Deep Learning Hands on - 2

Mishima.syk #13
@iwatobipen

今日のお題

Neural Style Transfer!

Ref

A Neural Algorithm of Artistic Style
https://arxiv.org/pdf/1508.06576.pdf
https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf

What’s neural style transfer?

これ

↓

Content / target　　Style / reference　　Results

https://medium.com/artists-and-machine-intelligence/neural-artistic-style-transfer-a-comprehensive-look-f54d8649c199

Definitions

- **Style: 画像の質感、色合い、パターン＝＞画風**
- **Content: 画像の全体的な構成**

Concept

- DeepLearningでStyle（特徴）を学習
- 目的の画像が生成できるようなLoss関数を定義
  - Styleはリファレンスとの距離、
  - Contentはターゲットとの距離

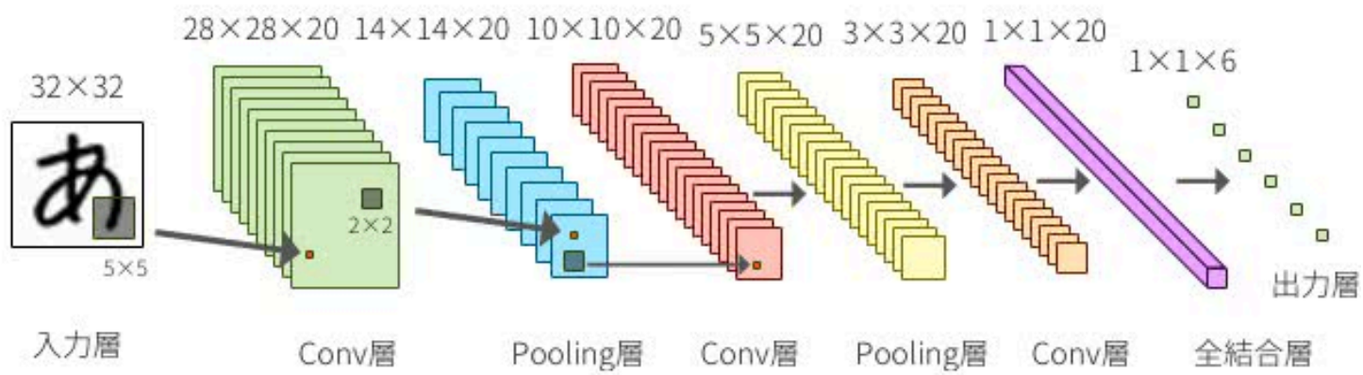
Style reference
Generated image
Target image

Loss = distance[style() - style()]

+ distance[content() - content()]

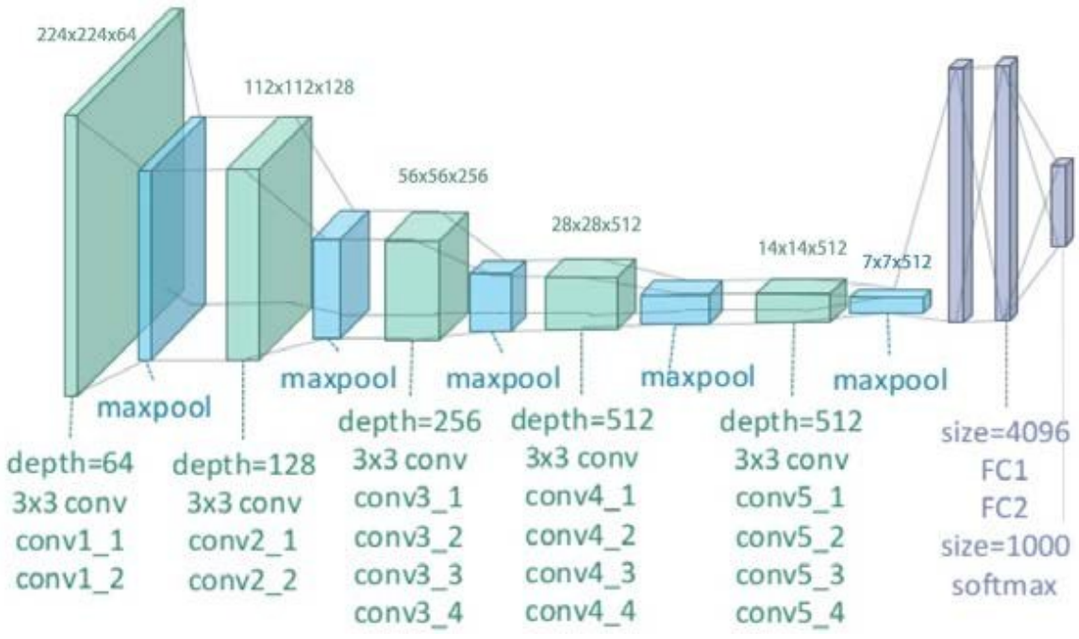
学習機が学んでいること

- 入力に近い側＝＞局所的な特徴
- 出力に近い側＝＞大局的、抽象的な特徴
- 今回は学習済みのVgg19の重みを使います。



https://deepage.net/deep_learning/2016/11/07/convolutional_neural_network.html

Vgg19



Style

Content

```
In [3]: model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (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_conv4 (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_conv4 (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_conv4 (Conv2D)        (None, None, None, 512)   2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, None, None, 512)   0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

Let's enjoy together!

- Google colabで動く。
- ローカルの環境でトライしてもOK。
- 画像の著作権などにはご留意ください。
- 前のセッションのネタを試したい方はそれでもいいですw。

https://colab.research.google.com/github/iwatobipen/playground/blob/master/mishimasyk_handson_2.ipynb

準備

