Style Transfer Report

CSCI611 Spring 2025 - Assignment 4

Boxi Chan

Intro and background

In this Assignment we will perform Style Transfer with Deep Neural Networking using PyTorch following by paper “Image Style Transfer Using Convolutional Neural Networks by Gatys”
We will use pre-trained VGG19 net to help us extract content or style features from a passed in image.

Loading Model

```
# get the "features" portion of VGG19 (we will not need the "classifier" portion)
vgg = models.vgg19(pretrained=True).features

# freeze all VGG parameters since we're only optimizing the target image
for param in vgg.parameters():
    param.requires_grad_(False)
```

```
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    vgg.to(device)
```

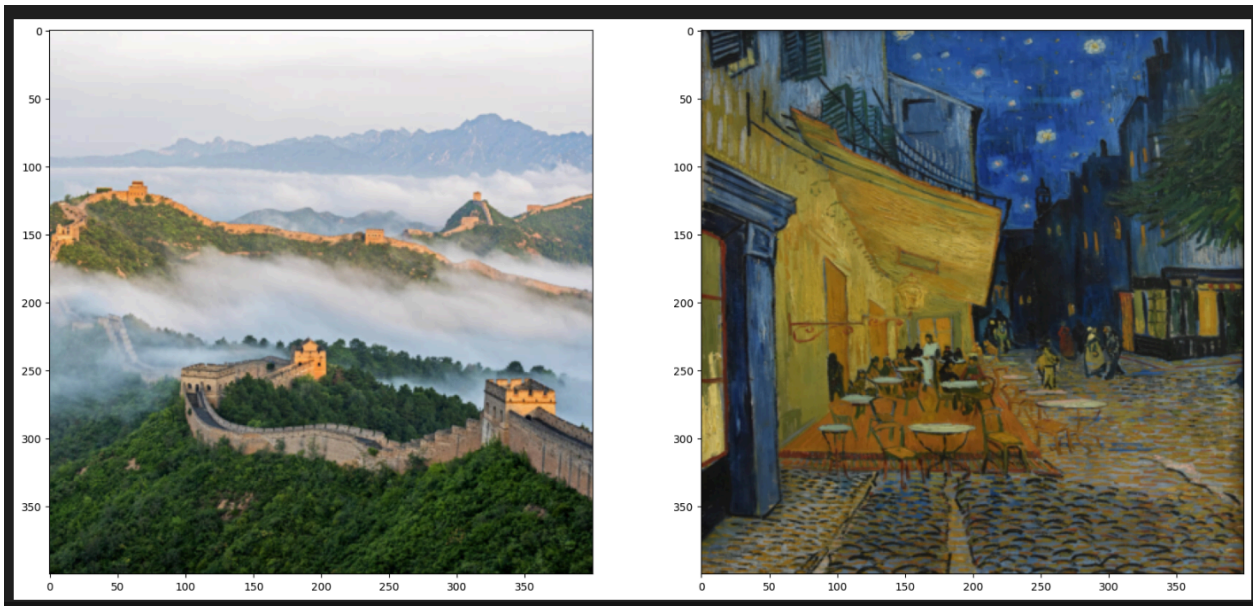
```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (11): ReLU(inplace=True)
  (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU(inplace=True)
  (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU(inplace=True)
  (16): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (17): ReLU(inplace=True)
  (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (19): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (20): ReLU(inplace=True)
  (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (22): ReLU(inplace=True)
  (23): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
...
  (33): ReLU(inplace=True)
  (34): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (35): ReLU(inplace=True)
  (36): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

After we loaded hidden layers in the neural network we can see network combine with multiple Convolutional layer , ReLU layer and MaxPool layer, there is 37 total layers and every new section always start after MaxPool2d such that Section 1 start on layer index 0, Section 2 start on layer index 5 after first MaxPool layer.

```python
def load_image(img_path, max_size=400, shape=None):
    ''' Load in and transform an image, making sure the image
       is <= 400 pixels in the x-y dims.'''
    if "http" in img_path:
        response = requests.get(img_path)
        image = Image.open(BytesIO(response.content)).convert('RGB')
    else:
        image = Image.open(img_path).convert('RGB')

    # large images will slow down processing
    if max(image.size) > max_size:
        size = max_size
    else:
        size = max(image.size)

    if shape is not None:
        size = shape

    in_transform = transforms.Compose([
                        transforms.Resize(size),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),
                                             (0.229, 0.224, 0.225))])

    # discard the transparent, alpha channel (that's the :3) and add the batch dimension
    image = in_transform(image)[:3,:,:].unsqueeze(0)

    return image
```

Our load image function will help convert and normalize the size of image when it is loading into our system, so the picture size will be fit into our input of our model.



On the left is the content of the target picture(The Great Wall from China), On the right is style(Café Terrace at Night by Vincent van Gogh 1888). The final product should be a

target picture(The Great Wall) displayed by the style(Café Terrace) we choose.

Content and Style Features

