GR5242 Final Project

Yanzhu Chen
yc3511
Columbia University
yc3511@columbia.edu

Guoyong Xu
gx2138
Columbia University
gx2138@columbia.edu

Lulu Wang
lw2836
Columbia University
lw2836@columbia.edu

Xiaoou Su
xs2340
Columbia University
xs2340@columbia.edu

Abstract

More recently, Neural Style Transfer has demonstrated impressive results in instance-based image blending. The method is based on a parametric texture model defined by summary statistics on Convolutional Neural Network(CNN) responses and appears to have several advantages over patch-based synthesis. This project is based on the work of Leon A. Gatys' paper [2], A Neural Algorithm of Artistic Style. We retrained different image network models with adjusted style layer weights and introduced a third component in loss function. Besides, we tried another pretrained neural network ResNet50 to compare with VGG19. Our result showed a successful image synthesis.

1 Introduction

Combining a style image with a content image is considered as a type of texture transfer. Some methods focus on low level image features, which are unable to extract semantic image content. While CNN provides a measure to solve this problem; an image representation derived from a pretrained network improves the image synthesis result. In this work, based on the previous work [2], we compared different modern classification network techniques with adjusted loss function and optimizer.

2 Method

We first implement by using original algorithm in the paper [2] which uses VGG19 as classification neural network and L2 regularization in content loss function. In addition, we change content loss function to L1 regularization and add total variation loss to total loss. Besides, we use Adam instead of L-BFGS optimizer because Adam optimizer performs well in Tensorflow implementation. The gradient descent of updating starts from a random white noise image which is tested to see its impact on final result. At last, we consider change in our model to get feature representation. It means that we extract two content layers in VGG19 instead of one layer. Then, We change VGG19 to Resnet50 with one content layer and eight style layers. The following sections explain some terms involved in this changing process. The comparison of different parameters in the model will be presented in the model comparison section.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2.1 Pretrained Image Classification Neural Network

First, as the paper [2] does, we utilize a pretrained neural network model called Visual Geometry Group (VGG-19). We train image data on 'ImageNet' and global average pooling is applied to the output of the last convolution layer. We choose 1 content layer conv5_2 and 5 style layers conv1_1, conv2_1, conv3_1, conv4_1, conv5_1. The default α/β ratio is 10^{-3} and variation weight is 10. Second, we pretrain model using ResNet50 [3], also a method mentioned in class. We train iamge data on 'ImageNet' and global average pooling is applied to the output of the last convolution layer. We choose 1 content layer conv5_block1_1_conv and 8 style layers conv1_relu, conv2_block1_1_conv, conv2_block3_2_conv, conv3_block1_1_conv, conv3_block3_3_conv, conv3_block3_3_conv, conv4_block6_3_conv, conv5_block2_2_conv. The default α/β ratio is 10^{-3} and variation weight is 10.

2.2 Loss Function

$$L_{total} = \alpha L_{content} + \beta L_{style} + \gamma L_{tv}$$

the loss function we minimize during image synthesis is a linear combination between the loss functions for content, style and total variation respectively, we can smoothly regulate the emphasis on the parameters.

2.2.1 Content Loss

The Neural Style Transfer methods [2] computed content loss as follows. Let X be any image, then $C_{nn}(X)$ is the network fed by X. Let $F_i^l(x) \in C_{nn}(x)$ and $P_{ij}^l(p) \in C_{nn}(p)$ describe the respective intermediate feature representation of the network with inputs x and p at layer l. Then the content distance (loss) formally as:

$$L_{content}^l(p, x) = \frac{1}{2} \sum_{i,j} (F_{ij}^l(x) - P_{ij}^l(p))^2$$

where C_{nn} be a pretrained deep convolutional neural network.

We modify the content loss to be ℓ_1 loss between output graph and the input content image:

$$L_{content}^l(p, x) = \sum_{i,j} |F_{ij}^l(x) - P_{ij}^l(p)|$$

where $P_{ij}^l(p)$ stands for the original image's feature representation in layer l and $F_{ij}^l(x)$ for the output image.

2.2.2 Style Loss

To obtain a representation of the style of an input image, we perform gradient descent from the content image to transform it into an image that matches the style representation of the original image. Let G_{ij}^l be the inner product between the vectored feature maps i and j in layer ℓ:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

We do so by minimizing the mean squared distance between the feature correlation map of the style image and the input image. The contribution of each layer to the total style loss is described by

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

where G_{ij}^l and A_{ij}^l are the respective style representation in layer l of x and a. N_l describes the number of feature maps, each of size $M_l = height * width$. Thus, the total style loss across each layer is

$$L_{style}(a, x) = \sum_{l \in L} w_l E_l$$

where we weight the contribution of each layer's loss by some factor w_l.