```
[ ]   !mkdir ./img
      !wget http://jamesrobertlloyd.com/images/neural-art/style/picasso-face.jpg -P ./img
      !wget http://jamesrobertlloyd.com/images/neural-art/content/eiffel-tower-1.jpg -P ./img
      !wget https://upload.wikimedia.org/wikipedia/commons/0/0a/The_Great_Wave_off_Kanagawa.jpg -P ./img
      from keras.preprocessing.image import load_img, img_to_array
      target_path = './img/eiffel-tower-1.jpg'
      #style_reference_image_path = './img/picasso-face.jpg'
      style_reference_image_path = './img/The_Great_Wave_off_Kanagawa.jpg'

      width, height = load_img(target_path).size
      img_height = 400
      img_width = int(width * img_height/ height)
```

変換したい画像

転移したいスタイルを持つ画像

Loss関数の定義

文献中の記載
↓

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

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

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}\left(G_{ij}^{l} - A_{ij}^{l}\right)^2$$

$$\mathcal{L}_{style}(\vec{a},\vec{x}) = \sum_{l=0}^{L} w_l E_l$$

```
[ ]  from keras import backend as K
     target_image = K.constant(preprocess_image(target_path))
     style_reference_image =  K.constant(preprocess_image(style_reference_image_path))
     combination_image = K.placeholder((1, img_height, img_width, 3))
     input_tensor = K.concatenate([target_image, style_reference_image, combination_image], axis=0)
```

入力
- ターゲット（Const）
- スタイルリファレンス（Const）
- 合成画像（PF）

```
[ ]  model = vgg19.VGG19( input_tensor=input_tensor,
                          weights='imagenet',
                          include_top=False)
```

```
[ ]  #model.summary()
```

```
[ ]  def content_loss(base, combination):
       return K.sum(K.square(combination-base))

     def gram_matrix(x):
       features = K.batch_flatten( K.permute_dimensions(x,(2, 0, 1)))
       gram = K.dot(features, K.transpose(features))
       return gram

     def style_loss(style, combination):
       S = gram_matrix(style)
       C = gram_matrix(combination)
       channels = 3
       size = img_height * img_width
       return K.sum(K.square(S - C)) / (4. * (channels ** 2) * (size ** 2))
```

Channel毎にFlatten

=> h, w, c /3D
=> c, h, w /3D
=> c, h * w /2D

ドット積を取る

```
[ ]  def total_varidation_loss(x):
       a = K.square(
           x[:, :img_height - 1, :img_width - 1, :] -
           x[:, 1:, :img_width - 1, :])
       b = K.square(
           x[:, :img_height - 1, :img_width - 1, :] -
           x[:, :img_height - 1, 1:, :])
       return K.sum(K.pow(a + b, 1.25))
```

生成された画像を滑らかにするためにある

ネットワークの定義

```
[ ]   outputs_dict = dict([(layer.name, layer.output) for layer in model.layers])
      content_layer = 'block5_conv2'
      style_layers = [
                      'block1_conv1',
                      'block2_conv1',
                      'block3_conv1',
                      'block4_conv1',
                      'block5_conv1'
                     ]
      total_validation_weight = 1e-4
      style_weight = 1.
      content_weight = 0.025
```

各Loss関数の重み

Content Loss

```
[ ]   loss = K.variable(0.)
      layer_features = outputs_dict[content_layer]
      target_image_features = layer_features[0, :, :, :]
      combination_features = layer_features[2, :, :, :]
      loss += content_weight * content_loss(target_image_features, combination_features)
```

Style Loss

```
[ ]   for layer_name in style_layers:
        layer_features = outputs_dict[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_reference_features, combination_features)
        loss += (style_weight / len( style_layers)) * sl
      loss += total_validation_weight * total_varidation_loss(combination_image)
```

Almost there!

```
[ ]   loss = K.variable(0.)
      layer_features = outputs_dict[content_layer]
      target_image_features = layer_features[0, :, :, :]
      combination_features = layer_features[2, :, :, :]
      loss += content_weight * content_loss(target_image_features, combination_features)
```

```
[ ]   for layer_name in style_layers:
        layer_features = outputs_dict[layer_name]
        style_reference_features = layer_features[1, :, :, :]
        combination_features = layer_features[2, :, :, :]
        sl = style_loss(style_reference_features, combination_features)
        loss += (style_weight / len( style_layers)) * sl
      loss += total_validation_weight * total_varidation_loss(combination_image)
```

```
[ ]   grads = K.gradients(loss, combination_image)[0]
      fetch_loss_and_grads = K.function([combination_image], [loss, grads])
```

入力からLossと勾配を出力する関数を作った

Param. optimization

```
[ ]   evaluator = Evaluator()
      from scipy.optimize import fmin_l_bfgs_b
      from scipy.misc import imsave
      import time
      result_prefix = 'my_result'
      iterations = 20
      x = preprocess_image(target_path)
      x = x.flatten()
      for i in range(iterations):
        print(i)
        start_time = time.time()
        x, min_val, info = fmin_l_bfgs_b(evaluator.loss,
                                         x,
                                         fprime=evaluator.grads,
                                         maxfun=20)
        print('current loss value {}'.format(min_val))
        img = x.copy().reshape((img_height, img_width, 3))
        img = deprocess_image(img)
        fname = result_prefix + '_at_iteration_{}_th.png'.format(i)
        imsave(fname, img)
        end_time = time.time()
        print('iteration {} completed in {}'.format(i, end_time-start_time))
```

原著論文でBFGSの最適化が良かったとあり、それを使う。
KerasにはないのでScipyから。

xが画像の入力で
これを最適化する

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fmin_l_bfgs_b.html

Enjoy!