Style Transfer

In this notebook we will implement the style transfer technique from "Image Style Transfer Using Convolutional Neural Networks" (Gatys et al., CVPR 2015) (http://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf).

The general idea is to take two images, and produce a new image that reflects the content of one but the artistic "style" of the other. We will do this by first formulating a loss function that matches the content and style of each respective image in the feature space of a deep network, and then performing gradient descent on the pixels of the image itself.

The deep network we use as a feature extractor is SqueezeNet (https://arxiv.org/abs/1602.07360), a small model that has been trained on ImageNet. You could use any network, but we chose SqueezeNet here for its small size and efficiency.

Here's an example of the images you'll be able to produce by the end of this notebook:

caption

We will then use this to create a video style transferer and then use it to automatic download videos and styles from the internet and upload them to my Youtube Channel (https://www.youtube.com/channel/UC2RKwvGB9hrVYQtmPNtCLkw/featured)

Setup

In [ ]:

```

```

In [1]:

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import torchvision
import torchvision.transforms as T
import PIL

import numpy as np


import cv2
import os
from collections import namedtuple
import matplotlib.pyplot as plt

from deeplearning.image_utils import SQUEEZENET_MEAN, SQUEEZENET_STD
%matplotlib inline
```

These are helper functions to deal with images, since we're dealing with real JPEGs, not CIFAR-10 data.

In [ ]:

In [2]:
```
def preprocess(img, size=512):
    transform = T.Compose([
        T.Scale((size,size)),
        T.ToTensor(),
        T.Normalize(mean=SQUEEZENET_MEAN.tolist(),
                    std=SQUEEZENET_STD.tolist()),
        T.Lambda(lambda x: x[None]),
    ])
    return transform(img)

def deprocess(img):
    transform = T.Compose([
        T.Lambda(lambda x: x[0]),
        T.Normalize(mean=[0, 0, 0], std=[1.0 / s for s in SQUEEZENET_
STD.tolist()]),
        T.Normalize(mean=[-m for m in SQUEEZENET_MEAN.tolist()], std=
[1, 1, 1]),
        T.Lambda(rescale),
        T.ToPILImage(),
    ])
    return transform(img)

def rescale(x):
    low, high = x.min(), x.max()
    x_rescaled = (x - low) / (high - low)
    return x_rescaled

def rel_error(x,y):
    return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.ab
s(y))))

def features_from_img(imgpath, imgsize):
    img = preprocess(PIL.Image.open(imgpath), size=imgsize)
    img_var = Variable(img.type(dtype))
    device = torch.device('cuda')
    return extract_features(img_var.to(device), cnn), img_var

# Older versions of scipy.misc.imresize yield different results
# from newer versions, so we check to make sure scipy is up to date.
def check_scipy():
    import scipy
    vnum = int(scipy.__version__.split('.')[1])

    assert vnum >= 16, "You must install SciPy >= 0.16.0 to complete
 this notebook."

#check_scipy()

answers = np.load('style-transfer-checks.npz')
```

We need to set the dtype to select either the CPU or the GPU

In [3]:
```
#dtype = torch.FloatTensor
#comment above or below depending if you're on a machine with a GPU s
et up for PyTorch!
dtype = torch.cuda.FloatTensor
```

In [4]:
```
# Load the pre-trained SqueezeNet model.
cnn = torchvision.models.squeezenet1_1(pretrained=True).features
cnn.type(dtype)



# We don't want to train the model any further, so we don't want PyTo
rch to waste computation
# computing gradients on parameters we're never going to update.
for param in cnn.parameters():
    param.requires_grad = False

#helper code which takes an image, a model (cnn), and returns a list
 of
# feature maps, one per layer.
def extract_features(x, cnn):
    """
    Use the CNN to extract features from the input image x.

    Inputs:
    - x: A PyTorch Variable of shape (N, C, H, W) holding a minibatch
of images that
      will be fed to the CNN.
    - cnn: A PyTorch model that we will use to extract features.

