Assignment 2: Style Transfer

CS20 SI: TensorFlow for Deep Learning Research (cs20si.stanford.edu)

Due 2/18 at 11:59pm

Prepared by Chip Huyen (huyenn@stanford.edu)

Since you guys seemed really enthusiastic about implementing Deep Dream in class, I think you'll enjoy this assignment. You'll be implementing the much hyped neural style transfer. Neural style is so cool even Kristen Stewart co-authored a paper about it. Fun fact, our guest lecturer, Justin Johnson, has a citation from the Twilight star. How many academics can say that about their career?

For those of you who have been unaware of the hype, style transfer is the task of transferring the style of an image to another image. You should definitely read the original paper "A Neural Algorithm of Artistic Style" by Gatys et al. to understand the motivation and the intuition for this.

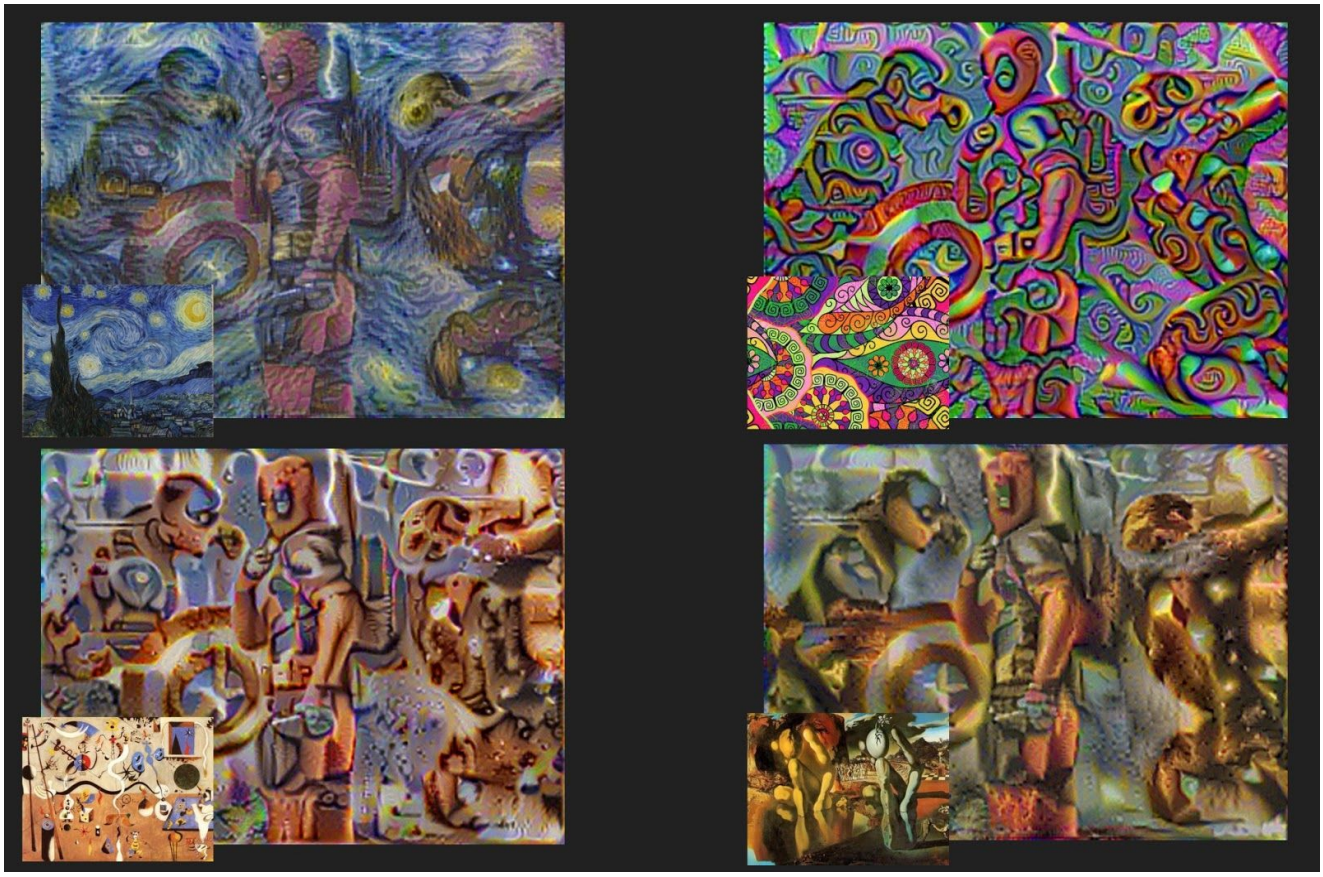
Basically, we will build a system that can take in two images (one content image and one style image), and output another image whose content is closest to the content of the content image while style is closest to the style of the style image.

