Style

August 23, 2020

1 Style Transfer

The idea for this notebook is to show how we can use a trained CNN to separate the content of an image from the style and therefore create a style transfer system, that can take 1 image and apply the style of another one.

https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_

1.1 How we are going to do it?

Well, the idea here is that we can use a trained CNN as a feature extractor (VGG19) and use different layers of the net as the representation of an image and their content.

Now we could take some target image (either the original image, random noise or a blank canvas) and try to minimize 2 loss functions, one related to content (we will want to make the content representation of the original and target image as similar as possible) and analogously minimize the loss of the gram matrices that represent the style of the images.

Finally, something interesting to note is that we are going to minimize a weighted sum of the losses, so we can tweak the weights to signal that we are more interested in presenting the content of on transferring as much style as possible.

[0]:
```
# import resources
%matplotlib inline

from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt
import numpy as np

import torch
import torch.optim as optim
import requests
from torchvision import transforms, models
```

2 VGG19

[0]:
```python
vgg = models.vgg19(pretrained=True).features

for param in vgg.parameters():
    param.requires_grad_(False)
```

[37]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

vgg.to(device)
print(vgg)
```

```
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))
  (24): ReLU(inplace=True)
  (25): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (26): ReLU(inplace=True)
  (27): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (29): ReLU(inplace=True)
  (30): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
  (31): ReLU(inplace=True)
  (32): 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)
)
```

3 Load Images

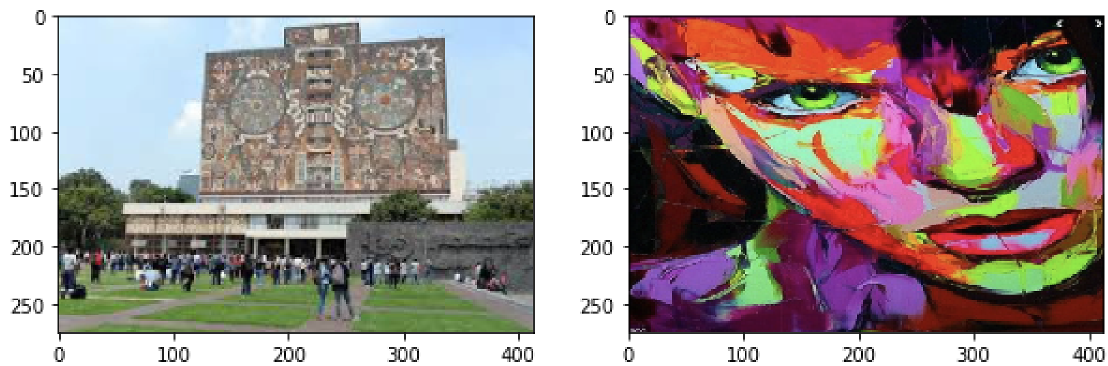
[0]:
```
# from udacity conv
def load_image(img_path, max_size=600, shape=None):
    image = Image.open(img_path).convert('RGB')
    if max(image.size) > max_size:
        size = max_size
    else:
        size = max(image.size)

    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))])

    # ignore transparent, alpha channel and add batch
    image = in_transform(image)[:3,:,:].unsqueeze(0)

    return image
```

[0]:
```
def im_convert(tensor):
    image = tensor.to("cpu").clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)

    #desnormal
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.
 ↪406))
    image = image.clip(0, 1)

    return image
```

[0]:
```
content = load_image('unam.jpeg').to(device)

style = load_image('cool.png', shape=content.shape[-2:]).to(device)
```

[41]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

ax1.imshow(im_convert(content))
ax2.imshow(im_convert(style))
```

[41]: <matplotlib.image.AxesImage at 0x7f7f81773198>



4 Content and Style Features

[0]:
```
def get_features(image, model, layers=None) -> dict:
    """ Run an image forward through a model and get the features for a set of␣
↪layers. """
    if layers is None:
        layers = {'0': 'conv1_1',
                  '5': 'conv2_1',
                  '10': 'conv3_1',
                  '19': 'conv4_1',
                  '21': 'conv4_2',  ## content representation
                  '28': 'conv5_1'}

    features = {}
    x = image

    for name, layer in model._modules.items():
        x = layer(x)
        if name in layers:
            features[layers[name]] = x
```

```
    return features
```

5 Gram Matrix

[0]:
```
def gram_matrix(tensor) -> torch.Tensor:
    """ Calculate the Gram Matrix of a given tensor
        Gram Matrix from :v https://en.wikipedia.org/wiki/Gramian_matrix
    """

    _, d, h, w = tensor.size()

    # reshape so we're multiplying the features for each channel
    tensor = tensor.view(d, h * w)

    gram = torch.mm(tensor, tensor.t())

    return gram
```