    Returns:
    - features: A list of feature for the input images x extracted us
ing the cnn model.
      features[i] is a PyTorch Variable of shape (N, C_i, H_i, W_i);
 recall that features
      from different layers of the network may have different numbers
of channels (C_i) and
      spatial dimensions (H_i, W_i).
    """
    features = []
    device = torch.device('cuda')
    cnn = cnn.to(device)
    prev_feat = x
    for i, module in enumerate(cnn._modules.values()):
        next_feat = module(prev_feat)
        features.append(next_feat)
        prev_feat = next_feat
    return features
```

Computing Loss

We're going to compute the three components of our loss function now. The loss function is a weighted sum of three terms: content loss + style loss + total variation loss.

Content loss

We can generate an image that reflects the content of one image and the style of another by incorporating both in our loss function. We want to penalize deviations from the content of the content image and deviations from the style of the style image. We can then use this hybrid loss function to perform gradient descent **not on the parameters** of the model, but instead **on the pixel values** of our original image.

Let's first write the content loss function. Content loss measures how much the feature map of the generated image differs from the feature map of the source image. We only care about the content representation of one layer of the network (say, layer ℓ), that has feature maps $A^\ell \in \mathbb{R}^{1 \times C_\ell \times H_\ell \times W_\ell}$. C_ℓ is the number of filters/channels in layer ℓ, H_ℓ and W_ℓ are the height and width. We will work with reshaped versions of these feature maps that combine all spatial positions into one dimension. Let $F^\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ be the feature map for the current image and $P^\ell \in \mathbb{R}^{N_\ell \times M_\ell}$ be the feature map for the content source image where $M_\ell = H_\ell \times W_\ell$ is the number of elements in each feature map. Each row of F^ℓ or P^ℓ represents the vectorized activations of a particular filter, convolved over all positions of the image. Finally, let w_c be the weight of the content loss term in the loss function.

Then the content loss is given by:

$L_c = w_c \times \sum_{i,j} (F_{ij}^{\ell} - P_{ij}^{\ell})^2$

In [5]:
```
def content_loss(content_weight, content_current, content_original):
    """
    Compute the content loss for style transfer.

    Inputs:
    - content_weight: Scalar giving the weighting for the content los
s.
    - content_current: features of the current image; this is a PyTor
ch Tensor of shape
      (1, C_l, H_l, W_l).
    - content_target: features of the content image, Tensor with shap
e (1, C_l, H_l, W_l).

    Returns:
    - scalar content loss
    """
#     sumy=0
    N ,C,H,W=content_current.shape
#     for k in range(C):
#         for i in range(H):
#             for j in range(W):
#                 sumy+=(content_current[0][k][i][j]-content_original
[0][k][i][j])**2

    sumy=torch.nn.functional.mse_loss(content_current,content_origina
l)
    return N*C*H*W*content_weight*sumy
```

In [ ]:
```

```

Style loss

Now we can tackle the style loss. For a given layer ℓ, the style loss is defined as follows:

First, compute the Gram matrix G which represents the correlations between the responses of each filter, where F is as above. The Gram matrix is an approximation to the covariance matrix -- we want the activation statistics of our generated image to match the activation statistics of our style image, and matching the (approximate) covariance is one way to do that. There are a variety of ways you could do this, but the Gram matrix is nice because it's easy to compute and in practice shows good results.

Given a feature map F^ℓ of shape $(1, C_\ell, M_\ell)$, the Gram matrix has shape $(1, C_\ell, C_\ell)$ and its elements are given by:

$$G_{ij}^\ell = \sum_k F_{ik}^\ell F_{jk}^\ell$$

Assuming G^ℓ is the Gram matrix from the feature map of the current image, A^ℓ is the Gram Matrix from the feature map of the source style image, and w_ℓ a scalar weight term, then the style loss for the layer ℓ is simply the weighted Euclidean distance between the two Gram matrices:

$$L_s^\ell = w_\ell \sum_{i,j} \left(G_{ij}^\ell - A_{ij}^\ell\right)^2$$

In practice we usually compute the style loss at a set of layers $\mathcal{L}$ rather than just a single layer ℓ; then the total style loss is the sum of style losses at each layer:

$$L_s = \sum_{\ell \in \mathcal{L}} L_s^\ell$$

We have implemented the Gram matrix computation below:

In [6]:
```
def gram_matrix(features, normalize=True):
    """
    Compute the Gram matrix from features.