We modify the weight to be equal for each layer:

$$L_{style}(a, x) = \sum_{l \in L} \frac{1}{|L|} E_l$$

2.2.3 Total variation loss

The variation loss is not proposed in [2]. The concept is in [4]. The calculation of total variation loss can be represented as

$$L_{tv} = \sum_{i}^{d_1} \sum_{j}^{d_2} \left[\frac{1}{d_1} (x_{i+1,j} - x_{i,j})^2 + \frac{1}{d_2} (x_{i,j+1} - x_{i,j})^2 \right]$$

It was observed that optimization to reduce only the style and content losses led to highly pixelated and noisy outputs. The total variation loss is useful to suppress noise of the output. It measures how much each neighboring pixels differ from each other. Measuring total variation loss will exhibit if there is a meaningful relationship, some sort of "spatial continuity" in the image. This component is used to smooth the output as a regularization method.

2.2.4 Total Loss

The Loss function in the paper [2] defined as:

$$L_{total} = \alpha L_{tcontent} + \beta L_{style}$$

We added total variation loss as a regularization method in favor of spatial smoothness, and separated content loss into type l1 and l2.

$$L_{total} = \alpha L_{content} + \beta L_{style} + \gamma L_{tv}$$

3 Implementation

The Python-Tensorflow based artistic style transfer system is built in Python3, Tensorflow2 with GPU. In order to access the intermediate layers corresponding to our style and content feature maps, we get the corresponding outputs and use the Keras.

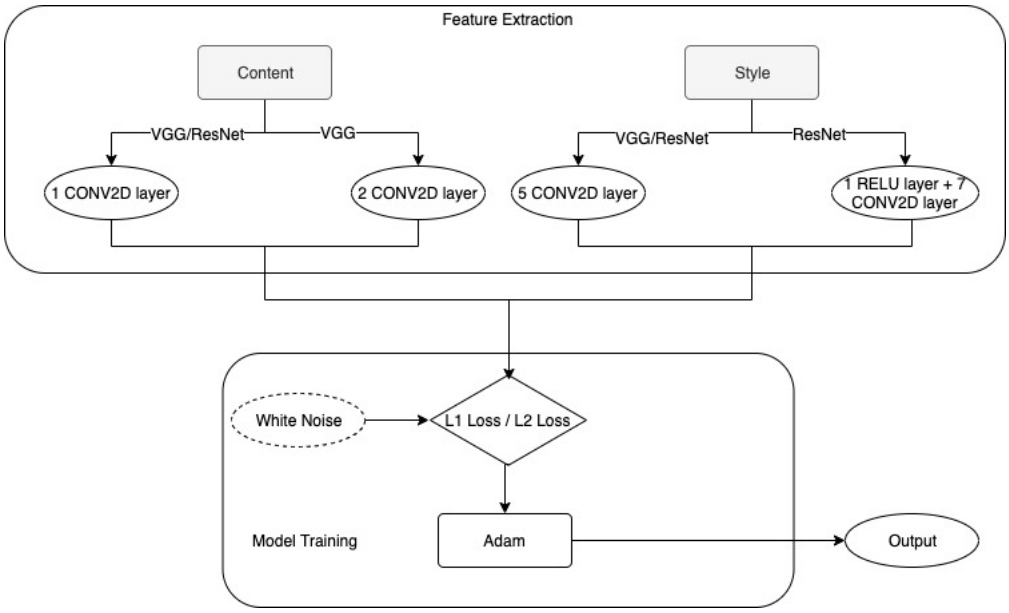


Figure 1: Tensorflow Implementation

We show (Table 1) pre-defined model parameters with 4 different combinations of α/β ratios (Table2) In Table 3, we compare different combinations of layers used VGG19 and ResNet.

parameter type	parameter value
content weight	Table 2
style weight	Table 2
total variation weight	10
Adam learning rate	15
maximum iteration	2000
pooling layer method	avg

Table 1: Pre-defined model parameters

parameter type	parameter value
content weight	1e5, 1e4, 1e2, 1e3
style weight	1e3, 1e3, 1e3, 1e3
ratio	100, 10, 0.1, 1

