Restylization Network: RestyNet Final Report

Batuhan Özçömlekçi (21703297)

Introduction

In this project, we aimed to achieve a technique that allows us to alter the style of one image to a desired style. In this context, we focused on style transfer where the content of an image is blended with the style of another image. This style transfer process is an image stylization problem. Image stylization problem can be divided into two subcategories which are photorealistic image stylization and non-photorealistic image stylization. The first aim of the project was to achieve non-photorealistic image stylization. Basically, the network planned to be developed will take two images, one as content image and the other as style reference, then the content image will be blended with the style of the other image. The resulting image will contain the core elements of the content image and will appear to be painted in the style of the other image. For this purpose, we reproduced CNN based style transfer networks which are based on [1]. Then, taking the benefit of perceptual loss and instance normalization techniques discussed in [5] and [**?**], the style transfer speed has been increased. Finally, some qualitative stylization results are put on the report for comparison. For the quantitative results, time taken by algorithms' style transfer are compared.

Problem Description

The aim of this project is to reproduce current style transfer methods and measure their performance qualitatively and quantitatively. Although it is not apparent to extract quantitative measurements, one may compare current methodologies in the literature with respect to their time spent for style transfer task. In summary, some of these problems includes:

- How can we transfer the style of an image to another image automatically by a network?
- What are the convolutional methods that enables style transfer task?
- Which convolutional methods yields better stylization result and why?
- What are some time efficient style transfer networks and how they achieve it?

Methods

There are various approaches to the problem of image stylization. A fundamental approach can be utilizing Convolutional Neural Networks (CNN) to achieve this purpose as stated by [1]. In this method a randomized output image is tuned by tuning style loss and content loss steming from the Cost(style image, output image) and Cost(content image, output image) tuples. Throughout the project, CNN based methods are used for the style transfer task. Among these methods, [1] was our starting point and then we introduced perceptual loss [5] and instance normalization [6] techniques to this network.

Initially, we adopt a CNN type of style transfer network which is put forth by [1]. Before moving on with more complex solutions, the team worked on the fundamentals described in this CNN approach to the style transfer problem. In this methodology, there are two cost functions which are content cost function and style cost function. They can be defined as

$$J = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

where S is the style image, C is the content image, G is the resulting image. α and and β are hyperparameters which the programmer needs to tune for this cost function relationship.

Initially, we are randomly assigning the output image G. Then by the following update rule, we are tuning our image:

$$G^{(t+1)} = G^{(t)} + \nabla G^{(t)}$$

Particularly, content cost of a hidden layer l can be calculated as the following. If the activation layers which belongs to content and output image, $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content. Specifically, the content cost as a mean square error can be written as:

$$J_{content}(C, G) = \frac{(a^{[l](C)} - a^{[l](G)})^2}{2}$$

On the other hand, style cost of a hidden layer l can be calculated by describing the style as correlation between activations across (RGB) channels. Specifically, the style matrices (gram matrix) as unnormalized cross covariance for images S and G are:

$$Gram_{kk'}^{[l](G)} = \sum_{i=1}^{n_H^{(l)}} \sum_{j=1}^{n_W^{(l)}} a_{i,j,k}^{[l](G)} a_{i,j,k'}^{[l](G)}$$

$$Gram_{kk'}^{[l](S)} = \sum_{i=1}^{n_H^{(l)}} \sum_{j=1}^{n_W^{(l)}} a_{i,j,k}^{[l](S)} a_{i,j,k'}^{[l](S)}$$

where S and G denote style and output images; i and j are the indices for width and height of the image; k and k' are indices for different channels; n_H, n_W and n_C denote ranges of i,j,k indices at layer l respectively. Finally the style cost function for a single layer and general style function can be expressed as:

$$J_{\text{style}}^{[l]}(S, G) = \frac{1}{\left(2n_H^{[l]} n_W^{[l]} n_C^{[l]}\right)^2} \sum_k \sum_{k'} \left(Gram_{kk'}^{[l](S)} - Gram_{kk'}^{[l](G)}\right)$$

$$J_{\text{style}}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G)$$

The λ in the latter equation are hyperparameters which tune the contribution of each layer (each complex feature) to the style of the output image.