[0]:
```
content_features = get_features(content, vgg)
style_features = get_features(style, 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}

# create a target image and prep it for change
target = content.clone().requires_grad_(True).to(device)
```

6 Loss

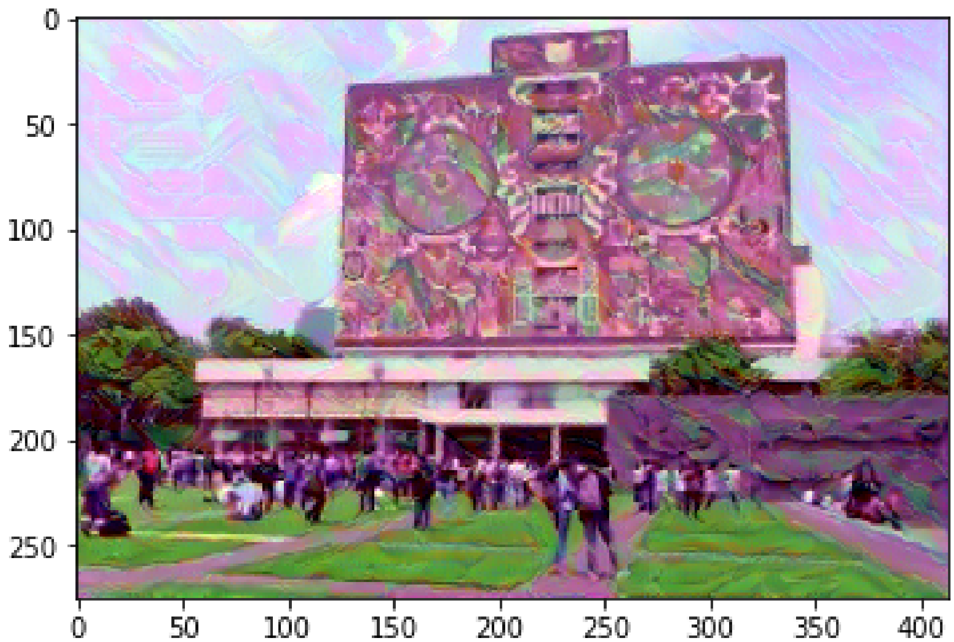
[0]:
```
style_weights = {'conv1_1': 1.,
                 'conv2_1': 0.85,
                 'conv3_1': 0.25,
                 'conv4_1': 0.2,
                 'conv5_1': 0.2}

content_weight = 1  # alpha
style_weight = 1e6  # beta
```

7 Train loop (from Udacity course)

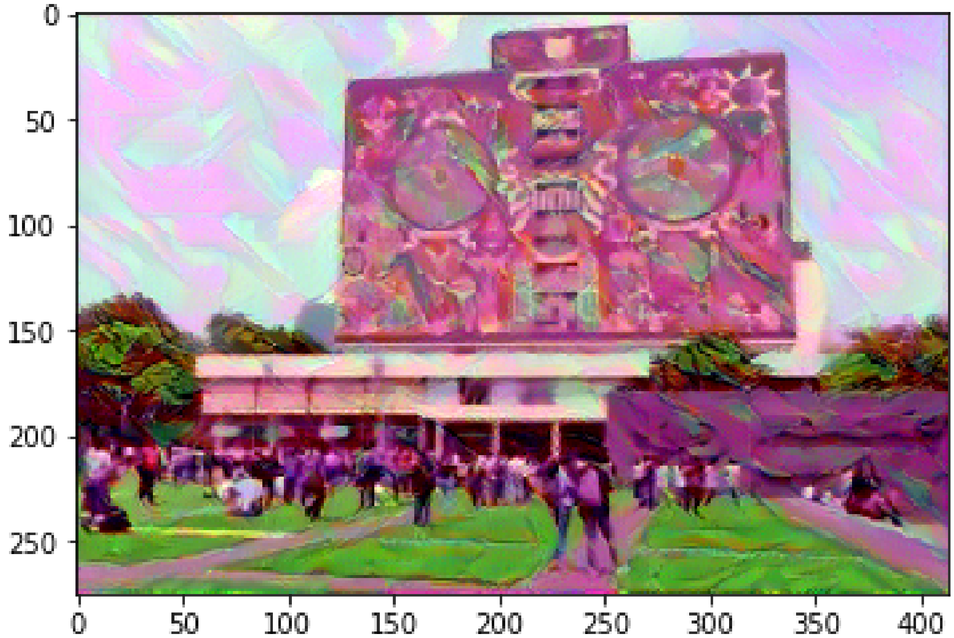
[46]:
```
show_every = 500

optimizer = optim.Adam([target], lr=0.002)
steps = 2000

for ii in range(1, steps + 1):
    target_features = get_features(target, vgg)
    content_loss = torch.mean((target_features['conv4_2'] -␣
↪content_features['conv4_2'])**2)

    # the style loss
    style_loss = 0

    if ii % 50 == 0: print(f"{round(ii / steps * 100, 2)}%")

    for layer in style_weights:
        # get the "target" style representation for the layer
        target_feature = target_features[layer]
        target_gram = gram_matrix(target_feature)
        _, d, h, w = target_feature.shape

        style_gram = style_grams[layer]

        # mean square
        layer_style_loss = style_weights[layer] * torch.mean((target_gram -␣
↪style_gram)**2)

        style_loss += layer_style_loss / (d * h * w)

    total_loss = (content_weight * content_loss) + (style_weight * style_loss)

    # update target
    optimizer.zero_grad()
    total_loss.backward()
    optimizer.step()

    # display intermediate images and print the loss
    if  ii % show_every == 0:
        print('Total loss: ', total_loss.item())
        plt.imshow(im_convert(target))
        plt.show()
```

```
2.5%
5.0%
7.5%
```

