# Image Style Transfer Using Convolutional Neural Networks

Yuan-Ting Hu, Wenzhao Xu, Mao-Chuang Yeh, Chen Zhang

## 1. Introduction

In this project, we first implement the algorithm of image style transfer using convolutional neural network (CNN), described in the paper “Image style transfer using convolutional neural networks” [1]. Then we modify the algorithm, i.e., the initialization, the use of the CNN model, the use of layers in the CNN model and the loss function, and we give a comprehensive evaluation on the modification in this report.

In this project, we re-implemented this image style transfer method using Tensorflow and explored the effects of different parameters to fully understand this novel approach. We organize this report as followings. We first give the review of literature on style transfer in Section 2 and the method we used in section 3. Then we report our experimental results in section 4, where we discuss the effects of initialization (Section 4.1), the effects of using different layers in the CNN model for style transfer (Section 4.2), the style representation and the content representation by the CNN model (Section 4.3), the effects and variants of loss function (Section 4.4). We also report the results using the GoogleNet (Inception) [9] model in Section 4.5. We summarize the discussion and conclusion in Section 5.

## 2. Review of Literature

Image style transfer has been extensively studied in recent years. Gatys et al. [1] first demonstrate the effectiveness of image style transfer using CNN. They show that using CNN can transfer the style from the artistic image (e.g., painting) as well as the style from the realistic image.

Following the work in [1], [2,3,10] also show remarkable results on tone, season, rendering style and illumination transfer using CNN. One of the advantages of the CNN based methods is that the CNN representation contains rich semantic information which is favorable for transferring style. This research used the feature space provided by the 19 layer VGG network [8] which was already trained with the data in ImageNet Challenge 2014. The output in each layer can be visualized from a white noise image to match the feature responses of the original image by gradient response. Following this approach, Nikulin and Novak [2] explored neural algorithm by using different CNN structures such as VGG16 and demonstrated transferring illumination and season using CNN. The work in [10] implemented a res-net model to fit the training process in the original approach, so the new image can be generated by one feed-forward process of the res-net.

## 3. Methodology

The output of each layer in a pre-trained convolutional neural network captures some aspects of the input image. For example, the VGG_19 model, as indicated by its name, has 5 groups, totally 19 layers (Figure 1). The higher layers in the network captures the information from the

picture (content or feature) from a more general sense, while the lower layers in the network captures the information in a more detailed way.

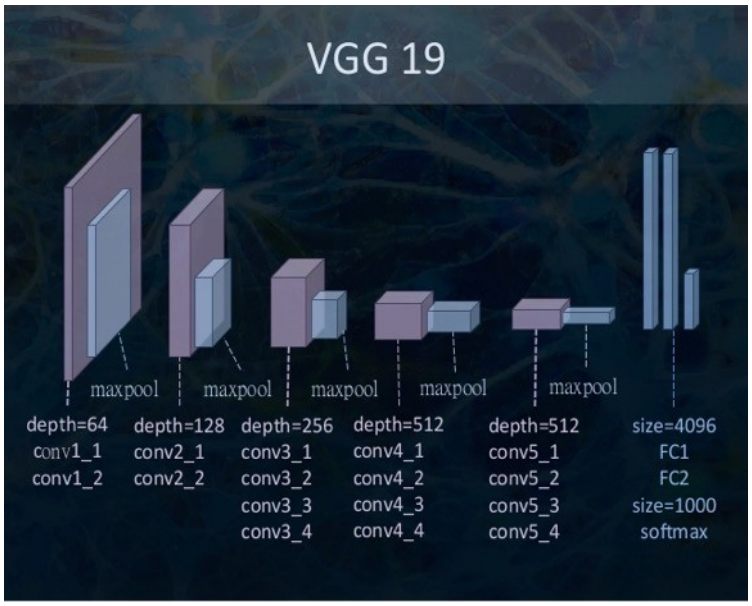



Figure 1. Structures of VGG 19 model
(from http://www.slideshare.net/ckmarkohchang/neural-art-english-version)

The layer outputs are used to construct the content representations (CR) and style representations (SR) of the image. The original paper constructed the content and style representation based on the output of each layer as:

$$CR = F^l_{i,j} \qquad SR = G^l_{i,j} = \sum_k F^l_{ik} F^l_{jk}$$

The content representation $F^l_{i,j}$ is just the output of layer $l$, i.e. the activation of the $i$ th filter at position $j$. The style representation $G^l_{i,j}$ is the correlation between vectorized feature maps by filter $i$ and filter $j$ in layer $l$. The content image and style image are fed into the network to generate the content representation and the style representation. The goal is to train an image from an initial image that generate similar content representation and style representation by minimize the weighted sum of L2 loss of content representation and style representation as

$$L = \alpha \frac{1}{2} (F^l - \acute{F}^l)^2 + \beta \frac{1}{4} \sum_{l=0}^{L} \omega^l \left( \frac{1}{N_l M_l} \right)^2 (G^l - \acute{G}^l)^2$$