All these equations above defines the CNN style transfer network put forth by Gatys et. al. [1]. The project is implemented this network in PyTorch and Tensorflow frameworks with the help of stated references [9], [8], [10], [4]. Initially, in order to obtain a network for the content loss pretrained VGG network with pretrained weights are utilized for these tasks [10], [8], [2].

Another method used is an extension for what Gatys et. al. proposes. This method is described in the paper, [5]. This method trains a transformer network for the VGG backbone and then instead of comparing pixels for the content loss, compares the feature maps that are extracted by the image

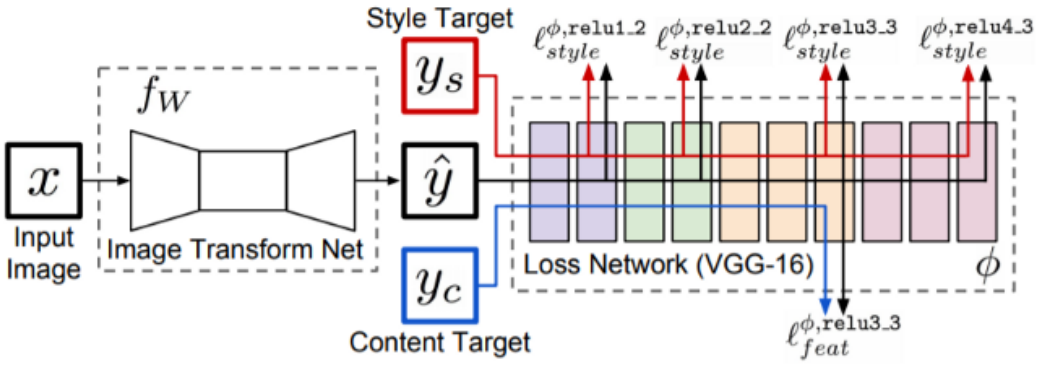


Figure 1: Network for the Perceptual Loss [5]

(a) Content (b) Style (c) Output

Figure 2: Style transfer with Gatys et al [1]

transformation network in the determination of feature loss (content loss). The overall structure of this network can be seen in Figure 1.

Basically, this network replaces the per pixel loss that is present in the Gatys et. al. with the perceptual loss which can be described as:

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

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

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

While training the Image Transform Net and optimizing the output image, Adam optimization algorithm is used with 0.001 learning rate in both of the methods. Instead iof VGG-16, VGG-19 is used as the Loss Network. Transformation network is initially trained with Cityscapes demo video images dataset but it yielded unsatisfactory results. Then this network is trained with approximately 9000 images from the MS COCO val2014 dataset (this is chosen due to time limitations). Finally, instance normalization described in [6] is added to the VGG network for better results.

Results

You can observe both qualitative and quantitative results below

Qualitative Results

(a) Content

(b) Style

(c) Output

Figure 3: Style transfer with Perceptual Loss (Cityscapes)[5]

(a) Content

(b) Style

(c) Output

Figure 4: Style transfer with Perceptual Loss (MS COCO)[5]

(a) Content

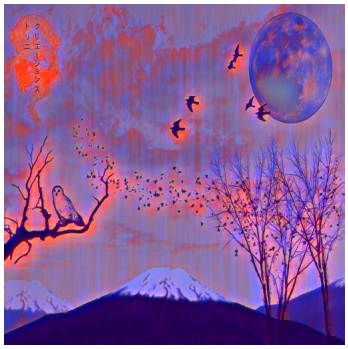
(b) Style

(c) Output

Figure 5: Style transfer with Perceptual Loss (MS COCO)[5]

(a) Content

(b) Style



(c) Output

Figure 6: Style transfer with Perceptual Loss (MS COCO)[5]

As it can be seen, Cityscapes dataset worked poorly with the style transfer task since presumably, it is harder to perform semantic segmentation on this dataset and its distribution doesn't align with neither ImageNet pretrained weights nor MS COCO.

