cl6

April 14, 2024

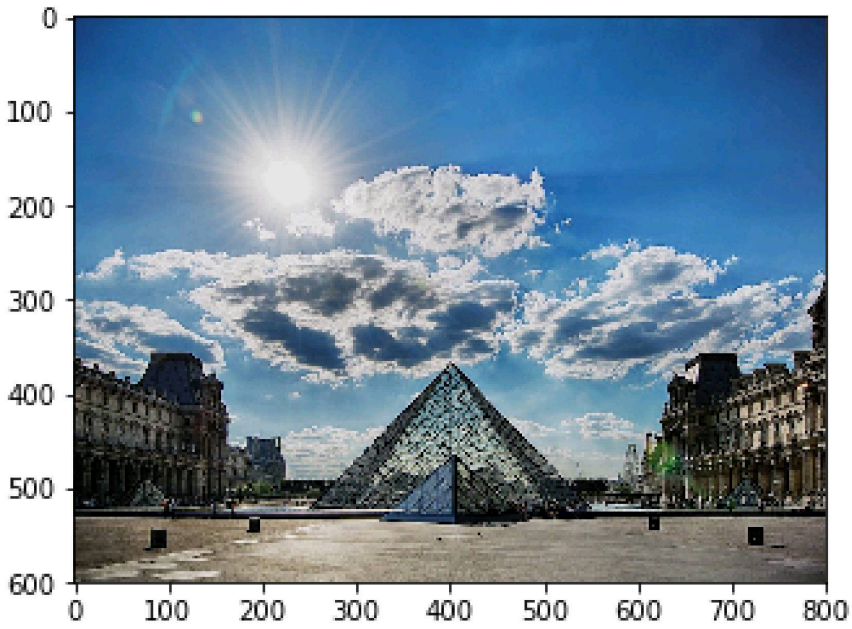
[1]:
```
import os
import sys
import scipy.io
import scipy.misc
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
from PIL import Image
from nst_utils import *
import numpy as np
import tensorflow as tf

%matplotlib inline
```

[2]:
```
model = load_vgg_model("pretrained-model/imagenet-vgg-verydeep-19.mat")
print(model)
```

```
{'input': <tf.Variable 'Variable:0' shape=(1, 300, 400, 3) dtype=float32_ref>,
'conv1_1': <tf.Tensor 'Relu:0' shape=(1, 300, 400, 64) dtype=float32>,
'conv1_2': <tf.Tensor 'Relu_1:0' shape=(1, 300, 400, 64) dtype=float32>,
'avgpool1': <tf.Tensor 'AvgPool:0' shape=(1, 150, 200, 64) dtype=float32>,
'conv2_1': <tf.Tensor 'Relu_2:0' shape=(1, 150, 200, 128) dtype=float32>,
'conv2_2': <tf.Tensor 'Relu_3:0' shape=(1, 150, 200, 128) dtype=float32>,
'avgpool2': <tf.Tensor 'AvgPool_1:0' shape=(1, 75, 100, 128) dtype=float32>,
'conv3_1': <tf.Tensor 'Relu_4:0' shape=(1, 75, 100, 256) dtype=float32>,
'conv3_2': <tf.Tensor 'Relu_5:0' shape=(1, 75, 100, 256) dtype=float32>,
'conv3_3': <tf.Tensor 'Relu_6:0' shape=(1, 75, 100, 256) dtype=float32>,
'conv3_4': <tf.Tensor 'Relu_7:0' shape=(1, 75, 100, 256) dtype=float32>,
'avgpool3': <tf.Tensor 'AvgPool_2:0' shape=(1, 38, 50, 256) dtype=float32>,
'conv4_1': <tf.Tensor 'Relu_8:0' shape=(1, 38, 50, 512) dtype=float32>,
'conv4_2': <tf.Tensor 'Relu_9:0' shape=(1, 38, 50, 512) dtype=float32>,
'conv4_3': <tf.Tensor 'Relu_10:0' shape=(1, 38, 50, 512) dtype=float32>,
'conv4_4': <tf.Tensor 'Relu_11:0' shape=(1, 38, 50, 512) dtype=float32>,
'avgpool4': <tf.Tensor 'AvgPool_3:0' shape=(1, 19, 25, 512) dtype=float32>,
'conv5_1': <tf.Tensor 'Relu_12:0' shape=(1, 19, 25, 512) dtype=float32>,
'conv5_2': <tf.Tensor 'Relu_13:0' shape=(1, 19, 25, 512) dtype=float32>,
'conv5_3': <tf.Tensor 'Relu_14:0' shape=(1, 19, 25, 512) dtype=float32>,
'conv5_4': <tf.Tensor 'Relu_15:0' shape=(1, 19, 25, 512) dtype=float32>,
'avgpool5': <tf.Tensor 'AvgPool_4:0' shape=(1, 10, 13, 512) dtype=float32>}
```