, where $\acute{F}^l$ and $\acute{G}^l$ is the content representations and style representations of the image to be trained in layer $l$. $N_l$ are $M_l$ are the number of filters and the size of feature maps in layer $l$. $\omega^l$ is the weights of each layer for style representations, which are the same across layers in our experiments. $\alpha$ and $\beta$ are the weights for content and style representations.

We implemented the style transfer code in Tensorflow. We wrote the code based on https://github.com/anishathalye/neural-style and our code can be found at https://github.com/stormxuwz/styleTransfer. Given two images, one content image and one style image, the basic steps in the code are:

*Step 1*. Preprocess the content and style image, including normalization and resizing the image to content image or model requirements (some pre-train models have specific dimension requirement of input image size)

*Step 2*. Create the graph based on a pre-train model, e.g. VGG 19 and inception, etc

*Step 3*. Calculate the content representation based on content image and style representation based on style image.

*Step 4*. Reset the graph

*Step 5*. Create the same graph but replace the input as a trainable variable ( $I$ ) rather than a place holder in Tensorflow

*Step 6*. Define the loss function as the weighted sum of L2 loss of content representations and style representations generated by the input images (i.e. content image and style image) and ( $I$ )

*Step 7*. Train $I$ to minimize the loss

For inception model, we downloaded the pre-train inception V3 model from "*http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz*" (this graph can be reconstructed in the tensorflow in bluewater). Other pre-trained models can be found at "*https://github.com/beniz/deepdetect/issues/89*" (protocol buffer file) or "*https://github.com/tensorflow/models/tree/master/slim*" (meta file), which requires the newer tensorflow to reconstruct.

During the implementation, we had some tips for using Tensorflow as:

(1) Tensorflow is convenient for calculating gradient automatically. However, it is not flexible in modifying the graph. Nodes can be appended to the existing graph but it may not be possible to modify the intermediate nodes. Nodes that appended should have a proper size that are compatible other tensor size.

(2) The graph saved by higher level APIs such as slim from newer version Tensorflow may not be compatible with the older version Tensorflow

(3) Tensorflow may write a lot of intermediate stuff to disk.

## 4. Results

We explored the how different initial image, different layers, different loss function and different models will affect the final images. The default layer for content representation used is "Conv 4-2" the same as in the original paper. Layers "Conv1-1", "Conv2-1", "Conv3-1", "Conv4-1" and "Conv5-1" are used to construct the style representations. We used Adam optimizer.

### 4.1. Effects of Initialization

In this section, we explore the effects of initialization. We test three different types of initialization, i.e., initialized with 1) random noises, 2) style image, and 3) content image. Results with the three types of initialization are shown in the Figure 2. The initialization can influence the converged output.

(a) Content Image

(b) Style Image

(c) Random noise

(d) Style image

(e) Content image

Figure 2. The effect of initialization. (a) is the content image and (b) is the style image. (c), (d) and (e) are the generated images initialized by random noises, style image and content image, respectively.

We can observe that initialized with style image fails to converge to a solution which contains enough contents from the content images. This may due to the style and content difference between the input content and style images. In this case, the result initialized with the content image shown in Figure 2 (c) can remain more meaningful semantics in the content image comparing to the results initialized with random noises shown in Figure 2 (d) and style image shown in Figure 2 (e).

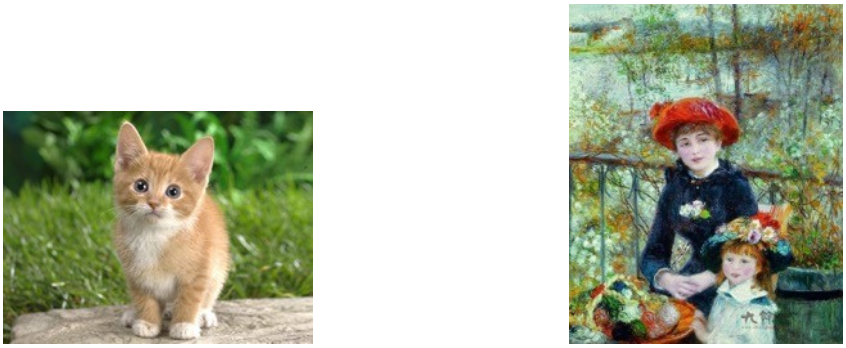
For more comparisons, we show 2 more examples in Figure 3 and Figure 4 to demonstrate the effects of initialization. In general, we would suggest to initialize with random noises or content image instead of style image.