    Inputs:
    - features: PyTorch Variable of shape (N, C, H, W) giving feature
s for
      a batch of N images.
    - normalize: optional, whether to normalize the Gram matrix
        If True, divide the Gram matrix by the number of neurons (H *
W * C)

    Returns:
    - gram: PyTorch Variable of shape (N, C, C) giving the
      (optionally normalized) Gram matrices for the N input images.
    """
    a, b, c, d = features.size()
    features = features.view(a * b, c * d)  # resise F_XL into \hat F
_XL

    G = torch.mm(features, features.t())  # compute the gram product

    # we 'normalize' the values of the gram matrix
    # by dividing by the number of element in each feature maps.
    if normalize:
        return G.div(a * b * c * d)
    return G

#     trans=features.transpose(1,2)
#     print(trans.shape)
#     gram=trans.dot(features)
```

In [ ]:

In [ ]:

Next, we implement the style loss:

In [7]:
```
# Now put it together in the style_loss function...
def style_loss(feats, style_layers, style_targets, style_weights):
    """
    Computes the style loss at a set of layers.

    Inputs:
    - feats: list of the features at every layer of the current imag
e, as produced by
      the extract_features function.
    - style_layers: List of layer indices into feats giving the layer
s to include in the
      style loss.
    - style_targets: List of the same length as style_layers, where s
tyle_targets[i] is
      a PyTorch Variable giving the Gram matrix the source style imag
e computed at
      layer style_layers[i].
    - style_weights: List of the same length as style_layers, where s
tyle_weights[i]
      is a scalar giving the weight for the style loss at layer style
_layers[i].

    Returns:
    - style_loss: A PyTorch Variable holding a scalar giving the styl
e loss.
    """
#     for l in range(len(style_layers)):
#         G=gram_matrix(feats[style_layers[l]])

#         C,C2=G.size()
#         summy=0

#         for i in range(C):
#             for j in range(C2):
#                 #print((i,j))
#                 summy+=(G[i][j]-style_targets[l][i][j])**2
#         losses.append(style_weights[l]*summy)
#     return sum(losses)
    losses=[]
    for l in range(len(style_layers)):
        G=gram_matrix(feats[style_layers[l]])
        C,_=G.shape
        #print(G,style_targets[l],)
        #print(G.shape,style_targets[l].shape)
        losses.append(style_weights[l]*(C**2*torch.nn.functional.mse_
loss(G, style_targets[l])))
    #print(losses)
    return sum(losses)
```

In [ ]:

In [ ]:

Total-variation regularization

It turns out that it's helpful to also encourage smoothness in the image. We can do this by adding another term to our loss that penalizes wiggles or "total variation" in the pixel values.

You can compute the "total variation" as the sum of the squares of differences in the pixel values for all pairs of pixels that are next to each other (horizontally or vertically). Here we sum the total-variation regualarization for each of the 3 input channels (RGB), and weight the total summed loss by the total variation weight, w_t:

$$L_{tv} = w_t \times \sum_{c=1}^{3}\sum_{i=1}^{H-1}\sum_{j=1}^{W-1} \left((x_{i,j+1,c} - x_{i,j,c})^2 + (x_{i+1,j,c} - x_{i,j,c})^2 \right)$$

In the next cell, we fill in the definition for the TV loss term.

In [8]:
```
def tv_loss(img, tv_weight):
    """
    Compute total variation loss.

    Inputs:
    - img: PyTorch Variable of shape (1, 3, H, W) holding an input im
age.
    - tv_weight: Scalar giving the weight w_t to use for the TV loss.