For example, take this image of Deadpool in Civil War as the content image (I really hope whoever holds the copyright to this image doesn't sue me) and combine it with Guernica by Picasso (1937) as the style image.

After 160 iterations, you get this:

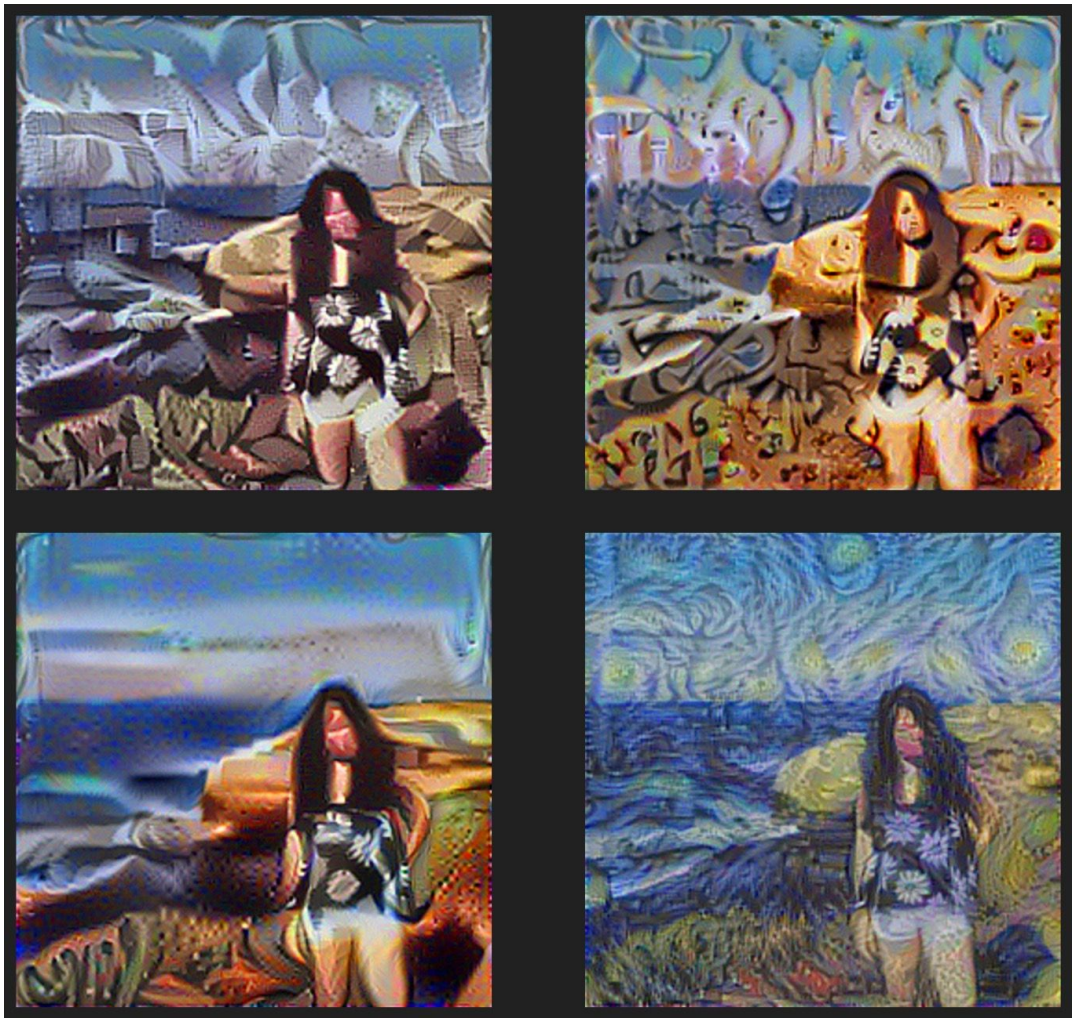
Because this image is big, 600 x 800 in dimension, so 160 iterations took approximately 1.5 hours. With smaller images, for example, 250 x 333, that'd take only around 15 minutes.

