encoded in each layer of a CNN by that layer's filter responses to that image. A layer with N_l distinct filters has N_l feature maps each of size M_l, where M_l is the height times the width of the feature map. Therefore, the response in a layer l can be stored in a matrix $F_l \in R^{N_l \times M_l}$, where F^l_{ij} is the activation of ith filter at position j at layer l.

The F_ls can be used to represent the content of the image. The filter response of lower layers of VGG net(meaning layers closer to the input) look very close to the input image, while the filter representation of the the higher layers of VGG net captures the high level content information(e.g. objects in the scene, the relative positions of objects) and largely ignores the pixel-by-pixel information of the input image. Figure 2 provides a very good illustration of this phenomenon.

Let $\hat{o}$ and $\hat{g}$ be the original image and the image that is generated respectively, and O^l and G^l be their respective filters at layer l. The contribution of each layler l to the total content loss is:

$$C_l(\hat{o}, \hat{g}) = \frac{1}{2} \sum_{i,j} (O^l_{ij} - G^l_{ij}) \quad (1)$$

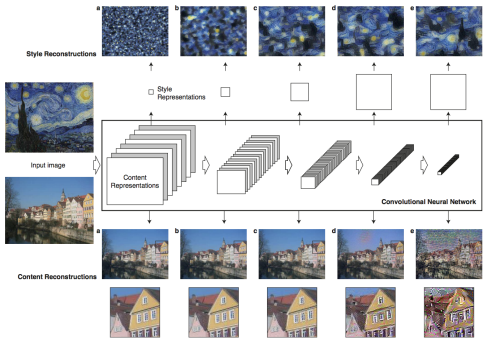


Figure 2: The Representation of Content and Style by High and Low Layers of Neural Network[5]

and the total content loss between $\hat{o}$ and $\hat{g}$ is:

$$L_{content}(\hat{o}, \hat{g}) = \sum_{l=0}^{L} c_l C_l(\hat{o}, \hat{g}) \tag{2}$$

where c_l is a hyperparameter that specifies the weighting factor of the contribution of each layer(note that some c_ls could be 0, indicating that we don't use the filter response of that layer).

2.1.2 Style

The style of an image is captured by the correlations between the different filter responses. The feature correlations are given by the Gram Matrix $G_l \in R^{N_l \times N_l}$, where G^l_{ij} is the inner product between the vectorized feature maps i and j at layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk} \tag{3}$$

By including feature correlations of multiple layers, we capture the style information of the image while ignoring the content information(e.g. objects present in scenes, global arrangements of objects).

Let $\hat{o}$ and $\hat{g}$ be the original image and the image that is generated respectively, let O_l and G_l be their style representations in layer l respectively, the contribution of layer l to the total style loss is($N_l \times M_l$ is the size of the filter at layer l):

$$S_l(\hat{o}, \hat{g}) = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (O^l_{ij} - G^l_{ij})^2 \tag{4}$$

and the total style loss between $\hat{o}$ and $\hat{g}$ is:

$$L_{style}(\hat{o}, \hat{g}) = \sum_{l=0}^{L} s_l S_l(\hat{o}, \hat{g}) \tag{5}$$

where s_l is a hyperparameter that specifies the weighting factor of the contribution of each layer(note that some s_ls could be 0, indicating that we don't use the filter response of that layer).

2.1.3 Style Transfer Algorithm

In deep learning, we normally optimize the parameters of some neural network. However, to transfer the style of an artwork $\hat{a}$ onto a photograph $\hat{p}$, we repeatly optimize the pixel values of the constructed image, $\hat{x}$, so that $\hat{x}$ simultaneously matches the style representation of $\hat{a}$ and the content representation of $\hat{p}$. At the beginning, $\hat{x}$ is initialized to random noise. In order to get a good stylized output, $\hat{x}$, we jointly minimize the content loss and the style loss. The final loss function we minimize is:

$$L_{total}(\hat{p}, \hat{a}, \hat{x}) = \alpha L_{content} + \beta L_{style} \tag{6}$$

where α and β are real numbers(they are hyperparamters to be set). A large $\frac{\alpha}{\beta}$ ratio means that we want to emphasize content of the photograph in the constructed image $\hat{x}$, while a small $\frac{\alpha}{\beta}$ means that we want to emphasize the style of the artwork in the constructed image $\hat{x}$.