(a) Content Image

(b) Style Image

(c) Random noise (d) Style image (e) Content image

Figure 3. The effect of initialization. (a) is the content image and (b) is the style image. (c), (d) and (e) are the generated images initialized by random noises, style image and content image, respectively.



(b) Content Image (b) Style Image

(c) Random noise (d) Style image (e) Content image

Figure 4. The effect of initialization. (a) is the content image and (b) is the style image. (c), (d) and (e) are the generated images initialized by random noises, style image and content image, respectively.

### 4.2. Effects of using different content layers

We conducted experiments to exam the ability of different content layers in capturing the content of the original image. We first selected one layer from each subgroup in the VGG19 model. The results are shown in Figure 5, 6 and 7.

Content Image Style Image

Conv 1-1 Conv 2-1 Conv 3-1 Conv 4-1 Conv 5-1

Figure 5. Example 1 using different layers as content representations

Content Image

Style Image

Conv 1-1 Conv 2-1 Conv 3-1 Conv 4-1 Conv 5-1

Figure 6. Example 2 using different layers as content representations

Content Image

Style Image

Conv 1-1 Conv 2-1 Conv 3-1 Conv 4-1 Conv 5-1

Figure 7. Example 3 using different layer as content representations

For the example in Figure 2 (cat), convolution 2_1, 3_1 and 4_1 all produce relatively acceptable results. But for the samples in Figure 5 (Mona Lisa) and Figure 4 (city), only convolution 4_1 has good result, with content and feature well-combined.

Now that we know the sub-group 4 has the best result in the output, we looked at the effects of individual layers in the sub-group 4 (Figure 8).

Content Image

Style Image

Conv 4-1

Conv 4-2

Conv 4-3

Conv 4-4

Figure 8. Results of experiments using different layers in Group 4

When we look at the behavior of using the four layers in sub-group 4 (Figure 8), we can see that convolution 4_1 has the best results. We can see the content of the picture with the desired feature style. But for images with convolution 4_3 and 4_4, the content is not identifiable. Therefore, we reached a pr8eliminary conclusion that convolution layer 4_1 best preserves the content of the picture to an extent that it still combines well with the feature style we're interested in. The paper used Conv 4-2, but it seems 4-1 may be better for this content image

### 4.3. Reconstruction from content and style representation

Recall that in the loss function we have $L_{total} = \alpha L_{content} + \beta L_{style}$ . We assign the value of β to be 0, so that we are only considering the loss due to content. In this way, we can look at how well the content layers can reproduce the original picture. Starting from a random image, the reconstructed image using different layers as content representations are shown in Figure 9.

Conv 1-1

Conv 1-2

Conv 2-1

Conv 2-2

Conv 3-1

Conv 3-2 Conv 4-1 Conv 4-2 Conv 5-1

Figure 9. Reconstruction from only content representations

Similarly, we can study what kind of images best reflect the style representations by setting $\alpha = 0$ (i.e. only to minimize style loss from a random image). We use the default style layers to calculate the style representations.