More Deadpool in arts:



The style images, from left to right, top to bottom are:
‘The Starry Night’ by Vincent van Gogh (1889)
Pattern by Olga Drozdova (201x?)
‘The Harlequin's Carnival’ by Joan Miró (1924-1925)
‘Metamorphosis of Narcissus’ by Salvador Dalí (1937)

Or if you’re as self-indulgent as I am, you can train your model on your own images to create this artsy collage of yourself. Can you guess which paintings are used as the style images for these photos?



The above are the images I generated from the model I implemented for this assignment. If you build your model correctly, you'll be able to generate other similar but much cooler images. Note that to speed up the computation, I used quite low resolution images (256 x 256 or 250 x 333), and that took me about 10 - 15 minutes to generate each image. When you have time at home, you should use higher resolution images to maximize the awesomeness.

The model is not mathematically difficult. In fact, it's rather straight-forward. However, the implementation is a bit involved since it's different from most of the models we've implemented in three aspects:

1. For previous models, we read in data and train variables -- we don't try to modify our original inputs. For this model, you have 2 fixed inputs: content image and style image, but also have a trainable input which will be trained to become the artwork. In this aspect, it's similar to Deep Dream. You input a random noise image and train it to look like a flower canvas.

2. There is not a clear distinction between the assembling the graph and the execution of the graph. For example, we can't assemble the graph at once, and then go on to run a session without ever having to add anything to the graph again.

All the 3 input (content image, style image, and trainable input) have the same dimensions and share the same kind of computations, so to save us from having to assemble the same subgraph multiple times, we will use one variable for all three of them. The variable is already defined for you in the model as:

```
input_image = tf.Variable(np.zeros([1, IMAGE_HEIGHT, IMAGE_WIDTH, 3]), dtype=tf.float32)
```

I know you're thinking, "What do you mean all three inputs share the same variable? How does TF know which input it's dealing with?" You remember that in TF we have the assign op for Variable. When we need to do some computation that takes in the content image as the input, we first assign the content image to that variable, and so on. You'll have to do that to calculate content loss (since it depends on the content image), and style loss (since it depends on the style image).

3. We don't train the weights from a random distribution but we'll use the weights and biases already trained for the object recognition task of the VGGNet of 19 layers. For more context on this model, you should read about it here. We'll only use their weights for the convolution layers. The paper by Gatys et al. suggested that average pooling is better than max pooling, so we'll have to do pooling ourselves.

But other than that, this is a typical model. Remember that in a model, you have several steps:
Step 1: Define inference
Step 2: Create loss functions
Step 3: Create optimizer
Step 4: Create summaries to monitor the progress of your training process
Step 5: Train your model

You'll progressively do all of those in this assignment. I suggest that you do all the tasks in order.

There are 3 files and 2 folders in the starter code:
style_transfer.py is the main file. You'll call python style_transfer.py to run your model.
load_vgg.py is where you load the VGG weights to your model and do all the convolution, relu, and pooling.
utils.py contains utils for the assignment. You should read it to know what each util function does, but you shouldn't need to modify this file.

The folder **styles** contains several paintings that you can use as your style images, and the folder content contains just one image deadpool.jpg that you can use as your content image. Feel free to add more style and content images.

