ADJUSTABLE REAL-TIME STYLE TRANSFER

Mohammad Babaeizadeh[1] and Golnaz Ghiasi[1]

[1]Google Brain

ABSTRACT

Artistic style transfer is the problem of synthesizing an image with content similar to a given image and style similar to another. Although recent feed-forward neural networks can generate stylized images in real-time, these models produce a single stylization given a pair of style/content images, and the user doesn't have control over the synthesized output. Moreover, the style transfer depends on the hyper-parameters of the model with varying "optimum" for different input images. Therefore, if the stylized output is not appealing to the user, she/he has to try multiple models or retrain one with different hyper-parameters to get a favorite stylization. In this paper, we address these issues by proposing a novel method which allows adjustment of crucial hyper-parameters, after the training and in real-time, through a set of manually adjustable parameters. These parameters enable the user to modify the synthesized outputs from the same pair of style/content images, in search of a favorite stylized image. Our quantitative and qualitative experiments indicate how adjusting these parameters is comparable to retraining the model with different hyper-parameters. We also demonstrate how these parameters can be randomized to generate results which are diverse but still very similar in style and content. The source-code of the project is available at the project website: https://goo.gl/PVWQ9K.

1 INTRODUCTION

Style transfer is a long-standing problem in computer vision with the goal of synthesizing new images by combining the *content* of one image with the *style* of another (Efros & Freeman, 2001; Hertzmann, 1998; Ashikhmin, 2001). Recently, neural style transfer techniques (Gatys et al., 2015; 2016; Johnson et al., 2016; Ghiasi et al., 2017; Li et al., 2018; 2017b) showed that the correlation between the features extracted from the trained deep neural networks is quite effective on capturing the visual styles and content that can be used for generating images *similar* in style and content. However, since the definition of similarity is inherently vague, the objective of style transfer is not well defined (Dumoulin et al., 2017) and one can imagine multiple stylized images from the same pair of content/style images.

Existing real-time style transfer methods generate only one stylization for a given content/style pair and while the stylizations of different methods usually look distinct (Sanakoyeu et al., 2018; Huang & Belongie, 2017), it is not possible to say that one stylization is better in all contexts since people react differently to images based on their background and situation. Hence, to get favored stylizations users must try different methods that is not satisfactory. It is more desirable to have a single model which can generate *diverse* results, but still *similar* in style and content, in real-time, by adjusting some input parameters.

One other issue with the current methods is their high sensitivity to the hyper-parameters. More specifically, current real-time style transfer methods minimize a weighted sum of losses from different layers of a pre-trained image classification model (Johnson et al., 2016; Huang & Belongie, 2017) (check Sec 3 for details) and different weight sets can result into very different styles (Figure 6). However, one can only observe the effect of these weights in the final stylization by retraining the model with the new set of weights. Considering the fact that the "optimal" set of weights can be different for any pair of style/content (Figure 4) and also the fact that this "optimal" truly doesn't exist (since the goodness of the output is a personal choice) retraining the models over and over until the desired result is generated is not practical.

The primary goal of this paper is to address these issues by providing a novel mechanism which allows for adjustment of the stylized image, in ***real-time*** and ***after*** training. To achieve this, we use an auxiliary network which accepts additional parameters as inputs and changes the style transfer process by adjusting the weights between multiple losses. We show that changing these parameters at inference time results to stylizations similar to the ones achievable by retraining the model with different hyper-parameters. We also show that a random selection of these parameters at run-time can generate a random stylization. These solutions, enable the end user to be in full control of how the stylized image is being formed as well as having the capability of generating multiple stochastic stylized images from a fixed pair of style/content. The stochastic nature of our proposed method is most apparent when viewing the transition between random generations. Therefore, we highly encourage the reader to check the project website https://goo.gl/PVWQ9K.

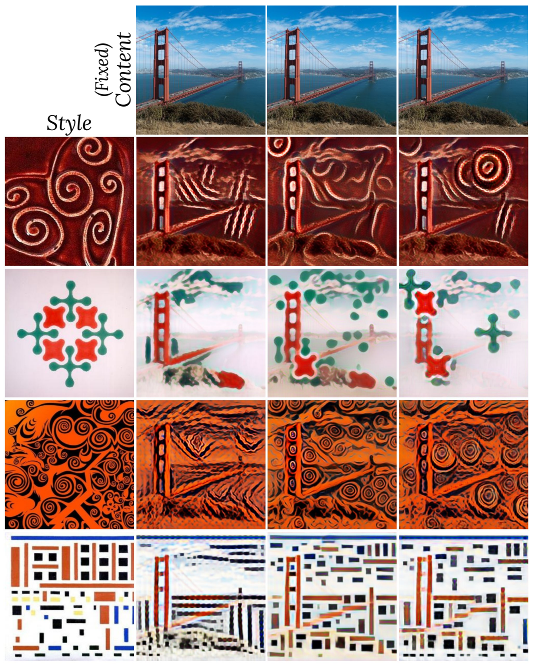

Figure 1: Adjusting the output of the synthesized stylized images in real-time without retraining. Each column shows a different stylized image for the same content and style image. Note how each row still resembles the same content and style while different in details.

2 Related Work

The strength of deep networks in style transfer was first demonstrated by Gatys et al. (Gatys et al., 2016). While this method generates impressive results, it is too slow for real-time applications due to its optimization loop. Follow up works speed up this process by training feed-forward networks that can transfer style of a single style image (Johnson et al., 2016; Ulyanov et al., 2016) or multiple styles (Dumoulin et al., 2017). Other works introduced real-time methods to transfer style of arbitrary style image to an arbitrary content image (Ghiasi et al., 2017; Huang & Belongie, 2017). Although, these methods can generate stylization for the arbitrary inputs, they can only produce one stylization for a single pair of content/style images. In the case that the user does not like the result, it is not possible to get a different result without retraining the network for a different set of hyper-parameters. Our goal in this paper is to train a single network that user can get different stylization without retraining the network.

Generating diverse results have been studied in multiple domains such as colorizations (Deshpande et al., 2017; Cao et al., 2017), image synthesis (Chen & Koltun, 2017), video prediction (Babaeizadeh et al., 2017; Lee et al., 2018), and domain transfer (Huang et al., 2018; Zhang, 2018). Domain transfer is the most similar problem to the style transfer. Although we can generate multiple outputs from a given input image (Huang et al., 2018; Zhu et al., 2017), we need a collection of target or style images for training. Therefore we cannot use it when we do not have a collection of similar styles. For instance, when we want to generate multiple stylizations for the Stary Night painting, it is hard to find different similar paintings.

Style loss function is a crucial part of style transfer which affects the output stylization significantly. The most common style loss is Gram matrix which computes the second-order statistics of the feature activations (Gatys et al., 2016), however many alternative losses have been introduced to measure distances between feature statistics of the style and stylized images such as correlation alignment loss (Peng & Saenko, 2018), histogram loss (Risser et al., 2017), and MMD loss (Li et al., 2017a). More recent work (Liu et al., 2017) has used depth similarity of style and stylized images as a part of the loss. We demonstrate the success of our method using only Gram matrix; however, our approach can be expanded to utilize other losses as well.

To the best of our knowledge, the only previous work which generates multiple stylizations is (Ulyanov et al., 2017) which utilized Julesz ensemble to explicitly encourage diversity in stylizations. However their results are quite similar in style and only differ in minor details. A qualitative comparison in Figures 8,14 show that our proposed method is more effective in diverse stylization.