Figure 3 clearly illustrates the style transfer algorithm.

Here is a very good description of the algorithm from [5] that accompanies figure 3: "First content and style features are extracted and stored. The style image $\vec{a}$ is passed through the network and its style representation A^l on all layers included are computed and stored (left). The content image $\vec{p}$ is passed through the network and the content representation P^l in one layer is stored (right). Then a random white noise image $\vec{x}$ is passed through the network and its style features G^l and content features F^l are computed. On each layer included in the style representation, the element-wise mean squared difference between G^l and A^l is computed to give the style loss L_{style} (left). Also the mean squared difference between F^l and P^l is computed to give the content loss $L_{content}$ (right). The total loss L_{total} is then a linear combination between the content and the style loss. Its derivative with respect to the pixel values can be computed using error back-propagation (middle). This gradient is used to iteratively update the image $\vec{x}$ until it simultaneously matches the style features of the style image $\vec{a}$ and the content features of the content image $\vec{p}$ (middle, bottom)". The inputs to this network are the style target, content target, and the initialized image.

Figure 5(c) shows an example output of this algorithm.

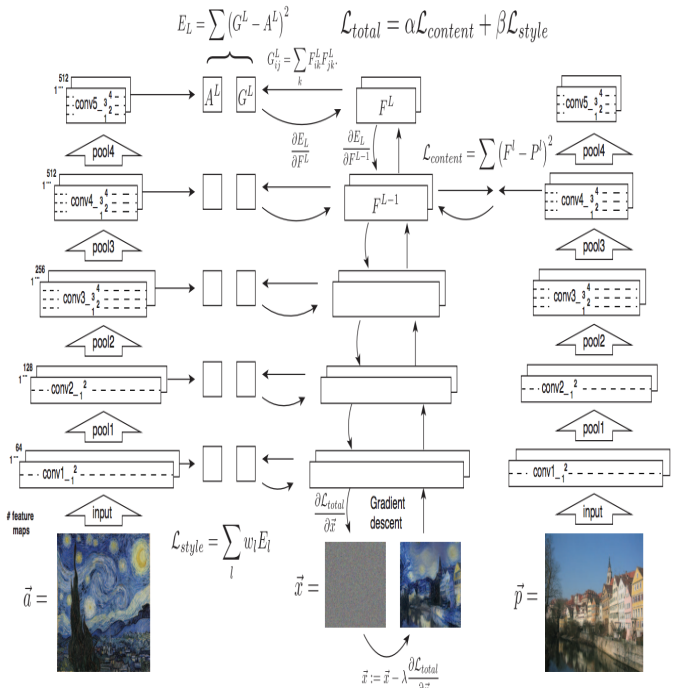


Figure 3: Style Transfer Algorithm[5]

Although this algorithm outputs images of very high quality, this algorithm takes 3-5 minutes to generate an output every single time. If one wants to generate a lot of stylized images, this algorithm will take a lot of time. We will look at another approach to neural style transfer that can achieve real-time style transfer after some training.

2.2. Perceptual Losses for Real-Time Style Transfer and Super-Resolution

In this paper [8], Johnson et al. trained a feedforward convolutional neural network in a supervised manner in order to achieve real time style transfer.

2.2.1 Dataset

The dataset used to train the feedforward convolutional neural network is the MS COCO dataset [12]. This dataset contains about 80K images of complex everyday scenes containing common objects in their natural context. This dataset is used as the content images.