Table 2: Weight. Relative weighting of matching content and style where ratio = α/β, we compared different combinations of ratios. (See section 4)

		layers used
content representation	VGG19 (1)	conv5_2
	VGG19 (2)	block4_conv4, block5_conv2
	ResNet	conv5_block1_1_conv
	Paper	conv4_2
style representation	VGG19 (1,2)	conv1_1, conv2_1, conv3_1, conv4_1, conv5_1
	ResNet	conv1_relu, conv2_block1_1_conv, conv2_block3_2_conv
		conv3_block1_1_conv conv3_block3_3_conv, conv3_block3_3_conv
		conv4_block6_3_conv, conv5_block2_2_conv
	Paper	conv1_1, conv2_1, conv3_1, conv4_1, conv5_1

Table 3: Pre-defined feature extraction layers

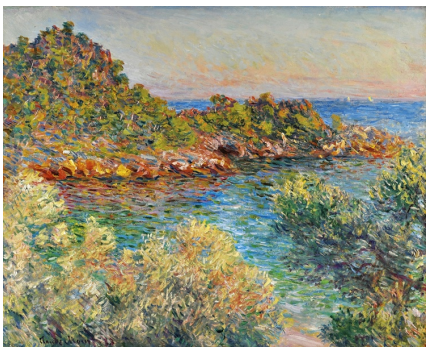


Figure 2: Monet.

Figure 3: Ashin.

4 Model comparison

This section will compare impacts of different parameters. The basic model runs algorithm in original paper [2]. Following subsections present result images with changing parameters respectively. The output here overlays Monet's picture on a famous singer's figure.

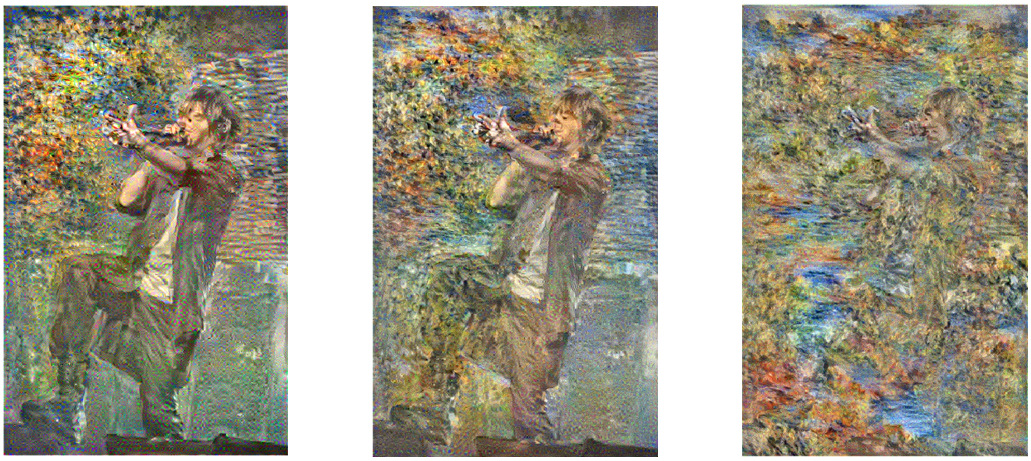
4.1 Basic model

The basic model implements with α/β = 100, L2 regularization, learning rate of optimizer =15. Layers used are denoted as content representation VGG19(1) and style representation VGG19(1,2) in Table 3. The random seed is set to 30. Style layer weight is equally weight for each layer, i.e. 0.2 for all layers. The iteration number is 1500. The following subsections will only change parameters denoted in the title and maintain other parameters same, unless otherwise stated. The figure shows the output of basic model output.