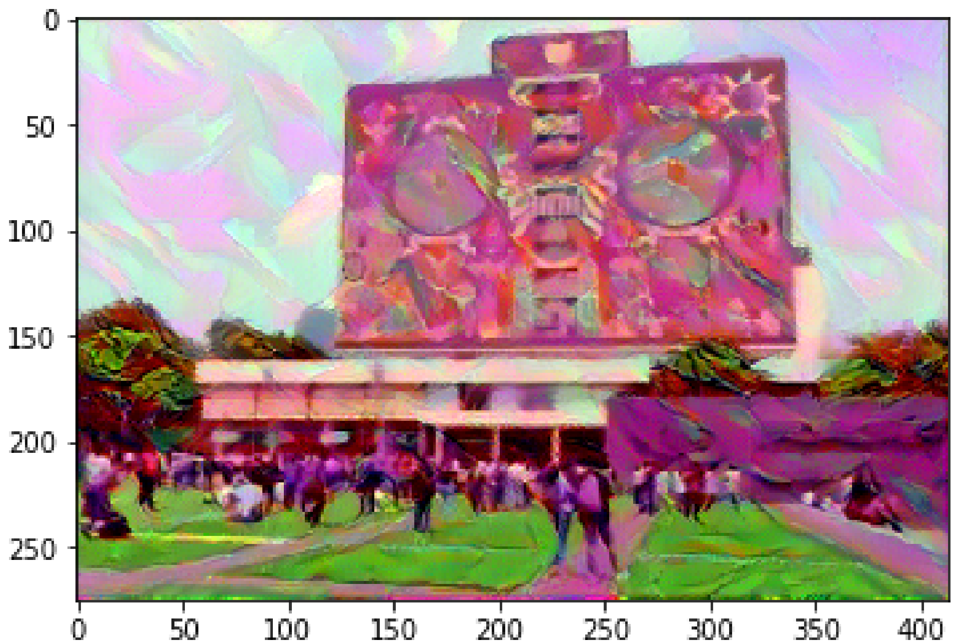
```
10.0%
12.5%
15.0%
17.5%
20.0%
22.5%
25.0%
Total loss:  41016948.0
```