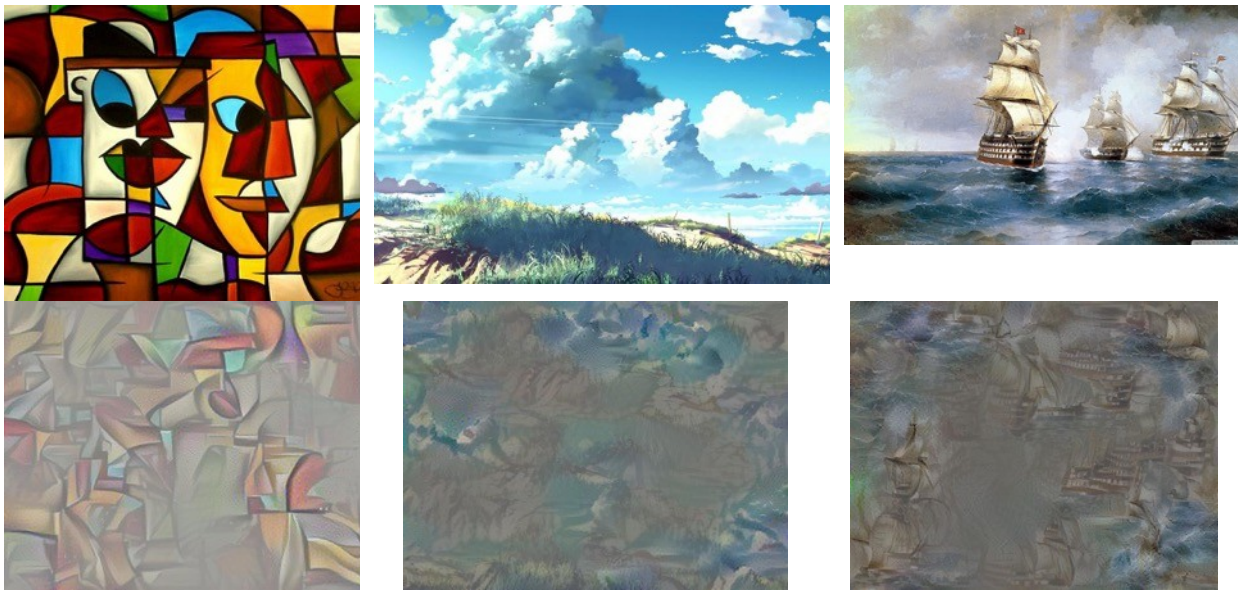


Figure 10. Reconstruction by only minimize style loss (upper image is the style image; lower image is the reconstructed image)

By only minimize the style loss, the output images (shown in Figure 10) reveals a pattern that the style representations not only preserve the style, but also contain information of the content of the style image. For example, for the rightmost image, the reconstructed image had some patches of “ships”, which are the content of the image.

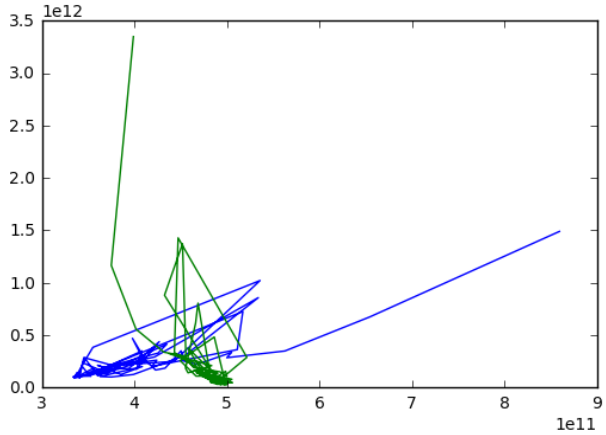
### 4.4. Results of different loss function (the loss multiplication vs loss addition)

In most of the style transfer paper, they get the best result by controlling the relative weight between the content loss and style loss in the loss additive model. Actually, for different style the best relative weight may be not the same. Therefore, it will be a big task to find the optimal weight of each style, which is a big disadvantage of the loss additive model. In our experiment, we choose the weight ratio as 1000 (i.e. $\beta / \alpha = 1000$), which is fixed and may be not the best.

The motivation of why we consider the multiplication of content and style loss:

1. we are curious about how tensor flow works with loss multiplication in back propagation. In most cases, the loss we deal l with are additive.
2. Because in the loss-multiplicative model: $L_{total} = L_{content} * L_{style}$ , we have $dL_{total} = dL_{content} * L_{style} + L_{content} * dL_{style}$ , the variance of the style loss automatically multiplied by the content loss so that the effect of style can be comparable to the content. We expect the loss multiplication will solve the weight-tuning problem.

The following plot is the lot of how the content loss(x-axis) and style loss(y-axis) evolve with iterations. The blue line represents the trajectory of the additive total loss, which start from the right side of the plot and then decreases the losses with iterations. The green line shows the multiplication of losses will eventually sacrifice some increase of content loss to reduce the style loss is less than what of additive loss structure. The optimizer is Adam.




Ideally, we expect $L_{total} = L_{content} * L_{style}$ can minimize both content loss and style loss. But there is a possible outcome with either very large content loss but perfect style loss or very large style loss but perfect content loss. That's one of reason why we sometimes cannot get the stylized result in loss multiplicative model. To avoid this problem, we can use the Adam optimizer to improve it.

Furthermore, if we assume the hyper parameter s as the degree of our painting, then we can have :

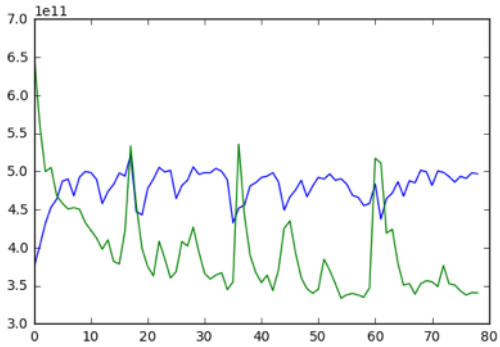
$$\frac{dL_{total}}{ds} = \frac{dL_{content}}{ds} * L_{style} + \frac{L_{content} * dL_{style}}{ds}$$

, and

$$Grandient = \left( \frac{dL_{content}}{ds} * L_{style}, \frac{L_{content} * dL_{style}}{ds} \right) \neq (L_{style}, L_{content})$$

Therefore, we can see the gradient of total loss relative to hyper parameter 's' is not just the gradient for $L_{content} * L_{style} = const$ .