Figure 4: Ashin Monet L2

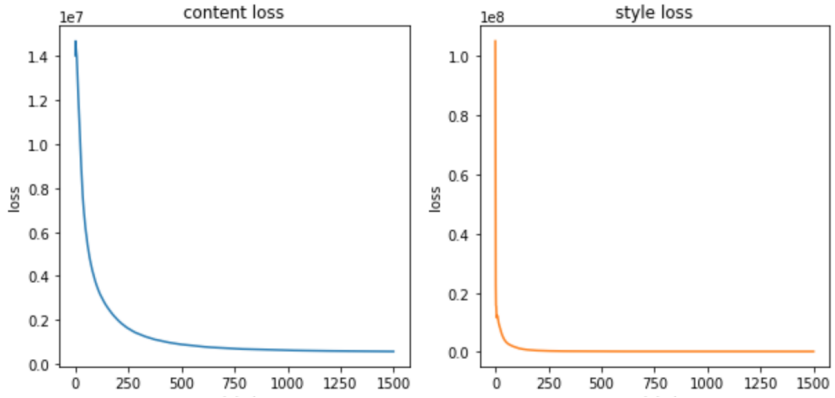
4.2 Loss weights

As mentioned in [2], since the loss function we minimized during image synthesis is a linear combination between the loss functions for content and style respectively, we can smoothly regulate the emphasis on either reconstructing the content or the style. the relative high ratio of content & style loss(α/β) produces a synthesised image with a little stylizing, and the low ratio produces a synthesised image with more obvious stylizing. In order to see the impact of weight change, we set α/β to 10 and 0.1. Since if the ratio is greater than 1, it means that content loss will contribute more to total loss. The gradient of total loss can be decomposed to linear combination of gradient of content loss and style loss. So when implementing gradient descent, the image will update to decrease more in content loss. It means that the image will update more to match content feature. The performance of content loss in different ratios are presented below. It is obvious that content loss decreases faster in ratio=10. From the output image, the content is more obvious in ratio=10. Note that in original paper, the ratio is set to 10^{-4}. However, we tried 10^{-4} but the result is more mosaic and therefore we tried 10^{-1}.

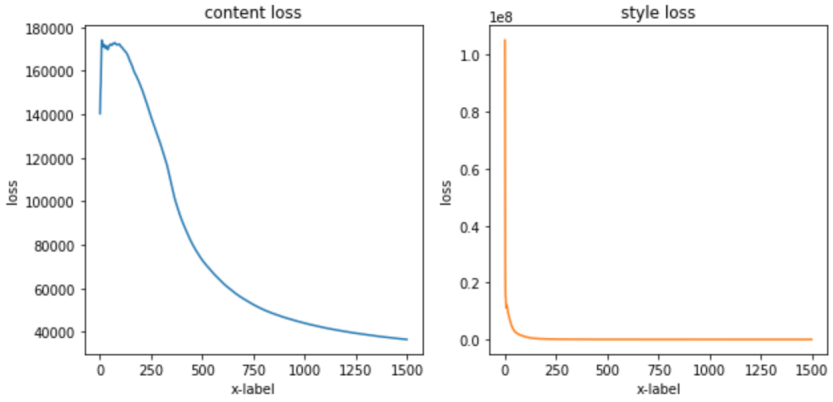


(a) Ashin Monet L2 (b) alpha:beta=10 (c) alpha:beta=0.1

Figure 5: Ashin Style Transfer With Loss Weight Comparison



(a) Loss Plot of alpha:beta=10



(b) Loss Plot of alpha:beta=0.1

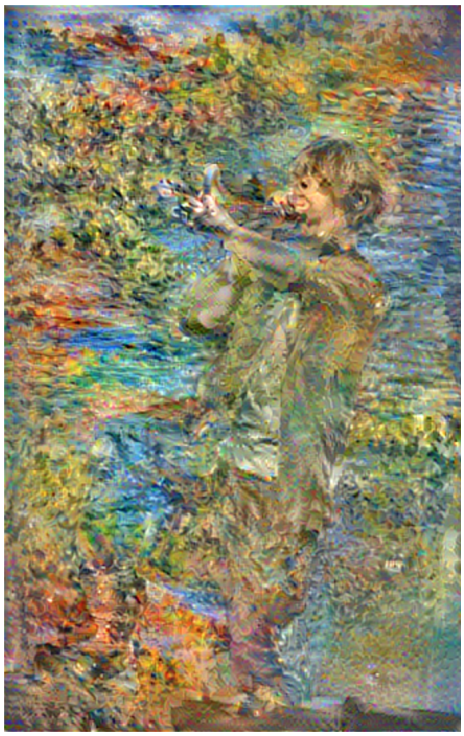
Figure 6: Comparison of α/β ratio

4.3 Variation loss

Variation loss is a regularization term to smooth image, which ensures that the contours of synthesized image are well configured. We try variation loss=0 and 10, the pixels of images turns out to mix better

with variation loss=10. This corresponds to the utility of variation loss which will make neighboring pixels become similar with each other. But if we add more weights on variations loss, for example 100, the texture transfer output dissatisfies us because the contour of content in the output image is obscure. Therefore, we optimize the model by setting variation weight=10.

