The Project Final Report of Image Style Migration

Final Presentation Video: https://youtu.be/fRmPoCoWuYo

Fei Ma UIUC Computer Science feima2@illinois.edu	Yucheng Ma UIUC Computer Science ym19@illinois.edu	Yuqing Zhang UIUC Mechanical Engineering yuqingz7@illinois.edu	Ziheng Song UIUC Computer Science zihengs2@illinois.edu

1. Introduction

Humans have long mastered the skill of drawing a specific object with different sorts of artistic style, but it could be hard for a computer to complete a mission like that. After the introduction of neural networks, especially deep neural networks, image style migration has walked into the sight of many people. It has also been implemented in different areas for various reasons.

Due to this reason and adds to our interests, we decided to choose image style migration as our main theme of the project. We selected two different methods from past papers and tried to research and understand how they work. Then we attempted to implement them through programs and code. The two methods that we have worked on in this project are listed below, more detailed introduction of methods will be presented in section 2:

1) An artificial system based on a Deep Neural Network that uses neural representations to separate and recombine content and style of arbitrary images. It achieves regular style transfer of fixed style and fixed content.
2) A feed-forward network with perceptual loss functions that achieves fast style transfer of fixed style and arbitrary content.

In this report, we will describe the progress and present results of our project below. Through our research and implementation we found that these two methods have a few similar points but also uniquenesses. By analyzing their own advantages and disadvantages, both methods have suitable usage scenarios. In this final report and video presentation, we will compare the performance, timecost and other characteristics of these two methods.

2. Details of the approach

Either way, we need to involve the construction of a convolutional neural network, so we have chosen *Pytorch* as our main package, which has powerful features and offers numbers of pre-trained models to choose from. This will help us considerably in our development process.

2.1 Model Architecture

The models we used as a convolutional neural network for classification and detection are VGG16 and VGG19, the architecture is shown in the *Fig.1* (Figure shows VGG16 structure, VGG19 adds three more convolution layers to VGG16). VGG was proposed by the Visual Geometry Group of Oxford on ILSVRC 2014. Their main contribution was to prove the increasing depth of the network can affect the final performance to some extent.

Compared to AlexNet, VGG16 uses consecutive 3×3 convolution kernels replacing the larger convolution kernels in AlexNet. Given a receptive field, the stacked smaller convolution kernel is

better than the larger kernel, because multiple nonlinear layers increase the network depth and ensure more complex learning.

The **advantages** of VGG:
1) Simple structure, network uses same convolution kernel size and max pooling size,
2) Combination of smaller filter is better than using a single larger filter,
3) Network performance is improved by deepening the network structure.

The **disadvantages** of VGG:
1) Consume more computing resources, memory usage and requires more parameters,
2) Most parameters come from the first fully connected layer.

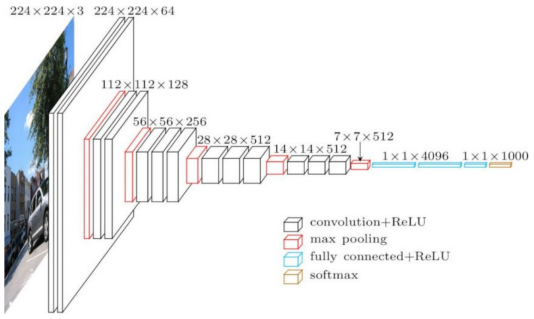


Fig 1. VGG Architecture

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Fig 2. VGG16 Network architecture

2.2 Method 1: A Neural Algorithm of Artistic Style

In paper *A neural Algorithm of Artistic Style* (Leon, Alexander)[1], it introduced an artificial system based on a Deep Neural Network that uses neural representations to separate and recombine content and style of arbitrary images. This paper mentioned that creating a Convolutional Neural Network to filter the information of images. Each layer of Convolutional Neural Network consists of small units that process visual information in a feed-forward manner. Thus, the outputs of different layers are differently filtered versions of the input image.

Content and Style Reconstructions:
While the different filters increase along the processing hierarchy, the size of the filtered images is reduced by some downsampling mechanism leading to a decrease in the total number of units per layer of the neural network. The experimental results in the article show that the deeper the image is reconstructed by the VGG19-network, the image suffers from severe detail loss, but the content of the image is preserved.

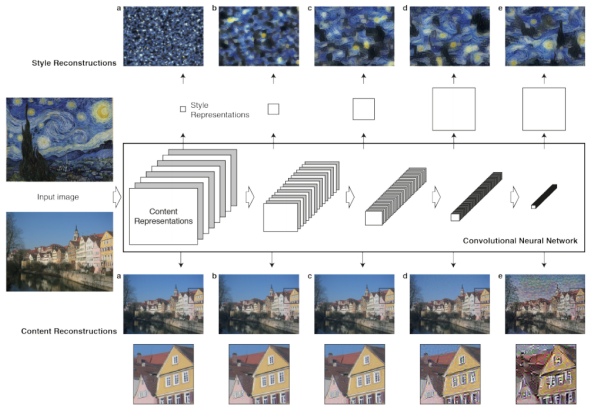


Fig 3. Convolutional Neural Network (CNN)

Again, the shallow layer of CNN filtered more "style" information and gave up the content from the images. Therefore, the key point of this method is that the representations of content and style in the Convolutional Neural Network are separable. That is, we can extract style and content

from two images separately, and then generate a blended image to achieve style migration.

In our programed CNN model, we used a 19 layer VGG network which was the one used in the paper[1]. PyTorch's implementation of VGG has two child Sequential modules: features and classifier. Vgg19.features contains a sequence (conv2d, relu, maxpool) layers. So we added our content loss and style loss layers after the conv2d layer after they were detecting. Following is a more detailed introduction of loss functions.

Loss functions:
Simply put, in order to make our style transformation successful and retain the most content. We need the loss function to be minimal when:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

So let p and x be the original image and the image that is generated, F^l and P^l are their respective feature representation in layer L

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l\right)^2 .$$

For the style, building a style representation on top of the CNN responses in each layer of the network. To compute the correlations between the different filter responses, these feature correlations are given by the Gram matrix $G^l \in R^{N^l \times N^l}$. The total loss is:

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Using Gaussian noise as the initial input image, optimize the mixing error of content + style, and perform multiple forward/backward iterations to optimize using the L-BFGS method to achieve style transfer. The architecture of our network has been output by print(net) and listed below:

```
Sequential(
  (0): Normalization()
  (conv_1): Conv2d(3, 64, kernel_size=(3, 3),
stride=(1, 1), padding=(1, 1))
  (style_loss_1): StyleLoss()
  (relu_1): ReLU()
  (conv_2): Conv2d(64, 64, kernel_size=(3, 3),
stride=(1, 1), padding=(1, 1))
  (style_loss_2): StyleLoss()
  (relu_2): ReLU()
  (pool_2): MaxPool2d(kernel_size=2, stride=2,
padding=0, dilation=1, ceil_mode=False)
  (conv_3): Conv2d(64, 128, kernel_size=(3, 3),
stride=(1, 1), padding=(1, 1))
  (style_loss_3): StyleLoss()
  (relu_3): ReLU()
  (conv_4): Conv2d(128, 128, kernel_size=(3, 3),
stride=(1, 1), padding=(1, 1))
  (style_loss_4): StyleLoss()
  (content_loss_4): ContentLoss()
  (relu_4): ReLU()
  (pool_4): MaxPool2d(kernel_size=2, stride=2,
padding=0, dilation=1, ceil_mode=False)
  (conv_5): Conv2d(128, 256, kernel_size=(3, 3),
stride=(1, 1), padding=(1, 1))
  (style_loss_5): StyleLoss()
)
```

2.3 Method 2: Perceptual Losses for Real-Time Style Transfer

Following the paper *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*[3], we solved the image transformation problem by training a feed-forward network with perceptual loss functions.This system consists of two main components as marked in *Fig.4*, the Image Transformation Network and the Loss Network(VGG-16).

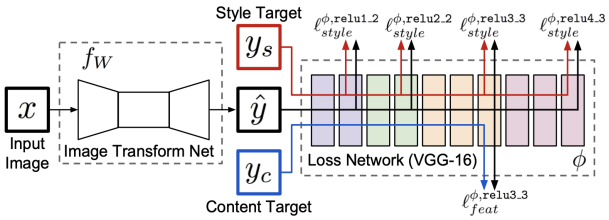


Fig.4 System Structure of Real-Time Style transfer

The Transformation Network is structured like the table in *Fig.5*. The first and the last layer have kernel size of 9, and others with size of 3. It involves downsampling and upsampling processes implemented with 2 downsampling layers by using stride 2 and 2 upsampling layers by using the upsampling nearest neighbor method, before and after the residual blocks. The benefits for downsampling and upsampling are that it can save

computational costs and have smaller receptive field size. The residual block contains 2 3x3 convolutional layers, it uses the skip-connection technique which gives access of lower-level features to high-level. The idea is adopted from paper Deep Residual Learning for Image Recognition[8].

Layer	Activation size
Input	$3 \times 256 \times 256$
$32 \times 9 \times 9$ conv, stride 1	$32 \times 256 \times 256$
$64 \times 3 \times 3$ conv, stride 2	$64 \times 128 \times 128$
$128 \times 3 \times 3$ conv, stride 2	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
Residual block, 128 filters	$128 \times 64 \times 64$
$64 \times 3 \times 3$ conv, stride 1/2	$64 \times 128 \times 128$
$32 \times 3 \times 3$ conv, stride 1/2	$32 \times 256 \times 256$
$3 \times 9 \times 9$ conv, stride 1	$3 \times 256 \times 256$

Fig.5 Transformation Network Structure

The Loss Network(VGG-16) is used to extract image features in order to measure perceptual differences between the content and the style. It is constructed by splitting the pretrained VGG-16 model to 4 blocks, and saving the results after applying each block of filters to the image. There will be 4 outputs relu1_2, relu2_2, relu3_3 and relu4_3.

Loss The image transformation network transforms an input image x to an output image $\hat{y}$.

Style Reconstruction Loss is between $\hat{y}$and the style target y_s, it is calculated by taking the Euclidean distance of them, the formula is below

$$\ell_{feat}^{\phi,j}(\hat{y},y) = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{y}) - \phi_j(y)\|_2^2$$

Feature Reconstruction Loss is between $\hat{y}$and the feature target y_s, we calculated the gram matrix G of $\hat{y}$ and y_s, and then took the Euclidean distance as mentioned in paper[3].The formula is below

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

$$\ell_{style}^{\phi,j}(\hat{y},y) = \|G_j^{\phi}(\hat{y}) - G_j^{\phi}(y)\|_F^2.$$

We used *MSELoss* for calculating the Euclidean distance.

Training Details:

***Dataset*:** Train2014[83K/13GB] from COCO dataset. Download
Timecost: ~8 hours on Google Colab with GPU
***Inputs and Outputs*:** Images with size 3x256x256
Epoch*:**2, ***Batch Size*:**4, ***Feature Weight*:**1e5, ***Style Weigh:1e10
Optimizer: Atom with learning rate 1e-3, everything else in default.

3. Results

3.1 Method 1 Result

We implemented the neural style and have some conclusion for this method. Because of the excellent performance of the VGG19 model, it is quite convenient for us to complete the style transformation of images without relying on other training sets.

At first, due to the optimization and iterations, this method is very slow in transformation. In transforming the image *Almas statue* into Vincent van Gogh's *The Starry Night style*, it took 6.5 hours for 30 iterations without GPU acceleration, and still 1.5 hours after acceleration.

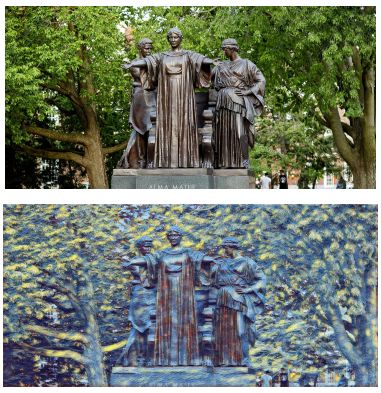
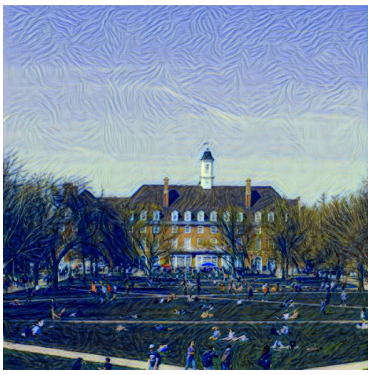
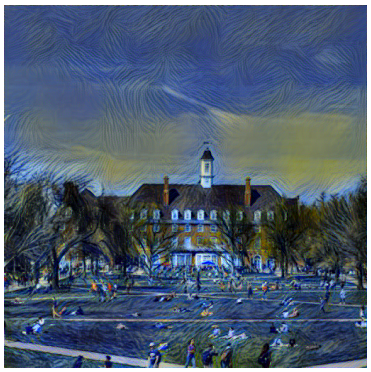


Fig. 6 Alma Statue Style Transfer

We then decided to put our project on google colab and run it in GPU mode. It turns out that in GPU mode we can produce more accurate images and iterate more to achieve better results.

Epoches:100 & Total loss = 16.3802



Epoches=600 & Total loss = 9.1213

Fig 7. Starry Quad Style Transfer

As the results show, the total loss decreases as the epoches rises. As we mentioned in Approach, a lower total loss means a better fusion of the two images. However, when we tried to increase the epoches in our experiments, we found that increasing them did not result in better visualisation. Therefore, the ultimate criterion for style migration may still be human aesthetic standards.

3.2 Method 2 Result

We implemented the training and style transforming process on Google Colab with GPU. We trained models on COCO 2014 Train images dataset [83K/13GB] without annotations. It took around 8 hours to train the whole dataset with a style image, and can produce transfer results instantly once we have the trained model. We implemented various image style extraction and applied fast style transfer on arbitrary input images. Some of our successful image style transfer results are shown below.

Fig. 8. Landscape Style Transfer

Fig 9. Brad Pitt Style Transfer

Fig.10. Kitten Style Transfer

4. Discussion and Conclusions

Method 1, A Neural Algorithm of Artistic Style, requires both style and content images to be input into the VGG19 neural network, which means that it is not possible to save a particular style alone for frequent use, and each style change requires new style information to be extracted from the style image. Everytime style migration needs two pre-processed images as a style image and content image. Both of them should have the same size and dimension so that we can put them into style loss layers and content loss layers.
Advantage: Style migration can be carried out by simply providing the style image and the target image, iterating through the loss functions without the need for long training models.
Disadvantage: Cannot store picture styles. Each time an image is generated it needs to be done with the help of the GPU's arithmetic power, which requires a not short waiting time. The size of the epoches determines the visual quality of the resulting image. But it is difficult to estimate a good epoch accurately until the results are available.

Method 2, Real-time style transfer, requires training on a dataset, and has a deeper network than method 1, with a transformation network and a VGG 16 network. It produces similar results as method 1.
Advantage: Real-time style transfer allows the users to transform the image with a style model in a few seconds even without GPU. The transformation process is fast and has almost no requirement for the environment.
Disadvantage: The training process took around 8 hours for our train2014 dataset on GPU. It has requirements on memory due to the huge amount of training data, and can be time consuming. It needs good computational power; GPU is required.

In conclusion, both of the two methods show satisfactory performance. Compared to method 2, we think method 1 is simpler in principle, but the inability to store the style model is a major drawback. This means that a large number of calculations per operation cannot be achieved on mobile devices without a powerful GPU. We believe that method 2 is more suitable for image style migration software, as the trained model can be applied directly to the content image, although the training cost is high.

5. Statement of Individual Contributions

Yucheng Ma implemented transfer method 1, Regular style transfer of fixed style and fixed content;
Fei Ma implemented the transfer method 2, Fast style transfer of fixed style and arbitrary content;
Yuqing Zhang is responsible for fine-tuning and training model with various style image using transfer method 2;
Ziheng Song is responsible for the final presentation and video.
For code and data maintenance, our group shared via cloud drive. For group interaction, our group uses Zoom to communicate and set up weekly meetings. All team members discuss papers and programming ideas as scheduled.

7. Reference

[1] Gatys, L. A. (2015, August 26). A Neural Algorithm of Artistic Style. ArXiv.Org. https://arxiv.org/abs/1508.06576
[2]https://github.com/Kautenja/a-neural-algorithm-of-artistic-style
[3] Johnson, J. (2016, March 27). Perceptual Losses for Real-Time Style Transfer and Super-Resolution. ArXiv.Org. https://arxiv.org/abs/1603.08155
[4] Shen, F. (2017, September 13). Meta Networks for Neural Style Transfer. ArXiv.Org. https://arxiv.org/abs/1709.04111
[5]https://github.com/FalongShen/styletransfer.
[6]https://cocodataset.org/#download.
[7]https://pytorch.org/tutorials/advanced/neural_style_tutorial.html
[8]Kaiming He (2015, December 10). Deep Residual Learning for Image Recognition. https://arxiv.org/pdf/1512.03385.pdf
[9]Karen Simonyan (2014, September 4). Very Deep Convolutional Networks for Large-Scale Image Recognition. https://arxiv.org/abs/1409.1556