Reinforcement Learning

For the second part of this project we will be implementing a simple Q-Learning algorithm on an RL environment called Cart Pole. The idea of Q-Learning is to try to estimate the expected future reward or Q-value of taking a certain action. Then at any given step we take the action with the most expected future reward.

In reinforcement learning, we refer to algorithms that attempt to solve environments as "agents", so in this part of the project we will be making a Deep Q Network Agent that will solve the Cart Pole environment.

In [1]:

```
!pip install gym tqdm
```

```
Collecting gym
  Downloading
https://files.pythonhosted.org/packages/9b/50/ed4a03d2be47ffd043be2ee514f32
5d98a30fe2d1b9c61dea5a9d861/gym-0.10.5.tar.gz (1.5MB)
    100% |████████████████████████████████| 1.5MB 566kB/s
Collecting tqdm
  Downloading
https://files.pythonhosted.org/packages/78/bc/de067ab2d700b91717dc5459d86a1
2df31abfb90ab01a5a5a5ce30b4/tqdm-4.23.0-py2.py3-none-any.whl (42kB)
    100% |████████████████████████████████| 51kB 1.6MB/s
Requirement already satisfied: numpy>=1.10.4 in
/anaconda3/envs/py36/lib/python3.6/site-packages (from gym)
Requirement already satisfied: requests>=2.0 in
/anaconda3/envs/py36/lib/python3.6/site-packages (from gym)
Requirement already satisfied: six in
/anaconda3/envs/py36/lib/python3.6/site-packages (from gym)
Collecting pyglet>=1.2.0 (from gym)
  Downloading
https://files.pythonhosted.org/packages/1c/fc/dad5eaaab68f0c21e2f906a94ddb9
662cc5a654eee404d59554ce0fa/pyglet-1.3.2-py2.py3-none-any.whl (1.0MB)
    100% |████████████████████████████████| 1.0MB 827kB/s
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in
/anaconda3/envs/py36/lib/python3.6/site-packages (from requests>=2.0->gym)
Requirement already satisfied: idna<2.7,>=2.5 in
/anaconda3/envs/py36/lib/python3.6/site-packages (from requests>=2.0->gym)
Requirement already satisfied: urllib3<1.23,>=1.21.1 in
/anaconda3/envs/py36/lib/python3.6/site-packages (from requests>=2.0->gym)
Requirement already satisfied: certifi>=2017.4.17 in
/anaconda3/envs/py36/lib/python3.6/site-packages (from requests>=2.0->gym)
Collecting future (from pyglet>=1.2.0->gym)
  Downloading
https://files.pythonhosted.org/packages/00/2b/8d082ddfed935f3608cc61140df6d
edea1bc3ab52fb6c29ae3e81e85/future-0.16.0.tar.gz (824kB)
    100% |████████████████████████████████| 829kB 1.0MB/s
Building wheels for collected packages: gym, future
  Running setup.py bdist_wheel for gym ... done
  Stored in directory:
/Users/chapatel/Library/Caches/pip/wheels/cb/14/71/f4ab006b1e6ff75c2b54985c
```

```
d0644fffe9c1dddc670925
  Running setup.py bdist_wheel for future ... done
  Stored in directory:
/Users/chapatel/Library/Caches/pip/wheels/bf/c9/a3/c538d90ef17cf7823fa51fc7(
a7a910a80f6a405bf15b1a
Successfully built gym future
Installing collected packages: future, pyglet, gym, tqdm
Successfully installed future-0.16.0 gym-0.10.5 pyglet-1.3.2 tqdm-4.23.0
You are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Part 1: Setup the Environment

In [2]:

```python
import gym
env = gym.make('CartPole-v0')
```

```
WARN: gym.spaces.Box autodetected dtype as <class 'numpy.float32'>. Please
provide explicit dtype.
```

Part 2: Create The DQN Agent

In [3]:

```python
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Activation
from collections import deque
import random
from keras.optimizers import Adam

import numpy as np


class DQNAgent:

    def __init__(self, env, replay_size=1000, epsilon=1.0, epsilon_min=0.01
, epsilon_decay=0.995, gamma=0.99):
        self.state_size = env.observation_space.shape[0]
        self.num_actions = env.action_space.n
        self.model = self.build_model()
        self.replay_buffer = deque(maxlen=replay_size)
        self.epsilon = epsilon
        self.epsilon_min = epsilon_min
        self.epsilon_decay = epsilon_decay
        self.gamma = gamma


    def build_model(self):
        model = Sequential()
        # TODO: add 2 dense layers each with 32 neurons, the input dim to t
he first
        # layer should be the state size, also add relu activations, for bo
th these layers
```

```
        # Then add another Dense layer with num_actions neurons.
        # Then use model.compile to compile the model with mse loss and an
Adam optimizer
        # with learning rate 0.001.
        model.add(Dense(32, input_dim = self.state_size))
        model.add(Activation("relu"))
        model.add(Dense(32))
        model.add(Activation("relu"))
        model.add(Dense(self.num_actions))
        keras.optimizers.Adam(lr=0.001)
        model.compile(optimizer = "Adam", loss="mse")


        return model

    def action(self, state):
        # Whenever a random number between 0 and 1 is less than epsilon we
want to return
        # a random action. This means that with probability epsilon we retu
rn a random action.
        if np.random.random() <= self.epsilon:
            return np.random.randint(self.num_actions)
            #TODO: return random action here
        # Now we want to use our model to get the q values
        # HINT: we want to do prediction

        q_values = self.model.predict(state)
        return np.argmax(q_values[0])

    def add_to_replay_buffer(self, state, action, reward, next_state, done)
:
        self.replay_buffer.append((state, action, reward, next_state,
done))

    def train_batch_from_replay(self, batch_size):
        # if we don't have enough samples in our replay buffer just return
        if len(self.replay_buffer) < batch_size:
            return False
        # TODO: randomly sample batch_size samples from the replay buffer
        # hint: use random.sample
        minibatch = random.sample(self.replay_buffer, batch_size)
        for state, action, reward, next_state, done in minibatch:
            target = reward
            if not done:
                next_Qs = self.model.predict(next_state)[0]
                # TODO: we want to add to our target GAMMA * max
Q(next_state)
                target += self.gamma * np.max(next_Qs)

            # our target should only take into account the current action
            # so we set all the Q values except the current action, to the
            # current output of our model so that they get ignored in the l
oss function.
            target_Qs = self.model.predict(state)
            target_Qs[0][action] = target
            self.model.fit(state, target_Qs, epochs=1, verbose=0)

        # Now we want to slowly decay how many random actions we take
        # to do this we can multiply epsilon by our epsilon decay parameter
        # each iteration
```

```
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

```
Using TensorFlow backend.
```

Part 3: Train the Model

In [4]:

```
agent = DQNAgent(env)
```

In [5]:

```
from tqdm import tqdm

done = False
batch_size = 32
num_episodes = 800

for episode in tqdm(range(num_episodes)):
    state = env.reset()
    state = np.reshape(state, [1, agent.state_size])

    for t in range(200):
        action = agent.action(state)
        next_state, reward, done, _ = env.step(action)
        reward = reward if not done else 100
        next_state = np.reshape(next_state, [1, agent.state_size])
        agent.add_to_replay_buffer(state, action, reward, next_state, done)
        # TODO: add this sample to the replay buffer

        state = next_state

        # TODO: train on a batch from the replay buffer
        agent.train_batch_from_replay(batch_size)
        if done:
            break
```

```
100%|██████████| 800/800 [34:59<00:00,  2.62s/it]
```

Part 4: Test the Model

In [6]:

```
#TODO: set the agent's epsilon so that we don't take any random actions.
for _ in range(10):
    state = env.reset()
    state = np.reshape(state, [1, agent.state_size])
    agent.episilon = -1
    total_reward = 0
    for t in range(200):
        action = agent.action(state)
        next_state, reward, done, _ = env.step(action)
        total_reward += reward
        state = np.reshape(next_state, [1, agent.state_size])
        # TODO: if you want to see the rendered version of your agent runni
ng
```

```
        # uncomment this line
        #env.render()
    print(total_reward)
```

```
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
40.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
23.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
27.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
21.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
35.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
21.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
24.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
46.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
20.0
WARN: You are calling 'step()' even though this environment has already ret
urned done = True. You should always call 'reset()' once you receive 'done
= True' -- any further steps are undefined behavior.
27.0
```

Part 5: Writeup

Now for the writeup portion write a paragraph of your understanding of how Deep Q Learning works.

