Pytorchst: A Library for Nerual Style Transfer

Licheng Luo Jiarui Sun Qihao Wang
University of Illinois at Urbana, Champaign
{ll6, jsun57, qw2}@illinois.edu

Abstract

Despite that many implementations for neural style transfer models are available, these implementations are limited in terms of providing users an easy-to-use interface or a comprehensive library to play with those models, not to mention they are implemented by different people with different implementation details thus making fair comparisons between different models is hard. We implemented Pytorchst, a library that contains two neural style transfer models under the same training and evaluation interface. This library allows us to evaluate the implemented models thoroughly using same set of hyperparameters and with the same criteria, which allows us to lay out key contributions of the selected models, demonstrate model differences and provide previously unknown observations through fair comparisons and evaluations.

Our code can be accessed at https://github.com/LectroniC/pytorchst.

Our narrated video can be accessed at https://www.youtube.com/watch?v=iwlusxqI7VE.

1. Introduction

Style transfer is a process which combines a content image with a style image so that the semantic content of the output image comes from the content image while the style of the output image is similar to the style image. Pioneering works on style transfer utilize methods including pixel re-sampling [2] [7] and multi-scale feature matching [1] [4], both of which employ the idea of fusing image pyramids as traditional blending technique. With the recent advances in deep neural network, a surge of interests in applying neural networks on style transfer task emerged [3] [5] [8], which significantly improve the quality of transferred images. For those neural network based techniques, there exist implementations of many different flavors. Some are implemented by the authors while others are reproduced according to those papers by other developers. In most scenarios, these implementations are limited in terms of providing users an easy-to-use interface or a comprehensive library to play with those models. It is also hard to fairly compare different solutions using existing implementations.

This project focuses on implementing different neural image style transfer models under a same training interface and a shared implementation perspective which allows a fair comparison between different models. Specifically, we implement and compare two deep learning based popular image style transfer models [5] [13], and produce an evaluation survey around these two methods. Our main project contribution can be summarized as follows:

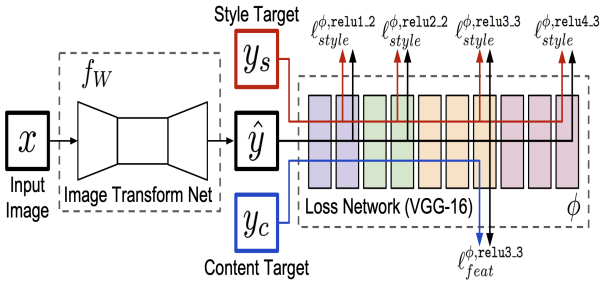
- A unified framework of style transfer in PyTorch and implementations of two models based on two selected style-transfer papers from scratch ([5] and [13]).
- A report which lays out key contributions of the selected papers, demonstrates model differences and provides previously unknown observations through fair comparisons and evaluations.

2. Details of the Approach

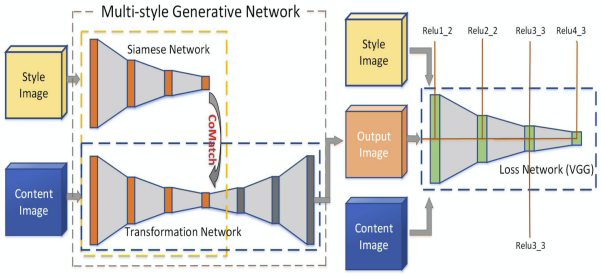
In this section, we present details of the overall library for image style transfer, which currently combines two models presented in [5] and [13] into one unified framework. We start from introducing the models responsible for generating style transferred images, followed by the idea of perceptual loss along with details of the loss network. The unified training and evaluation pipeline is then discussed. We also highlight some similarities and differences of the two models we consider.

2.1. Problem Definition

The objective of image style transfer task is to synthesize texture and semantic information from different images. Specifically, given a style image y_s and a content image y_c, we aim to learn a function $F(\cdot)$ which produces a style transferred image $\hat{y}$. In both implemented models, $F(\cdot)$ consists of a network $f_W(\cdot)$ responsible for image generation, and a loss network $f_L(\cdot)$ responsible for loss calculation.



(a) Image Transformation Net with VGG loss [5], $x = y_c$.