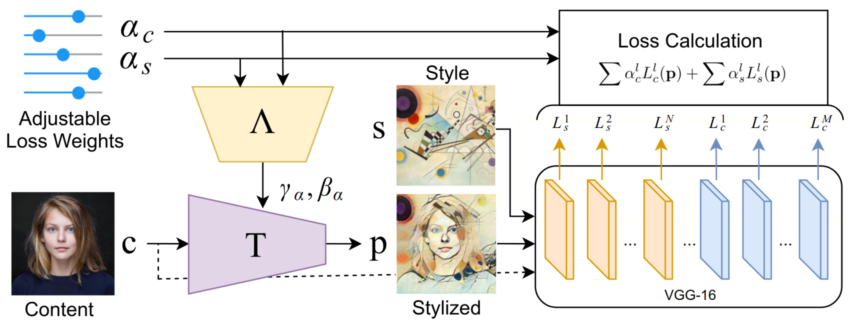
α_c
α_s
Adjustable Loss Weights
Loss Calculation
$\sum \alpha_c^l L_c^l(\mathbf{p}) + \sum \alpha_s^l L_s^l(\mathbf{p})$
Style
Λ
s
$\gamma_\alpha, \beta_\alpha$
L_s^1 L_s^2 L_s^N L_c^1 L_c^2 L_c^M
c
T
p
Content
Stylized
VGG-16

Figure 2: Architecture of the proposed model. The loss adjustment parameters $\boldsymbol{\alpha}_c$ and $\boldsymbol{\alpha}_s$ is passed to the network Λ which will predict activation normalizers $\gamma_{\boldsymbol{\alpha}}$ and $\beta_{\boldsymbol{\alpha}}$ that normalize activation of main stylizing network T. The stylized image is passed to a trained image classifier where its intermediate representation is used to calculate the style loss $\mathcal{L}_s$ and content loss $\mathcal{L}_c$. Then the loss from each layer is multiplied by the corresponding input adjustment parameter. Models Λ and T are trained jointly by minimizing this weighted sum. At generation time, values for $\boldsymbol{\alpha}_c$ and $\boldsymbol{\alpha}_s$ can be adjusted manually or sampled randomly.

3 Background

3.1 Style transfer using deep networks

Style transfer can be formulated as generating a stylized image $\mathbf{p}$ which its content is similar to a given content image $\mathbf{c}$ and its style is close to another given style image $\mathbf{s}$. The similarity in style can be vaguely defined as sharing the same spatial statistics in low-level features, while similarity in content is roughly having a close Euclidean distance in high-level features (Ghiasi et al., 2017). These features are typically extracted from a pre-trained image classification network, commonly VGG-19 (Simonyan & Zisserman, 2014). The main idea here is that the features obtained by the image classifier contain information about the content of the input image while the correlation between these features represents its style.

In order to increase the similarity between two images, Gatys et al. (Gatys et al., 2016) minimize the following distances between their extracted features:

$$\mathcal{L}_c^l(\mathbf{p}) = \left|\left|\phi^l(\mathbf{p}) - \phi^l(\mathbf{s})\right|\right|_2^2, \qquad \mathcal{L}_s^l(\mathbf{p}) = \left|\left|G(\phi^l(\mathbf{p})) - G(\phi^l(\mathbf{s}))\right|\right|_F^2 \tag{1}$$

where $\phi^l(\mathbf{x})$ is activation of a pre-trained classification network at layer l given the input image $\mathbf{x}$, while $\mathcal{L}_c^l(\mathbf{p})$ and $\mathcal{L}_s^l(\mathbf{p})$ are content and style loss at layer l respectively. $G(\phi^l(\mathbf{p}))$ denotes the Gram matrix associated with $\phi^l(\mathbf{p})$.

The total loss is calculated as the weighted sum of losses across a set of *content layers* C and *style layers* S:

$$\mathcal{L}_c(\mathbf{p}) = \sum_{l \in C} w_c^l \mathcal{L}_c^l(\mathbf{p}), \qquad \mathcal{L}_s(\mathbf{p}) = \sum_{l \in S} w_s^l \mathcal{L}_s^l(\mathbf{p}) \tag{2}$$

where w_c^l, w_s^l are hyper-parameters to adjust the contribution of each layer to the loss. Layers can be shared between C and S. These hyper-parameters have to be manually fine tuned through try and error and usually vary for different style images (Figure 4). Finally, the objective of style transfer can be defined as:

$$\min_{\mathbf{p}} \left(\mathcal{L}_c(\mathbf{p}) + \mathcal{L}_s(\mathbf{p})\right) \tag{3}$$

This objective can be minimized by iterative gradient-based optimization methods starting from an initial $\mathbf{p}$ which usually is random noise or the content image itself.

3.2 Real-time style transfer

Solving the objective in Equation 3 using an iterative method can be very slow and has to be repeated for any given pair of style/content image. A much faster method is to directly train a deep network T which maps a given content image $\mathbf{c}$ to a stylized image $\mathbf{p}$ (Johnson et al., 2016). T is usually a feed-forward convolutional network (parameterized by θ) with residual connections between down-

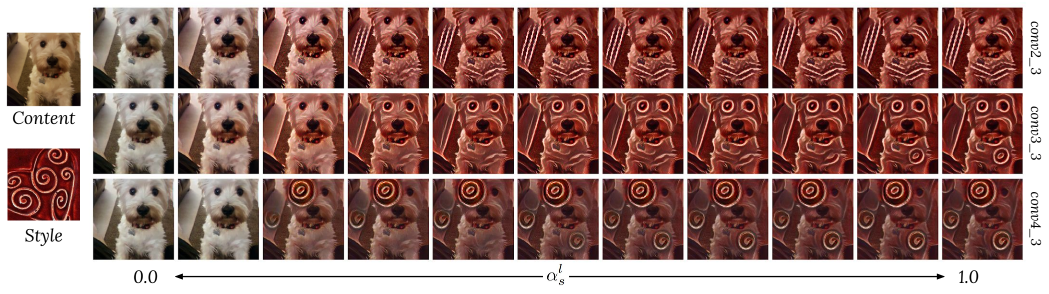

Figure 3: Effect of adjusting the input parameters $\boldsymbol{\alpha}_s$ on stylization. Each row shows the stylized output when a single α_s^l increased gradually from zero to one while other $\boldsymbol{\alpha}_s$ are fixed to zero. Notice how the details of each stylization is different specially at the last column where the value is maximum. Also note how deeper layers use *bigger* features of style image to stylize the content.

sampling and up-sampling layers (Ruder et al., 2018) and is trained on many content images using Equation 3 as the loss function:

$$\min_{\theta} \big(\mathcal{L}_c(T(\mathbf{c})) + \mathcal{L}_s(T(\mathbf{c}))\big) \tag{4}$$

The style image is assumed to be fixed and therefore a different network should be trained per style image. However, for a fixed style image, this method can generate stylized images in real-time (Johnson et al., 2016). Recent methods (Dumoulin et al., 2017; Ghiasi et al., 2017; Huang & Belongie, 2017) introduced real-time style transfer methods for multiple styles. But, these methods still generate only one stylization for a pair of style and content images.