Q-learning uses a simple update rule to perform q-value iteration, which allows us to bypass the need to keep track of values, transition functions, and reward functions. We use Deep Q-Learning to approximate our Q-value function with the use of a Neural Network. We choose the neuron from out network that has the highest value and take an action corresponding to this neuron.

STYLE TRANSFER

In this project we are going to be using Convolutional Neural Networks to implement Neural Style Transfer, a technique for creating a new image with the contenet of one input image and the style of another input. The idea behind style transfer is as follows: take three input images, one our style image, one our content image, and one output image which starts as random noise and we iteratively update it until it looks like the content of the content image in the style of the style image. To do this we run all three images through a pretrained VGG16 model trained to classify images. Then for a selected convolutional or pooling layer of the VGG16 model we compare the activations (values of the neurons) at that layer for the three different images. Specifically, we have what is called a feature reconstruction loss that compares the activations of the current output image and the content image, and what is called a style loss that compares the activations of the current output image and the desired style image. Then we use the gradient of these loss functions to update our current output image. Hopefully, that gives you an overview of what we will be doing, and you should gain a more in depth understanding as we go along.

The paper we will be implementing is found here: https://arxiv.org/pdf/1508.06576.pdf

Part 1: Build Model and Define Losses

First we want to initialize a VGG16 model we can use for style transfer.

In [28]:

```
from keras.applications import vgg16
from keras.layers import Input, Concatenate
from keras.models import Model, Sequential
from keras import backend as K
import tensorflow as tf
```

In [29]:

```
K.clear_session()
content_input = Input(batch_shape=(1, 224, 224, 3))
style_input = Input(batch_shape=(1, 224, 224, 3))
output_tensor = tf.get_variable("output_tensor", [1, 224, 224, 3])
output_input = Input(tensor=output_tensor)
## TODO: use a concatenate layer to concatenate the three inputs on the
first axis.
input_tensor = tf.concat([style_input,output_input,content_input], 1)
```

If you get an error for the cell below about SSL PROTOCOL VERSIONS or something similar you can try downloading this file https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5 and putting it in the folder `~/.keras/models` on your computer. Then the cell below should work after that

In [30]:

```
# We now create a pretrained VGG 16 model, which is really easy to do in Ke
ras
# include_top=False ensures we don't use the fully connected layers.
vgg_model = vgg16.VGG16(input_tensor=input_tensor, weights='imagenet', incl
ude_top=False)
# We can now look at the structure of this model
vgg_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_4 (InputLayer)         (None, None, None, 3)     0         
_________________________________________________________________
block1_conv1 (Conv2D)        (None, None, None, 64)    1792      
_________________________________________________________________
block1_conv2 (Conv2D)        (None, None, None, 64)    36928     
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, None, None, 64)    0         
_________________________________________________________________
block2_conv1 (Conv2D)        (None, None, None, 128)   73856     
_________________________________________________________________
block2_conv2 (Conv2D)        (None, None, None, 128)   147584    
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, None, None, 128)   0         
_________________________________________________________________
block3_conv1 (Conv2D)        (None, None, None, 256)   295168    
_________________________________________________________________
block3_conv2 (Conv2D)        (None, None, None, 256)   590080    
_________________________________________________________________
block3_conv3 (Conv2D)        (None, None, None, 256)   590080    
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, None, None, 256)   0         
_________________________________________________________________
block4_conv1 (Conv2D)        (None, None, None, 512)   1180160   
_________________________________________________________________
block4_conv2 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block4_conv3 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, None, None, 512)   0         
_________________________________________________________________
block5_conv1 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block5_conv2 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block5_conv3 (Conv2D)        (None, None, None, 512)   2359808   
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0         
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

In [31]:

```
print([layer.name for layer in vgg_model.layers])
```

```
['input_4', 'block1_conv1', 'block1_conv2', 'block1_pool', 'block2_conv1',
'block2_conv2', 'block2_pool', 'block3_conv1', 'block3_conv2',
'block3_conv3', 'block3_pool', 'block4_conv1', 'block4_conv2',
'block4_conv3', 'block4_pool', 'block5_conv1', 'block5_conv2',
'block5_conv3', 'block5_pool']
```

In [115]:

```
# now select one of the above listed layers to be the layer to use for
content information
# and select some number of layers (maybe 2 or 3 layers) from the above
layers to be the
# style information. If you choose layers closer to the input this will use

# more simplistic features, and choosing layers closer to the end will use
more complicated
# abstracted features.
content_layer = vgg_model.layers[4].name
style_layers = [
    vgg_model.layers[len(vgg_model.layers) - 3].name,
    vgg_model.layers[len(vgg_model.layers) - 2].name,
    vgg_model.layers[len(vgg_model.layers) - 1].name
]
# you can also play with the_content and style loss weights if you want
to. this will effect
# how stylized vs similar to the content image the output will look.
content_loss_weight = 100
style_loss_weight = 750
content_layer,style_layers
```

Out[115]:

```
('block2_conv1', ['block5_conv2', 'block5_conv3', 'block5_pool'])
```

In [116]:

```
layers_dict= dict([(layer.name, layer.output) for layer in vgg_model.layers
])
layers_dict
```

Out[116]:

```
{'block1_conv1': <tf.Tensor 'block1_conv1/Relu:0' shape=(1, 672, 224, 64) d
type=float32>,
 'block1_conv2': <tf.Tensor 'block1_conv2/Relu:0' shape=(1, 672, 224, 64) d
type=float32>,
 'block1_pool': <tf.Tensor 'block1_pool/MaxPool:0' shape=(1, 336, 112, 64)
dtype=float32>,
 'block2_conv1': <tf.Tensor 'block2_conv1/Relu:0' shape=(1, 336, 112, 128)
dtype=float32>,
 'block2_conv2': <tf.Tensor 'block2_conv2/Relu:0' shape=(1, 336, 112, 128)
dtype=float32>,
 'block2_pool': <tf.Tensor 'block2_pool/MaxPool:0' shape=(1, 168, 56, 128)
dtype=float32>,
 'block3_conv1': <tf.Tensor 'block3_conv1/Relu:0' shape=(1, 168, 56, 256) d
type=float32>,
 'block3_conv2': <tf.Tensor 'block3_conv2/Relu:0' shape=(1, 168, 56, 256) d
type=float32>,
 'block3_conv3': <tf.Tensor 'block3_conv3/Relu:0' shape=(1, 168, 56, 256) d
```

```
type=float32>,
 'block3_pool': <tf.Tensor 'block3_pool/MaxPool:0' shape=(1, 84, 28, 256) d
type=float32>,
 'block4_conv1': <tf.Tensor 'block4_conv1/Relu:0' shape=(1, 84, 28, 512) dt
ype=float32>,
 'block4_conv2': <tf.Tensor 'block4_conv2/Relu:0' shape=(1, 84, 28, 512) dt
ype=float32>,
 'block4_conv3': <tf.Tensor 'block4_conv3/Relu:0' shape=(1, 84, 28, 512) dt
ype=float32>,
 'block4_pool': <tf.Tensor 'block4_pool/MaxPool:0' shape=(1, 42, 14, 512) d
type=float32>,
 'block5_conv1': <tf.Tensor 'block5_conv1/Relu:0' shape=(1, 42, 14, 512) dt
ype=float32>,
 'block5_conv2': <tf.Tensor 'block5_conv2/Relu:0' shape=(1, 42, 14, 512) dt
ype=float32>,
 'block5_conv3': <tf.Tensor 'block5_conv3/Relu:0' shape=(1, 42, 14, 512) dt
ype=float32>,
 'block5_pool': <tf.Tensor 'block5_pool/MaxPool:0' shape=(1, 21, 7, 512) dt
ype=float32>,
 'input_4': <tf.Tensor 'concat:0' shape=(1, 672, 224, 3) dtype=float32>}
```

Loss Functions

We want to define our style transfer losses now. First, we are going to define a feature reconstruction loss based on our content features and our output features. Using tensorflow functions implement the following loss function:

$$\frac{1}{2}\sum_{i,j,k}(F_{ijk} - P_{ijk})^2$$

where F is the 3D tensor of content features and P is the 3D tensor of our output image features. HINT: tf.reduce_sum and tf.square will be helpful here.

In [117]:

```
def feature_reconstruction_loss(content_img_features, output_img_features):
    """Takes a tensor representing a layer of VGG features from the content
image
    and a tensor representing a layer of VGG features from the current outp
ut image and returns a loss value.
    """
    return 0.5 *
tf.reduce_sum(tf.square(content_img_features-output_img_features))
```

Now we wish to define our style loss function. First, we have to take our features and represent them as a Gram Matrix, for more information on Gram Matrices and this loss function you can read the paper if you like. Then we wish to implement the loss function:

$$\frac{1}{4H^2W^2C^2}\sum_{ij}(G_{ij} - A_{ij})^2$$

where G is the Gram matrix of the output image features and A is the Gram Matrix of the style image features. Note that we have written a Gram matrix function for you so you only need to call it.

In [118]:

```
def gram_matrix(x):
    # make channels first dimension
    x = tf.transpose(x, (2, 0, 1))
    # flatten everything but channels so x is now (C, H*W)
    x = tf.reshape(x, tf.stack([-1, tf.reduce_prod(tf.shape(x)[1:])]))
    return tf.matmul(x, tf.transpose(x))
```

In [119]:

```
def style_loss(style_img_features, output_img_features, img_shape):
    """Takes a tensor representing a layer of VGG features from the style i
mage and a tensor
    representing a layer of VGG features from the current output image and
returns
    the style loss for these features.
    """
    G = gram_matrix(output_img_features)
    A = gram_matrix(style_img_features)
    return tf.reduce_sum(tf.square(G-A))/(4* img_shape[0]**2 *img_shape[1]**
2 *img_shape[2]**2)
```

In [120]:

```
print(layers_dict[content_layer].shape)
print(tf.transpose(layers_dict[content_layer])[2, :, :, :])
```

```
(1, 336, 112, 128)
Tensor("strided_slice_30:0", shape=(112, 336, 1), dtype=float32)
```

In [121]:

```
content_features = layers_dict[content_layer]
content_img_features = tf.transpose(content_features)[0, :, :, :]
# output_content_features = content_features[:, :, :, 2]
output_content_features = tf.transpose(content_features)[2, :, :, :]
```

In [122]:

```
content_loss = feature_reconstruction_loss(content_img_features, output_con
tent_features)
```

In [123]:

```
total_style_loss = tf.zeros(1)
weight = 1.0 / len(style_layers)
for style_layer in style_layers:
    style_features = layers_dict[style_layer]
#     style_img_features = style_features[:, :, :, 1]
#     output_img_features = style_features[:, :, :, 2]
    style_img_features = tf.transpose(content_features)[1, :, :, :]
    output_img_features = tf.transpose(content_features)[2, :, :, :]
    total_style_loss += weight * style_loss(style_img_features,
output_img_features, (224, 224, 3))
```

Now we need to combine our two loss functions using the weightings we defined earlier. HINT: don't overthink this it should be a very simple operation.

In [124]:

```
total_loss = content_loss_weight*content_loss + style_loss_weight*total_sty
le_loss
```

In [125]:

```
optimize = tf.train.AdamOptimizer(learning_rate=10).minimize(total_loss, va
r_list=[output_tensor])
```

Part 2: Feeding in Images

We now want to load and preprocess our images. keras provides a `load_img` function that conviently loads our image and then cuts it down to our target size. Keras also provides a `vgg16.preprocess_input` that preprocesses images to be in the format vgg16 expects. Use these two functions to write the load_image function below.

In [126]:

```
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
import numpy as np
def load_image(img_path):
    img = load_img(target_size=(224,224,3),path=img_path)#YOUR CODE HERE
call load img and set the target size to be (224,224,3)
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    return img

def deprocess_image(x):
    x = x.reshape((224, 224, 3))
    x[:, :, 0] += 103.939
    x[:, :, 1] += 116.779
    x[:, :, 2] += 123.68
    # 'BGR'->'RGB'
    x = x[:, :, ::-1]
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

In [127]:

```
content_img_path = 'images/sproul.jpg'
style_img_path = 'images/monet_style.jpg'
```

In [128]:

```
content_img = load_image(content_img_path)
style_img = load_image(style_img_path)
```

In [129]:

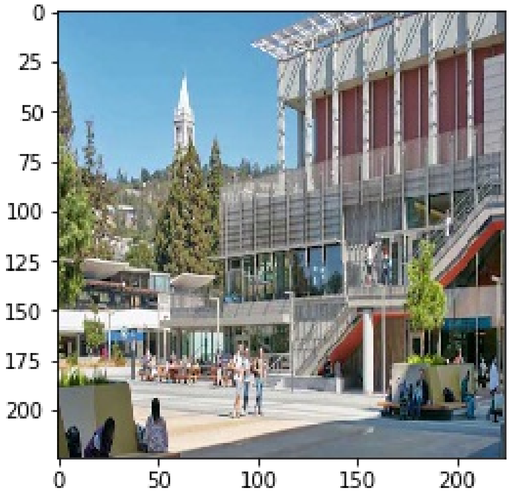
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [130]:

```
plt.imshow(deprocess_image(content_img))
```

Out[130]:

```
<matplotlib.image.AxesImage at 0x11c938080>
```

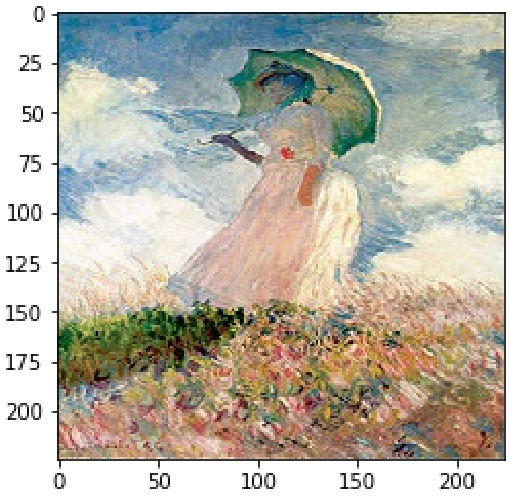


In [131]:

```
plt.imshow(deprocess_image(style_img))
```

Out[131]:

```
<matplotlib.image.AxesImage at 0x11c9cd3c8>
```



Part 3: Stylize Images

In [132]:

```
assign_var = tf.assign(output_tensor, content_img)
sess = K.get_session()
var = sess.run(assign_var)
```

Running the cell below will update the image 10 times. Since the initialization code is in the cell above, if you run the cell below and you're output isn't great you can run for another 10 iterations simply by rerunning the cell below.

In [133]:

```
n_iterations = 10
```

```
for i in range(n_iterations):
    print("Running iteration: {}".format(i))
    _, output_val, loss = sess.run([optimize, output_tensor, total_loss], f
eed_dict={content_input: content_img, style_input: style_img})
```

```
Running iteration: 0
Running iteration: 1
Running iteration: 2
Running iteration: 3
Running iteration: 4
Running iteration: 5
Running iteration: 6
Running iteration: 7
Running iteration: 8
Running iteration: 9
```

In [134]:

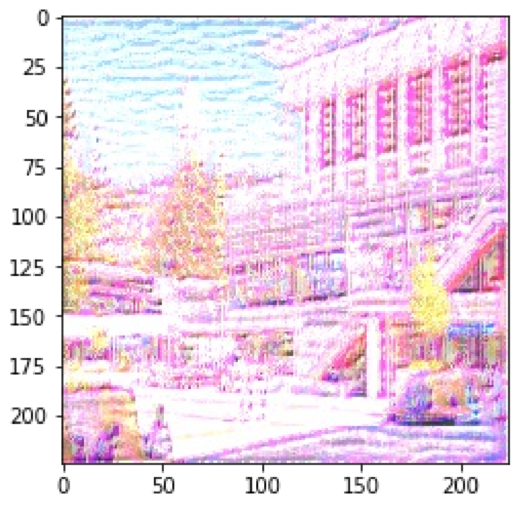
```
output_img = deprocess_image(output_val)
```