2.2.2 Architecture

In this algorithm, the input image, x, which is the same as the content image, is passed through the Image Transform Net. The output of the the Image Transform Net, $\hat{y}$ is passed throught the VGG-16 [15] net along with the style target and

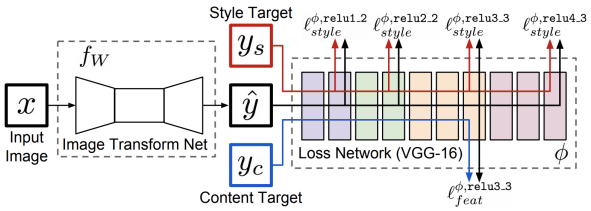


Figure 4: Feedforward Network[8]

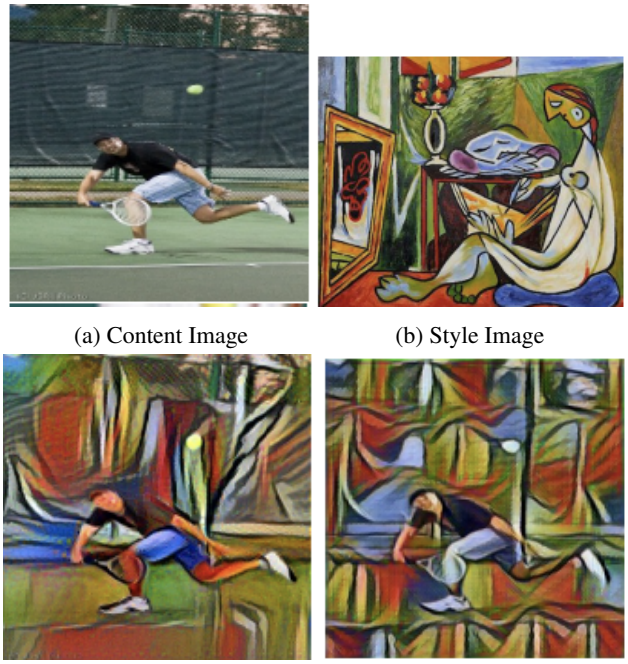
content target. Note that during training, the style target is fixed, and the content target is the same as the input image x. The content loss and the style loss is then calculated in the same way as the algorithm described in section 2.1(using a VGG net[15]). In this algorithm, we are optimizing the parameters of the Image Transform Net. It takes about 4-6 hours to train the Image Transform Net, depending on the hardware you are using. The input to the Image Transform Net after it is trained is simply the content image you want to use. We can use the Image Transform Net to transfer the style of the style target used to train this network onto any content image in real time. The quality of images obtained using this algorithm is comparable to the algorithm described in section 2.2(see Figure 5). Please refer to [8] for the specific architecture of the Image Transform Net.

The main drawback of this algorithm is that each network is tied to a single style. If we want to perform style transfer for multiple styles, we have to train a separate network using each style. Not only is this extremely inefficient, it is impractical in many situations. For example, in mobile applications, there is not enough memory in mobile phones to store the parameters of many separate Image Transform Networks. In order to address this drawback, we will next look at algorithms that can perform style transfer for multiple styles using only one network in real time.

can we fix style hyperparams earlier than actual i/p

2.3. Mix Fast Style Transfer

Mix fast style transfer is the task of performing real-time style transfer for multiple styles using one network. In this section we will look at two techniques that can achieve mix fast style transfer. Both techniques stem from the intuition that "many styles probably share some degree of computation, and that this sharing is thrown away by training N networks from scratch when building an N-styles style transfer system." [4]. Both techniques propose a conditional style transfer network, where the network is given a content image and the identity of style to apply to produce a corresponding stylized image. The difference between the two techniques lies in how they build the conditional network.



(a) Content Image (b) Style Image

(c) Gatys et al (d) Feedforward Network

Figure 5: Example Results Produced by Gatys et al's Algorithm and Feedforward Network[8]

2.3.1 Conditional Instance Normalization

This paper [4] proposes the conditional instance normalization technique, which is an extension to the instance normalization technique [16]. The goal fo the procedure is to transform a layer's filter response, F, into a normalized filter response, F_N, specific to a painting style s. The key insight of this technique is: "to model a style, it is sufficient to specialize scaling and shifting parameters after normalization to each specific style." [4] That is, we can share the the convolution weights of the style transfer network across many styles, and all we need to tune are two $N \times C$ matrices, γ and β, where N is the number of styles being modeled and C is the number of output feature maps. γ is the scaling matrix and β is the shifting matrix here, and γ_i and β_i represent the parameters of $style_i$. Conditioning on one style is achieved by:

$$z = \gamma_s(\frac{x-u}{\sigma}) + \beta_s \tag{7}$$

where u and σ are x's mean and standard deviation taken across x's spatial axes(that is, if $x \in R^{N \times C \times W \times H}$, where N is the number of batches, C is the number of channels, W is the width, and H is the height, we take the mean and standard deviation across W and H) and γ_s, β_s are the sth row of γ and β that correspond to style s.

The input to this network is simply the content image and the network will output N stylized images, where N is the number of styles the network is trainied on.

This approach can be applied to any style transfer network, and to apply this technique, we simply replace the batch normalization layer in the original network by conditional instance normalization.

Because conditional instance normalization only acts on the scaling and shifting parameters, training a single style transfer network on N styles require far fewer parameters than training N separate networks for scratch. Furthermore, when we already have a trained style transfer network on N styles and we want to incorporate $N+1$th style into our network, we can just keep the convolutional weights of the network fixed and finetune the γ and β matrices(the dimensions of γ and β matrices need to change from $N \times C$ to $(N+1) \times C$). As shown in figure 6, finetuning is much faster than training from the scratch as finetuning for 5000 steps achieves similar qualitative results as training for 40000 steps from scratch.

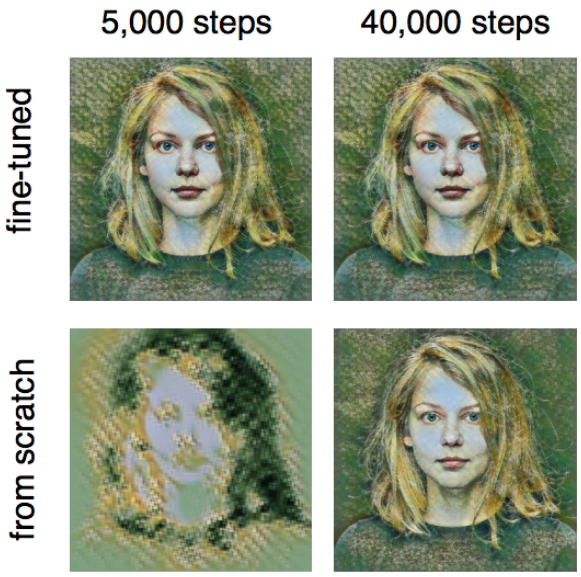


Figure 6: Training from Scratch vs Fine Tuning γ and β[4]

Another interesting feature of this algorithm is that this algorithm allows for arbitrary combination of painting styles. We combine different painting styles by calculating convex combinations of the γ and β values. Suppose (γ_1, β_1) and (γ_2, β_2) are the parameters that correspond to two different styles. If we want to stylize a content image using the combination of two different styles, we will calculate $\gamma_{1+2} = \alpha \times \gamma_1 + (1-\alpha) \times \gamma_2$ and $\beta_{1+2} = \alpha \times \beta_1 + (1-\alpha) \times \beta_2$. α is a hyperparameter to be tuned based on how much you want to emphasize ei-

ther style.

Figure 7 shows some example outputs of this technique. Note that all the different stylized images are produced by a single network.