4 Proposed Method

This paper addresses the following issues in real-time feed-forward style transfer methods:
1. The output of these models is sensitive to the hyper-parameters w_c^l and w_s^l and different weights significantly affect the generated stylized image as demonstrated in Figure 6. Moreover, the ”optimal” weights vary from one style image to another (Figure 4) and finding a good set of weights should be repeated for each style image. Note that for each set of w_c^l and w_s^l the model has to be retrained that limits the practicality of style transfer models.
2. Current methods generate a *single* stylized image given a content/style pair. While the stylizations of different methods usually look very distinct (Sanakoyeu et al., 2018), it is not possible to say which stylization is better for every context since it is a matter of personal taste. To get a favored stylization, users may need to try different methods or train a network with different hyper-parameters which is not satisfactory and, ideally, the user should have the capability of getting different stylizations in real-time.

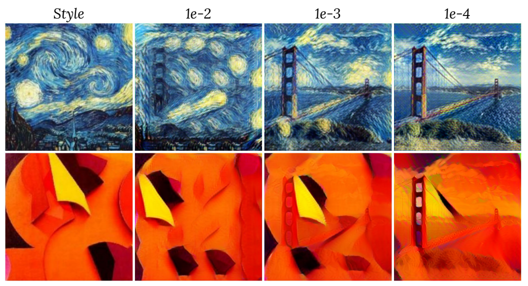

Figure 4: Effect of adjusting the style weight in style transfer network from (Johnson et al., 2016). Each column demonstrates the result of a separate training with all w_s^l set to the printed value. As can be seen, the ”optimal” weight is different from one style image to another and there can be multiple ”good” stylizations depending on ones’ personal choice. Check supplementary materials for more examples.

We address these issues by conditioning the generated stylized image on additional input parameters where each parameter controls the share of the loss from a corresponding layer. This solves the problem (1) since one can adjust the contribution of each layer to adjust the final stylized result after the training and in real-time. Secondly, we address the problem (2) by randomizing these parameters which result in different stylizations.

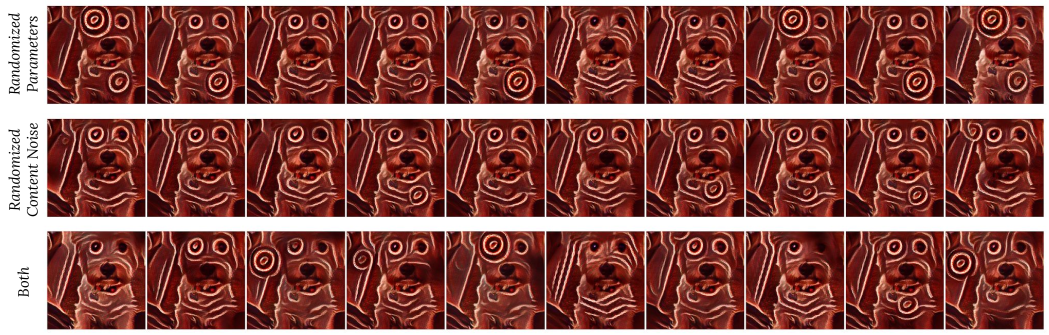

Figure 5: Effect of randomizing $\boldsymbol{\alpha}$ and additive Gaussian noise on stylization. *Top*: randomizing $\boldsymbol{\alpha}$ results to different stylizations while the style features appear in the same spatial position (e.g., look at the swirl effect on the left eye). *Middle*: the effect of adding random noise to the content image in moving these features with fixed $\boldsymbol{\alpha}$. *Bottom*: combination of this two randomization techniques can generate highly versatile outputs. Notice how each image in this row differs in both style and the spatial position of style elements.

4.1 Style transfer with adjustable loss

We enable the users to adjust w_c^l,w_s^l without retraining the model by replacing them with input parameters and conditioning the generated style images on these parameters:

$$\mathbf{p} = \Psi(\mathbf{c}, \mathbf{s}, \boldsymbol{\alpha}_c, \boldsymbol{\alpha}_s)$$

$\boldsymbol{\alpha}_c$ and $\boldsymbol{\alpha}_s$ are vectors of parameters where each element corresponds to a different layer in content layers C and style layers S respectively. α_c^l and α_s^l replace the hyper-parameters w_c^l and w_s^l in the objective Equation 2:

$$\mathcal{L}_c(\mathbf{p}) = \sum_{l \in C} \alpha_c^l \mathcal{L}_c^l(\mathbf{p}) \qquad \mathcal{L}_s(\mathbf{p}) = \sum_{l \in S} \alpha_s^l \mathcal{L}_s^l(\mathbf{p}) \tag{5}$$

To learn the effect of $\boldsymbol{\alpha}_c$ and $\boldsymbol{\alpha}_s$ on the objective, we use a technique called *conditional instance normalization* (Ulyanov et al.). This method transforms the activations of a layer x in the feed-forward network T to a normalized activation z conditioned on extra inputs $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_c, \boldsymbol{\alpha}_s]$:

$$z = \gamma_{\boldsymbol{\alpha}}\left(\frac{x - \mu}{\sigma}\right) + \beta_{\boldsymbol{\alpha}} \tag{6}$$

where μ and σ are mean and standard deviation of activations at layer x across spatial axes (Ghiasi et al., 2017) and $\gamma_{\boldsymbol{\alpha}}, \beta_{\boldsymbol{\alpha}}$ are the learned mean and standard deviation of this transformation. These parameters can be approximated using a second neural network which will be trained end-to-end with T:

$$\gamma_{\boldsymbol{\alpha}}, \beta_{\boldsymbol{\alpha}} = \Lambda(\boldsymbol{\alpha}_c, \boldsymbol{\alpha}_s) \tag{7}$$

Since $\mathcal{L}^l$ can be very different in scale, one loss term may dominate the others which will fail the training. To balance the losses, we normalize them using their exponential moving average as a normalizing factor, i.e. each $\mathcal{L}^l$ will be normalized to:

$$\mathcal{L}^l(\mathbf{p}) = \frac{\sum_{i \in C \cup S} \overline{\mathcal{L}^i}(\mathbf{p})}{\overline{\mathcal{L}^l}(\mathbf{p})} * \mathcal{L}^l(\mathbf{p}) \tag{8}$$

where $\overline{\mathcal{L}^l}(\mathbf{p})$ is the exponential moving average of $\mathcal{L}^l(\mathbf{p})$.

5 Experiments

In this section, first we study the effect of adjusting the input parameters in our method. Then we demonstrate that we can use our method to generate random stylizations and finally, we compare our method with a few baselines in terms of generating random stylizations.