In [135]:

```
plt.imshow(output_img)
```

Out[135]:

```
<matplotlib.image.AxesImage at 0x117cb6160>
```



Part 4: Style Transfer Writeup

Now you need to writeup your project. First, write a short paragraph about your understanding of how style transfer works. Feel free to refer to the paper if it helps but your paragraph needs to be in your own words.

Then attach 3 sets of images to your writeup. For each set show the original content image, the original style image, and the style transfer result. One set should be the images we provided here, include the content and style layers you used as well as the content and style weights you used. Another set should be the images we provided here but with different content layers, style layers and different content and style weights, include your choices for the layers and weights in your writeup. Finally, include a set of images that is based on a new content image and a new style image that you choose yourself. There will be an award for the group with the coolest style transfer result.

Set 1

Content Layer: 'block2_conv2'

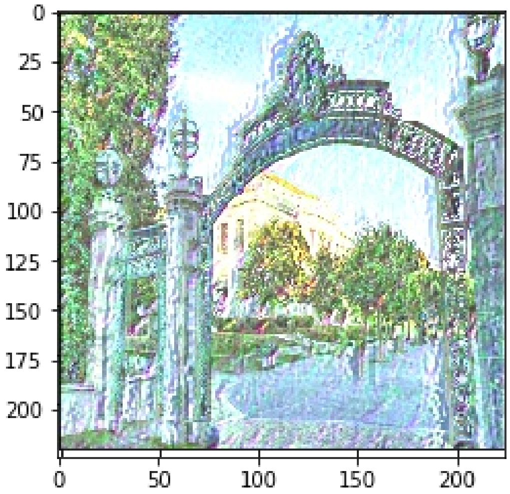
Three Styles: 'block2_pool', 'block3_conv1', 'block3_conv2'

In [137]:

```
plt.imshow(deprocess_image(load_image("images/satherGateStyle.png"))) #styl
ized
```

Out[137]:

```
<matplotlib.image.AxesImage at 0x11bb24940>
```

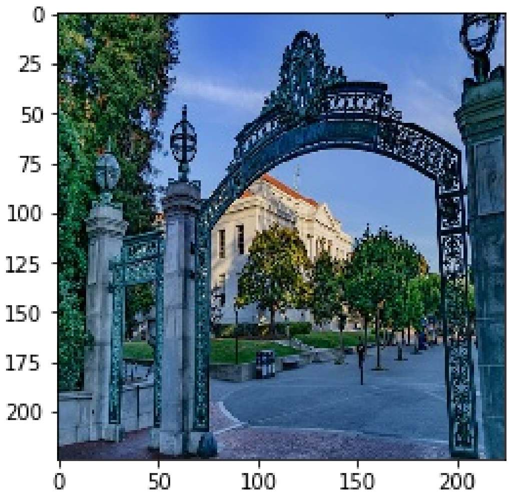


In [138]:

```
plt.imshow(deprocess_image(load_image("images/satherGate.jpg"))) #original
```

Out[138]:

```
<matplotlib.image.AxesImage at 0x11cb12d68>
```



Set 2

Content Layer: 'block1_conv2'

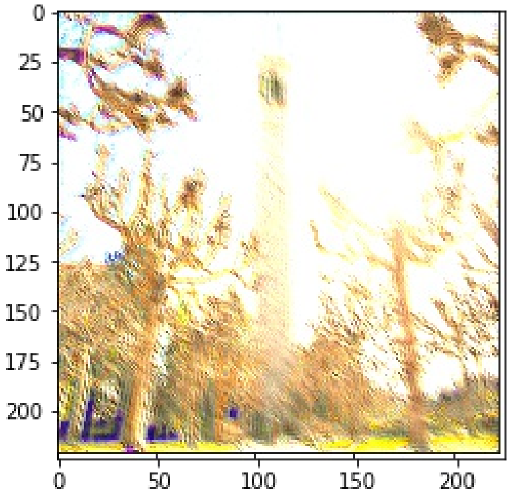
Three Layers: 'block1_pool', 'block2_conv1', 'block2_conv2'