[3]:
```
content_image = scipy.misc.imread("images/louvre.jpg")
imshow(content_image)
```

[3]: <matplotlib.image.AxesImage at 0x7fc364fda400>



[4]:
```
# GRADED FUNCTION: compute_content_cost

def compute_content_cost(a_C, a_G):
    """
    Computes the content cost

    Arguments:
    a_C -- tensor of dimension (1, n_H, n_W, n_C), hidden layer activations␣
 ↪representing content of the image C
    a_G -- tensor of dimension (1, n_H, n_W, n_C), hidden layer activations␣
 ↪representing content of the image G

    Returns:
    J_content -- scalar that you compute using equation 1 above.
    """

    ### START CODE HERE ###
    # Retrieve dimensions from a_G ( 1 line)
    m, n_H, n_W, n_C = a_G.get_shape().as_list()
```

```
    # Reshape a_C and a_G (2 lines)
    a_C_unrolled = tf.transpose(a_C)
    a_G_unrolled = tf.transpose(a_G)

    # compute the cost with tensorflow (1 line)
    J_content = (1/ (4* n_H * n_W * n_C)) * tf.reduce_sum(tf.pow((a_G_unrolled
 ↪- a_C_unrolled), 2))
    ### END CODE HERE ###

    return J_content
```

[5]:
```
tf.reset_default_graph()

with tf.Session() as test:
    tf.set_random_seed(1)
    a_C = tf.random_normal([1, 4, 4, 3], mean=1, stddev=4)
    a_G = tf.random_normal([1, 4, 4, 3], mean=1, stddev=4)
    J_content = compute_content_cost(a_C, a_G)
    print("J_content = " + str(J_content.eval()))
```

```
J_content = 6.76559
```

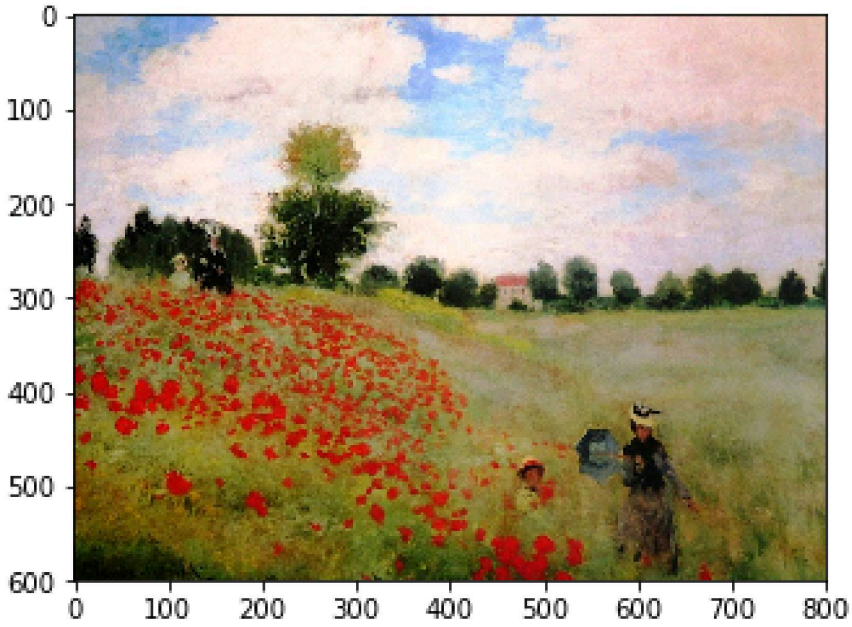
Expected Output:

J_content

6.76559

[6]:
```
style_image = scipy.misc.imread("images/monet_800600.jpg")
imshow(style_image)
```

[6]:
```
<matplotlib.image.AxesImage at 0x7fc3646f5cf8>
```



[7]:
```
# GRADED FUNCTION: gram_matrix

def gram_matrix(A):
    """
    Argument:
    A -- matrix of shape (n_C, n_H*n_W)

    Returns:
    GA -- Gram matrix of A, of shape (n_C, n_C)
    """

    ### START CODE HERE ### ( 1 line)
    GA = tf.matmul(A, tf.transpose(A))
    ### END CODE HERE ###

    return GA
```

[8]:
```
tf.reset_default_graph()

with tf.Session() as test:
    tf.set_random_seed(1)
    A = tf.random_normal([3, 2*1], mean=1, stddev=4)
    GA = gram_matrix(A)

    print("GA = " + str(GA.eval()))
```

```
GA = [[  6.42230511  -4.42912197  -2.09668207]
 [ -4.42912197  19.46583748  19.56387138]
 [ -2.09668207  19.56387138  20.6864624 ]]
```

Expected Output:

GA

[[6.42230511 -4.42912197 -2.09668207] [-4.42912197 19.46583748 19.56387138] [-2.09668207 19.56387138 20.6864624]]

[9]:
```
# GRADED FUNCTION: compute_layer_style_cost

def compute_layer_style_cost(a_S, a_G):
    """
    Arguments:
    a_S -- tensor of dimension (1, n_H, n_W, n_C), hidden layer activations representing style of the image S
    a_G -- tensor of dimension (1, n_H, n_W, n_C), hidden layer activations representing style of the image G

    Returns:
    J_style_layer -- tensor representing a scalar value, style cost defined above by equation (2)
    """

    ### START CODE HERE ###
    # Retrieve dimensions from a_G (1 line)
    m, n_H, n_W, n_C = a_G.get_shape().as_list()

    # Reshape the images to have them of shape (n_H*n_W, n_C) (2 lines)
    a_S = tf.transpose(tf.reshape(a_S, [n_H*n_W, n_C]))
    a_G = tf.transpose(tf.reshape(a_G, [n_H*n_W, n_C]))

    # Computing gram_matrices for both images S and G (2 lines)
    GS = gram_matrix(a_S)
    GG = gram_matrix(a_G)

    # Computing the loss (1 line)
    J_style_layer = (1./(4 * n_C**2 * (n_H*n_W)**2)) * tf.reduce_sum(tf.pow((GS - GG), 2))

    ### END CODE HERE ###

    return J_style_layer
```

[10]:
```
tf.reset_default_graph()

with tf.Session() as test:
```

```
    tf.set_random_seed(1)
    a_S = tf.random_normal([1, 4, 4, 3], mean=1, stddev=4)
    a_G = tf.random_normal([1, 4, 4, 3], mean=1, stddev=4)
    J_style_layer = compute_layer_style_cost(a_S, a_G)

    print("J_style_layer = " + str(J_style_layer.eval()))
```

```
J_style_layer = 9.19028
```

Expected Output:

J_style_layer

9.19028

[11]:
```
STYLE_LAYERS = [
    ('conv1_1', 0.2),
    ('conv2_1', 0.2),
    ('conv3_1', 0.2),
    ('conv4_1', 0.2),
    ('conv5_1', 0.2)]
```

[12]:
```
def compute_style_cost(model, STYLE_LAYERS):
    """
    Computes the overall style cost from several chosen layers

    Arguments:
    model -- our tensorflow model
    STYLE_LAYERS -- A python list containing:
                        - the names of the layers we would like to extract␣
 ↪style from
                        - a coefficient for each of them

    Returns:
    J_style -- tensor representing a scalar value, style cost defined above by␣
 ↪equation (2)
    """

    # initialize the overall style cost
    J_style = 0

    for layer_name, coeff in STYLE_LAYERS:

        # Select the output tensor of the currently selected layer
        out = model[layer_name]

        # Set a_S to be the hidden layer activation from the layer we have␣
 ↪selected, by running the session on out
        a_S = sess.run(out)
```

```
        # Set a_G to be the hidden layer activation from same layer. Here, a_G␣
↪references model[layer_name]
        # and isn't evaluated yet. Later in the code, we'll assign the image G␣
↪as the model input, so that
        # when we run the session, this will be the activations drawn from the␣
↪appropriate layer, with G as input.
        a_G = out

        # Compute style_cost for the current layer
        J_style_layer = compute_layer_style_cost(a_S, a_G)

        # Add coeff * J_style_layer of this layer to overall style cost
        J_style += coeff * J_style_layer

    return J_style
```

[13]:
```
# GRADED FUNCTION: total_cost

def total_cost(J_content, J_style, alpha = 10, beta = 40):
    """
    Computes the total cost function

    Arguments:
    J_content -- content cost coded above
    J_style -- style cost coded above
    alpha -- hyperparameter weighting the importance of the content cost
    beta -- hyperparameter weighting the importance of the style cost

    Returns:
    J -- total cost as defined by the formula above.
    """

    ### START CODE HERE ### (1 line)
    J = alpha * J_content + beta * J_style
    ### END CODE HERE ###

    return J
```

[14]:
```
tf.reset_default_graph()

with tf.Session() as test:
    np.random.seed(3)
    J_content = np.random.randn()
    J_style = np.random.randn()
    J = total_cost(J_content, J_style)
    print("J = " + str(J))
```

```
J = 35.34667875478276
```

Expected Output:

J

35.34667875478276

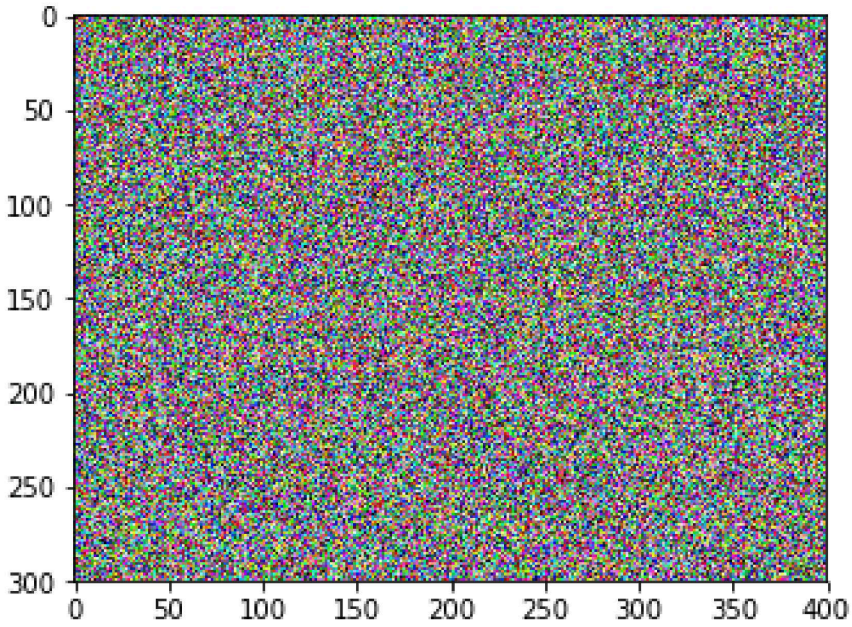
[15]:
```
# Reset the graph
tf.reset_default_graph()

# Start interactive session
sess = tf.InteractiveSession()
```

[16]:
```
content_image = scipy.misc.imread("images/louvre_small.jpg")
content_image = reshape_and_normalize_image(content_image)
```

[17]:
```
style_image = scipy.misc.imread("images/monet.jpg")
style_image = reshape_and_normalize_image(style_image)
```

[18]:
```
generated_image = generate_noise_image(content_image)
imshow(generated_image[0])
```

[18]:
```
<matplotlib.image.AxesImage at 0x7fc35c789128>
```



Next, as explained in part (2), let's load the VGG16 model.