    Returns:
    - loss: PyTorch Variable holding a scalar giving the total variat
ion loss
      for img weighted by tv_weight.
    """
    # Your implementation should be vectorized and not require any lo
ops!
    diff_i = torch.sum(torch.square((img[:, :, :, 1:] - img[:, :, :,
:-1])))
    diff_j = torch.sum(torch.square(torch.abs(img[:, :, 1:, :] - img
[:, :, :-1, :])))
    tv_loss = tv_weight*(diff_i + diff_j)
    return tv_loss
```

In [ ]:

In [ ]:

Now we're ready to string it all together:

In [9]:
```
def style_transfer(content_image, style_image, image_size, style_size
, content_layer, content_weight,
                   style_layers, style_weights, tv_weight, init_rando
m = False, quiet=False):
    """
    Run style transfer!

    Inputs:
    - content_image: filename of content image
    - style_image: filename of style image
    - image_size: size of smallest image dimension (used for content
 loss and generated image)
    - style_size: size of smallest style image dimension
    - content_layer: layer to use for content loss
    - content_weight: weighting on content loss
    - style_layers: list of layers to use for style loss
    - style_weights: list of weights to use for each layer in style_l
ayers
    - tv_weight: weight of total variation regularization term
    - init_random: initialize the starting image to uniform random no
ise
    """

    device = torch.device('cuda')
    # Extract features for the content image
    content_img = preprocess(PIL.Image.open(content_image), size=imag
e_size)
    content_img_var = Variable(content_img.type(dtype)).to(device)
    feats = extract_features(content_img_var, cnn)
    content_target = feats[content_layer].clone().to(device)

    # Extract features for the style image
    style_img = preprocess(PIL.Image.open(style_image), size=style_si
ze).to(device)
    style_img_var = Variable(style_img.type(dtype)).to(device)
    feats = extract_features(style_img_var, cnn)
    style_targets = []
    for idx in style_layers:
        style_targets.append(gram_matrix(feats[idx].clone()))

    # Initialize output image to content image or nois
    if init_random:
        img = torch.Tensor(content_img.size()).uniform_(0, 1)
    else:
        img = content_img.clone().type(dtype).to(device)
    # We do want the gradient computed on our image!
    img_var = Variable(img, requires_grad=True).to(device)


    # Set up optimization hyperparameters
    initial_lr = 3.0
    decayed_lr = 0.1
    decay_lr_at = 180

    # Note that we are optimizing the pixel values of the image by pa
ssing
```

```
    # in the img_var Torch variable, whose requires_grad flag is set
 to True
    optimizer = torch.optim.Adam([img_var], lr=initial_lr)
    if not quiet:
        f, axarr = plt.subplots(1,2)
        axarr[0].axis('off')
        axarr[1].axis('off')
        axarr[0].set_title('Content Source Img.')
        axarr[1].set_title('Style Source Img.')
        axarr[0].imshow(deprocess(content_img.cpu()))
        axarr[1].imshow(deprocess(style_img.cpu()))
        plt.show()
        plt.figure()

    for t in range(200):
        if t < 190:
            img.clamp_(-1.5, 1.5)
        optimizer.zero_grad()

        feats = extract_features(img_var, cnn)
        #print(t)
        # Compute loss
        c_loss = content_loss(content_weight, feats[content_layer], c
ontent_target)
        #print(t)
        s_loss = style_loss(feats, style_layers, style_targets, style
_weights)
        #print(t)
        t_loss = tv_loss(img_var, tv_weight)
        #print(t)
        loss = c_loss + s_loss + t_loss

        loss.backward()
        #print(t)
        # Perform gradient descents on our image values
        if t == decay_lr_at:
            optimizer = torch.optim.Adam([img_var], lr=decayed_lr)
        optimizer.step()