Figure 7: Mixed Style Transfer Using Conditional Instance Normalization [4]

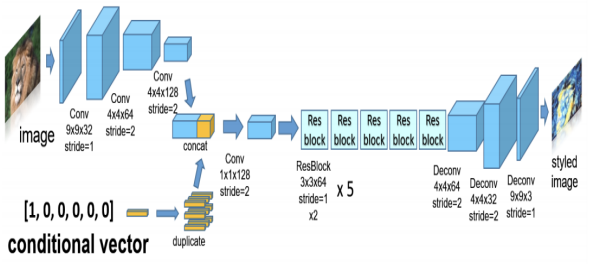


Figure 8: Fusion Network [9]

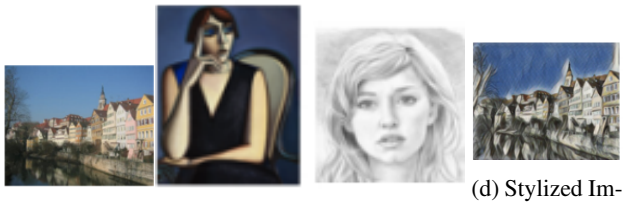
2.3.2 Mix Fast Style Transfer Via Fusion

The conditional instance normalization approach is efficient and produces very good outputs. However, conditional instance normalization layer needs to be manually implemented in most of today's popular deep-learning frameworks. This paper [9] proposes an approach for Mix Fast Style Transfer that is simpler to implement that the conditional normalization approach.

This algorithm uses a similar structure as Johnson's feed-forward neural network. There is an Image Transform Network followed by a VGG-16 network [15]. Figure 8 is the Image Transform Net used in this algorithm. During training, a minibatch is formed with one content image and all N style images the network is associated with. To train the network, we present the network with a content image and a N-dimensional binary vector where the corresponding element to each style is 1 and the other elements are 0. That is, if we are presenting the network the content image and style 1, the first element of the binary vector should be 1 and all other elements of the binary vector should be 0. The fusion step simply concatenates the conditional vector and the filter response of the content image. The output of the Image Transform Network, along with the content target and style target, is being passed to a VGG-16 network [15] to calculate the content loss and style loss in the same way as in the algorithm described in section 2.1.

There is no need to manually implement a special layer if we are using one of the modern deep-learning frameworks. Furthermore, it is also easier to combine different styles in this approach. If the network is trained on 4 different styles, to combine the 4 styles, we simply present the network with a content image and a 4-dimensional vector where each element is nonzero(e.g. $[0.1, 0.2, 0.3, .0.4]$, the specific value of each element of the vector depends on how much you want to emphasize each style). However, in order to incoporate additional elements into this style transfer network, we have to train the network from scratch instead of finetuning like we did in section 2.3.1. This technique's outputs are qualitatively similar to the one presented in section 2.3.1.

Instead of showing stylized images using any one particular style, we will show an example of stylizing an image using a mixture of two styles(Figure 9).



(a) Content Image (b) Style Image1 (c) Style Image2 (d) Stylized Image using Style 1 and 2

Figure 9: Stylizing an Image Using a Mixture of Two Styles[9]

2.4. Universal Fast Style Transfer

All the real-time style transfer techniques presented so far can only do style transfer for painting styles that the network has seen before, and we will now present several techniques that can train a network to perform style transfer for any painting style(including the ones the network has not seen before) in real time.

2.4.1 Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization

This paper [7] introduces a technique known as adaptive instance normalization(AdaIN). The insight of this method is that feature statistics of generated image can control the style of the generated image. In conditional instance normalization, the specific styles are specificied using a scaling matrix γ and a shifting matrix β. Both scaling and shifting are affine transformations, and we will call γ and β affine parameters from now on. The high level idea of AdaIN is to find ways of coming up affine parameters for arbitrary styles.

A transformation that preserves lines and parallelism (maps parallel lines to parallel lines) is an affine transformation.