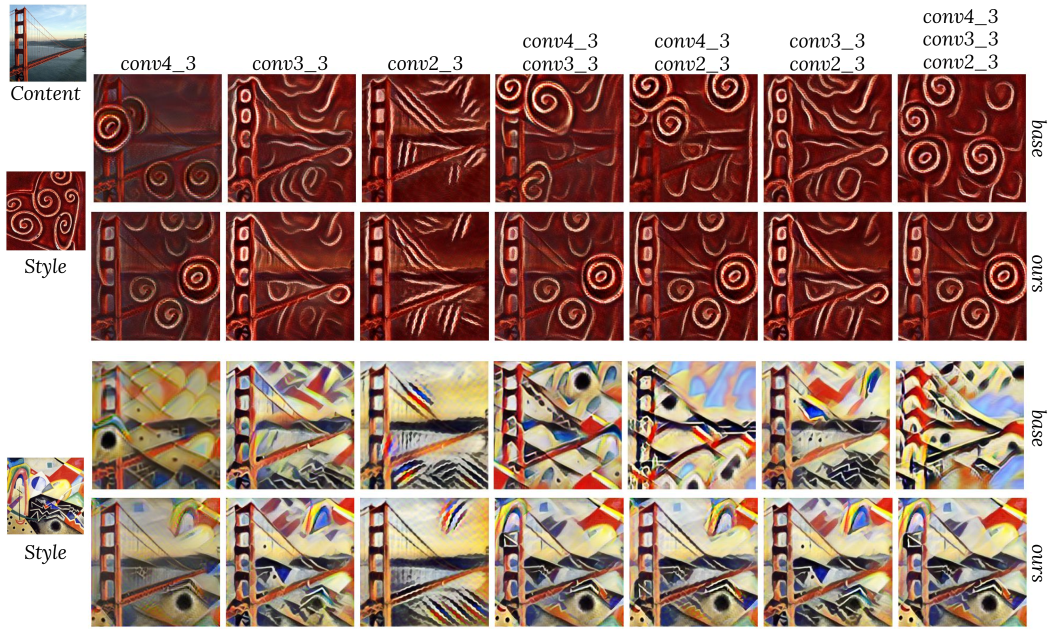

Figure 6: Qualitative comparison between the base model from (Johnson et al., 2016) with our proposed method. For the base model, each column has been retrained with all the weights set to zero except for the mentioned layers which has been set to 1e−3. For our model, the respective parameters α_s^l has been adjusted. Note how close the stylizations are and how the combination of layers stays the same in both models.

5.1 Implementation details

We implemented Λ as a multilayer fully connected neural network. We used the same architecture as (Johnson et al., 2016; Dumoulin et al., 2017; Ghiasi et al., 2017) for T and only increased number of residual blocks by 3 (look at supplementary materials for details) which improved stylization results. We trained T and Λ jointly by sampling random values for $\boldsymbol{\alpha}$ from $U(0, 1)$. We trained our model on ImageNet (Deng et al., 2009) as content images while using paintings from Kaggle Painter by Numbers (Kaggle) and textures from Descibable Texture Dataset (Cimpoi et al., 2014) as style images. We selected random images form ImageNet test set, MS-COCO (Lin et al., 2014) and faces from CelebA dataset (Liu et al., 2018) as our content test images. Similar to (Ghiasi et al., 2017; Dumoulin et al., 2017), we used the last feature set of $conv3$ as content layer C. We used last feature set of $conv2$, $conv3$ and $conv4$ layers from VGG-19 network as style layers S. Since there is only one content layer, we fix $\boldsymbol{\alpha}_c = 1$. Our implementation can process 47.5 fps on a NVIDIA GeForce 1080, compared to 52.0 for the base model without Λ.

5.2 Effect of adjusting the input parameters

The primary goal of introducing the adjustable parameters $\boldsymbol{\alpha}$ was to modify the loss of each separate layer manually. Qualitatively, this is demonstrable by increasing one of the input parameters from zero to one while fixing the rest of them to zero. Figure 3 shows one example of such transition. Each row in this figure is corresponding to a different style layer, and therefore the stylizations at each row would be different. Notice how deeper layers stylize the image with *bigger* stylization elements from the style image but all of them still apply the coloring. We also visualize the effect of increasing two of the input parameters at the same time in Figure 9. However, these transitions are best demonstrated interactively which is accessible at the project website https://goo.gl/PVWQ9K.

To quantitatively demonstrate the change in losses with adjustment of the input parameters, we rerun the same experiment of assigning a fixed value to all of the input parameters while gradually increasing one of them from zero to one, this time across 100 different content images. Then we calculate the median loss at each style loss layer S. As can be seen in Figure 7-(top), increasing

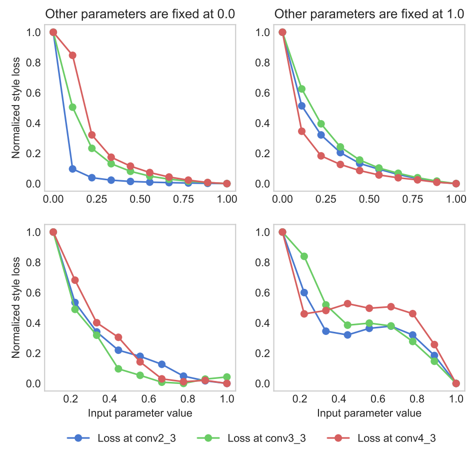

Figure 7: Effect of adjusting the input parameters $\boldsymbol{\alpha}_s$ on style loss from different layers across single style image of Figure 3 (top) or 25 different style images (bottom). In each curve, one of the input parameters α_s^l has been increased from zero to one while others are fixed at to zero (left) and to one (right). Then the style loss has been calculated across 100 different content images. As can be seen, increasing α_s^l decreases the loss of the corresponding layer. Note that the losses is normalized in each layer for better visualization.

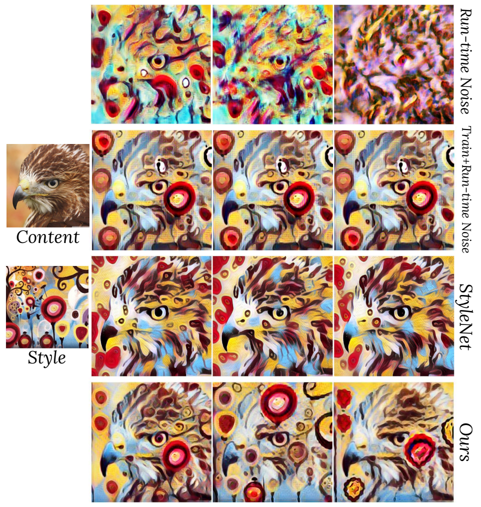

Figure 8: Diversity comparison of our method and baselines. First row shows results for a baseline that adds random noises to the style parameters at run-time. While we get diverse stylizations, the results are not similar to the input style image. Second row contains results for a baseline that adds random noises to the style parameters at both training time and run-time. Model is robust to the noise and it does not generate diverse results. Third row shows stylization results of StyleNet (Ulyanov et al., 2017). Our method generates diverse stylizations while StyleNet results mostly differ in minor details. More examples can be seen at Figure 14.

α_s^l decreases the measured loss corresponding to that parameter. To show the generalization of our method across style images, we trained 25 models with different style images and then measured median of the loss at any of the S layers for 100 different content images (Figure 7)-(bottom). The similarly exhibited trends show that the model can generate stylizations conditioned on the input parameters.

Finally, we verify that modifying the input parameters $\boldsymbol{\alpha}_s$ generates visually similar stylizations to the retrained base model with different loss weights w_s^l. To do so, we train the base model (Johnson et al., 2016) multiple times with different w_s^l and compare the generated results with the output of our model when $\forall l \in S, \alpha_s^l = w_s^l$ (Figure 6). Note how the proposed stylizations in test time and without retraining match the output of the base model.