        if t % 100 == 0:
            if not quiet:
                print('Iteration {}'.format(t))
                plt.axis('off')
                plt.imshow(deprocess(img.cpu()))
                plt.show()
    return img
```

Generate some pretty pictures!

- The `content_image` is the filename of content image.
- The `style_image` is the filename of style image.
- The `image_size` is the size of smallest image dimension of the content image (used for content loss and generated image).
- The `style_size` is the size of smallest style image dimension.
- The `content_layer` specifies which layer to use for content loss.
- The `content_weight` gives weighting on content loss in the overall loss function. Increasing the value of this parameter will make the final image look more realistic (closer to the original content).
- `style_layers` specifies a list of which layers to use for style loss.
- `style_weights` specifies a list of weights to use for each layer in style_layers (each of which will contribute a term to the overall style loss). We generally use higher weights for the earlier style layers because they describe more local/smaller scale features, which are more important to texture than features over larger receptive fields. In general, increasing these weights will make the resulting image look less like the original content and more distorted towards the appearance of the style image.
- `tv_weight` specifies the weighting of total variation regularization in the overall loss function. Increasing this value makes the resulting image look smoother and less jagged, at the cost of lower fidelity to style and content.

In [ ]:
```
def styletrans(im1,im2, show=False):
    content_size = 224
    style_size = 224
    params1 = {
        'content_image' : im1,
        'style_image' : im2,
        'image_size' : content_size,
        'style_size' : style_size,
        'content_layer' : 3,
        'content_weight' : 1e-1,
        'style_layers' : [1, 4, 6, 7],
        'style_weights' : [300000, 1000, 15, 3],
        'tv_weight' : 5e-2,
        'quiet': True
    }


    img = style_transfer(**params1)
    arr = deprocess(img.cpu())
    if show:
        plt.axis('off')
        plt.imshow(arr)
        plt.show()
    return cv2.cvtColor(np.asarray(arr), cv2.COLOR_RGB2BGR)

#Movie Maker
def styletrans_a_mp4(movie, style, outputmovie,s=-1):
    cam = cv2.VideoCapture(movie)
    frame_ct= int(cam.get(cv2.CAP_PROP_FRAME_COUNT))
    # initialize video writer
    fourcc = cv2.VideoWriter_fourcc('M','P','E','G')
    fps = int(cam.get(cv2.CAP_PROP_FPS))
    print("FPS:",fps)
    video_filename = outputmovie
    width = 224
    height = 224
    out = cv2.VideoWriter(video_filename, fourcc, fps, (width, height
))
    done_ct = 0
    print(str(done_ct)+"/"+str(int((frame_ct if s==-1 else s*fps))))
    while(True):
        # reading from frame
        ret,frame = cam.read()
        if ret:
            if s!=-1 and done_ct > int(s*fps):
                print("Stopping after " + str(s) + " seconds of foota
ge")
                out.release()
                return
            done_ct+=1
            name = 'xyzfdsf.jpg'
            cv2.imwrite(name, frame)
            if done_ct % int(fps/8) == 0:
                print(str(done_ct)+"/"+str(int((frame_ct if s==-1 els
e s*fps))))
            img = styletrans(name, style, show = (done_ct ==1))
```

```
            out.write(img)
            os.remove(name)
        else:
            out.release()
            return
```

In [11]:

```
#test on the basics--------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
```

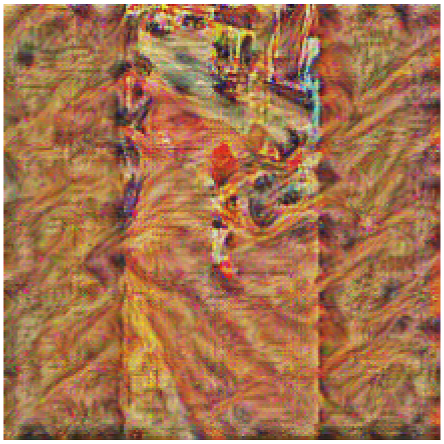
```
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
---------------------------------------------------------------------
------------------------------------------------
arr = styletrans('styles/tubingen.jpg','styles/starry_night.jpg', sho
w=True)
print(arr.shape)
```

```
/home/rishirules/anaconda3/lib/python3.7/site-packages/torchvision/tr
ansforms/transforms.py:285: UserWarning: The use of the transforms.Sc
ale transform is deprecated, please use transforms.Resize instead.
  warnings.warn("The use of the transforms.Scale transform is depreca
ted, " +
```