The bellow is the plot of content loss with iteration (every 20 steps), where the green line represents for loss multiplication. Here we can see the content loss in loss multiplicative model is higher and less stable than the additive loss model.



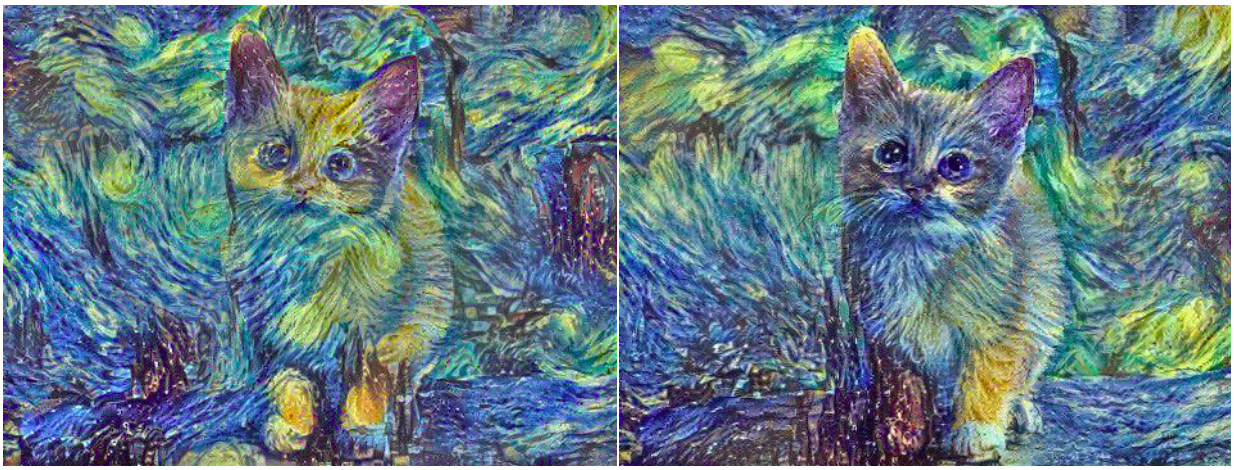

Although, the loss is not critical one to determine a good style transformation, it can help us to understand how loss multiplication sometime works better on optimization of style transform.

The content loss is more less stable in the multiplicative case.

The image comparison of two type of loss arrangements:

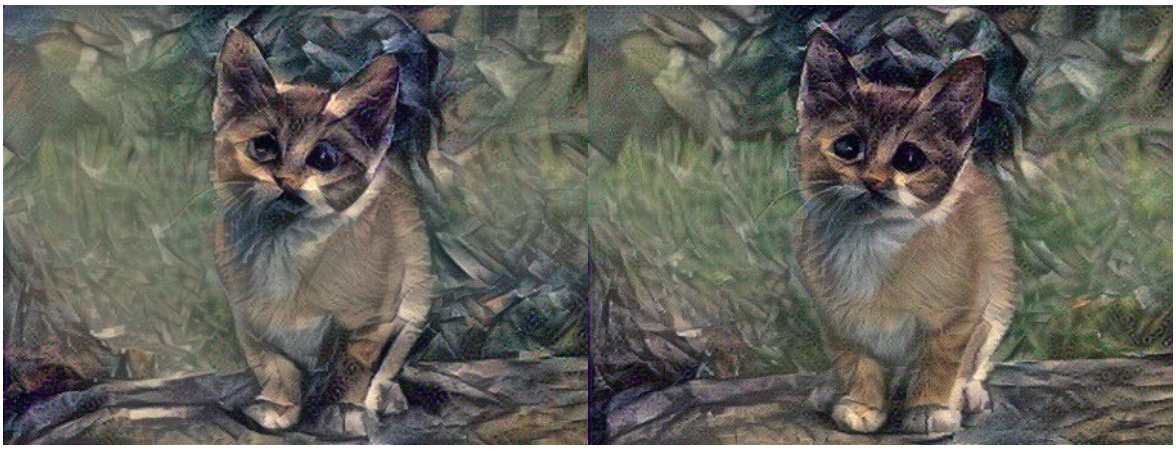
The left is generated by loss multiplication; the right is generated by loss addition.