5.3 Generating randomized stylizations

One application of our proposed method is to generate multiple stylizations given a fixed pair of content/style image. To do so, we randomize $\boldsymbol{\alpha}$ to generate randomized stylization (top row of Figure 5). Changing values of $\boldsymbol{\alpha}$ usually do not randomize the position of the "elements" of the style. We can enforce this kind of randomness by adding some noise with the small magnitude to the content image. For this purpose, we multiply the content image with a mask which is computed by applying an inverse Gaussian filter on a white image with a handful (< 10) random zeros. This masking can shadow sensitive parts of the image which will change the spatial locations of the "elements" of style. Middle row in Figure 5 demonstrates the effect of this randomization. Finally, we combine these two randomizations to maximizes the diversity of the output which is shown in the bottom row of Figure 5. More randomized stylizations can be seen in Figure 10 and at https://goo.gl/PVWQ9K.

5.4 Comparison with other methods

To the best of our knowledge, generating diverse stylizations at real-time is only have been studied at (Ulyanov et al., 2017) before. In this section, we qualitatively compare our method with this baseline. Also, we compare our method with a simple baseline where we add noise to the style parameters.

The simplest baseline for getting diverse stylizations is to add noises to some parameters or the inputs of the style-transfer network. In the last section, we demonstrate that we can move the locations of elements of style by adding noise to the content input image. To answer the question that if we can get different stylizations by adding noise to the style input of the network, we train (Dumoulin et al., 2017) which uses conditional instance normalization for transferring style. We train this model with only one style image and to get different stylizations, we add random noise to the style parameters ($\gamma_{\boldsymbol{\alpha}}$ and $\beta_{\boldsymbol{\alpha}}$ parameters of equation 6) at run-time. The stylization results for this baseline are shown on the top row of Figure 8. While we get different stylizations by adding random noises, the stylizations are no longer similar to the input style image.

To enforce similar stylizations, we trained the same baseline while adding random noises at the training phase. As it can be seen in the second row of Figure 8, adding noise at the training time makes the model robust to the noise and the stylization results are similar. This indicates that a loss term that encourages diversity is necessary.

We also compare the results of our model with StyleNet (Ulyanov et al., 2017). As visible in Figures 8, although StyleNet's stylizations are different, they vary in minor details and all carry the same level of stylization elements. In contrast, our model synthesizes stylized images with varying levels of stylization and more randomization.

6 Conclusion

Our main contribution in this paper is a novel method which allows adjustment of each loss layer's contribution in feed-forward style transfer networks, in real-time and after training. This capability allows the users to adjust the stylized output to find the favorite stylization by changing input parameters and without retraining the stylization model. We also show how randomizing these parameters plus some noise added to the content image can result in very different stylizations from the same pair of style/content image.

Our method can be expanded in numerous ways e.g. applying it to multi-style transfer methods such as (Dumoulin et al., 2017; Ghiasi et al., 2017), applying the same parametrization technique to randomize the correlation loss between *the features of each layer* and finally using different loss functions and pre-trained networks for computing the loss to randomize the outputs even further. One other interesting future direction is to apply the same ”loss adjustment after training” technique for other classic computer vision and deep learning tasks. Style transfer is not the only task in which modifying the hyper-parameters can greatly affect the predicted results and it would be rather interesting to try this method for adjusting the hyper-parameters in similar problems.

References

Michael Ashikhmin. Synthesizing natural textures. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pp. 217–226. ACM, 2001.

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

Yun Cao, Zhiming Zhou, Weinan Zhang, and Yong Yu. Unsupervised diverse colorization via generative adversarial networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2017.

Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *ICCV*, 2017.

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

Aditya Deshpande, Jiajun Lu, Mao-Chuang Yeh, Min Jin Chong, and David A Forsyth. Learning diverse image colorization. In *CVPR*, 2017.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *Proc. of ICLR*, 2017.

Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 341–346. ACM, 2001.

Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 262–270, 2015.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, pp. 2414–2423, 2016.

Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *arXiv preprint arXiv:1705.06830*, 2017.

Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 453–460. ACM, 1998.

Xun Huang and Serge J Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. *arXiv preprint arXiv:1804.04732*, 2018.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pp. 694–711. Springer, 2016.

Kaggle. Kaggle Painter by numbers kernel description. `www.kaggle.com/c/painter-by-numbers`. 2016.

Alex X Lee, Richard Zhang, Frederik Ebert, Pieter Abbeel, Chelsea Finn, and Sergey Levine. Stochastic adversarial video prediction. *arXiv preprint arXiv:1804.01523*, 2018.

Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036*, 2017a.

Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. In *Advances in Neural Information Processing Systems*, pp. 386–396, 2017b.

Yijun Li, Ming-Yu Liu, Xueting Li, Ming-Hsuan Yang, and Jan Kautz. A closed-form solution to photorealistic image stylization. *arXiv preprint arXiv:1802.06474*, 2018.

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*, pp. 740–755. Springer, 2014.

Xiao-Chang Liu, Ming-Ming Cheng, Yu-Kun Lai, and Paul L Rosin. Depth-aware neural style transfer. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, 2017.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15:2018, 2018.

Xingchao Peng and Kate Saenko. Synthetic to real adaptation with generative correlation alignment networks. In *WACV*, 2018.

Eric Risser, Pierre Wilmot, and Connelly Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *arXiv preprint arXiv:1701.08893*, 2017.

Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. Artistic style transfer for videos and spherical images. *International Journal of Computer Vision*, pp. 1–21, 2018.

Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Björn Ommer. A style-aware content loss for real-time hd style transfer. *arXiv preprint arXiv:1807.10201*, 2018.

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

D Ulyanov, A Vedaldi, and VS Lempitsky. Instance normalization: the missing ingredient for fast stylization. corr abs/1607.0 (2016).

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, pp. 1349–1357, 2016.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017.

Yongqi Zhang. Xogan: One-to-many unsupervised image-to-image translation. *arXiv preprint arXiv:1805.07277*, 2018.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