```
(224, 224, 3)
```

In [ ]:
```
#test movie style transfer
styletrans_a_mp4('styles/dog.mp4','styles/fire_demon_van_goh.jpg','st
yles/fire_dog.mp4')
```

```
FPS: 59
0/3537
```



```
7/3537
14/3537
21/3537
28/3537
35/3537
42/3537
49/3537
56/3537
63/3537
70/3537
77/3537
84/3537
91/3537
98/3537
105/3537
112/3537
119/3537
126/3537
133/3537
140/3537
147/3537
154/3537
161/3537
168/3537
175/3537
182/3537
189/3537
196/3537
203/3537
210/3537
217/3537
224/3537
231/3537
238/3537
245/3537
252/3537
259/3537
266/3537
273/3537
280/3537
287/3537
294/3537
301/3537
308/3537
315/3537
322/3537
329/3537
336/3537
343/3537
350/3537
357/3537
364/3537
371/3537
378/3537
385/3537
392/3537
399/3537
```

1.00

0:00

Other Stuff:

In [ ]:
```
# # Composition VII + Tubingen
# params1 = {
#     'content_image' : 'styles/tubingen.jpg',
#     'style_image' : 'styles/composition_vii.jpg',
#     'image_size' : 192,
#     'style_size' : 512,
#     'content_layer' : 3,
#     'content_weight' : 5e-2,
#     'style_layers' : (1, 4, 6, 7),
#     'style_weights' : (20000, 500, 12, 1),
#     'tv_weight' : 5e-2
# }

# style_transfer(**params1)
```

In [ ]:
```
!which ffmpeg
```

In [ ]:
```
# # Scream + Tubingen
# params2 = {
#     'content_image':'styles/tubingen.jpg',
#     'style_image':'styles/the_scream.jpg',
#     'image_size':192,
#     'style_size':224,
#     'content_layer':3,
#     'content_weight':3e-2,
#     'style_layers':[1, 4, 6, 7],
#     'style_weights':[200000, 800, 12, 1],
#     'tv_weight':2e-2
# }

# style_transfer(**params2)
```

In [ ]:
```
# # Starry Night + Tubingen
# params3 = {
#     'content_image' : 'styles/tubingen.jpg',
#     'style_image' : 'styles/starry_night.jpg',
#     'image_size' : 192,
#     'style_size' : 192,
#     'content_layer' : 3,
#     'content_weight' : 6e-2,
#     'style_layers' : [1, 4, 6, 7],
#     'style_weights' : [300000, 1000, 15, 3],
#     'tv_weight' : 2e-2
# }

# style_transfer(**params3)
```

Feature Inversion

The code written can do another cool thing. In an attempt to understand the types of features that convolutional networks learn to recognize, a recent paper [1] attempts to reconstruct an image from its feature representation. We can easily implement this idea using image gradients from the pretrained network, which is exactly what we did above (but with two different feature representations).

Now, if you set the style weights to all be 0 and initialize the starting image to random noise instead of the content source image, you'll reconstruct an image from the feature representation of the content source image. You're starting with total noise, but you should end up with something that looks quite a bit like your original image.

(Similarly, you could do "texture synthesis" from scratch if you set the content weight to 0 and initialize the starting image to random noise)

[1] Aravindh Mahendran, Andrea Vedaldi, "Understanding Deep Image Representations by Inverting them", CVPR 2015

Example below

In [ ]:
```
# # Feature Inversion -- Starry Night + Tubingen
# params_inv = {
#     'content_image' : 'styles/tubingen.jpg',
#     'style_image' : 'styles/starry_night.jpg',
#     'image_size' : 192,
#     'style_size' : 192,
#     'content_layer' : 3,
#     'content_weight' : 6e-2,
#     'style_layers' : [1, 4, 6, 7],
#     'style_weights' : [0, 0, 0, 0], # we discard any contributions
 from style to the loss
#     'tv_weight' : 2e-2,
#     'init_random': True # we want to initialize our image to be ran
dom
# }

# style_transfer(**params_inv)
```

In [ ]:

In [ ]: