COMP 3359
Artificial Intelligence Applications Proposal
Group 1

Pan Huijie(3035534622)   Li ShangGen(3035551228)   Yang Yuezhi(3035603033)

Report of tasks achieved

Because we have 3 model implements, the report of tasks achieved will be introduced model by model.
See source code from github https://github.com/yyuezhi/HKU-COMP-3359-Group-I

- ### Image-Optimization-Based Online Neural Methods

1. Understand the idea of the paper
   The idea underlying is basically to optimize the pixels in an image followed by a well-derived loss function which reflects the relative difference in content between new generated images and relative style differences between style images, with the aid of a descriptive network, in which case we adopt VGG 16 here.
   The formula is given by

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

$$J_{\text{content}}(C,G) = \frac{1}{2}||a^{[l][C]} - a^{[l][G]}||^2$$

   J $_{\text{content}}$ is the sum of square 2-norm of the pixel wise difference of feature maps extracted from L layer of descriptive network, between the newly generated images and the content images.

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

   Also,$J_{style}^{[l]}$, where $G_{kk'}^{[l](S)}$ is the gram matrix calculated from a particular layer, given

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

   by the formulae

   that is the inner product among the channels in one particular feature map. This gram matrix value measures the correlation of between different features in one feature map, hence, measures the style of an image. Thus, by minimizing the square difference between the gram matrix value of generated images and style images, we can make the generated images close to the style imaged in terms of the style.

   By changing the appropriate hyperparameterαandβ, we can modify the different importance of content loss and style loss, thus generate images with more styled or more realistic to content images

2. Implement of the networks

   Initialization: Instead of initializing the generated pictures by a completely noise picture, we initialize the images by a weighted sum of content images and gaussian noise images according to hyperparameter noise ratio. This ensures quicker convergence during the experiment and tends to have better resultant image quality, as the pixel value would be similar to its neighbor, making the whole image smoother.

Weight sum of style layers: Instead of uniformly averaging the style loss of each layer, we assign each layer style loss a weight and calculate the weighted sum of them. The detail justification is explained in the preliminary experiment section.

Total variation loss: The total variation loss term is an extra regularization term in the loss function, which penalizes the loss function when there are too many higher frequency artifacts or checkerboard noise within the function. It is actually an edge detector which detects the appearance of edges, that is, sharp changes in pixel value in the images. Qualitatively speaking, an increase in the total variation loss weight could remove the rough texture of the image and the resultant image looks much smoother.

- Model-Optimization-Based Offline Neural Methods

1. Understand the idea of the paper
   This method implements the idea developed by Dmitry Ulyanov's paper in 2016: Texture Networks: Feed-forward Synthesis of Textures and Stylized Images.

   This method is developed upon Gaty's idea on images based optimization model.

   Instead of directly performing optimization on pixels of generated images, which takes a longer time for a single image to be generated, a generative network is built and trained so that it would take the inputs of a content image and several noise images and then generate an image of the desired content.

   The training process of this network is more complicated than Gaty's idea. A separate evaluation network called descriptor network is used to calculate the loss function. Usually, a well-trained image classification network, like VGG 19 trained on ImageNet is used as this descriptive network. This network takes input of the generative images, content and style images separately and calculates the loss using exactly the formulae given by Gaty's idea. Then, the generative network, instead of the descriptive network is updated through backpropagation from the loss function.

   The generative models follow a pyramidal architecture shape, that is, each noise image input of different shape, after going through a series of convolutional networks would be upsampled and concatenate the other noise images of the same shape by channels. This process is done repeatedly and final merges to one single tensor and converted to generated images of desired shape. The detailed architecture is given in the following diagram.

