Assignment 2: Style Transfer

CS20: TensorFlow for Deep Learning Research (cs20.stanford.edu)
Due 2/19 at 11:59pm
Prepared by Chip Huyen (chiphuyen@cs.stanford.edu)

In this assignment, you'll be implementing the much hyped neural style transfer. Neural style is so cool even the Twilight star co-authored a paper about it.

Bringing Impressionism to Life with Neural Style Transfer in *Come Swim*

Bhautik J Joshi*
Research Engineer, Adobe

Kristen Stewart
Director, *Come Swim*

David Shapiro
Producer, Starlight Studios

Figure 1: *Usage of Neural Style Transfer in Come Swim; left: content image, middle: style image, right: upsampled result. Images used with permission, (c) 2017 Starlight Studios LLC & Kristen Stewart.*

For those who have been unaware of the hype, style transfer is the task of transferring the style of an image to another image. Please read the paper A Neural Algorithm of Artistic Style (Gatys et al., 2016) to understand the motivation and intuition behind this.

We want to build a system that takes two images and outputs another image whose content is closest to one image while style is closest to the style of the other.

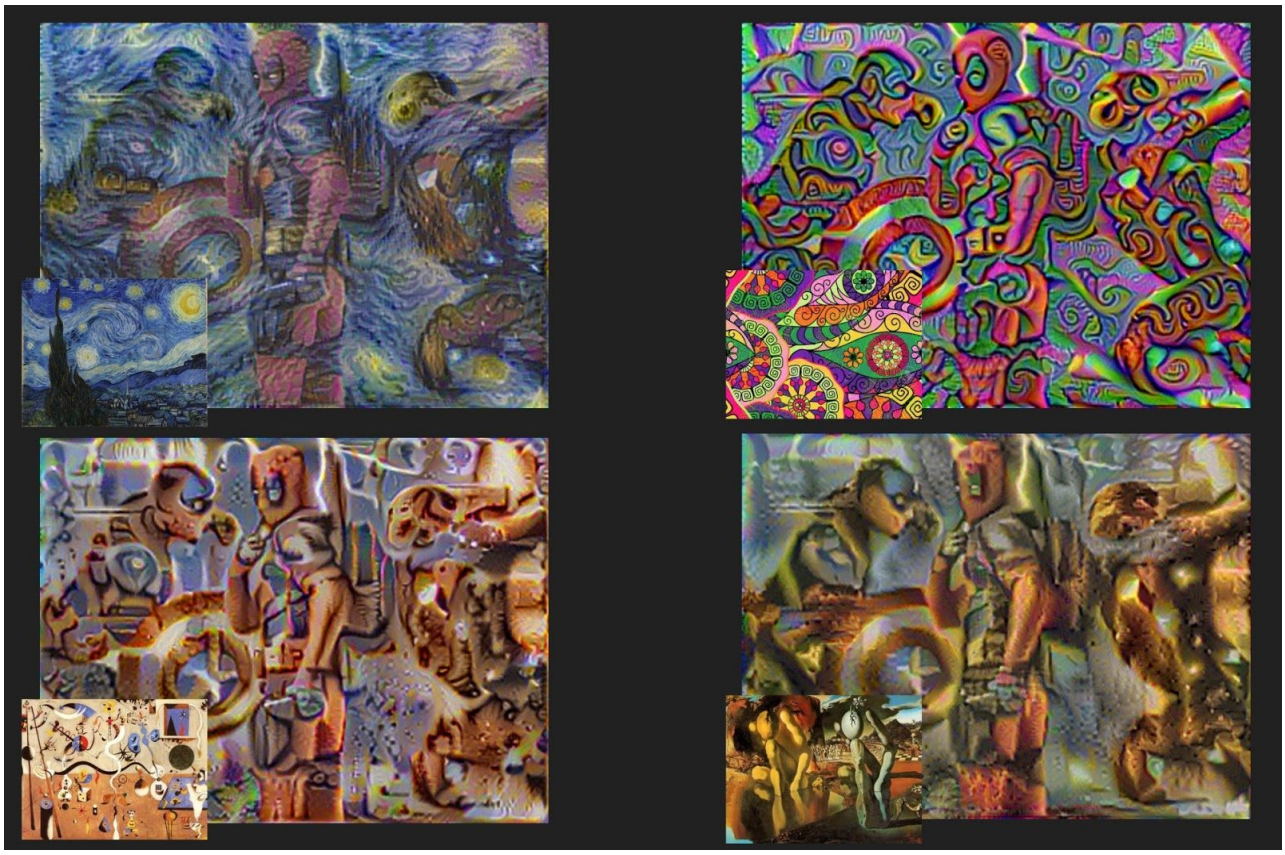
For example, take this image of Deadpool in Civil War[1] as the content image and combine it with the style of Guernica by Picasso (1937).