Unlike conditional instance normalization, AdaIN has no learnable affine parameters. Instead, AdaIN receives a content image c and a style image s and matches the channel-wise mean and variance of c with those of s:

$$AdaIN(c, s) = \sigma(s)(\frac{c - u(c)}{\sigma(c)}) + u(s) \tag{8}$$

where $\sigma(x)$ and $u(x)$ denote the standard deviation and mean of x across x's spatial axes.

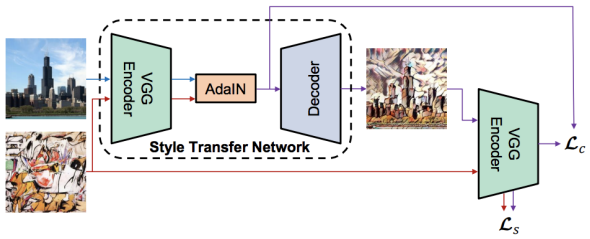


Figure 10: AdaIN Style Transfer Architecture[7]

Figure 10 illustrates the architecture of AdaIN style transfer network. The style transfer network T takes a content image and an arbitrary style image as inputs and stylizes the content image using the style of the style image. The network adopts an encoder-decoder architecture, in which the encoder f is fixed to first few layers(up to $relu4_1$) of a VGG-19 [15] that is pretrained on Imagenet [2]. After the encoder encodes the content image and the style image as $f(c)$ and $f(s)$ respectively, we pass them to AdaIn and get:

$$t = \text{AdaIn}(f(c), f(s)) \tag{9}$$

A randomly initialized decoder network is trained to generate the stylized image. The decoder network is mostly the same as the encoder network, with all pooling layer replaced by nearest up-sampling to reduce checkerboard effects.

During training, the MS-COCO dataset [12] is used as the content images and a set of paintings mostly collected from WikiArt [14] is used as the style images. Both datasets contain about 80000 images. To train the decoder network, equation (6) is used as the loss function and the style and content loss are calculated using a VGG-19 [15] that is pretrained on Imagenet [2] in the same way as the algorithm described in section 2.1. In this case, the content target is the output of AdaIN, t(we don't use the actual content image as content target because in practice, using t as content target leads to faster convergence speed), the style target is the style image, and the generated image is the output of the decoder network.

Figure 11 shows an example output of AdaIN. Note that the network has not seen the particular style shown in Figure 11.

(a) Content Image (b) Style Image (c) Style Transfer

Figure 11: Results of AdaIN [7]

2.4.2 Universal Fast Style Transfer Via Fusion

This technique [17] is an extension of the technique presented in section 2.3.2.

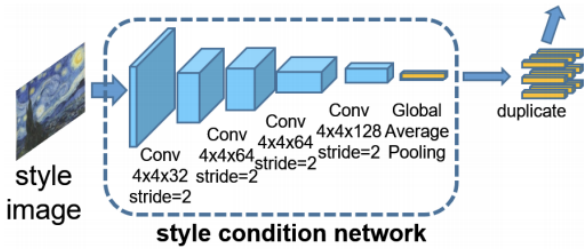


Figure 12: Style Conditional Network[17]

In the algorithm presented in section 2.3.2, we supply the conditional vectors as input. In this algorithm, as illustrated in figure 12, there is an additional style conditional network. We pass in a style image as input and the style conditional network would extract a conditional vector. The rest of architecture is the same as before.

In this algorithm, we train the style conditional network along with the rest of architecture. The goal of training the style conditional network is to teach it to generate the right conditional vector for arbitrary styles. During training, the MS-COCO dataset [12] is used as the content images and a set of paintings mostly collected from WikiArt [14] is used as the style images.

This method achieves qualitatively similar results as the method described in section 2.4.2.