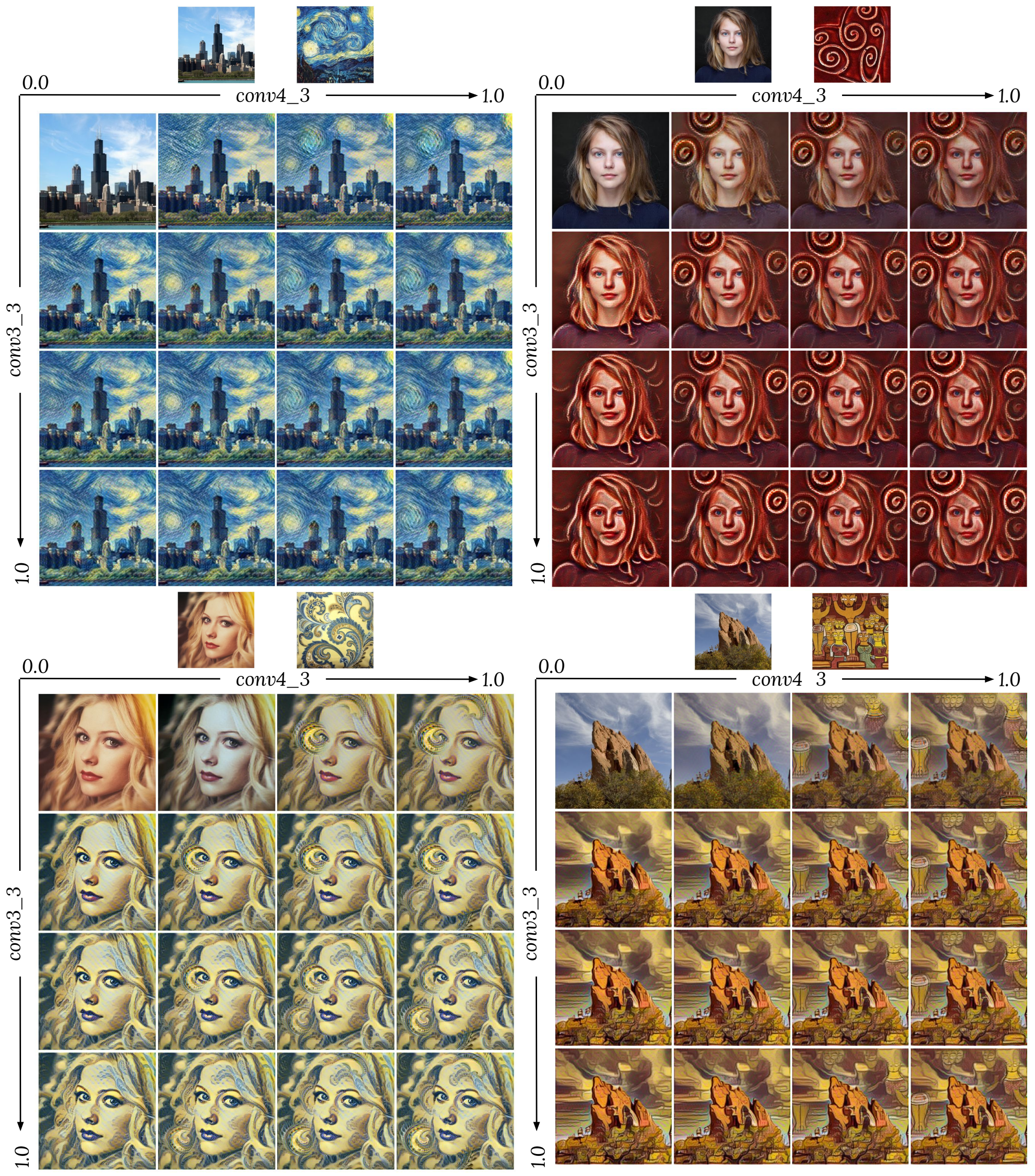

Figure 9: More results for adjusting the input parameters in real-time and after training. In each block the style/content pair is fixed while the parameters corresponding to $conv3$ and $conv4$ increases vertically and horizontally from zero to one. Notice how the details are different from one layer to another and how the combination of layers may result to more favored stylizations. For an interactive presentation please visit https://goo.gl/PVWQ9K.

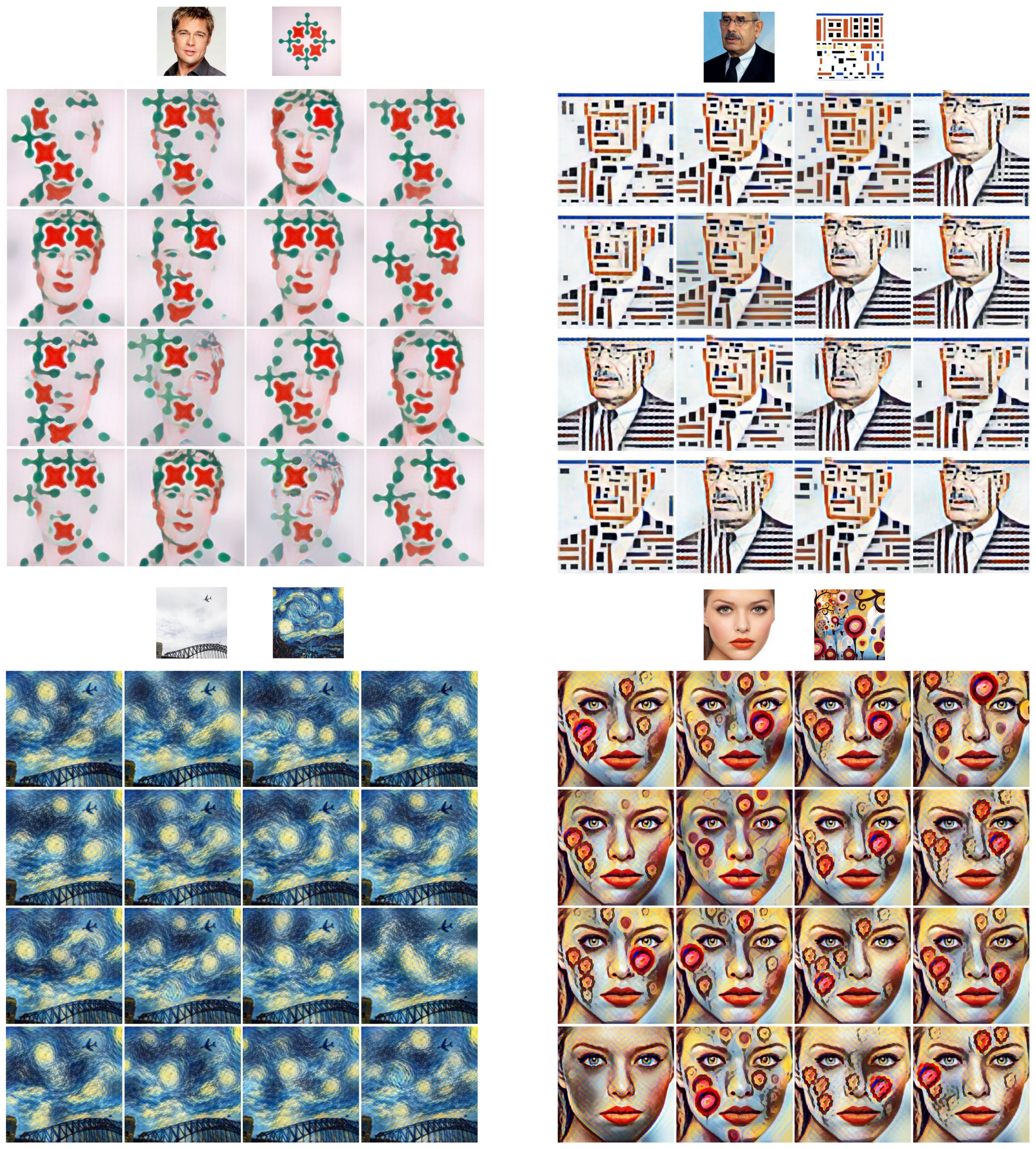

Figure 10: More results of stochastic stylization from the same pair of content/style. Each block represents randomized stylized outputs given the fix style/content image demonstrated at the top. Notice how stylized images vary in style granularity, the spatial position of style elements while maintaining similarity to the original style and content image. For more results please visit https://goo.gl/PVWQ9K.