Quantitative Results

The style transfer with classic method Gatys et. al. worked stylized image in 39 s, while the perceptual loss function did the same job in 0.601 s when it is pretrained. Therefore, perceptual loss yields a faster stylization result.
The code of the project can be found on this github page.

Discussion

The style transfer task with these two methods showed that perceptual loss method is faster for single style settings yet it has downsides of training an additional network.

The results of the both networks are not good due to some hyperparameter optimization problems and bugs in the code.

The style transfer task on these CNN networks outputted decent results yet they were poor for computational efficiency. Both of the methods require some time to process arbitrary images. Also, the outputted image doesn't seem realistic. Among CNN methods, the method with perceptual loss outperform the classic neural style transfer network.

Conclusion

Throughout the project, CNN based style transfer networks are tested. The style transfer task on these CNN networks outputted decent results yet they were poor for computational efficiency. Both of the methods require some time to process arbitrary images. Also, the outputted image doesn't seem realistic. Among CNN methods, the method with perceptual loss [5] outperform the classic neural style transfer network [1].

For the future, I would like to move on with a different approach including GANs. For this purpose, VGG-19 and VGG-16 models can be utilized again. For example, the style transfer task can be achieved using a closed form solution as it is described in PhotoWCT [11]. The initial focus of the project is non-photorealistic image stylization yet photo-realistic style transfer experiments can be done as well. For the photo-realistic style transfer experiments, AMT perceptual studies and FCN scores are some of the metrics which the outputs of the network can be evaluated on.

References

[1] L. A. Gatys, A. S. Ecker, and M. Bethge, "A Neural Algorithm of Artistic Style," 02-Sep-2015. [Online]. Available: https://arxiv.org/abs/1508.06576. [Accessed: 16-Nov-2020].

[2] "torch.utils.data," torch.utils.data - PyTorch 1.7.0 documentation. [Online]. Available: https://pytorch.org/docs/stable/data.html. [Accessed: 25-Dec-2020].

[3] dxyang, "dxyang/StyleTransfer," GitHub. [Online]. Available: https://github.com/dxyang/StyleTransfer. [Accessed: 16-Nov-2020].

[4] "Neural Transfer Using PyTorch," Neural Transfer Using PyTorch - PyTorch Tutorials 1.7.1 documentation. [Online]. Available: https://pytorch.org/tutorials/advanced/neural_style_tutorial.html. [Accessed: 25-Dec-2020].

[5] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," Computer Vision – ECCV 2016 Lecture Notes in Computer Science, pp. 694–711, 2016.

[6] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization," arXiv.org, 06-Nov-2017. [Online]. Available: https://arxiv.org/abs/1607.08022. [Accessed: 25-Dec-2020].

[7] TensorFlow Implementation of "A Neural Algorithm of Artistic Style". [Online]. Available: http://www.chioka.in/tensorflow-implementation-neural-algorithm-of-artistic-style. [Accessed: 16-Nov-2020].

[8] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," arXiv.org, 10-Apr-2015. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 16-Nov-2020].

[9] H. Narayanan, "Convolutional neural networks for artistic style transfer," Convolutional neural networks for artistic style transfer - Harish Narayanan, 31-Mar-2017. [Online]. Available: https://harishnarayanan.org/writing/artistic-style-transfer/. [Accessed: 16-Nov-2020].

[10] "Pretrained models," Pretrained CNNs - MatConvNet. [Online]. Available: https://www.vlfeat.org/matconvnet/pretrained/. [Accessed: 16-Nov-2020].

[11] Nvidia, "NVIDIA/FastPhotoStyle," GitHub, 26-Jul-2018. [Online]. Available: https://github.com/NVIDIA/FastPhotoStyle/blob/master/photo_wct.py. [Accessed: 16-Nov-2020].