2.5. Photorealistic Style Transfer

The algorithms we surveyed in previous sections give very cool artistic outputs. However, other than for enter-

tainment purposes, it is not clear how neural style transfer is applicable. Photorealistic style transfer is the same problem as neural style transfer, except that when both the style images and content images are photos, the stylized image should look photo realistic. The solution to photo realistic style transfer is very applicable for photo editing. For example, photorealistic style transfer can be used to alter the time of the day or weather of a picture. An output of the algorithm described in section 2.1 and an output photorealisitic algorithm is shown in figure 13. Other neural style transfer algorithms explored do not produce qualitatively different results from the algorithm described in section 2.1. As one can see, the existing style transfer techniques are not suitable for photorealistic style transfer. Even when both the content image and the style image are photos, the output of regular style transfer algorithms still contains distortions that are reminiscent of paintings(e.g. the spill in the sky). The photorealistic style transfer successfully stylizes the daytime photo so that it looks like it has been taken at night.

(a) Content Image (b) Style Image

(c) Style Transfer (d) Photorealistic Style Transfer

Figure 13: Comparison Between Regular Style Transfer and Photorealistic Style Transfer [13]

2.5.1 Key Challenges

There are two key challenges present in this problem. The first one has to do with two competing objectives we are trying to accomplish. On one hand, we are trying to achieve very drastic local effects(Light up the window of the skyscraper in figure 13). On the other hand, we do not want the transfer to have any geometric effect(e.g. the grids of windows stay as grids and don't get distorted). This work [13] proposes the first solution to this challenge.

Another challenge is raised by the complexity of scenes in the real world. "The transfer should respect the semantics of the scene. For instance, in a cityscape, the appearance of buildings should be matched to buildings, and sky to sky" [13].

2.5.2 Algorithm

Recall that in the algorithm described in section 2.1, there are two terms in the loss function, namely content loss and style loss. This paper [13] has two core technical contributions.

The first is the introduction of a photorealism regularization term into the loss function. The insight is that the input content image is already photorealistic, all we need to do is to ensure that we do not lose the photorealism during the style transfer process. This is accomplished by adding a term to equation (6) that penalizes image distortion. This loss term is built upon the Matting Laplacian proposed by Levin et al. [11]. In [11], a least-square penalty function that penalizes image distortion is proposed:

$$L_m = \sum_{c=1}^{3} V_c[O]^T M_I V_c[O] \tag{10}$$

$V_c[O]$ is the vectorized version($N \times 1$) of the output image O in channel c. M_I is a matrix that depends only on the input image I. For the specific formula of M_I, please refer to this paper [11] and we will only present the high level idea in this literature review due to spatial constraints. The high level idea of this loss term is to penalize things that are not well explained by locally affine transformation(An affine transformation is a linear transformation that preservers points, straight lines, and planes). Output that is not well explained by locally affine transformation includes things like making a straight edge curvy.

A limitation of the style loss in equation (6) is that the Gram matrix is computed over the entire image. By calculating Gram matrix over the entire image, the Gram matrix is limited in terms of its ability to adapt to variations of semantic context and can cause "spillovers". This problem is addressed using a semantic segmentation method [1] to generate image segmentation tasks for content and style images for a set of common labels(e.g. sky, buildings, water, and etc). We add the masks to the input image as additional channels and we update the style loss as follows:

$$L_s^l = \sum_{c=1}^{C} \frac{1}{2N_{l,c}^2} \sum_{ij} (G_{l,c}[O] - G_{l,c}[S])_{ij}^2 \tag{11}$$

$$F_{l,c}[O] = F_l[O] M_{l,c}[I] \qquad F_{l,c}[S] = F_l[O] M_{l,c}[S] \tag{12}$$

where C is the number of channels in the semantic segmentation mask, $M_{l,c}[\cdot]$ denotes the channel c of the segmentation mask of input $\cdot$ in layer l, $F_{l,c}[\cdot]$ denote the the filter response of channel c of input $\cdot$ in layer l. In each layer,

there are N_l filters each with a vectorized feature map of size D_l. $G_{l,c}[\cdot]$ is the Gram matrix corresponding to $F_{l,c}[\cdot]$. $N_{l,c}$ is the number of filters in channel c of layer l.