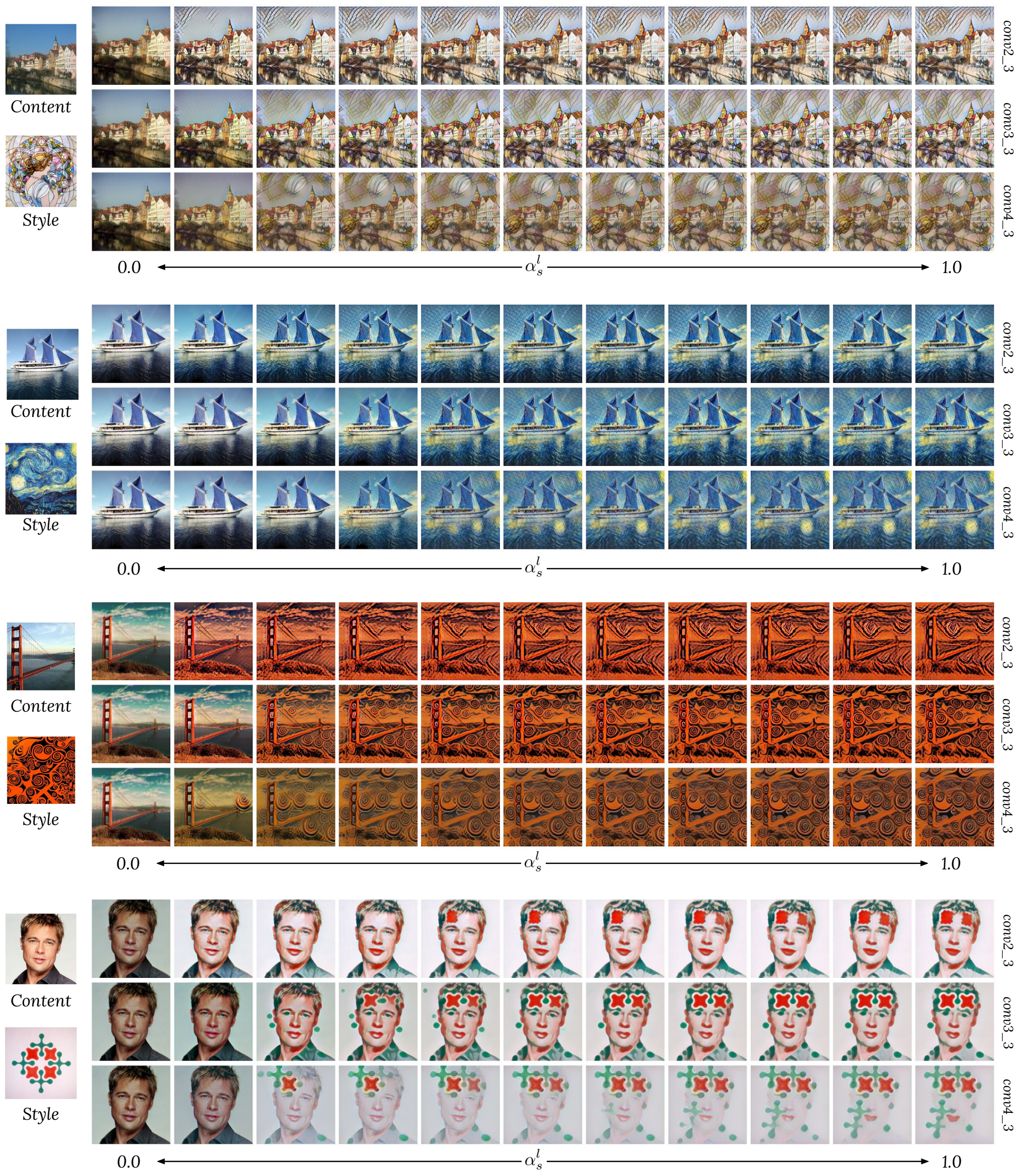

Figure 11: More examples for effect of adjusting the input parameters $\boldsymbol{\alpha}_s$ in real-time. Each row shows the stylized output when a single α_s^l increased gradually from zero to one while other $\boldsymbol{\alpha}_s$ are fixed to zero. Notice how the details of each stylization is different specially at the last column where the weight is maximum. Also how deeper layers use *bigger* features of style image to stylize the content.

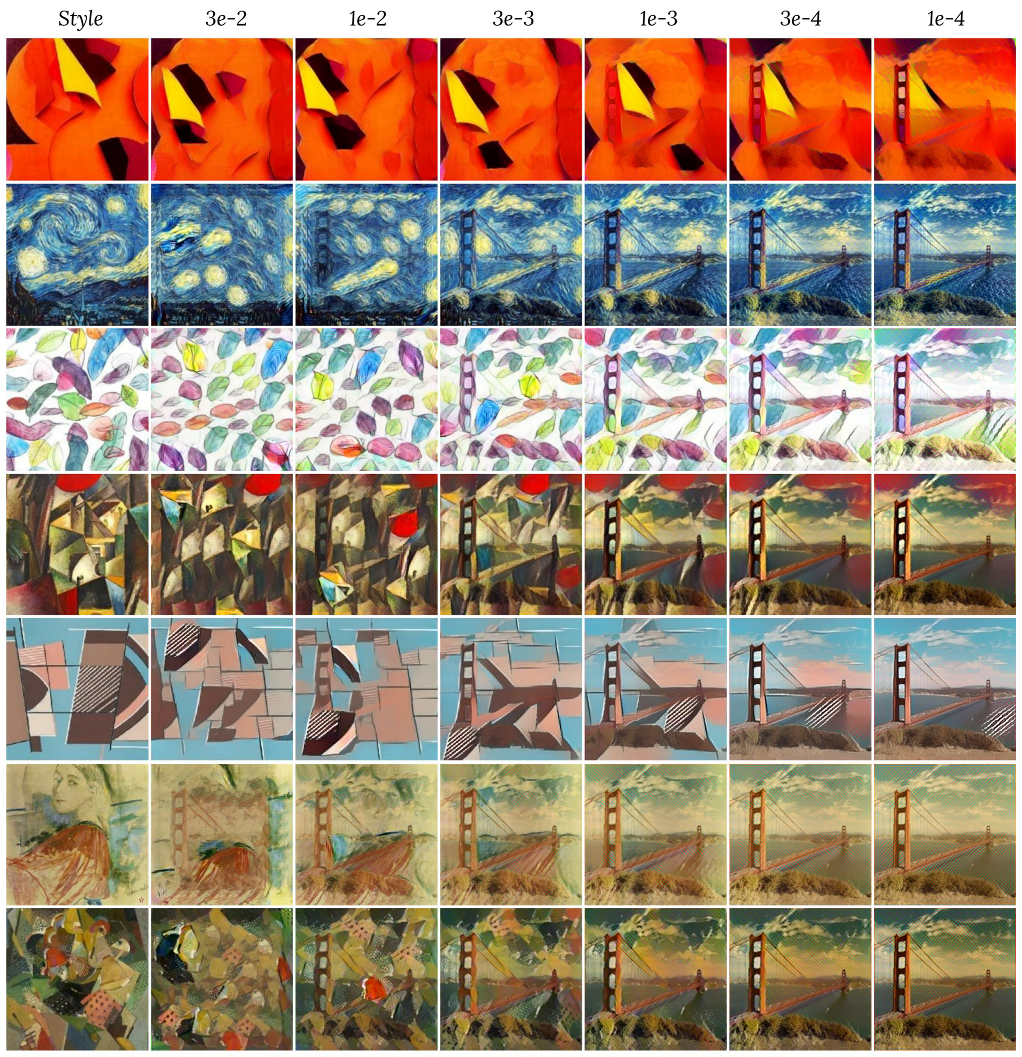

Figure 12: More examples for effect of adjusting the style weight in style transfer network from (Johnson et al., 2016). Each column demonstrates the result of a separate training. As can be seen, the ”optimal” weight is different from one style image to another and there can be more than one ”good” stylization depending on ones personal choice.