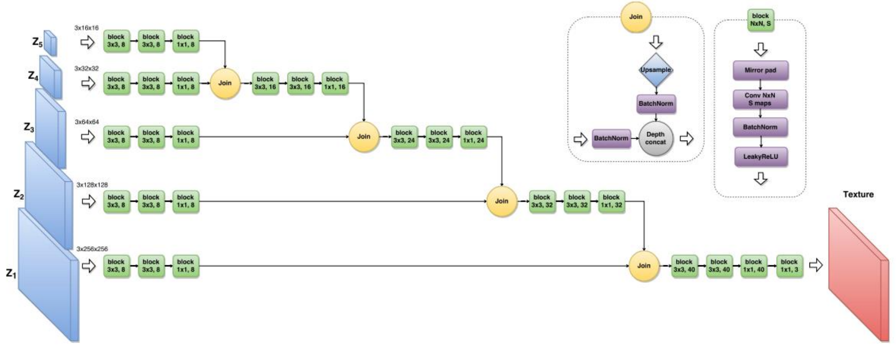
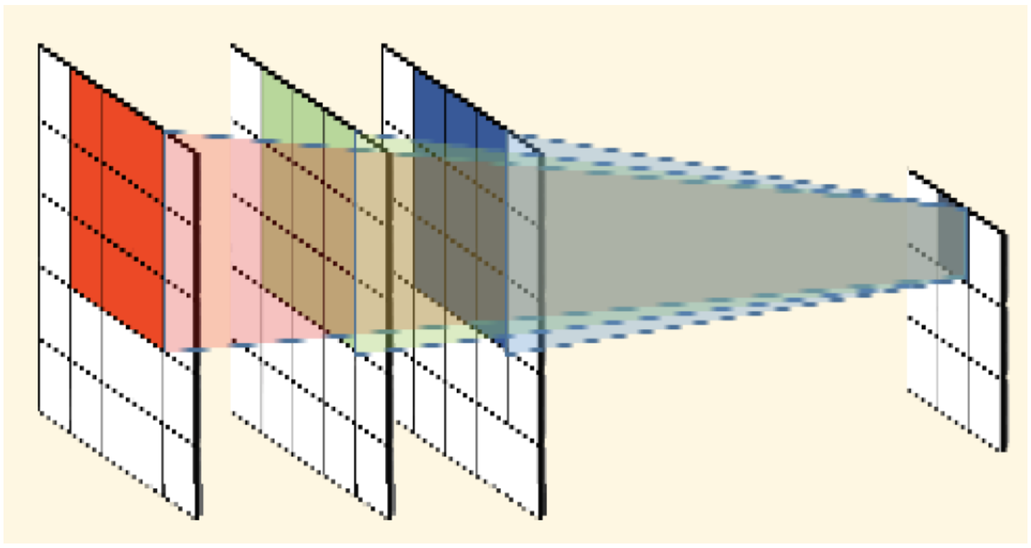


Figure 8. A more exact scheme of the architecture used for texture generation in our experiments.

2. Implementation detail

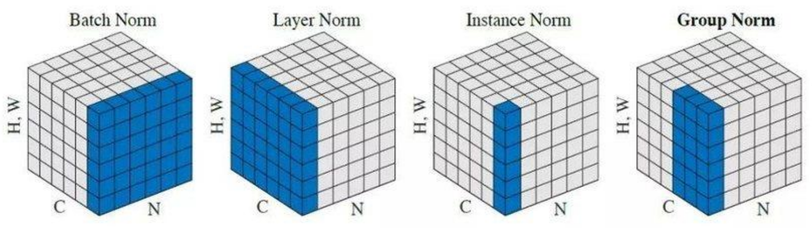
Circular Padding

Circular padding is used during the convolutional layers of the generator network which essentially wraps the image from top to bottom, from left to right, to preserve the shape of the images.



- Instance Normalization

Instead of batch normalization layer, instance normalization is used in the generative network. Another paper written by Ulyanov suggests that instance normalization (normalizes pixels across the channel) outperforms batch normalization (normalizes pixels across the batches of images), in both speed and quality, during the training when the number of images per batch is relatively small.



3. Implement of the networks

Due to the tight timeline, our group still needs some time to debug the whole implementation and train the network. We plan to train two to three networks with each one that could produce images of particular style. The style images, which only requires 2-3 pieces, would be from great artwork of famous artists. The content images would be retrieved from a small portion of ImageNet data.