In [112]:

```
plt.imshow(deprocess_image(load_image("images/campBerkStyle.png"))) #styliz
ed
```

Out[112]:

```
<matplotlib.image.AxesImage at 0x11b381908>
```

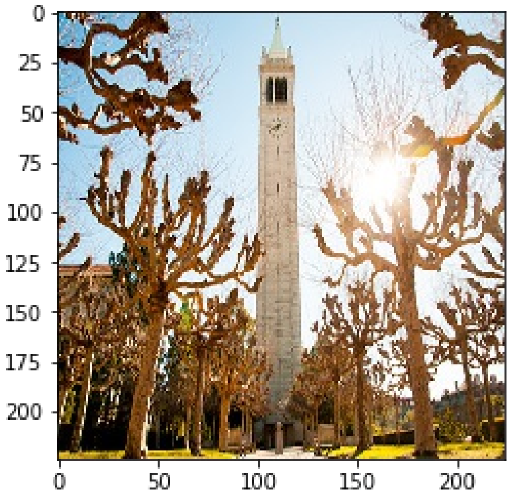


In [113]:

```
plt.imshow(deprocess_image(load_image("images/campBerk.jpg"))) #original
```

Out[113]:

```
<matplotlib.image.AxesImage at 0x11b3e1cc0>
```



Set 3

Content Layer: 'block2_conv1'

Three Layers: 'block5_conv2', 'block5_conv3', 'block5_pool'

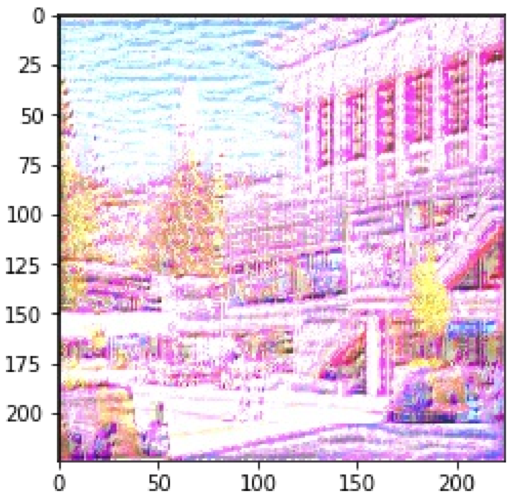
In [139]:

```
plt.imshow(deprocess_image(load_image("images/sproulStyle.png"))) #stylized
```

Out[139]:

```
<matplotlib.image.AxesImage at 0x11da53208>
```

<matplotlib.image.AxesImage at 0x11da95200>

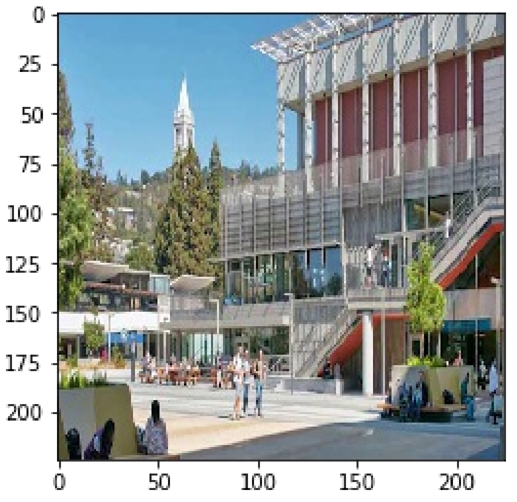


In [136]:

```
plt.imshow(deprocess_image(load_image("images/sproul.jpg"))) #original
```

Out[136]:

<matplotlib.image.AxesImage at 0x11ba49518>



Style transfer works by starting off with a random noise image, then after every iteration our original image changes a bit by bit to eventually adopt the style of the other image. Using convolution neural networks we use backpropogation to minimize our defined loss functions and then use the gradient of these loss function to compute our output image. We played around with different layer values to see the different results we can get.