You can download the starter code from the class GitHub repository.

While building your model, you should create respective folders to hold 3 things:

1. Checkpoints
2. Outputs (generated images at certain step intervals). For example, you want to save the generated image after every 10 intervals to see how it changes.
3. Graphs (so you can visualize your model on Tensorboard)

Okay, are you ready?

Step 1: Define inference
You should modify the function **_conv2d_relu()** and **_avgpool()** in **load_vgg.py**.
If you have problems with this part, you should refer to the convolutional model we built for MNIST.

Step 2: Create loss functions
You'll have to modify the function **_create_content_loss()**, **_create_style_loss()** and the helper functions **_single_style_loss()** and **_gram_matrix()** in **style_transfer.py**. This part is tricky, so please read the following instructions very carefully.

You have two losses and you try to minimize the combination of them. The content loss is defined as following:

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

where F is the feature representation of the generated image and P is the feature representation of the content image layer l. The paper suggests that we use the feature map from the layer **'conv4_2'** (I've already extracted the feature map for you in starter code). So this is basically the mean squared error of F and P.

However, in practice, we've found that this function makes it really slow to converge, so people often replace the coefficient ½ with 1/(4s) in which s is the product of the dimension of P. If P has dimension [5, 5, 3] then s = 5 * 5 * 3 = 75.

The style loss is defined as following:

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

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

N is the third dimension of the feature map, and M is the product of the first two dimensions of the feature map. However, remember that in TensorFlow, we have to add one extra dimension to make it 4D to make it work for the function tf.nn.conv2d, so the first dimension is actually the second, and the second is the third, and so on.

A is the Gram matrix from the original image and G is the Gram matrix of the image to be generated. To obtain the gram matrix, for example, of the style image, we first need to get the feature map of the style image at that layer, then reshape it to 2D tensor of dimension M x N, and take the dot product of 2D tensor with its transpose. You do the same thing to get the gram matrix from the feature map of the generated image. If you don't know what gram matrix does, you should look it up to understand. I recommend the Wikipedia article and this video.

The subscript l stands the layer whose feature maps we want to incorporate into the generated images. In the paper, it suggests that we use feature maps from 5 layers:

```
['conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1']
```

After you've calculated the tensors E's, you calculate the style loss by summing them up with their corresponding weight w's. You can tune w's, but I'd suggest that you give more emphasis to deep layers. For example, w for 'conv1_1' can be 1, then weight for 'conv2_2' can be 2, and so on.

After you've got your content loss and style loss, you should combine them with their corresponding weights to get the total loss -- and the total loss is what we'll try to minimize.

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

The paper suggests that we use alpha and beta such that alpha/beta = 0.001 or 0.0001, but I've found that the ratio alpha/beta = 1/20 or 1/50 works just fine.

Step 3: Create optimizer
I suggest AdamOptimizer but you can be creative and see what you find. You should find this part in the **main()** function.

Step 4: Create summaries

You need to summary ops for the values that you want to monitor through your training process in TensorBoard. You should modify the TO DO part in the **main()** function of **style_transfer.py**

Train your model for at least 200 iterations and submit the content loss graph, style loss graph, the total loss graph, and the graph of your model. You should justify in at most 3 sentences what you see in the loss graphs and why.

Step 5: Train your model
You should modify the TO DO parts in the **train()** function of **style_transfer.py**

There are a lot of parameters that you can play around with. For example, you can change the weights for the content loss and the style loss, change the coefficients for content and style losses, change the learning rate, use different layers for style and content, etc.

Deliverables
The finished code

1. The content loss graph, style loss graph, and the total loss graph. Explanation of what you see in the loss graphs.
2. The graph of your model.
3. Change at least two parameters, explain what you did and how that changed the results.
4. 3 artworks generated using at least 3 different styles.

Submission instruction
After you've finished, zip up all deliverables and name it using your SUNet ID and send it to my email huyenn@stanford.edu.

Have fun!