- Multiple-Style Per Model Method

1. Understand the idea of the paper (StyleBank)

Before the emergence of neural networks, style transfer was defined as a process of texton (known as basic element of texture), which is mapping the texton to the target location.

The goal of it is to derive a generalization process from a sample texture to generate arbitrary new images with that texture. After the input image is given, the sampling function is defined to sample the small part of the input image's feature and put the texture of the small part of the corresponding place. Finally, we will get a large texture based on each small part.

It's just that the previous method was done in the image space. The authors adapt this idea and use the neural network (VGG16) to sample the style images. Here are the three parts of about this network

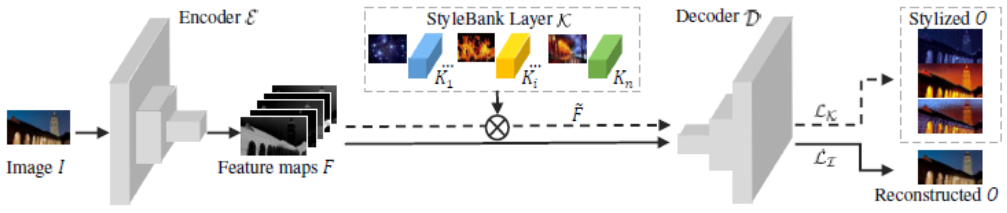
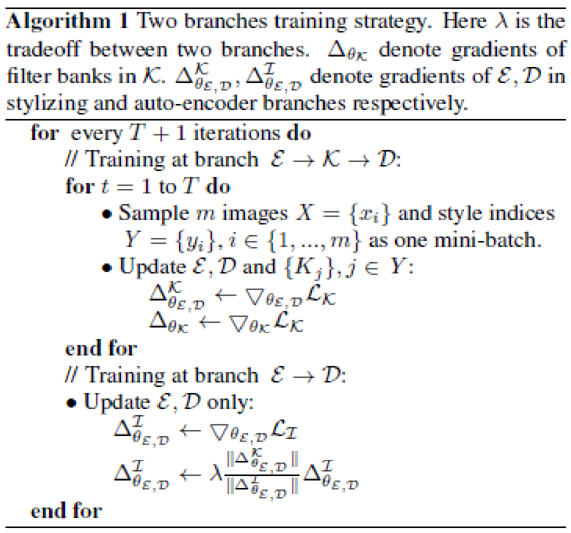


Figure 1. Our network architecture consists of three modules: image encoder $\mathcal{E}$, StyleBank layer $\mathcal{K}$ and image decoder $\mathcal{D}$

- Image encoder: consists of 3 convolution layers, it encodes the image into the feature map.
- StyleBank layer: every filter bank will represent a style and each bank consists of several kernels (which represents a texton). Each texton will convolve with the feature map
- Image decoder: after putting each texton on the feature map, it will go through the decoder. And then we will get the styled output images

To make sure the style information only can be stored in the StyleBank layer, the authors add a new branch into the networks, which doesn't go through the StyleBank. This branch will demand that after an image only go through the encoder and decoder, there should be no difference. Therefore, the encoder and decoder could be separate from the StyleBank.

In order to train the StyleBank network, the author defines T + 1 method, which is using



