Lecture 6 Summary

김형근 (@engiecat)

2018.12.16

PyTorch Challenge with Facebook #sg-korea study group

Neural Style Transfer

- Gatys et al. 2015에서 제안됨
  - L. A. Gatys, A. S. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 2414-2423.
- **최초**로 그림의 스타일을 다른 그림으로 전달한 연구

우리나라에서도 보도! →

NST in Nutshell

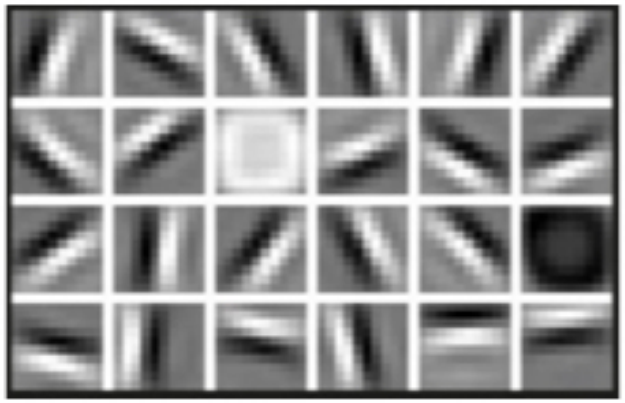
content image

style image

target image

Inspirations – Defining Content

- CNN이 사물을 인식하는 방식에서 착안
  - “사물 인식용 CNN은 층이 깊어질수록 더 추상적인 feature들로 변환됨“



First Layer Representation

Second Layer Representation

Third Layer Representation

Defining Content

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^2 .$$

- How?
  - 해당 레이어에서의 feature map이 주어진 것과 비슷하게 나오도록
  - White noise이미지부터 Gradient Descent 시행 – **잘됨!**

'깊은' 층에서의 feature map으로 재현한 이미지도 **추상적 내용(Content)은 보존**!

Defining Style (1)

- 이전에 진행하던 Texture Generation 연구
  - Parametric Texture Generation (Portilla & Simoncelli, 2000)
    - http://people.csail.mit.edu/siracusa/tmp/9.912-simoncelli-texture.pdf
    - 다양한 스케일, 다양한 방향에서의 feature을 추출 (**아주 잘 고른 filter을** 써서)
    - 이러한 feature들에 대한 **다양한 통계를** 이용해서 Iterative하게 텍스처를 생성

- Gatys 팀 (DeepTexture, 2015)
  - **Object Detection 용 CNN을 써서 feature map**을 만들자!
    - Filter을 고를 필요 없이 그냥 뽑아 쓰기
  - 여러 통계 쓸 필요없이 **각 레이어의 feature response간의 correlation** 확인
    - Gram matrix!

Defining Style (2) - DeepTexture

- Gram Matrix
  - Layer L에서의 Feature map i와 j사이의 Correlation (k는 map 내의 위치)
    - 같은 층에서의 서로 다른 filter 들 간의 correlation

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

- 마찬가지로 iterative하게 생성

- **매우 성공적!**

Defining Style (3)

아랫 레이어에서 뽑으면 단순한 텍스처 정보

윗 레이어에서 뽑으면 더 큰 스케일의 구조가 나옴

기존 방식에 비해 재현성도 상당히 좋음

Defining Style (3)

아랫 레이어에서 뽑으면 단순한 텍스처 정보

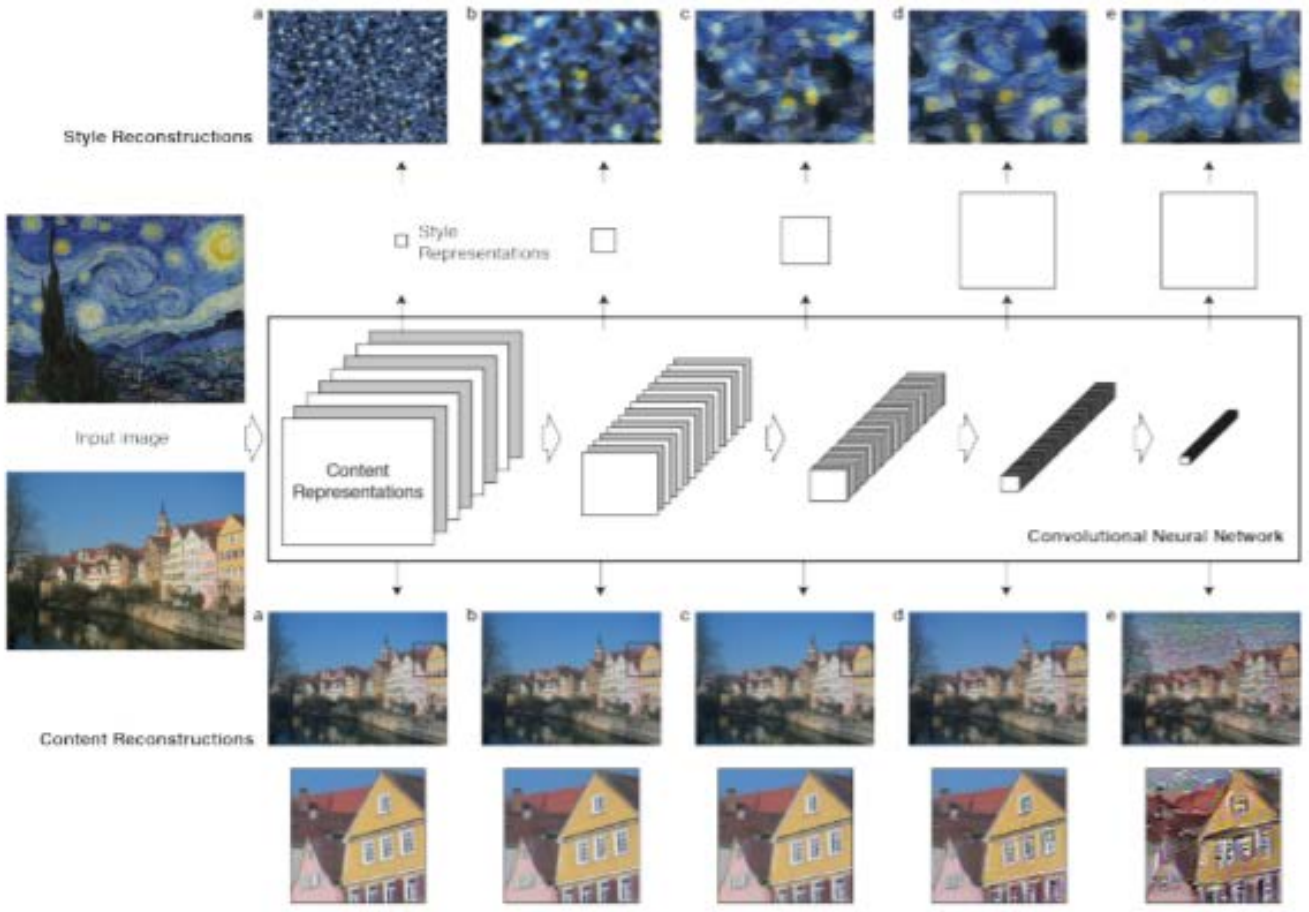
윗 레이어에서 뽑으면 더 큰 스케일의 구조가 나옴

그렇다면 여러 레이어에서 뽑은 Correlation이 Style 아닐까?
(Layer 별로 다른 weight를 줘 보자)

$$\mathcal{L}_{style}(a, x) = \sum_{l=0}^{L} w_l E_l \qquad E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2.$$

기존 방식에 비해 재현성도 상당히 좋음

Merging Style and Content



다양한 레이어에서 스타일 추출
(Gram Matrix)