```
def get_features(image, model, layers=None):
    """ Run an image forward through a model and get the features for
        a set of layers. Default layers are for VGGNet matching Gatys et al (2016)
    """

    ## TODO: Complete mapping layer names of PyTorch's VGGNet to names from the paper
    ## Need the layers for the content and style representations of an image
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  '21': 'conv4_2',
                  '28': 'conv5_1',
                  }


    ## -- do not need to change the code below this line -- ##
    features = {}
    x = image
    # model._modules is a dictionary holding each module in the model
    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x

    return features
```

```
## TODO: Complete mapping layer names of PyTorch's VGGNet to names from the paper
## Need the layers for the content and style representations of an image
```

Based on the paper, we will need to map layers of PyTorch's VGGNet for Content and style representations in order to help model to identify which is for content learning and which is style learning.
So we follow the path of every Section after the MaxPool layer, so we set index 0,5,10,19,28 as different section layers and they always start with a Convolutional layer.

1. Conv1_1, Conv1_2: learning pixel detail (color and edges)
2. Conv3_1: learning content detail (object size, circle)
3. Conv4_1,Conv5_1 learning more abstract(style, content, type)
4. Conv4_2 choose as style layer(Conv4_2 as index 21 layer, this layer is middle of network, and model already learning general what object gonna looks like)

Gram Matrix

The output of every convolutional layer is a Tensor with dimensions associated with the `batch_size`, a depth, `d` and some height and width (`h`, `w`). The Gram matrix of a convolutional layer can be calculated as follows:
* Get the depth, height, and width of a tensor using `batch_size, d, h, w = tensor.size()`
* Reshape that tensor so that the spatial dimensions are flattened
* Calculate the gram matrix by multiplying the reshaped tensor by it's transpose

Note: You can multiply two matrices using `torch.mm(matrix1, matrix2)`.

TODO: Complete the `gram_matrix` function.

```python
def gram_matrix(tensor):
    """ Calculate the Gram Matrix of a given tensor
        Gram Matrix: https://en.wikipedia.org/wiki/Gramian_matrix
    """

    ## get the batch_size, depth, height, and width of the Tensor
    batch_size, depth, height, width = tensor.size()
    ## reshape it, so we're multiplying the features for each channel
    Flatten = tensor.view(depth, height * width)

    ## calculate the gram matrix
    gram = torch.mm(Flatten, Flatten.t())

    return gram
```

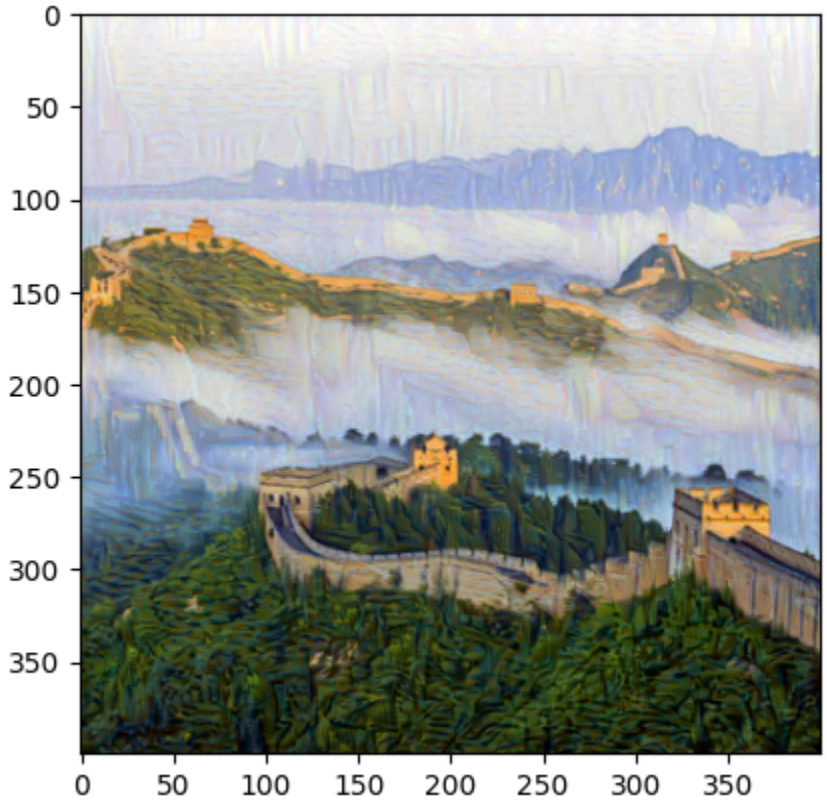
We will use build-in function tensor.view to reshaped the size, so it will become depth of height * width matrices, such that depth 4, height and weight are 10 then matrix will be come 4x(10*10) which is 4x100.
Then we have math for calculate gram matrix:
(depth, height * width) @ (height * width, depth) = (depth, depth)

We will using built-in function torch.mm to help us calculate the gram

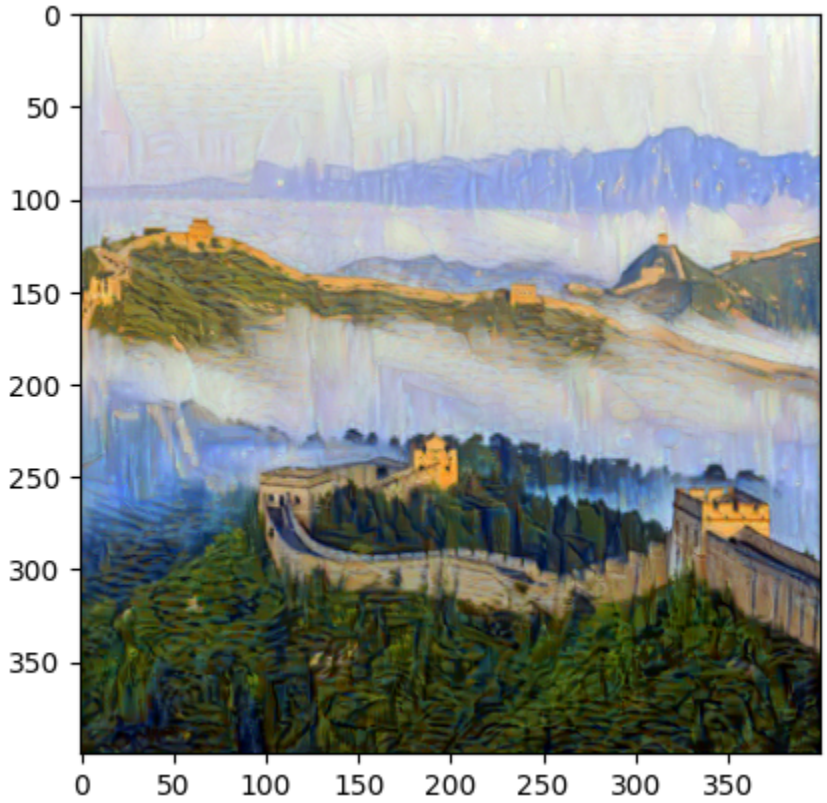
```
for ii in range(1, steps+1):