The loss function used in photorealistic style transfer is:

$$L_{total} = \alpha L_c + \beta L_s + \gamma L_m \tag{13}$$

where α, β, and γ are the hyperparameters that regulate the content weight, style weight, and photorealism weight respectively.

Other than the modified loss function, this algorithm uses the same architecture as the algorithm presented in section 2.1. The semantic segmentation is performed using DilatedNet [1].

2.6. Evaluation Metric

Since determining quality of images is a largely subjective task, most of evaluations of neural style transfer algorithms are qualitative. The most common approach is to qualitatively compare outputs of some current approach with some previous approaches by putting outputs of different algorithms side by side. The algorithm presented in section 2.1 is often used as a baseline method for qualitative evaluation.

Another common evaluation method is user study. The typical setup is to recruit some Amazon Mechanical Turk users, show them stylized images output by different algorithms, and ask them which images they prefer.

There are also some attempts for quantitative evaluation. The most common approach is compute the runtime or the convergence speed(how long it takes the loss function to converge) of different algorithms. Some paper also try to compare the final values of loss functions of different algorithms. However, the values of loss function do not always correspond precisely to the quality of output images.

Due to spatial constraints, we have only shown a limited number of outputs from each algorithm in this literature review. If you are interested in other kinds of evaluation metrics mentioned here, please refer back to the original papers listed under Reference.

3. Style Transfer On Videos

There are some works that focus on generalizing style transfer to videos. If we just naively apply existing style transfer techniques to individual frames of videos, the resulting video does not have temporal coherence. We will briefly look at one paper [6] that solves the problem doing style transfer on videos. However, since this literature review is not focusing on style transfer on videos, we will not go into great depth on this topic.

The key technical contribution of this paper [6] is that the neural network takes consecutive frames as input instead of single frame. The neural network takes previous stylized frame p_{t-1} and the current video frame p_t as input. The output at each timestep is fed as input at the next time step. At each time step, the loss function is the same as equation (6). Between timesteps, a new term known as temporal consistency loss is introduced into the loss function to ensure that the network will produce temporally consistent results. The high level idea of temporal consistent loss is to ensure that consecutive frames of a video should be very similar to each other. We will not go into the technical details of the architecture or the temporal consistency loss term in this literature review. This method does produce temporally consistent stylized videos.

4. Discussion

From a practical point of view, neural style transfer provides people with a way to create interesting artwork. The ability to generate artwork of different styles is one type of visual intelligence. Furthermore, as shown in section 2.5, style transfer can also have important applications in photo editing.

From a technical point of view, the body of research concerning style transfer deepens our understanding of CNN's ability to represent images(e.g. Feature statistics of a generator network, such as channelwise mean and variance, can represent the style of an image). By deepending our understanding of CNN's ability to represent images, the body of research concerning style transfer has made important contribution to the study of visual intelligence.

References

[1] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] S. Desai. Neural artistic style transfer: A comprehensive look.

[4] V. Dumoulin, J. Shlens, and M. Kudlur. A learned representation for artistic style. *ICLR*, 2017.

[5] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[6] A. Gupta, J. Johnson, A. Alahi, and L. Fei-Fei. Characterizing and improving stability in neural style transfer. *ICCV*, 2017.

[7] X. Huang and S. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[8] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, 2016.

[9] R. T. Keiji Yanai. Conditional fast style transfer network. *ICMR*, 2017.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[11] A. Levin, D. Lischinski, and Y. Weiss. A closed form solution to natural image matting. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2006.

[12] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. *Computer Vision–ECCV 2014. Springer (2014)*.

[13] F. Luan, S. Paris, E. Shechtman, and K. Bala. Deep photo style transfer. *arXiv preprint arXiv:1703.07511*, 2017.

[14] K. Nichol. Painter by numbers, wikiart. 2016.

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[16] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016.

[17] K. Yanai. Unseen style transfer based on a conditional fast style transfer network. *ICLR*, 2017.