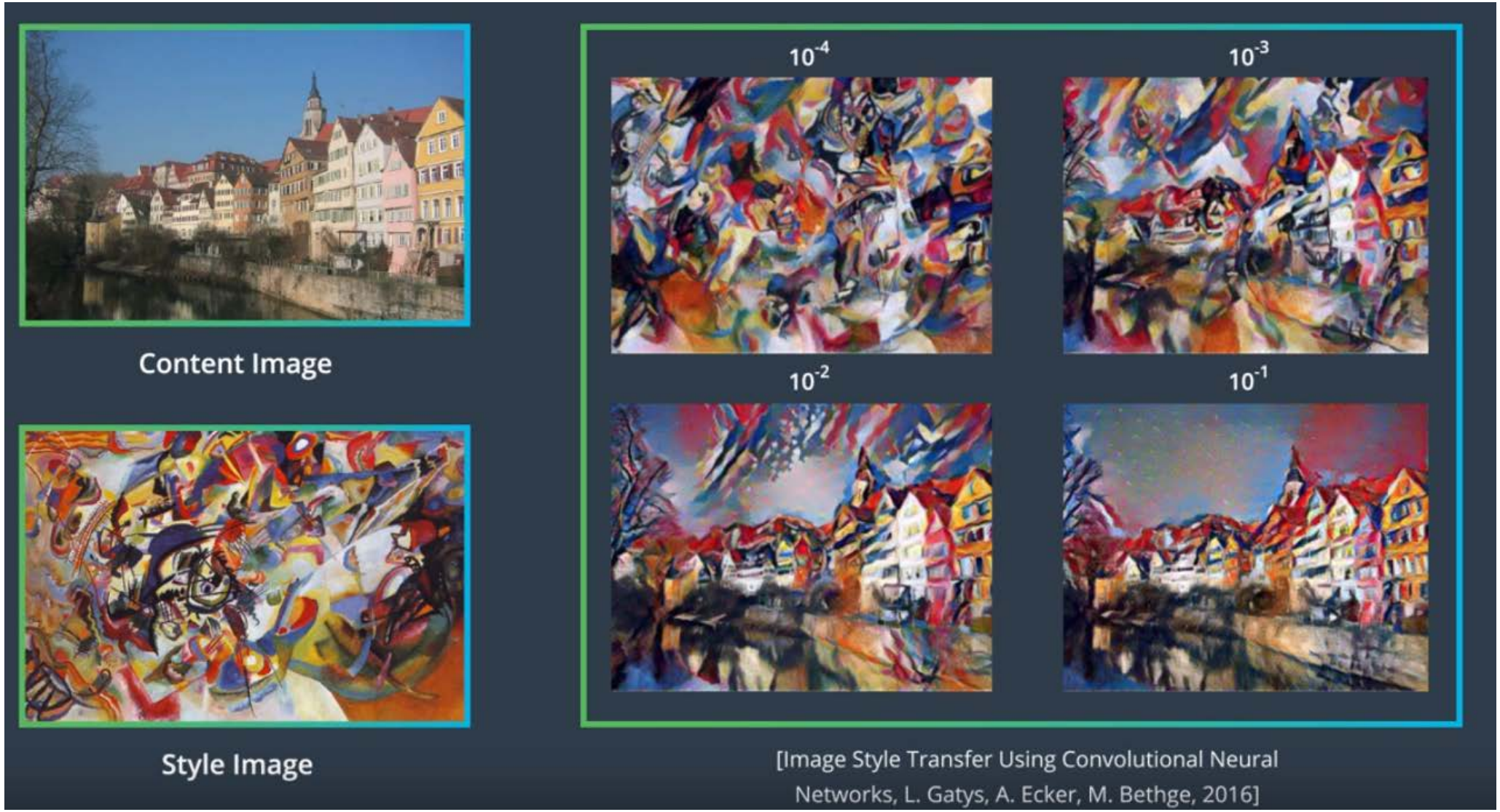
$$\alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

Content Weight & Style Weight

Often Much Larger

다양한 레이어에서 내용 추출
(MSE Loss사용)

Result (by alpha/beta)



Then, How to Code? (1)

- 이미지 불러오기/내보내기
  - ImageNet으로 훈련한 VGG-19를 사용
    - 훈련 전에 dataset들을 normalize 시킴

```
in_transform = transforms.Compose([
                        transforms.Resize(size),
                        transforms.ToTensor(),
                        transforms.Normalize((0.485, 0.456, 0.406),
                                             (0.229, 0.224, 0.225))])
```

  - 되돌리려면?

```
def im_convert(tensor):
    """ Display a tensor as an image. """

    image = tensor.to("cpu").clone().detach()
    image = image.numpy().squeeze()
    image = image.transpose(1,2,0)
    image = image * np.array((0.229, 0.224, 0.225)) + np.array((0.485, 0.456, 0.406))
    image = image.clip(0, 1)
```

Then, How to Code? (2)

- Gram Matrix 구하기
  - 위 식에서 k는 각 feature map에서의 위치
  - 근데 위치가 2차원이니 비효율적임 -> 1차원으로 unroll하자!

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

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

  - Gram Matrix는 그럼 단순하게 구해짐.

```
# calculate the gram matrix
gram = torch.mm(tensor, tensor.t())
```

Then, How to Code? (3)

- Iterative하게 Image 만들기 (White Noise에서 Gradient Desc.)

```
# get the features from your target image
target_features = get_features(target, vgg)

# the content loss
content_loss = torch.mean((target_features['conv4_2'] - content_features['conv4_2'])**2)

# the style loss
# initialize the style loss to 0
style_loss = 0
# then add to it for each layer's gram matrix loss
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
    # get the "style" style representation
    style_gram = style_grams[layer]
    # the style loss for one layer, weighted appropriately
    layer_style_loss = style_weights[layer] * torch.mean((target_gram - style_gram)**2)
    # add to the style loss
    style_loss += layer_style_loss / (d * h * w)

# calculate the *total* loss
total_loss = content_weight * content_loss + style_weight * style_loss

# update your target image
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()
```

Thank you!!