Algorithm 1 Two branches training strategy. Here λ is the tradeoff between two branches. $\Delta_{\theta_{\mathcal{K}}}$ denote gradients of filter banks in $\mathcal{K}$. $\Delta^{\mathcal{K}}_{\theta_{\mathcal{E},\mathcal{D}}}, \Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}}$ denote gradients of $\mathcal{E}, \mathcal{D}$ in stylizing and auto-encoder branches respectively.

```
for every $T+1$ iterations do
    // Training at branch $\mathcal{E} \rightarrow \mathcal{K} \rightarrow \mathcal{D}$:
    for $t = 1$ to $T$ do
        • Sample $m$ images $X = \{x_i\}$ and style indices
          $Y = \{y_i\}, i \in \{1, ..., m\}$ as one mini-batch.
        • Update $\mathcal{E}, \mathcal{D}$ and $\{K_j\}, j \in Y$:
            $\Delta^{\mathcal{K}}_{\theta_{\mathcal{E},\mathcal{D}}} \leftarrow \nabla_{\theta_{\mathcal{E},\mathcal{D}}} \mathcal{L}_{\mathcal{K}}$
            $\Delta_{\theta_{\mathcal{K}}} \leftarrow \nabla_{\theta_{\mathcal{K}}} \mathcal{L}_{\mathcal{K}}$
    end for
    // Training at branch $\mathcal{E} \rightarrow \mathcal{D}$:
    • Update $\mathcal{E}, \mathcal{D}$ only:
        $\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}} \leftarrow \nabla_{\theta_{\mathcal{E},\mathcal{D}}} \mathcal{L}_{\mathcal{I}}$
        $\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}} \leftarrow \lambda \frac{\|\Delta^{\mathcal{K}}_{\theta_{\mathcal{E},\mathcal{D}}}\|}{\|\Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}}\|} \Delta^{\mathcal{I}}_{\theta_{\mathcal{E},\mathcal{D}}}$
end for
```

T step to the StyleBank branch, and using the 1 train the encoder and decoder.

2. Implement of the networks

   Considering that there are only Pytorch and TensorFlow 1.0 versions implementation of the StyleBank in GitHub. We decided to reimplement it based on TensorFlow 2.0 with the referring to the Pytorch and TensorFlow1.0 version.

   During this converting process, we have to understand the details of Pythorch and TensorFlow 1.0(eg: tf.session tf.placeholder), which is quite a lot of work.

Report of preliminary experimental results

Because of the difficulty of the last two models building, we only fully implement the first methods. Therefore, the following report will only include the first model.

Since we found that in both image optimization and model optimization-based models, the loss of generated pictures regarding original style and content images are assessed in a similar manner. That is, to pass all three images into a descriptor network and calculate content loss based on sum of residual norm of features map from intermediate layers and style loss based on sum of residual norm of the gram matrix. Thus, it would be beneficial for all three models if we investigate the effect of turning hyperparameters in the descriptor network qualitatively and hence better fine turning our model. As a result, our group conducted several preliminary experiments on hyperparameter turning in an image based optimization model, whose descriptor network is VGG 19 trained from ImageNet data, and we hope these experiment results could be generalized to other two models.

Content Images Figure 1

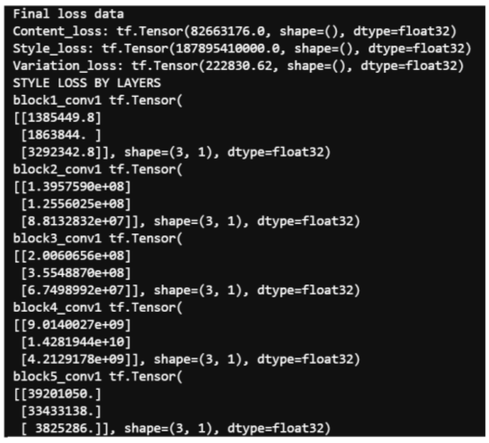
Style images Figure 2

In every round of these experiments, 3-4 style images representing different art styles are used to generate new images with 4 scenery images of HKU campus.

1. Style layers investigation

   We first try to generate 3 images using 5 style layers together and 1 content layer exactly described in Gaty 's paper. As shown by figure 3, the individual style loss of each layer is considerably different from each other. For example, The loss produced by block_1conv_1 is approximately 100 times smaller than that of block2_conv2.

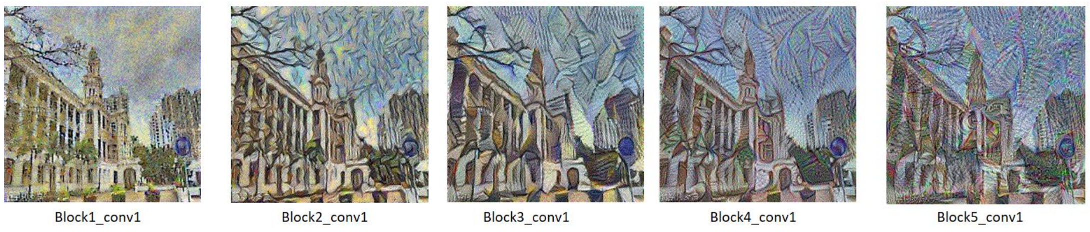
Thus, instead of uniformly averaging the each style layer loss to form the total style loss, a weighted sum scheme should be introduced in calculating the sum.



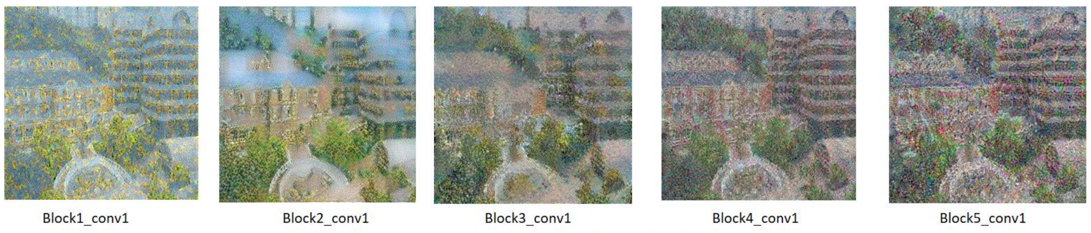
```
Final loss data
Content_loss: tf.Tensor(82663176.0, shape=(), dtype=float32)
Style_loss: tf.Tensor(187895410000.0, shape=(), dtype=float32)
Variation_loss: tf.Tensor(222830.62, shape=(), dtype=float32)
STYLE LOSS BY LAYERS
block1_conv1 tf.Tensor(
[[1385449.8]
 [1863844. ]
 [3292342.8]], shape=(3, 1), dtype=float32)
block2_conv1 tf.Tensor(
[[1.3957590e+08]
 [1.2556025e+08]
 [8.8132832e+07]], shape=(3, 1), dtype=float32)
block3_conv1 tf.Tensor(
[[2.0060656e+08]
 [3.5548870e+08]
 [6.7498992e+07]], shape=(3, 1), dtype=float32)
block4_conv1 tf.Tensor(
[[9.0140027e+09]
 [1.4281944e+10]
 [4.2129178e+09]], shape=(3, 1), dtype=float32)
block5_conv1 tf.Tensor(
[[39201050.]
 [33433138.]
 [ 3825286.]], shape=(3, 1), dtype=float32)
```

Figure 3

In addition, our group tries to experiment on different generated images if I train using only one of these layers. Note that the style weight parameter is configured so that each time the style loss converges at a similar level.