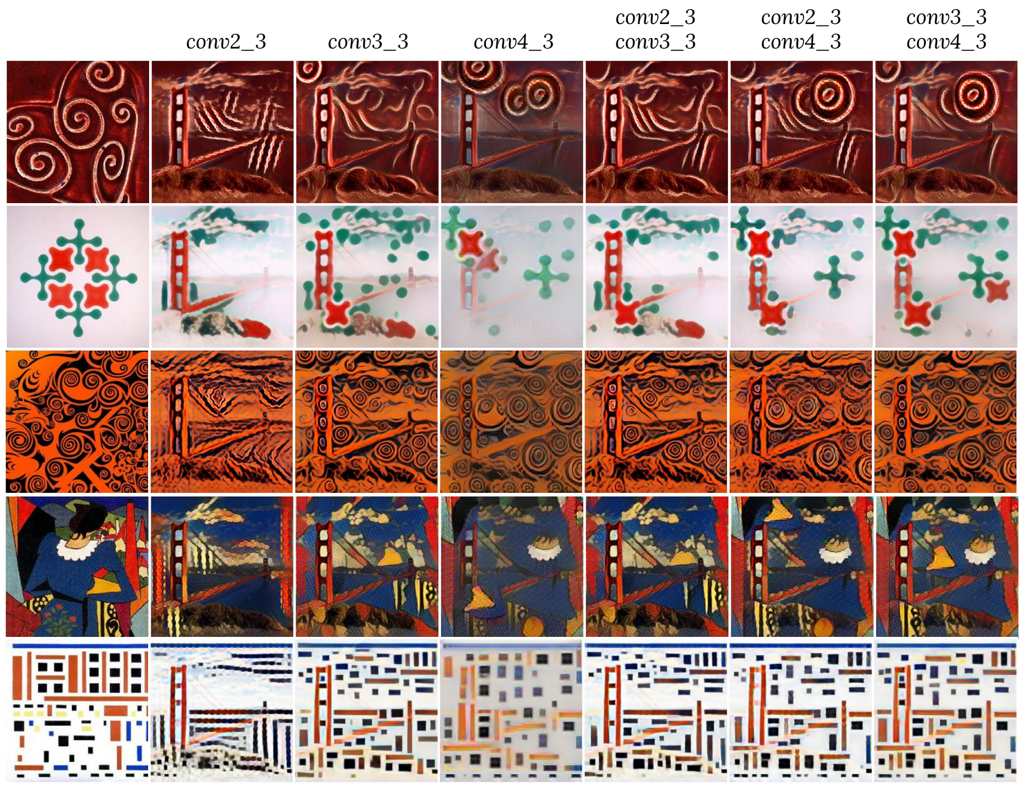

Figure 13: Results of combining losses from different layers at generation time by adjusting their corresponding parameters. The first column is the style image which is fixed for each row. The content image is the same for all of the outputs. The corresponding parameter for each one of the losses is zero except for the one(s) mentioned in the title of each column. Notice how each layer enforces a different type of stylization and how the combinations vary as well. Also note how a single combination of layers cannot be the ”optimal” stylization for any style image and one may prefer the results from another column.

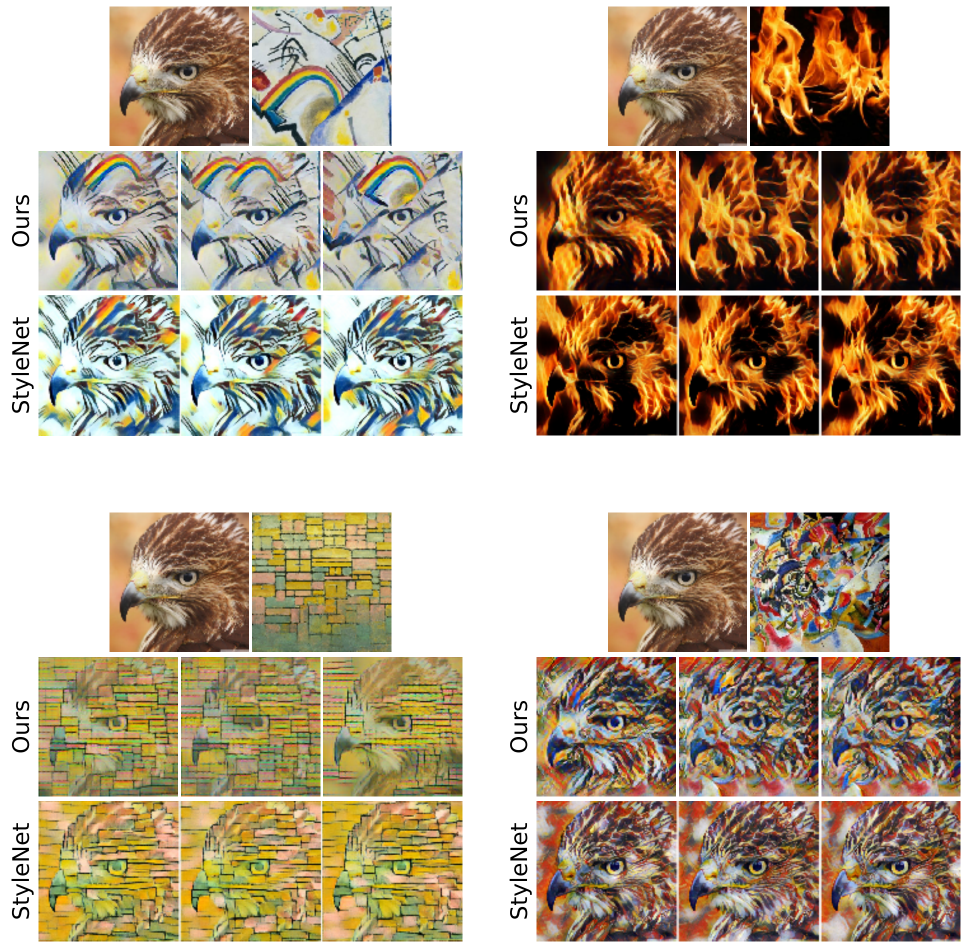

Figure 14: Diversity comparison of our method with StyleNet (Ulyanov et al., 2017). Our method generates diverse stylizations while StyleNet results mostly differ in minor details.

Operation	input dimensions	output dimensions
input parameters $\boldsymbol{\alpha}$	3	1000
$10\times$Dense	1000	1000
Dense	1000	$2(\gamma_{\boldsymbol{\alpha}}, \beta_{\boldsymbol{\alpha}})$
Optimizer	Adam ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)	
Training iterations	200K	
Batch size	8	
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)	

Table 1: Network architecture and hyper-parameters of Λ.

Operation	Kernel size	Stride	Feature maps	Padding	Nonlinearity
Network – $256 \times 256 \times 3$ input					
Convolution	9	1	32	SAME	ReLU
Convolution	3	2	64	SAME	ReLU
Convolution	3	2	128	SAME	ReLU
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Residual block			128		
Upsampling			64		
Upsampling			32		
Convolution	9	1	3	SAME	Sigmoid
Residual block – C feature maps					
Convolution	3	1	C	SAME	ReLU
Convolution	3	1	C	SAME	Linear
	Add the input and the output				
Upsampling – C feature maps					
	Nearest-neighbor interpolation, factor 2				
Convolution	3	1	C	SAME	ReLU
Normalization	Conditional instance normalization after every convolution				
Optimizer	Adam ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)				
Training iterations	200K				
Batch size	8				
Weight initialization	Isotropic gaussian ($\mu = 0$, $\sigma = 0.01$)				

Table 2: Network architecture and hyper-parameters of T.