[1] Image source: theodysseyonline.com

After 160 iterations, you get this:

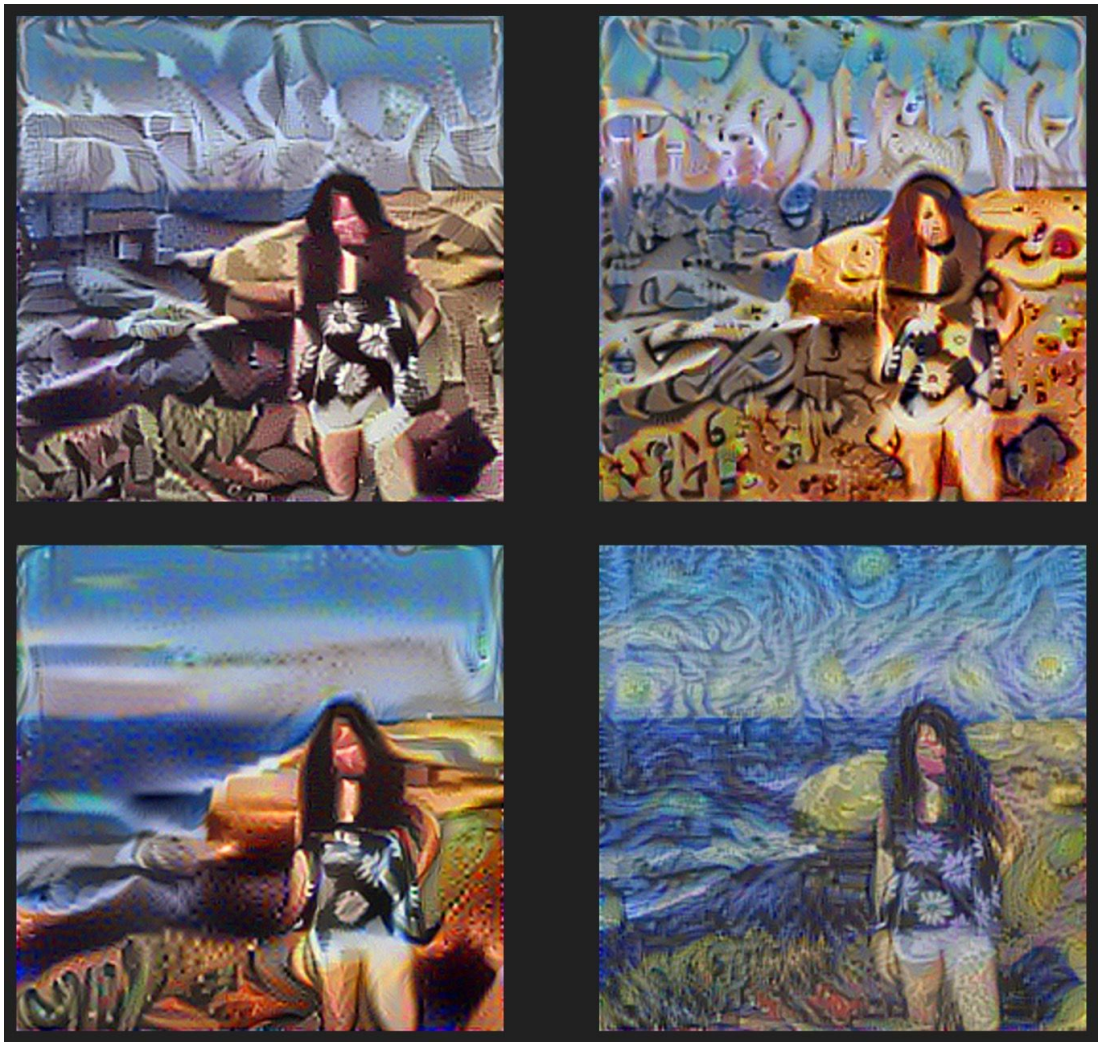
The above image is pretty big (600 x 800), 160 iterations took approximately 1.5 hours. Smaller images, for example, 250 x 333, would take around 15 minutes.

More Deadpool art



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. To speed up the computation during writing code, you might want to use lower resolution images (about 250 x 333). After you're done, you should feel free to use images with higher resolution to maximize the awesomeness.

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

1. In the models that we've learned so far, we import data and use it to train variables -- we don't try to modify our original inputs. For this model, you have two fixed inputs: content image and style image, but also have a trainable input which will be trained to become the generated artwork.

2. There is not a clear distinction between the two phases of a TensorFlow program: assembling the graph and executing it. All the 3 input (content image, style image, and trainable input) have the same dimensions and act as input to the same computation to extract the same sets of features. 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:

```
self.input_img = tf.get_variable('in_img',
                                 shape=([1, self.img_height, self.img_width, 3]),
                                 dtype=tf.float32,
                                 initializer=tf.zeros_initializer())
```

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 variables. 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. In this assignment, you'll get acquainted to transfer learning: we use the weights trained for another task for this task. We will use the weights and biases already trained for the object recognition task of the model VGG-19[2] (a convolutional network with 19 layers) to extract content and style layers for style transfer. 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 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 that order.

There are 3 files and 2 folders in the starter code, which you should be able to find on GitHub.

style_transfer.py is the main file. You'll call python style_transfer.py to run your model. You'll have to modify this file.

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

load_vgg.py is where you load in the trained VGG variables. You'll have to modify this file.
utils.py contains utils for the assignment. You should read it to know what utilities are offered. You shouldn't have to modify this file, but you're welcome to do so if you find it necessary.

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.

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 **_content_loss()**, **_style_loss()** and their corresponding 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'**. The loss function 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 with both optimizers and learning rate to see what you find. You can find this part in the **optimize**() method in **style_transfer.py.**

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 find this in the **create_summary**() method in **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()** method of **style_transfer.py**

There are a lot of hyperparameters that you can play around with. See them in __init__ method for **StyleTransfer**. 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.

You can see how the generated images look like after certain number of iterations in the folder **output**. The training progress looks like this:

Deliverables

1. The finished code.
2. The training curve of content loss, style loss, and the total loss. Write a few sentences about what you see.
3. The graph of your model.
4. Change at least two parameters, explain what you did and how that changed the results.
5. 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. Send it to cs20-win1718-staff@lists.stanford.edu with subject title: "[Your SUNetID]_assignment2"

Have fun!