(b) Multi-style Generative Net with VGG loss. [13]

Figure 1: Image style transfer model structures included in library.

2.2. Image Transformation Network

The image transformation network $f_W(\cdot)$ (PLST), presented in [5], takes a content image y_c as input and produces the style transferred image $\hat{y}$:

$$\hat{y} = f_W(y_c). \tag{1}$$

The network consists of multiple convolutional up-sampling and down-sampling layers, along with multiple residual blocks to facilitate training and enhance model capacity.

In terms of model implementation, we mainly follow the original paper and its associated supplementary material [1]. Note that instead of using batch normalization layers our implementation employs instance normalization layers, which is suggested in [12] to improve transferred image quality.

2.3. Multi-style Generative Network

The multi-style generative network $f_W(\cdot)$ (MSGNet) presented in [10] is different from the image transformation network in [5] as it enables multi-style transfer which allows both style image and content image as network input. In particular, we have:

$$\hat{y} = f_W(y_s, y_c), \tag{2}$$

where the additional input y_s is the style image. Except the standard convolutional up-sampling, down-sampling layers and residual blocks, the network introduces CoMatch layer, which is designed to explicitly match second order feature statistics of the style image and content image. It enables the network to receive different style images as input during both training and evaluation phase.

In terms of model implementation, we follow the original paper model description and code repository released by the authors.

[1] https://web.eecs.umich.edu/~justincj/papers/eccv16/JohnsonECCV16Supplementary.pdf

2.4. Loss Network

In order to better capture and measure perceptual and semantic differences between transferred images and input images, both implemented models employ the idea of perceptual loss, which uses another pretrained network as loss function to compute major losses. It is inspired by the fact that network that is pretrained on large computer vision dataset has already learned to encode the perceptual and semantic information, which is necessary to measure for image style transfer task. The loss network takes either the style, content or the transferred image, and outputs the corresponding feature maps $\phi_s, \phi_c, \hat{\phi}$ at K multiple resolutions:

$$\phi_s = f_L(y_s), \phi_c = f_L(y_c), \hat{\phi} = f_L(\hat{y}). \tag{3}$$

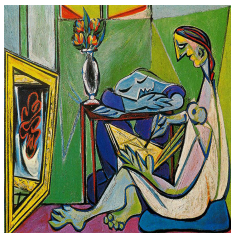
The feature maps are then used to calculate content loss (Also called the feature reconstruction loss) and style loss (Also called the style reconstruction loss) through mean squared error. Note that the loss network is only used for loss calculation and not trained anymore. The total loss for both models are calculated as follows:

$$\begin{aligned} L_{total} &= \lambda_c \left\| f_L^c\left(\hat{y}\right) - f_L^c\left(y_c\right) \right\|_F^2 \\ &+ \lambda_s \sum_{i=1}^{K} \left\| \mathcal{G}\left(f_L^i\left(\hat{y}\right)\right) - \mathcal{G}\left(f_L^i\left(y_s\right)\right) \right\|_F^2 \\ &+ \lambda_{TV} \ell_{TV}\left(\hat{y}\right), \end{aligned} \tag{4}$$

where the first term corresponds to content loss and second term corresponds to style loss. The content loss is considered at resolution index c, and the style loss in consider at multiple resolution indexes $i \in \{1...K\}$. λ_c, λ_s and λ_{TV} are balancing weight hyperparameters, $\mathcal{G}(\cdot)$ denotes gram matrix function, and $\ell_{TV}(\cdot)$ is the total variation regularization loss to encourage output smoothness.

In particular, the loss network we use is VGG16 [11] network pretrained on the ImageNet [9] dataset. In terms of implementation, content loss is computed at layer ReLU3_3

(a) Starry Night



(b) La Muse

Figure 2: Style Image

($c = 3$) and style loss is computed at layers ReLU1_2, ReLU2_2, ReLU3_3, and ReLU4_3 ($K = 4$). Note that both papers have inconsistencies among model diagram, paper description and actual implementation regarding content loss computation index. We compute the content loss at layer ReLU3_3. The overall models are shown in Figure 1.