You can see the left is more stylized than the right; the pattern follows more from style image.



The left is generated by loss multiplication; the right is generated by loss addition.

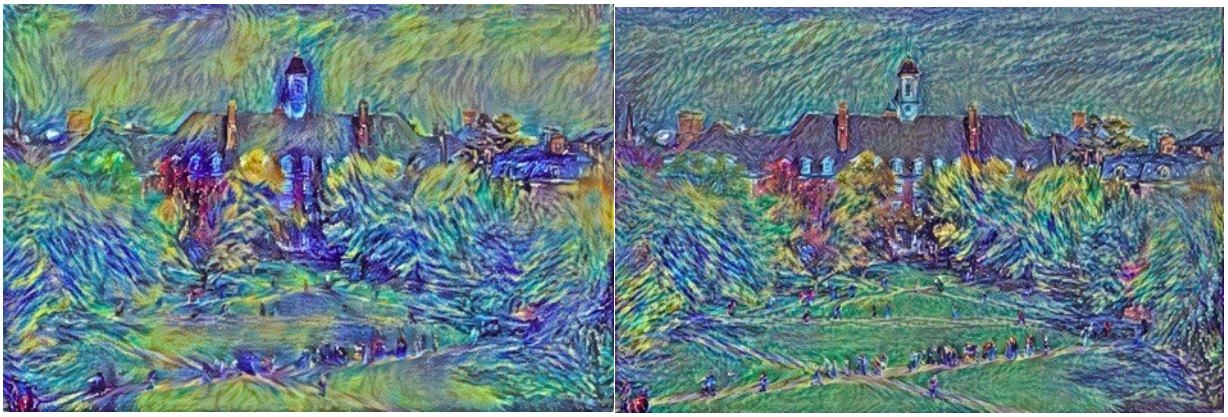
You can see the left is more stylized than the right, where the pattern follows more from style image.



The style images:

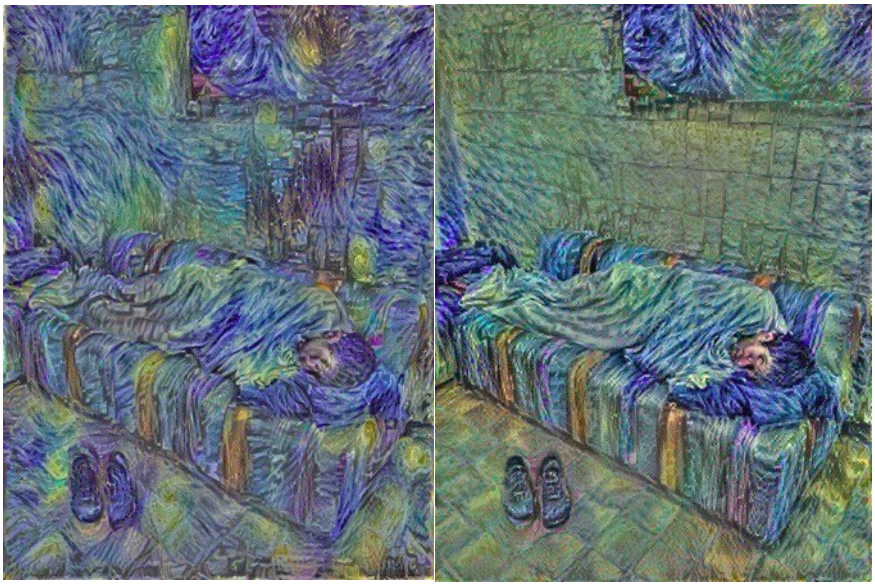

The left is generated by loss multiplication; the right is generated by loss addition.

You can see the left is more stylized than the right, where the pattern follows more from style image.

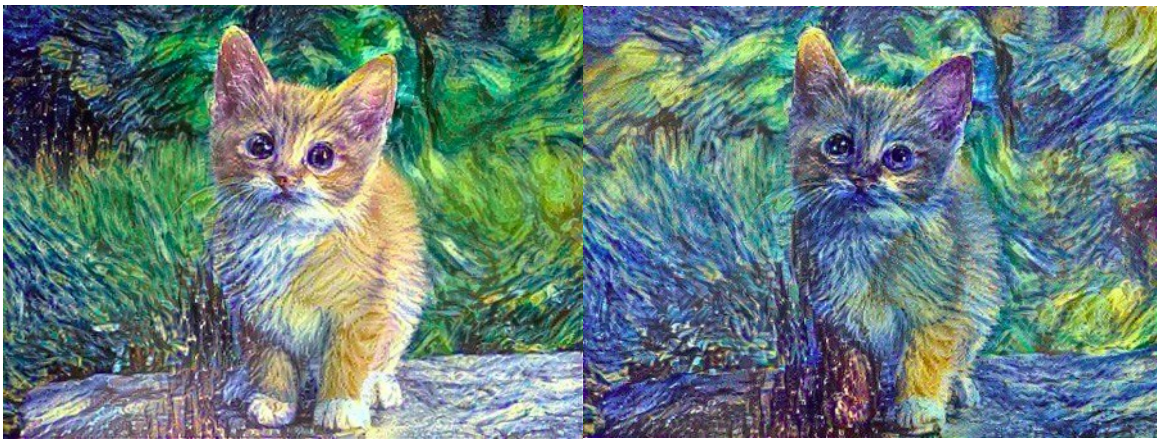


The left is generated by loss multiplication; the right is generated by loss addition.

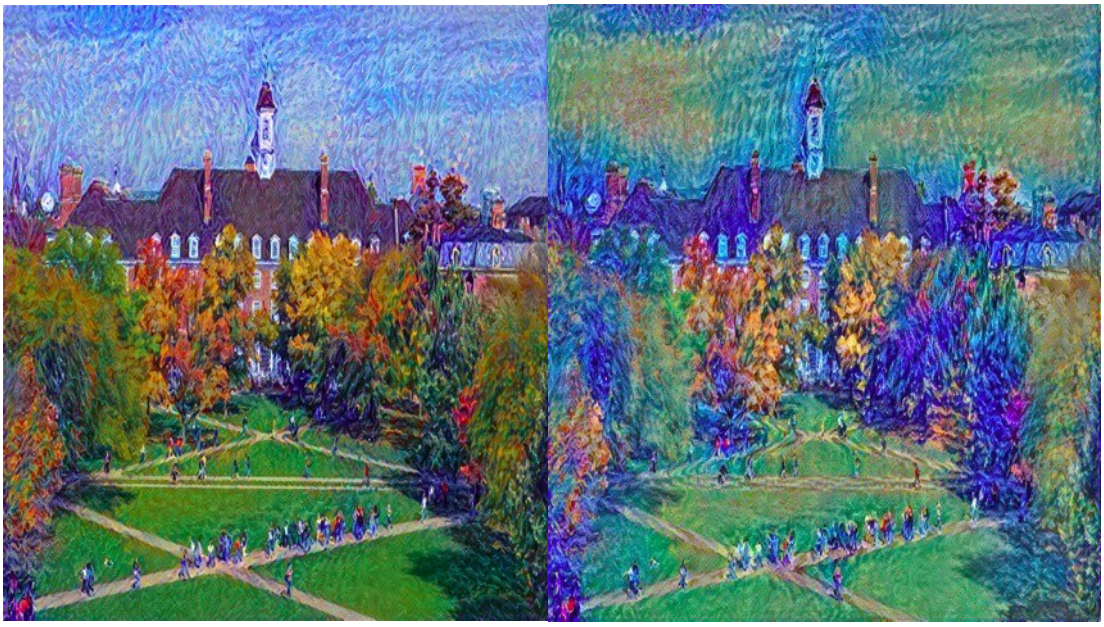
You can see the left is more stylized than the right, where the pattern follows more from style image.



On the other hand, we also put additional RGB L2 loss term to preserving the color. It works good. But we need to know that the color is part of the style. It doesn't make sense if we preserved too much on the color. That's why people do not focus too much on color preserving on style transfer. Here are our results:



The left is generated by the loss with RGB L2 loss.



The left is generated by the loss with RGB L2 loss.

Here is the conclusion about the properties of loss multiplication:

1. Dynamically tunes the weight between content loss and style loss!
2. Follow more creativity from style!
3. Paint faster (better quality at fewer iteration)!
4. Sometime fail to utilize the style transfer with some style
5. Less stable than the additive loss.

### 4.5. Results of the inception model

For Inception models, we use the different layers to construct the content and style representations. (Table 1)

| Model | **Inception V1** | **Inception V3** | **Inception V4** |
|---|---|---|---|
| Content Layer | MaxPool_3a_3x3/MaxPool | mixed_4/join | Mixed_5a/conca |
| Style Layer | Conv2d_1a_7x7/Relu,<br>Conv2d_2c_3x3/Relu,<br>MaxPool_3a_3x3/MaxPool<br>MaxPool_4a_3x3/MaxPool<br>MaxPool_5a_2x2/MaxPool | conv<br>pool<br>conv_4<br>pool_1 | Conv2d_1a_3x3/Relu<br>Conv2d_2a_3x3/Relu<br>Conv2d_2b_3x3/Relu |
| Input image size | 224 * 224 | 299 * 299 | 299 * 299 |
| Model Source | Source 1 | Source 2 | Source 1 |

Table 1. Layers used in Inception models. (Source1: https://github.com/beniz/deepdetect/issues/89, Source 2: http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz)

Table 1 shows the layers we used in each model. The layers in Inception V1 are from https://github.com/fzliu/style-transfer/blob/master/style.py. The results from different initializations are shown in Figure 11.

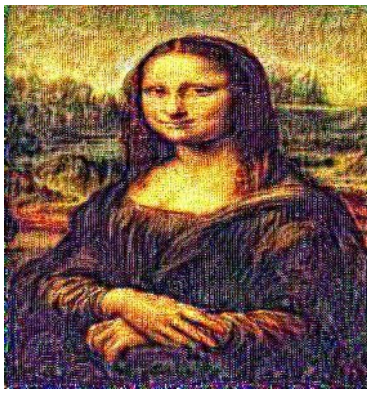
Content Image

Style Image

(a) From content (V1)

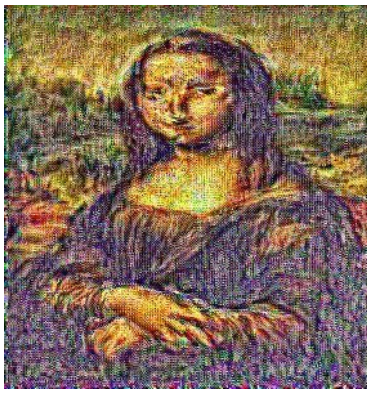
(b) From style (V1)

(c) From random (V1)

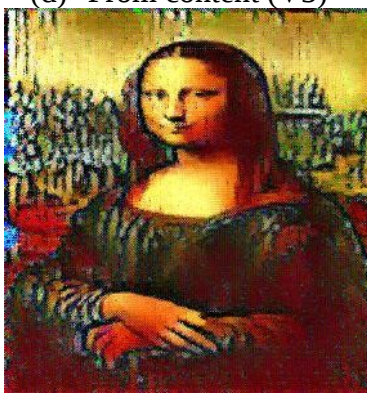


(d) From content (V3)

(e) From style (V3)



(f) From random (V3)



(g) From content (V4)

(h) From style (V4)

(i) From random (V4)

Figure 11. Results from different models using different initialization approaches. (a) to (c) are from Inception V1 model, (d) to (f) are from Inception V3 and (g) to (i) are from Inception V4

The layers chosen for inception V1 models perform better. It successfully transfers some styles to the content image. But for models in V3 and V4, we couldn't find good layers to represent a reasonable content and style representation. V3 and V4 are more complicated than V1 models, so a lot of layer combinations are available.

## 5. Discussion and Conclusions

We re-implemented the style transfer approach in Tensorflow. Through experiments, we found:

(1) Convolution 4_1 in VGG 19 is the best layer to represent the content, not too detailed, not too coarse.
(2) Changing loss function could have different results
(3) Style representations in the paper will also have content parts to some extent.
(4) Initialized with the content image seems to help to converge to a better result
(5) The style representation (the Gram matrix) may not be appropriate for other models rather than VGG

## Reference

[1] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "Image style transfer using convolutional neural networks." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[2] Nikulin, Yaroslav, and Roman Novak. “Exploring the Neural Algorithm of Artistic Style.” arXiv:1602.07188 [Cs], February 23, 2016. http://arxiv.org/abs/1602.07188.

[3] Ahmed Selim, Mohamed Elgharib, and Linda Doyle. 2016. Painting style transfer for head portraits using convolutional neural networks. ACM transactions on graphics (TOG). 2016.

[4] Laffont, P. Y., Ren, Z., Tao, X., Qian, C., & Hays, J. Transient attributes for high-level understanding and editing of outdoor scenes. ACM Transactions on Graphics (TOG). 2014.

[5] Shih, Y., Paris, S., Barnes, C., Freeman, W. T., & Durand, F. Style transfer for headshot portraits. ACM transactions on graphics (TOG). 2014

[6] Shih, Y., Paris, S., Durand, F., & Freeman, W. T. Data-driven hallucination of different times of day from a single outdoor photo. ACM Transactions on Graphics (TOG). 2013

[7] Radford, Alec, Luke Metz, Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434 (2015).

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

[9] Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[10] Johnson, Justin, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution." arXiv preprint arXiv:1603.08155 (2016).

## Appendix

Followings are the results by the modified loss function.