Images generated using different style layers Figure 4

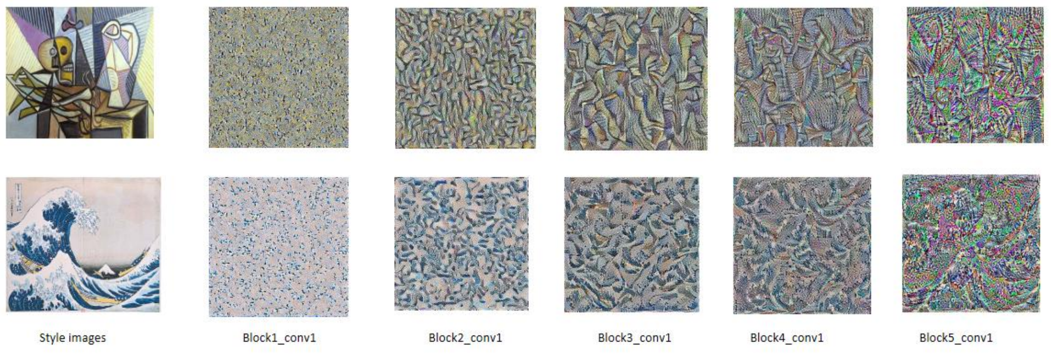


Images generated using different style layers Figure 5

We can observe that the size of the brushstroke increases as we move from block1_conv1 to block5_conv1, getting deeper to the VGG network. This is as expected, as the size of respective fields increases as we go deeper to the network. Also notice that in Block4_conv1 and Block5_conv1, there are more noise and checkerboard pattern appear, we think this is because the content of high level feature map tends to be very abstract with few detail low level information contained, thus, making such noise. For the case of Block1_conv1, our group think this layer is too shallow to extract the style information form the picture, but instead, it only extracts the colour information, making a large colour distortion of the images.

Also, in different images for example, the best quality images might happen in different layers, for example the best images occur at Block3_conv1 in figure 4 while Block4_conv1 at figure 5. This shows that fine turning needs to be better done to individual images, given such structure is not robust enough. Overall, The fact that the best result occurs when drawing style information form the middle layers agrees with the general impression that styles are more relevant to the low level features like shape, brushstroke, than more fundamental features like colour nor high level features like the object draw itself.

Below figure 6 images are generated from different style layers by setting the content weight and total variation weight to zeros. It agrees with the previous conclusion that the middle layer captures style information the best. From figure 6, the best style images using these two style images should use Block3_conv1.

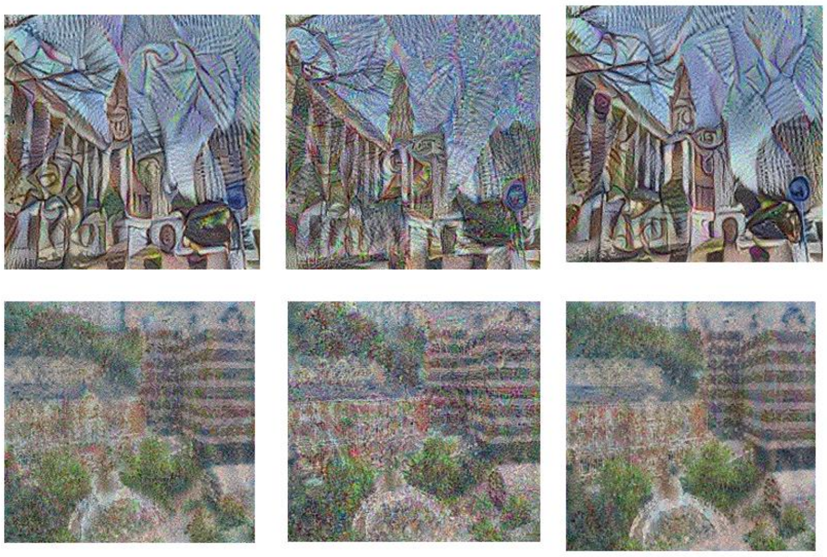


Images generated using solely style images Figure 6

2. Total variation loss experiment

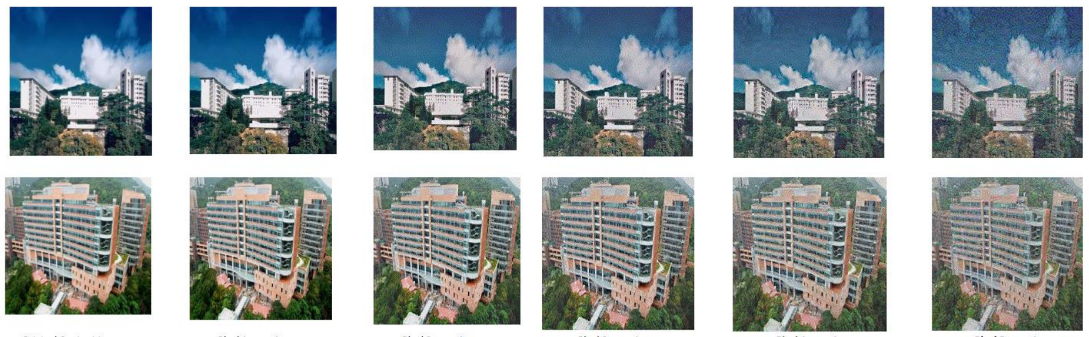
As suggests by Gaty's paper, one downside to this naive implementation is that it produces a lot of high frequency artifacts, resulting such checkerboard pattern and gaussian noise in the images. Our group tackles this problem by decrease these noise using an explicit regularization term on the high frequency components of the image, which is called total variation loss. Qualitatively speaking, an increase in the total variation loss weight could remove the rough texture of the image and the resultant image looks much smoother.