3. Experimental setup and results

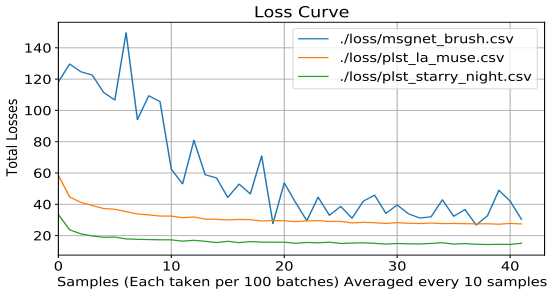
In this section, we perform necessary experiments for the two models implemented by ourselves. We first describe the detailed training procedures and the choice of hyperparameters. Then we perform quantitative and qualitative analysis regarding the experiments. Specifically, we will compare and analyze the output images derived by the two models from different aspects.

3.1. Experimental setup

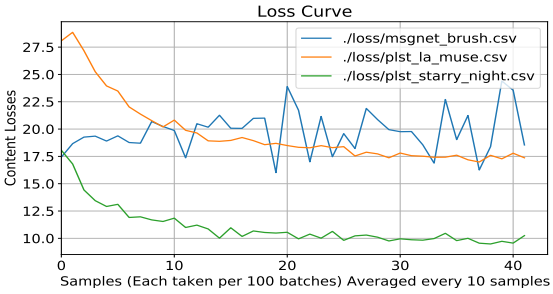
In terms of training, we implement a common interface which integrates the two model designs into one unified framework. Specifically, we train both models under the same number of epochs, batch sizes, optimizer and the same loss function with the same hyperparameters to balance the different loss components (The balancing hyperparameters we used are $\lambda_c = 1.0$, $\lambda_s = 7.5$, $\lambda_{TV} = 1.0$). As a side note also mentioned in previous section, in the original paper, [13] claimed to use the same loss as in [5] but the authors' released code uses ReLU2_2 instead of ReLU3_3 for the content loss which is inconsistent with the paper description. In our implementation, we use ReLU3_3 and we make sure that we use the same loss network for both models such that a fair model comparison can be performed.

We use the Microsoft COCO dataset [2] as the content images for training. We implement a data loader to provide an interface to retrieve images. In the content training set, there are 82783 images which are the photos of many different common objects such as chairs and trains. We do not need labels or annotations for our training purpose. Although the

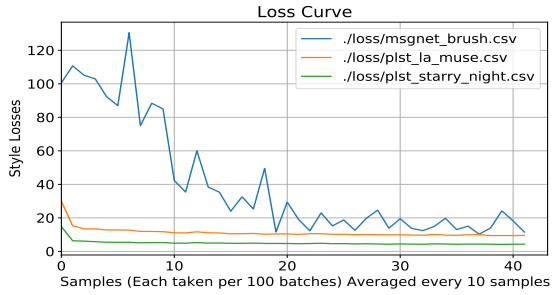
[2] http://images.cocodataset.org/zips/train2014.zip



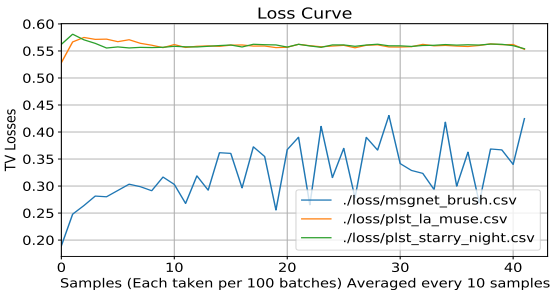
(a) Total Loss. The sum of the rest of the three loss.



(b) Content Loss.



(c) Style Loss.



(d) TV Loss.

Figure 3: The losses recorded during training (Over the 2 epochs). The losses are recorded for all three models where the *msgnet_brush.csv* refers to the loss of MSGNet and the other two refer to the losses of the PLST trained with Figure 2.a and Figure 2.b respectively.

Run-time performance comparison		
Model	Time (ms)	FPS
PLST	8.46	118.20
MSGNet	13.99	71.48

Table 1: Run-time Comparison

dataset also has a validation set, we do not use it since the training set is sufficiently large for style transfer purpose. For each image in the dataset, we resize it to the resolution of 256x256 and normalize it through ImageNet variance and mean. The style images are selected from image dataset wikiart.org[3]. During training, we pre-process the style images in the same way as the content images. Such pre-processing techniques facilitate neural network training and are a common practice in addressing computer vision tasks.