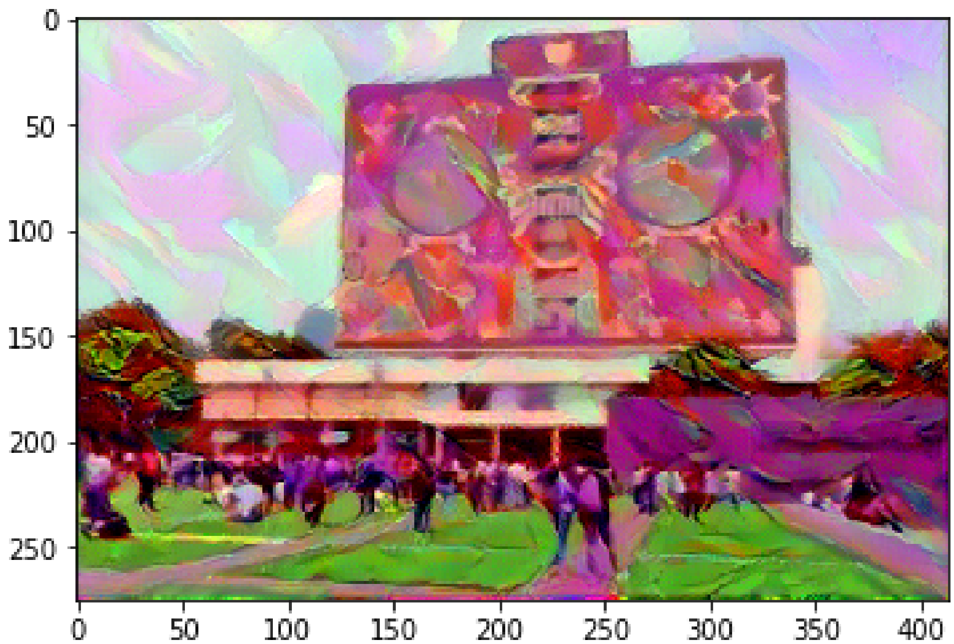

```
27.5%
30.0%
32.5%
35.0%
37.5%
40.0%
42.5%
45.0%
47.5%
50.0%
Total loss:  16994910.0
```




```
52.5%
55.0%
57.5%
60.0%
62.5%
65.0%
67.5%
70.0%
72.5%
75.0%
Total loss:  9602514.0
```

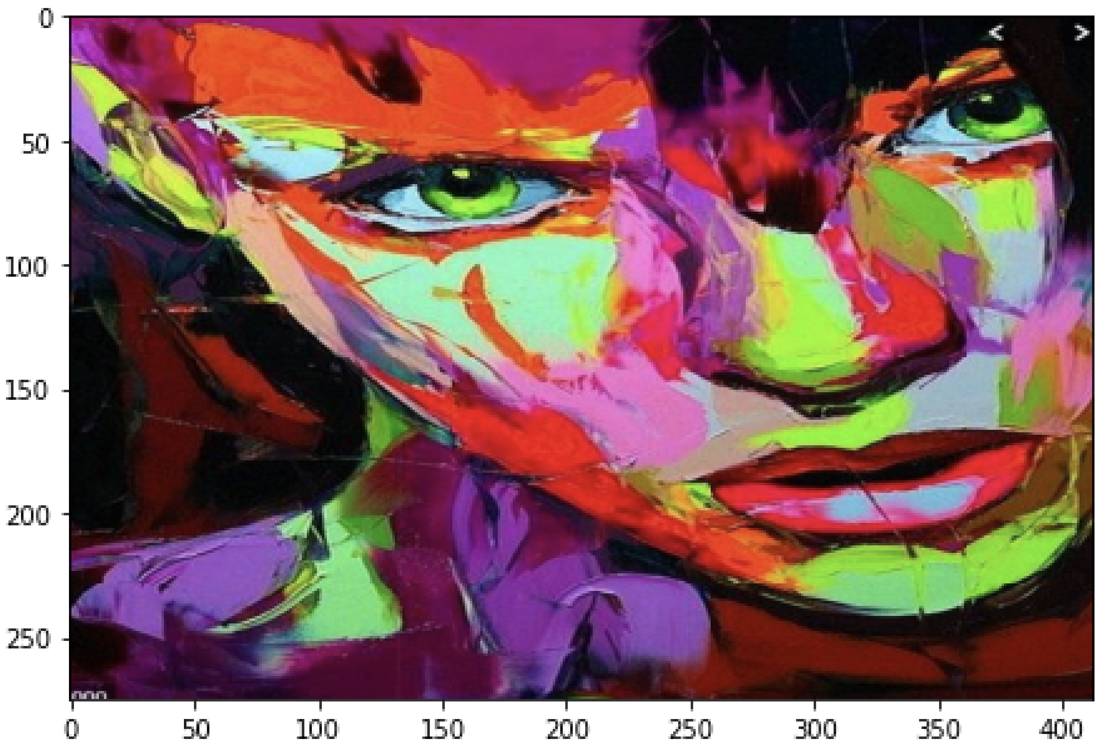


```
77.5%
80.0%
82.5%
85.0%
87.5%
90.0%
92.5%
95.0%
97.5%
100.0%
Total loss:  5961594.0
```



```
[47]: fig, ax1 = plt.subplots(1, 1, figsize=(10, 5))
      ax1.imshow(im_convert(style))
```

```
[47]: <matplotlib.image.AxesImage at 0x7f7f81722e48>
```



```
[48]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))
      ax1.imshow(im_convert(content))
      ax2.imshow(im_convert(target))
```

```
[48]: <matplotlib.image.AxesImage at 0x7f7f81640cc0>
```