    ## TODO: get the features from your target image
    ## Then calculate the content loss
    target_features = get_features(target, vgg)
    content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2'])**2)

    # the style loss
    # initialize the style loss to 0
    style_loss = 0
    # iterate through each style layer and add to the style loss
    for layer in style_weights:
        # get the "target" style representation for the layer
        target_feature = target_features[layer]
        _, d, h, w = target_feature.shape

        ## TODO: Calculate the target gram matrix
        target_gram = gram_matrix(target_feature)

        ## TODO:  get the "style" style representation
        style_gram = style_grams[layer]
        ## TODO: Calculate the style loss for one layer, weighted appropriately
        layer_style_loss = torch.mean((target_gram - style_gram)**2)

        # add to the style loss
        style_loss += layer_style_loss / (d * h * w)


    ## TODO:  calculate the *total* loss
    total_loss = content_weight * content_loss + style_weight * style_loss
```

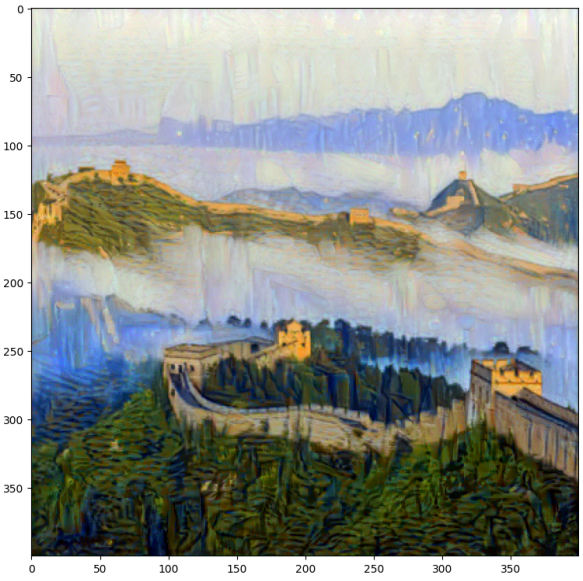
Next we just get target features for every loop and calculate the loss using the gram_matrix we built above and calculate the loss of each train(every generated picture compare to style image).

Total loss: 15159387.0



Total loss: 9810742.0



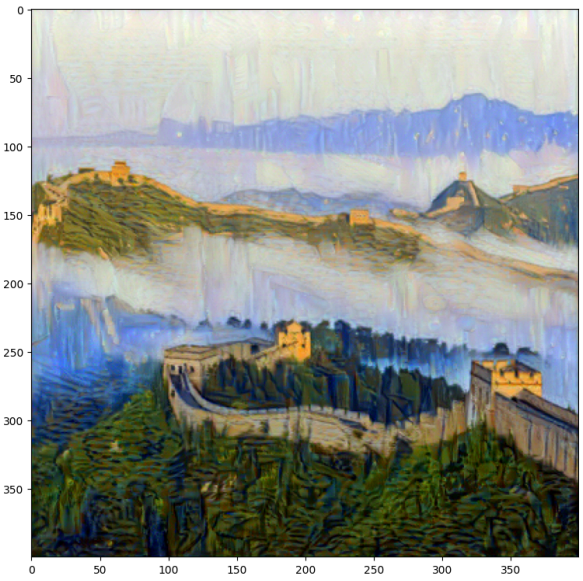


After initial training with steps 1000, we can see loss is really high because style weight loss is 1e6 which is really high.

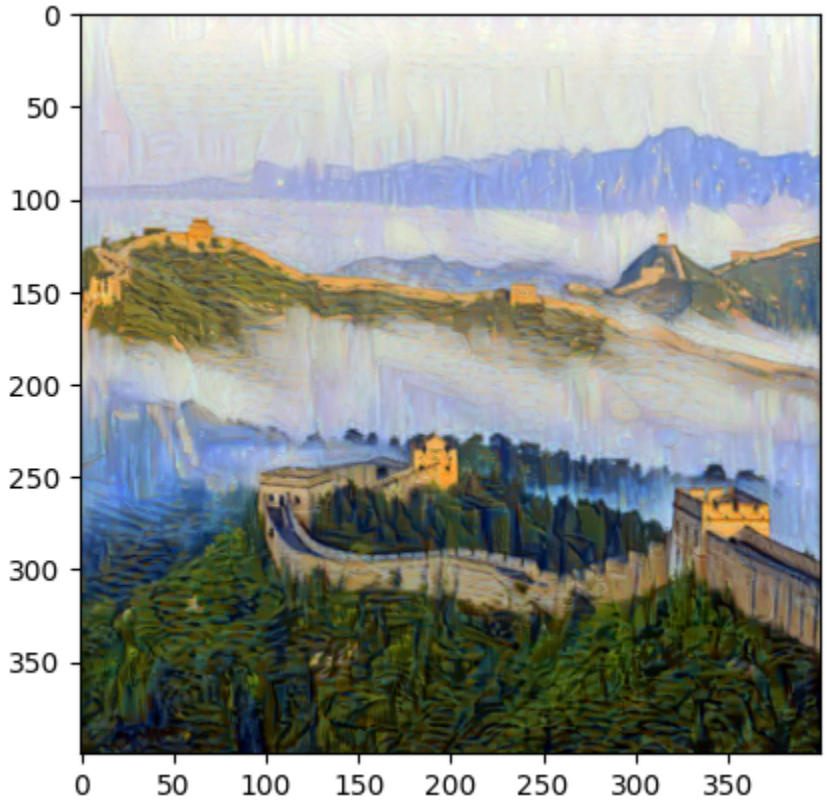
Train 2, we will remain everything be the same but style weight will be adjust, sense Gogh's draw more abstract, more general look of picture, so we will lower the weight for

coloring, size, more focus on abstract and style

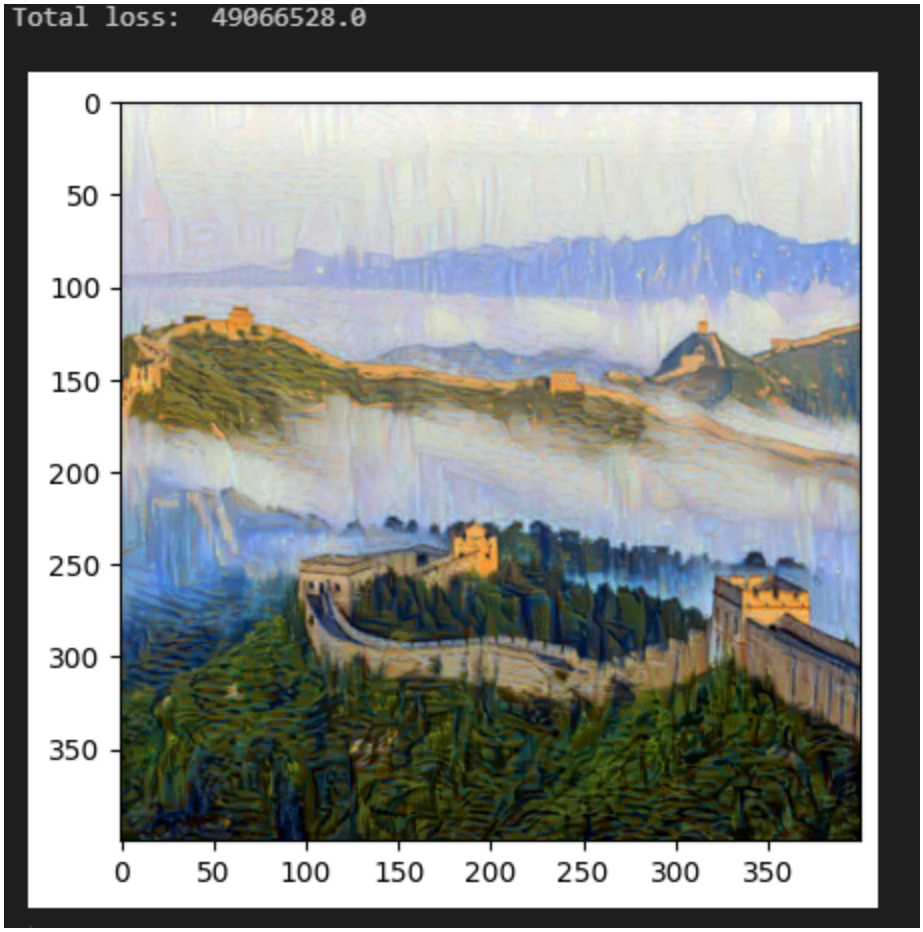
```
style_weights = {'conv1_1': 0.5,
                 'conv2_1': 0.4,
                 'conv3_1': 0.5,
                 'conv4_1': 0.8,
                 'conv5_1': 0.4}
```



Total loss: 9812405.0



Train 3:We can see there is not much changing after adjusting weight, maybe because of style_weight = 1e6 is really high, so on train 3 we will adjust style_weight up to 1e6 -> 5e6 so it will generate more Gogh's type, drop more feature and add more abstract style feature, which will make more like Gogh's drawing type.



We can see the image change to a more Oil painting style, and loss is increasing a lot that means we lost a lot of detail of the original target picture's features and more focus on style.

Train 4: we will decrease the optimizer lr to 0.001 and steps to 2000

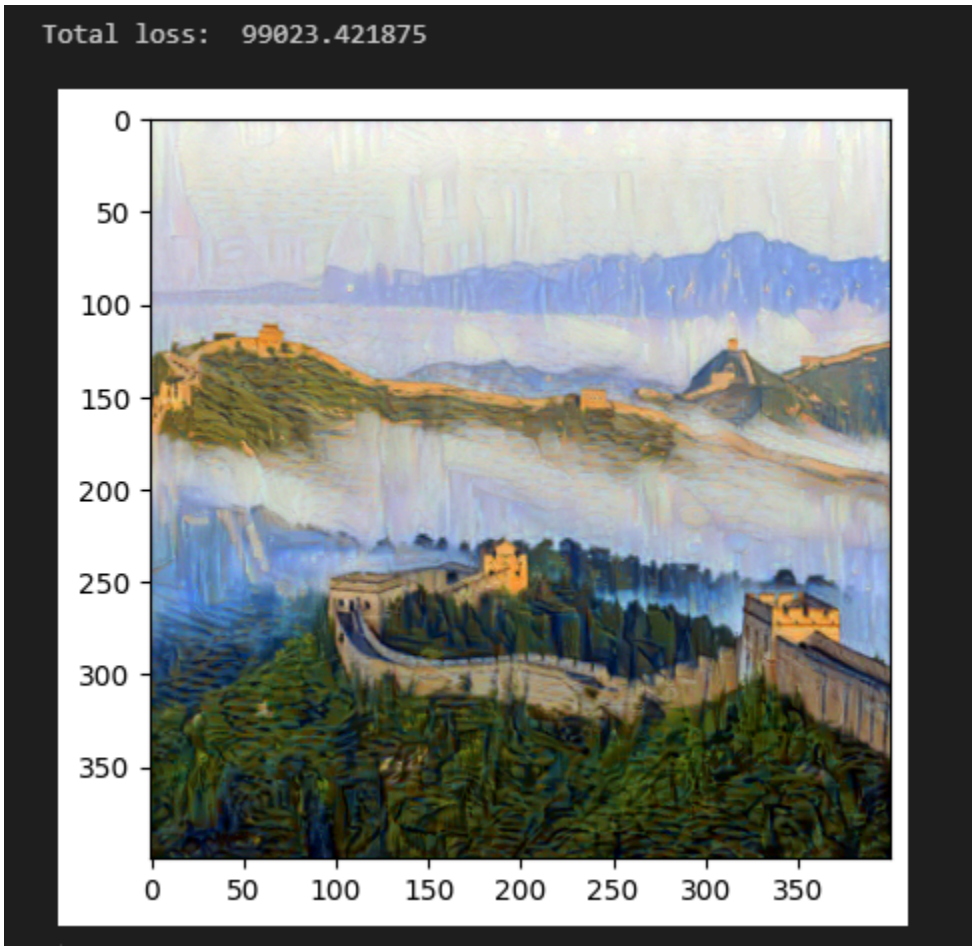
```
steps = 2000  # decide how many iterations to update your image (5000)
optimizer = optim.Adam([target], lr=0.001)
```

We got total loss: 24332762.0 ->17631904.0->14262982.0->11842479.0->9875554.0
We can see as steps increase the model having better time to train the model to perform better but optimizer learning step lower to 0.001 is will lead the model to more focus style type.

Train 5: we will shift model focus more on content and less for style(steps still 1000, lr is 0.003)

```
content_weight = 1e2  # alpha
style_weight = 1e4  # beta
```

So we lower the style weight and increase the content weight:

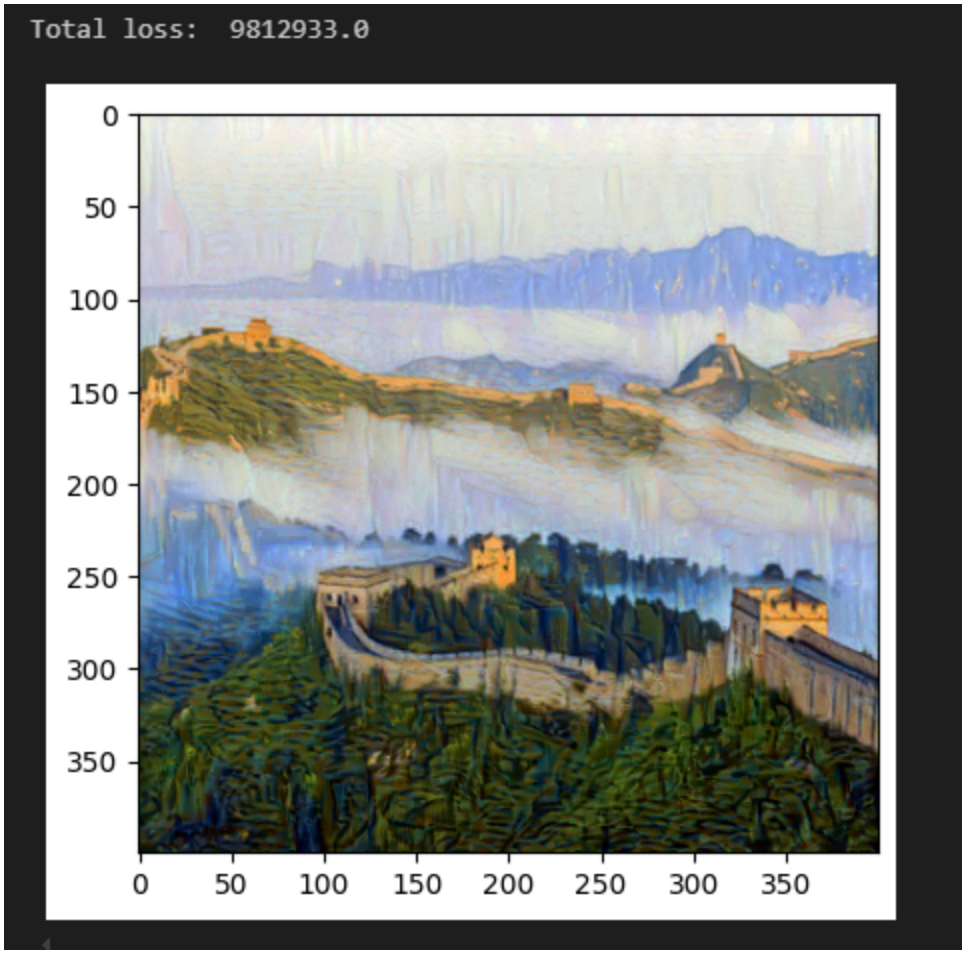


We can see the oil painting style is much less than training 4. The coloring and original image shape is more clear.

Train 6: we will try select different layer as style layer, the original layer was selected on conv4_2, now we switch to conv5_2 which is more deep in the neural network

```
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  #'21': 'conv4_2',
                  '28': 'conv5_1',
                  '30': 'conv5_2',
                  }
```

```
content_loss = torch.mean((target_features['conv5_2'] - content_features['conv5_2'])**2)
```



We can see it's similar to conv4_2, not really changing much, but it is supposed to be more abstract, more oil painting style.

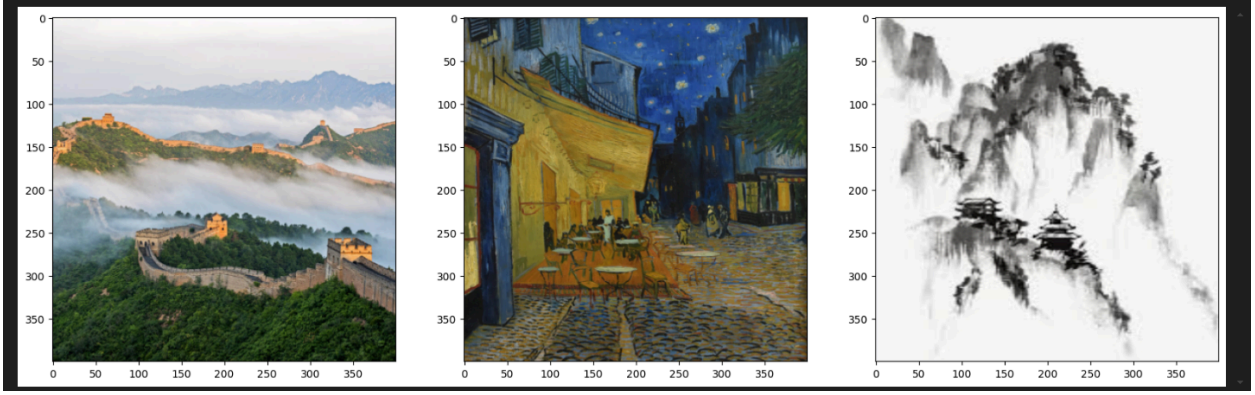
Train 7: we will try add another different type to mix up the final product which is using two different style into target picture

```
style = load_image('m.style.image.jpg', shape=content.shape[-2:]).to(device)
style2 = load_image('m.style.image2.jpg', shape=content.shape[-2:]).to(device)
```

```python
# display the images
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(20, 10))
# content and style ims side-by-side
ax1.imshow(im_convert(content))
ax2.imshow(im_convert(style))
ax3.imshow(im_convert(style2))
```

✓ 0.7s Python

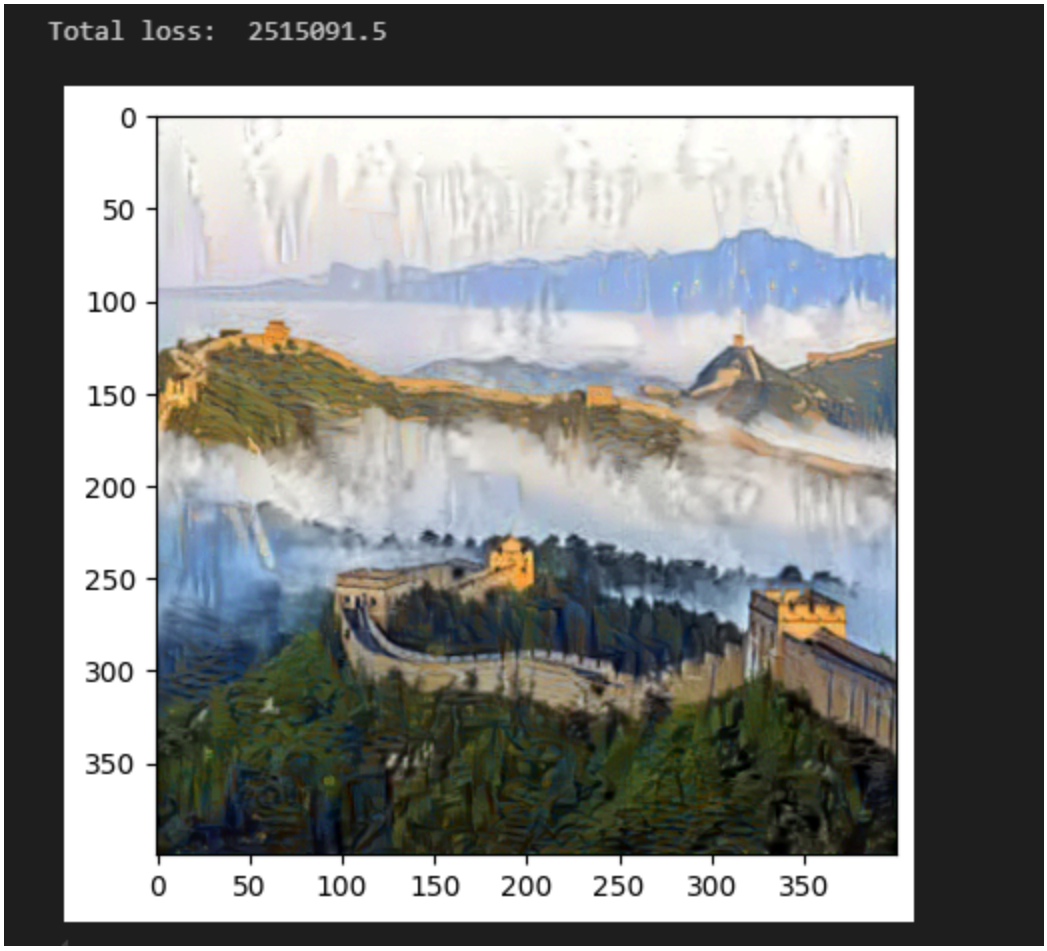
```
<matplotlib.image.AxesImage at 0x7f78031c48b0>
```



```python
style_features = get_features(style, vgg)
style2_features = get_features(style2, vgg)

# calculate the gram matrices for each layer of our style representation
style_grams = {layer: gram_matrix(style_features[layer]) for layer in style_features}
style2_grams = {layer: gram_matrix(style2_features[layer]) for layer in style2_features}
```

```python
style_gram = 0.5 * style_grams[layer] + 0.5 * style2_grams[layer]
```

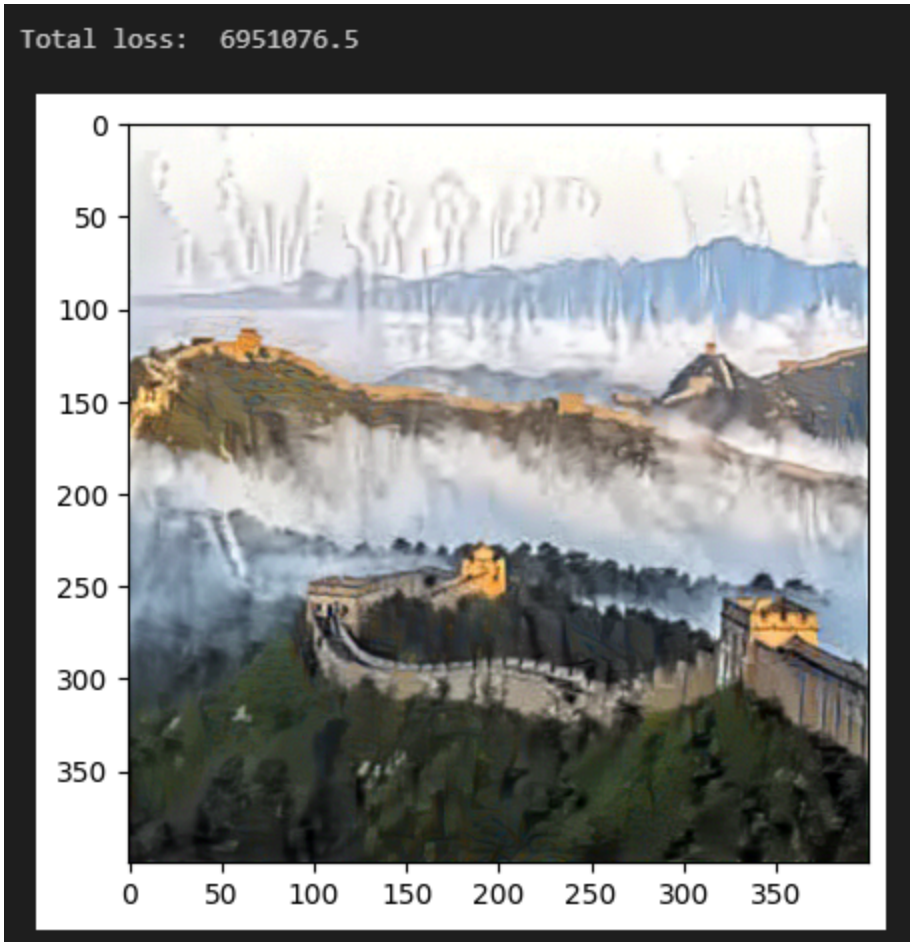
We will adding another type of style ink painting to see how it makes target pictures different, we will set both weight to 0.5 and 0.5 to generate new pictures



We can see we had really low loss and the picture also had more black blurring dots in the images.

```
style_gram = 0.2 * style_grams[layer] + 0.8 * style2_grams[layer]
```

When we focus more on ink painting instead of oil painting, weight becomes 0.2 for oil and 0.8 for ink.



We can see the final product of the image is more blurring not abstract as Gogh's type image.

Conclusion

In this assignment, we explored style transfer using a deep neural network base on VGG19 model following Gaty's paper, after perform the base training interactions, we adjusted few key parameters such as style weight, content weight, optimizer learning rate, training steps and layer selections to find their impact on the model.
Key finding:

- Style_weight
    - A high style weight 1e6 -> 5e6 will make a stronger style influences so that making image more abstract and oil-painting like
    - A low style weight 1e5 -> 1e4 will keep more of the original content.
- Layer_weight

    - Higher weight on lower layers(more towards start) will make model learning more focus on size, shape, colors
    - Higher weight on higher lawyer(more towards end) will make model more focus style and content
- Learning Rate and Steps
    - Lowering the learning rate 0.003->0.001 and increase steps 1000->2000 allowed for smoother and more stable convergence
    - Higher lr led to faster updates but will increase the style weight.
- Layer Selection
    - Using Conv4_2 vs Conv5_2 doesn't shows much difference but more deep layer select as style lawyer will lead to more abstract effects
- Combining Multiple Style
    - Bleeding two different styles(oil & ink painting) allowed for unique results, such that lower loss.
    - The weight different between two styles will significantly impacted texture style in the final output

The result shows that fine-tuning parameters and layer selections are the most important role for the model to achieving goal artistic effects in style transfer, while a higher style weight leads to a more intense transformation, balancing content and style weight will allows model to perform better on general image produce.