The training is done using an RTX2080 GPU. We train three models in total. One model is for MSGNet with 9 distinct style images (As opposed to 100 style images in the original paper evaluation for training speed considerations), and the other two are for PLST with two different style images which are contained in the aforementioned style image set. The two style images are presented in Figure 2. For MSGNet, we also resize the style images in different iterations according to [13] so it can generalize better for brush size control. We used a batch size of 4 and Adam optimizer [6] with default parameters. The total training time is roughly 4-5 hours for each model.

3.2. Quantitative Analysis

To compare the performance of the two model designs, we recorded the training losses for the three models we trained as in Figure 3. We can see that the style loss and the content loss starts to reduce quickly at the beginning and starts to converge at around 20000 iterations. The TV loss on the other hand contributes less comparing to other loss components, and it is only observed to reduce a relatively small amount as we train for more epochs. For the total loss training curve, we see that the overall training loss is less stable for the MSGNet while PLST is rather stable when trained with either style image. The MSGNet converges slower and PLST converges much quicker. We believe this is due to the fact that multiple style images are used in MSGNet, which result in the unstable weight updates during training phase. However, overall MSGNet demonstrates that it is able to learn the different styles to produce different transformations.

To compare the run-time performance post-training, we evaluate the forward inference time on the same hardware and the performance for each model is shown in Table 1.

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

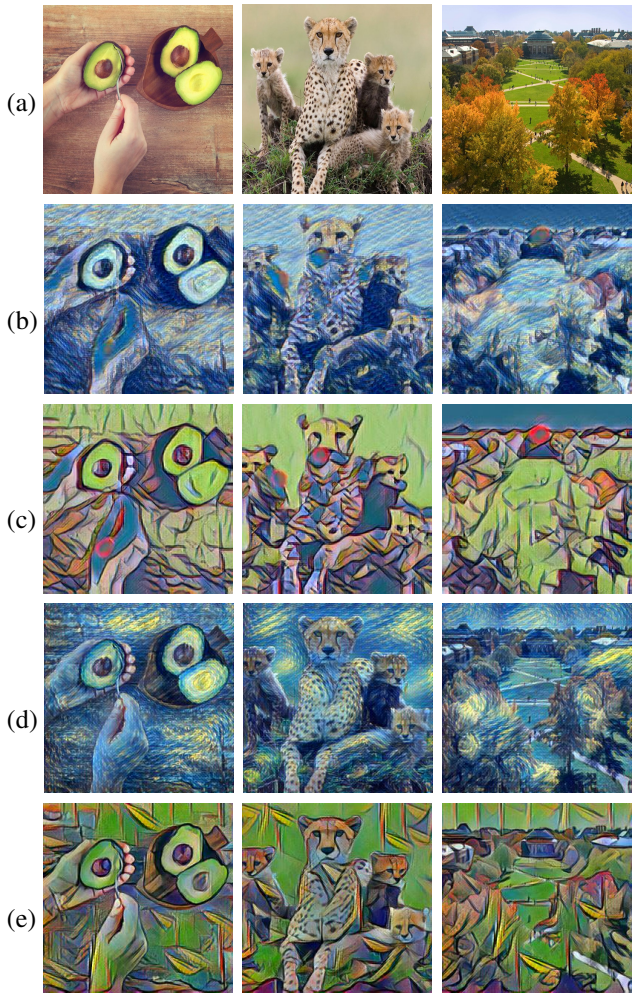
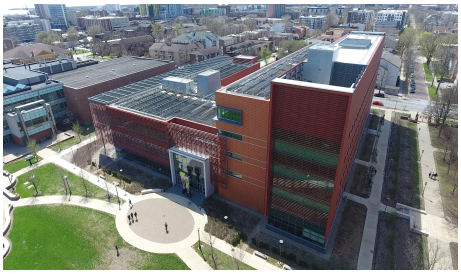


Figure 4: The output for three models we trained.
(a) The original content images.
(b) The output image for MSGNet with Starry Night as input style.
(c) The output image for MSGNet with La Muse as input style.
(d) The output image for PLST trained with Starry Night.
(e) The output image for PLST trained with La Muse.