(a) Ashin Monet L2



(b) Ashin Monet L2 with variation

Figure 7: Ashin Style Transfer With Variation Loss Comparison

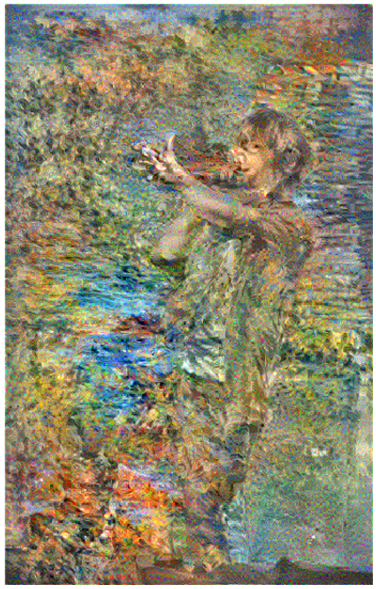
4.4 Iteration

Loss tends to decrease in general when iteration goes to large number, which indicates Image synthesis performs better with greater iterations. When iteration = 0, the image is white noise initialization, and content loss, style loss, total loss have about 1e8 variation to the initialized image. As iteration number becomes large, the speed of loss decrease is fast in earlier iterations(i.e. 500 iterations) while slow after a point. The reason behind decreasing of loss decline speed is that the synthesized image is more similar to original images with greater iterations, which ensures the model converges to right direction. Due to the regularization component variation loss and randomized initialization image, the losses could increase for a very short range, and this phenomenon especially happens in early stage of iterations. To achieve time efficiency and also make sure the output accuracy, we set the default number of iteration to 1500. Looking at the loss plots, the losses decrease slightly after 1000 iterations, and approach 0 at the end of iteration.

4.5 Loss function

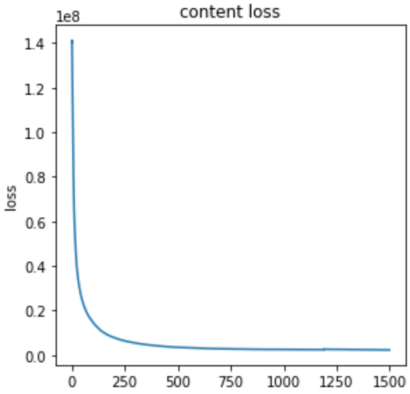
L2 type content loss has larger content loss since it calculates squared error loss. Therefore, with same alpha and beta ratio, L2 type content loss will make total loss larger and the gradient descent will become larger. For similar reason in impact of loss weights, the image will update more to content in L2 to decrease more in content loss function than in L1. In figure 8, L2 figure will have content picture slightly more obvious than L1 figure. Also, from the content loss comparison (figure 9), the L2 has higher value than L1 and decreases more rapidly.

(a) Ashin Monet L2 Loss

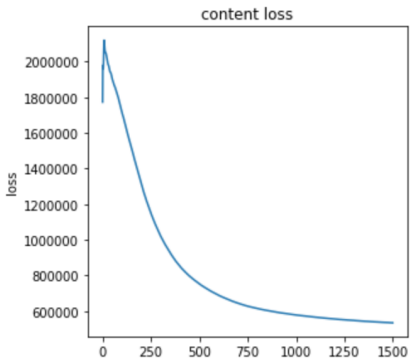


(b) Ashin Monet L1 Loss

Figure 8: L1, L2 comparison



(a) Ashin Monet L2 Loss

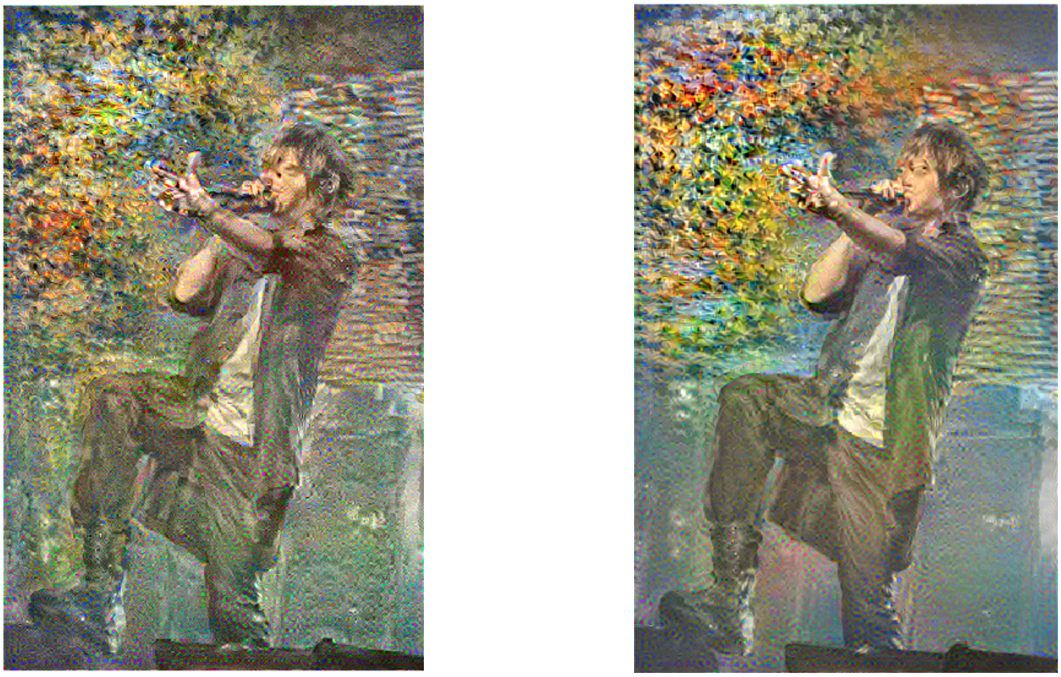


(b) Ashin Monet L1 Loss

Figure 9: L1, L2 content loss comparison

4.6 Convolution layer selection

In other sections, we only use one layer to get content representation. Now we add one more layers (block4_conv4) as show in Table 3 VGG19(2). By comparing output, we find that the content feature is much clearer than before. Therefore, the feature map depends on the the number of layers that are chosen. If user wants the output image to have more content features, one can choose to have more content layers in the model.



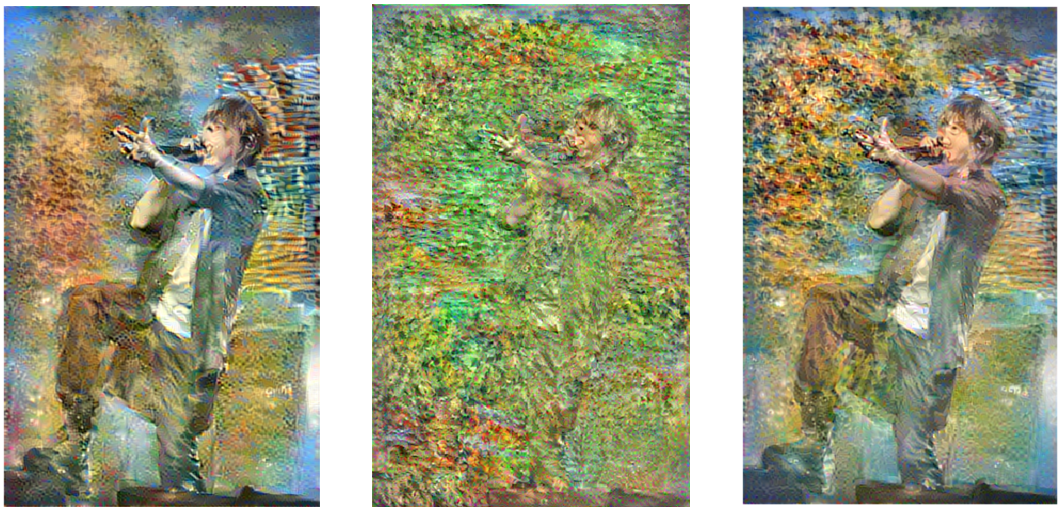
(a) Ashin Monet with 1 content layer

(b) Ashin Monet with 2 content layers

Figure 10: Ashin Monet 1 & 2 content layers comparison

4.7 Style layer weight

Here we try to figure out the influence of style layer weights by comparing mean-weight style layers with weighted style layers. First, let's consider an extreme position that put all weights on the first two level layers or the last two layers, i.e. [1/2, 1/2, 0, 0, 0] or [0, 0, 0, 1/2, 1/2]. Comparing the two images (Figure 11, (a)(b)), we found that the previous one seems have obvious content feature. The color of later image is more like style image. However, the first image has clearer style feature. We can see the shape of trees on the background of the image. Next, compering output images by implement random layer weights. The style layer weights are around [0.491, 0.154, 0.348, 0.001, 0.006]. Although the last high level layers relatively small, the final output improves a lot. The different style appears because the lower layers in extracting style features have more abstract representation. But last two layers will represent style features in more detailed and pixel-related. That is why the change of style layer weight will cause much difference. Therefore, users can choose depending on objective in output with more content or style features.



(a) Ashin weight [1/2, 1/2, 0, 0, 0] (b) Ashin weight [0, 0, 0, 1/2, 1/2] (c) Ashin Random weight

Figure 11: Ashin style layer transfer with different weight (VGG19, L2 loss)

4.8 Gradient descent parameters (learning rate)

Learning rate of optimizer can impressively improve the speed of image synthesis, which means if we set larger learning rate, loss decreases faster and a final output version appears with less iterations. In this project we compare learning rate 5 with 15. In the loss plots of learning rate=5, the content loss and total loss converge well while style loss and variation loss are still on the way to approach 0. As for loss plots of learning rate=15, the content loss and total loss curve look similar to loss plots with learning rate=5, but style loss and variation loss curves are more convex, which differs obviously by converging rapidly to 0. To save running time and ensure output accuracy, we set the default learning rate=15.

4.9 Random initialization

Different random seeds, which is used to create an initial white noise image, has no obvious impact on final results. For a well randomized initialization image, the updating always starts from a huge loss and then converges to 0. As long as the iteration is large enough and learning process is exhausted, the losses will achieve 0 at the end of updating and the effects of initial white noise image is ignored. The random images are scaled to same shape with content and style images before running algorithm.

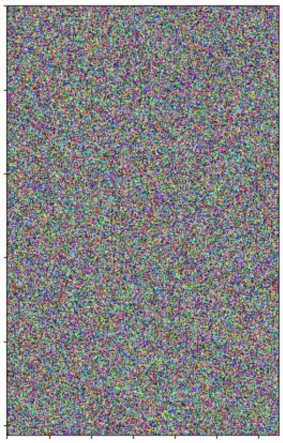
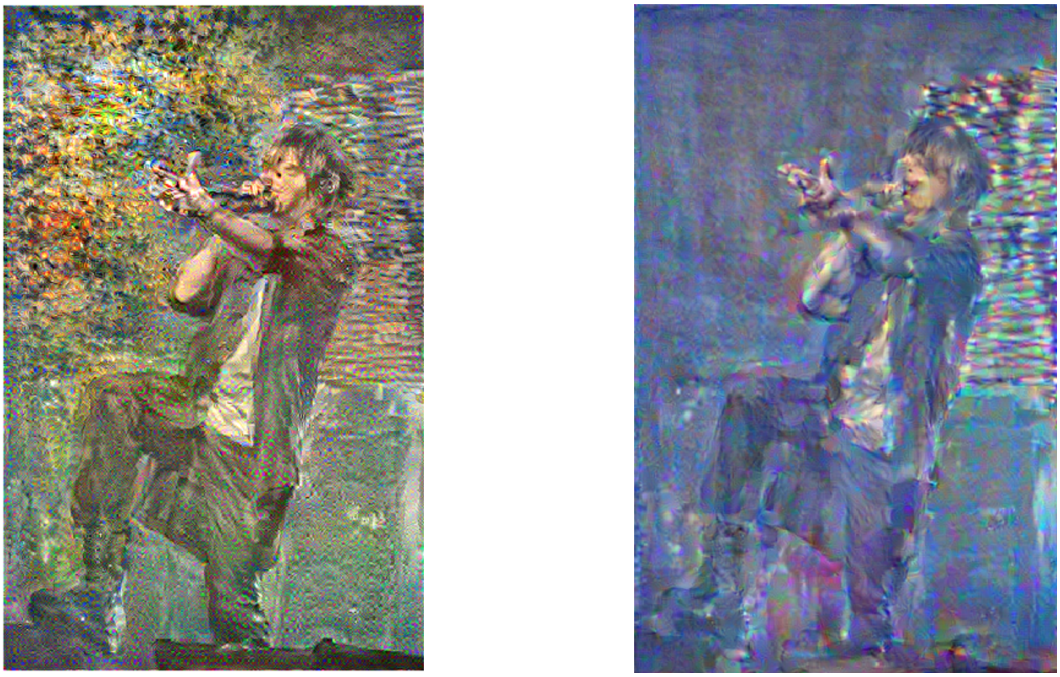


Figure 12: Random Initialization

4.10 Pretrained model

In addition to VGG19, we added another classification network ResNet50 to compared with VGG19. While ResNet50 presents good ability in extracting features and classification (we tried extracting from Resnet with 1 content layers and 8 style layers), ResNet cannot represent style features well which results in a combination of content image and white noise image (figure 13).

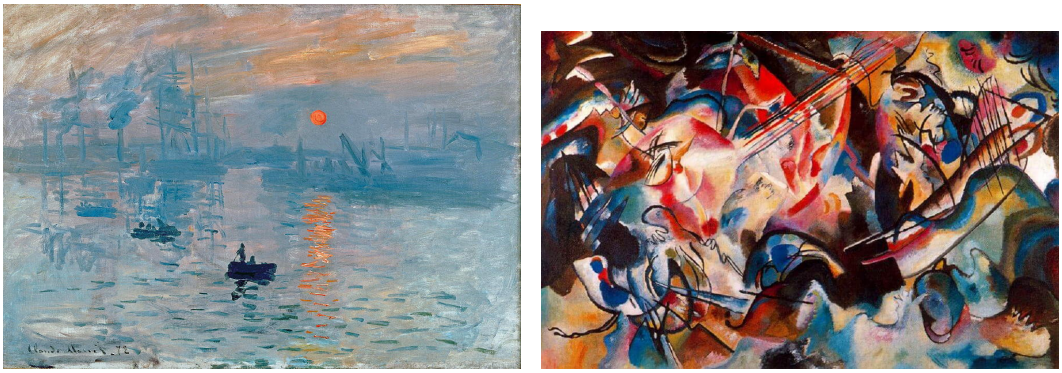


(a) Ashin Monet VGG19 (b) Ashin Monet ResNet

Figure 13: Ashin Monet VGG19 & ResNet comparison

5 Result

We showed, in Figure (15,16,17), neural style transfer using VGG19 (with adjusted α/β) and L2 loss.



(a) Monet (b) Wassily

Figure 14: Two style graphs

(a) Trump

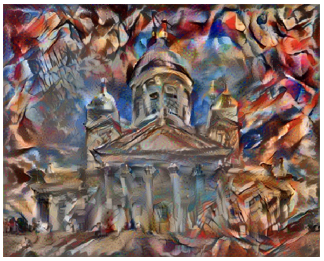
(b) Trump Monet

(c) Trump Wassily

Figure 15: Trump Style Transfer (VGG19, L2 loss)

(a) House

(b) House Monet

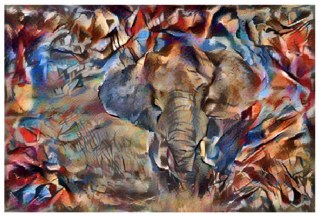


(c) House Wassily

Figure 16: House Style Transfer (VGG19, L2 loss)

(a) Elephant

(b) Elephant Monet



(c) Elephant Wassily

Figure 17: Elephant Style Transfer (VGG19, L2 loss)

6 Discussion

In this project, we tried two pretrianed model VGG19 and ResNet50, while the later is not as good as the former in the neural style transfer. The combined image weights more on content and little stylizing. Even though we tune larger iteration and smaller relative weight α/β (i.e. 0.1), the output is more like a combination of content image and white noise initialization rather than style transfer. We assume that the content representation is suitable in this situation, and the problem of the model lies on style representation. This situation may result from too many layers in ResNet50, more than 100 layers. It may need many different trials in extracting useful style layers or some other loss functions.

Moreover, we use max-pooling layers in VGG19 and white noise initialization in updating. To improve time efficiency, we can utilize average pooling layers in pretrained CNN, and start updating combination from content image or style image.

7 Conclusion

We modified the parametric texture model based on a high-performing convolution neural network. The synthesized image turns out to be a well combination of content image and style image. We discuss parameter tuning effects on results and in the future work, we can develop a better feature representation for pretrained model ResNet50, set VGG19 average pooling layers, and update texture transfer from content or style images.

References

[1] Andrea Vedaldi Aravindh Mahendran. *Understanding Deep Image Representations by Inverting Them*. eprint arXiv:1412.0035, 2014.

[2] Leon A. Gatys. *Image Style Transfer Using Convolutional Neural Networks*. CVPR, 2016.

[3] Kaiming He. *Deep Residual Learning for Image Recognition*. cs.CV, 2015.

[4] Justin Johnson. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. cs.CV, 2016.

[5] NA. *Neural Style Transfer: Creating Art with Deep Learning using tf.keras and eager execution*.

[1] [2] [3] [4] [5]