3. Investigation on content layer


Total variation weight = 100
Total variation weight = 0
Total variation weight = 1e4

Comparison for different total variation weight Figure 7

We try to investigate the effect of choosing different content layers in generating style transfer images.



Images generated using solely style images Figure 8

First, we set the total variation weight and the style weight to zeros to “visualize” the content generated in different layers.

As Figure 8 shows , we move from Block1_conv1 to Block5_conv1, the high level information like windows cloud trees are still captured, but low level information like edges, corners etc is increasingly blurred. Moreover, there is more noise within the picture, suggesting the low level features and information such as line and colour starts to lose when extracting from high level layers.

Moreover, our group also investigated the effect of using different content layers to generate images.

Block1_conv1
Block2_conv1
Block3_conv1
Block4_conv1

Images generated using different content layer Figure 9

Again, though it is not evident, but with detailed observation of magnified images, we could observe that, as we move from conv2_2 to conv4_1, edges of building become more blunt, colours are more faded and much less of the original content is captured, as a result of loss of information of lower level features.

Future experiment direction:

Due to the tight timeline, our group is unable to conduct furthermore experiments whose result could be helpful for fine turning the hyperparameter of the model. Thus, here we just list out some more direction we wish to investigate before the final submission.

1. Training using different descriptor network

   Up to now, all the experiments conducted on image-based optimization methods are on VGG 19 network, we wish to investigate the result using different networks. For example, we plan to use those with distinct network architecture such as ResNet 50 and Inception Net and those with similar network architecture like VGG 16 to better evaluate the effect of network architecture to the generation of images.

2. Training using cropped style images

   A small part cropping from original style images may change the correlation of features calculated by gram matrix, thus making a much difference to the generated images. For example, a primary feature of images generated by the style04, the Japanese painting style is that it tends to make the cloud look like a big wave. We plan to investigate whether this effect would get much stronger given we crop the images such that on waves part is preserved like Figure 10

3. Training using different resolution of style images

   Our group thinks that the information contained in one style image would lose with worsen resolution. While to which degree the quality of generated images would

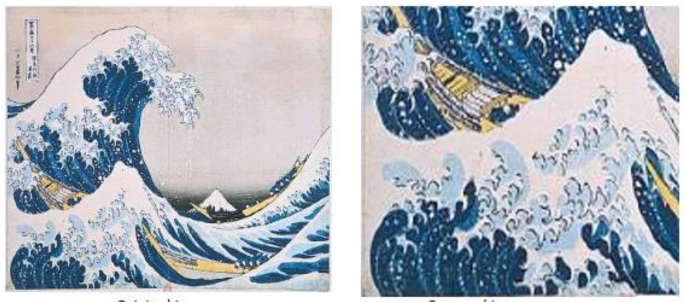


Figure 10

   deteriorate significantly due to poor quality of style images still worth further investigation.

4. Training using object images

   The descriptor network of this model, VGG 16 originally trained on ImageNet data is used for classification tasks, while in this context, we use it to extract the features of the images. Up to now all the style images we provide are classified as none, using VGG 16 networks.

Suggestive experiments/improvements for the next stage

Firstly, we will finalize our three models and will also tune the hyper-parameters. The proposed experiment in the previous section would be conducted and its conclusion result would be applied to all the models. Qualitative analysis of the generated picture would be performed to guide the fine turning.

Secondly, we would gather a uniform dataset of content and style images. Then three models would be compared by running on the same dataset. We need to work out the detail comparison scheme, which includes both qualitative and quantitative analysis.