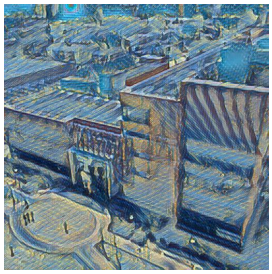
Opposing to what [13] claimed about having similar evaluation performance with [5], we observed a relatively large run-time performance gap that PLST has 118.20 FPS and MSGNet has 71.48 FPS. However in the original paper of [13], the author observed both designs provide roughly 92 FPS. [4]

[4] Our network is implemented effectively the same as MSG-Net-100 in [13].



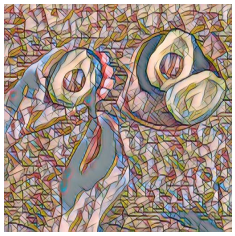
(a) Original high resolution content image

(b) Transferred Image from PLST



(c) Transferred Image from MSGNet

Figure 5: Transferred Image for High resolution content image with Starry Night style. The original image is resized to the highest resolution that can be passed through the network on our GPU.



(a) Style image resized to 256x256

(b) Style image resized to 512x512

(c) Style image resized to 768x768

(d) Style image resized to 1024x1024

Figure 6: MSGNet with different brush sizes.

3.3. Qualitative Analysis

In this section, we discuss the differences between the different models from the perspective of the generated style transferred images.

We first select three content images that are not used during training and the two style images that we used to train the two different PLST models as well as our MSGNet model. The output image for both models are shown in Figure 4, where images in first row correspond to the original content images and the outputs from different models with different styles are presented in the following rows. Here we observe that, given the same amount of training time, the output from the two PLST models are aesthetically better than the result of MSGNet and the details are more noticeable and are more consistently integrated with the style. For example, we can compare the images from Figure 4.(b) with Figure 4.(d), which are the output images generated by the two models with the same style image. Figure 4.(d) captures the color more faithfully than Figure 4.(b) and arguably looks closer to the original Starry Night painting. Specifically, comparing the two images of the main quad, the output image by PLST is much more fine-grained than the image produced by MSGNet. In PLST results, we can observe fine-grained details such as the pavement and even pedestrians. This is also true for the La Muse style if we compare row (c) and (e): The image generated by PLST keeps more features in content image such as spots in Cheetah's body while for MSGNet, the output image does not have clear spots in the cheetah's body.

We also use a high resolution image as the content image with the Starry Night style to illustrate the differences at a higher resolution. We resize down the original high resolution image to the highest resolution that our GPU would allow. In Figure 5, for the same high resolution input image, it is hard to argue which one necessarily looks better (Which happens to be a challenge for most style transfer tasks as it is difficult to argue quality of the outputs). But what we can say for sure is that PLST also preserves much more details in this scenario with more vivid colors while MSGNet destroys quite some details. For some features such as the grid outside the buildings, PLST keeps those features very well but for MSGNet, these parts are warped. Overall we can say that PLST preserves better details than MSGNet. Although this doesn't necessarily imply MSGNet is worse and could potentially be mitigated by more parameters and longer training, it suggests that with the many styles that MSGNet needs to learn, it might be hard to learn a specific style as good as a single style transfer model.

In the next experiment, we experiment with the brush size control as discussed as one important feature of [13]. The brush size control is done through resizing the style image before passing that into the model. Intuitively, when the style image is resized to a larger resolution, the brush size should increase as all features in the style image are scaled up. Since the output image needs to match the statistics of the style image, the scale-up effect should be reflected in

(a) Two Horses By Franz Marc

(b) Corresponding style-transferred Avocado

Figure 7: A failed generalization case for MSGNet. The left hand side is a style image that is relatively different from all the training style images. The content image is also not included during training.

the output transferred image. Figure 6 contains four output images from MSGNet with the same style image scaled to different resolutions. We can clearly see that, while the object (avocado) shape stays at the same scale, the small mosaic that came from the style image is constructed to different sizes. Specifically, Figure 6.(a) has the smallest mosaic while Figure 6.(d) has the largest.