[19]:
```
model = load_vgg_model("pretrained-model/imagenet-vgg-verydeep-19.mat")
```

[20]:
```
# Assign the content image to be the input of the VGG model.
sess.run(model['input'].assign(content_image))

# Select the output tensor of layer conv4_2
out = model['conv4_2']

# Set a_C to be the hidden layer activation from the layer we have selected
a_C = sess.run(out)

# Set a_G to be the hidden layer activation from same layer. Here, a_G␣
 ↪references model['conv4_2']
# and isn't evaluated yet. Later in the code, we'll assign the image G as the␣
 ↪model input, so that
# when we run the session, this will be the activations drawn from the␣
 ↪appropriate layer, with G as input.
a_G = out

# Compute the content cost
J_content = compute_content_cost(a_C, a_G)
```

[21]:
```
# Assign the input of the model to be the "style" image
sess.run(model['input'].assign(style_image))

# Compute the style cost
J_style = compute_style_cost(model, STYLE_LAYERS)
```

[22]:
```
### START CODE HERE ### (1 line)
J = total_cost(J_content, J_style, alpha = 10, beta = 40)
### END CODE HERE ###
```

[23]:
```
# define optimizer (1 line)
optimizer = tf.train.AdamOptimizer(2.0)

# define train_step (1 line)
train_step = optimizer.minimize(J)
```

[ ]:
```
def model_nn(sess, input_image, num_iterations = 200):

    # Initialize global variables (you need to run the session on the␣
 ↪initializer)
    ### START CODE HERE ### (1 line)
    sess.run(tf.global_variables_initializer())
    ### END CODE HERE ###

    # Run the noisy input image (initial generated image) through the model.␣
 ↪Use assign().
    ### START CODE HERE ### (1 line)
```

```
    sess.run(model['input'].assign(input_image))
    ### END CODE HERE ###

    for i in range(num_iterations):

        # Run the session on the train_step to minimize the total cost
        ### START CODE HERE ### (1 line)
        sess.run(train_step)
        ### END CODE HERE ###

        # Compute the generated image by running the session on the current␣
↪model['input']
        ### START CODE HERE ### (1 line)
        generated_image = sess.run(model['input'])
        ### END CODE HERE ###

        # Print every 20 iteration.
        if i%20 == 0:
            Jt, Jc, Js = sess.run([J, J_content, J_style])
            print("Iteration " + str(i) + " :")
            print("total cost = " + str(Jt))
            print("content cost = " + str(Jc))
            print("style cost = " + str(Js))

            # save current generated image in the "/output" directory
            save_image("output/" + str(i) + ".png", generated_image)

    # save last generated image
    save_image('output/generated_image.jpg', generated_image)

    return generated_image
```

[ ]:
```
model_nn(sess, generated_image)
```

```
Iteration 0 :
total cost = 5.05035e+09
content cost = 7877.67
style cost = 1.26257e+08
Iteration 20 :
total cost = 9.43272e+08
content cost = 15187.2
style cost = 2.3578e+07
Iteration 40 :
total cost = 4.84905e+08
content cost = 16785.1
style cost = 1.21184e+07
Iteration 60 :
```

```
total cost = 3.12573e+08
content cost = 17466.2
style cost = 7.80995e+06
Iteration 80 :
total cost = 2.28117e+08
content cost = 17715.2
style cost = 5.69849e+06
```

Expected Output:

Iteration 0 :

total cost = 5.05035e+09 content cost = 7877.67 style cost = 1.26257e+08