As MSGNet is advertised for multi-style transfer, we also experiment with its ability to generalize by transferring a style image which is unseen during training. In Figure 7, we can see that the output image is hard to be related back to the style image visually. We consider this a failure case. It suggests that although the MSGNet is able to learn the styles quite effectively of the many different training style images, it is hard to generalize to unseen style images in the first shot, especially for the unseen styles which are drastically different from the training style set. It implies that the network still need some training time to get used to the new style although the training can be easily done, since no labels are needed and training images can be easily acquired. But as suggested by the learning curve, incorporating more styles would be a challenge in the sense that it would mostlikely harm the transferring effects for other styles at some level. We believe this essentially becomes the trade-off that MSGNet has to make: It cannot be very proficient in transferring all styles. It can either have more parameters and therefore longer training time to be better at all styles provided, or admit the fact that it can do well on some styles, do badly on others and hard to generalize without more training.

4. Discussion and conclusions

To summarize, we implement the two models (MSGNet and PLST) under a unified framework in PyTorch and conduct a thorough analysis on the two models both quantitatively and qualitatively.

In our quantitative analysis, we observe that MSGNet takes longer to train and harder to converge comparing to PLST. We also find that MSGNet is hard to reach the inference speed mentioned in [13] and might have underestimated the inference speed for other models such as PLST [5].

In the qualitative analysis, we observe that, under the same training length PLST preserves much more details and produce images with better style transfered with higher quality while MSGNet is more versatile and can achieve brush-control easily with a few lines of code added. We further observe that MSGNet cannot generalize well. It cannot produce reasonable good output image if the style image is not seen during training or is drastically different from the training style images. With that in mind, we believe that this is fundamentally a design trade-off that constrains a multi-style network like MSGNet: The network cannot be small and efficient while catering to all different styling details at the same time unless more training time is paid.

Through our evaluation process, we also find that evaluating the result of one single-style transfer model is hard due to subjectivity and lacking qualitative metrics, while evaluating a multi-style transfer model is even harder. Potentially, more interesting questions can be asked about the multi-style transfer model. For example, what are things the network is essentially learning? How large is the conflict when learning different styles, and will it cause learning diverge? Are there any set of styles that are particularly hard to incorporate for the same network, and could there be any training routines to avoid such behavior? Could there be any architectures can more effectively learn multi-styles and generalize to unseen styles? These are all interesting followup questions to answer, and are worthy to explore in future studies.

5. Statement of individual contributions

In terms of current member roles and collaboration strategy, all team members help each other out. The code and data are shared and maintained via Github, and we communicate with each other daily, with a fixed weekly synchronization meeting. Our contributions are as follows:

- *Jiarui Sun* is responsible for model and loss implementations for both PLST and MSGNet. (35%)
- *Licheng Luo* is responsible for designing experiments, training models, implementing training interfaces, plotting necessary graphs, and help Jiarui on model debugging. (35%)
- *Qihao Wang* is responsible for data collection and dataset interface implementation and help Jiarui on model debugging. (30%)

We all worked on the report and the video.

References

[1] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In G. Scott Owen, Turner Whitted, and Barbara Mones-Hattal, editors, *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1997, Los Angeles, CA, USA, August 3-8, 1997*, pages 361–368. ACM, 1997. 1

[2] Alexei A. Efros and William T. Freeman. Image quilting for texture synthesis and transfer. In Lynn Pocock, editor, *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2001, Los Angeles, California, USA, August 12-17, 2001*, pages 341–346. ACM, 2001. 1

[3] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 262–270, 2015. 1

[4] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In Susan G. Mair and Robert Cook, editors, *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1995, Los Angeles, CA, USA, August 6-11, 1995*, pages 229–238. ACM, 1995. 1

[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*, pages 694–711. Springer, 2016. 1, 2, 3, 4, 6

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[7] Vivek Kwatra, Arno Schödl, Irfan A. Essa, Greg Turk, and Aaron F. Bobick. Graphcut textures: image and video synthesis using graph cuts. *ACM Trans. Graph.*, 22(3):277–286, 2003. 1

[8] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III*, volume 9907 of *Lecture Notes in Computer Science*, pages 702–716. Springer, 2016. 1

[9] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3):211–252, 2015. 2

[10] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Bjorn Ommer. A style-aware content loss for real-time hd style transfer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 698–714, 2018. 2

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 2

[12] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *CoRR*, abs/1607.08022, 2016. 2

[13] Hang Zhang and Kristin Dana. Multi-style generative network for real-time transfer. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pages 0–0, 2018. 1, 2, 3, 4, 5, 6