12.8

Style Transfer, Gan, AlexNet and VGGNet

1.STYLE TRANSFER:

http://blog.csdn.net/stdcoutzyx/article/details/53771471

主要分为三块内容：

1. 内容提取（content）
2. 风格提取（style）
3. 风格转换

最终在风格转换过程中调整 alpha 和 beta 参数。

内容提取

和之前类似，内容就是采用CNN的某一层或者某几层来表示，一般来说，层级越高，表示就越抽象。这里，需要有几个形式化的表达：

- M_l: 第l层的feature map的大小
- N_l: 第l层的filter的数目
- F^l: 图像在第l层的特征表示，是一个矩阵，矩阵大小为$M_l * N_l$.
- F^l_{ij}: 第l层第i个filter上位置j处的激活值。
- p: 原始内容图片
- x: 生成图片
- P^l: 原始图片在CNN中第l层的表示
- F^l: 生成图片在CNN中第l层的表示

因而，我们就得到了内容的loss。

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

求导即为：

$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F_{ij}^l} = \begin{cases} \left(F^l - P^l\right)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

风格提取

而风格的转换则是这篇论文的神来之笔，论文使用相关矩阵来表示图像的风格。当然，风格的抽取仍然是以层为单位的。

- a: 初始风格图片
- A^l: 风格图片某一层的风格特征表示。
- G^l: 生成图片某一层的风格特征表示，大小为$N_l * N_l$

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

其中，G^l_{ij}的值是l层第i个feature map和第j个feature map的内积。

从而，我们得到了风格损失函数。

单独某层的损失函数：

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

各层综合的损失函数：

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

求偏导：

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left((F^l)^{\mathrm{T}} \left(G^l - A^l\right)\right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}.$$

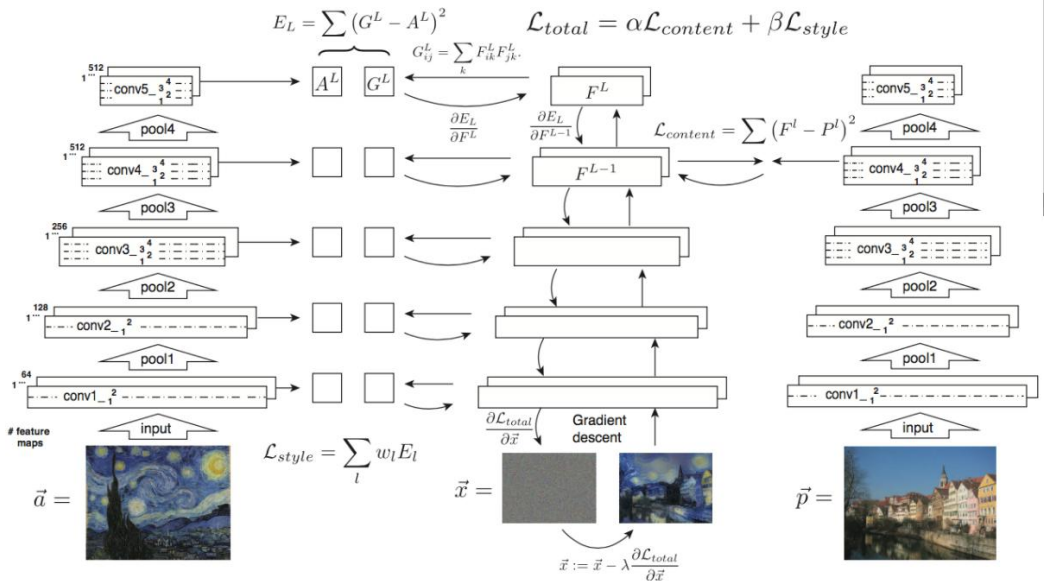
与内容表示类似，如果我们用随机初始化的x，保持CNN参数不变，将风格图片A和x输入进网络，然后对x求导，x就会在风格上趋近于A。

风格转换

有了内容与风格，风格转换就呼之欲出了，即两种loss的加权。

$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{\text{content}}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{\text{style}}(\vec{a}, \vec{x})$$

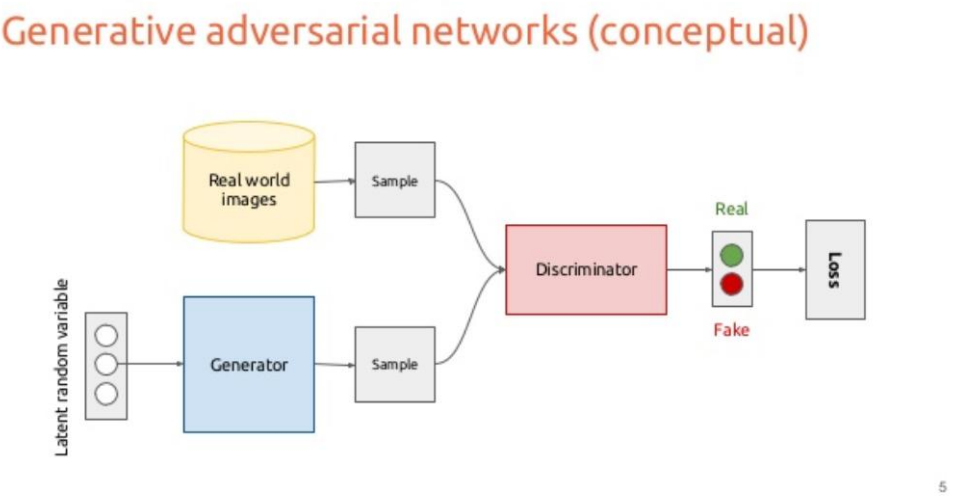
也可如图示：



即同时将三张图片(a, p, x)输入进三个相同的网络，对a求出风格特征，对p求出内容特征，然后对x求导，这样，得到的x就有a的风格和p的内容。

2. GAN

http://blog.csdn.net/jackytintin/article/details/61908718



GAN 的基本思想是，生成器和判别器玩一场“道高一尺，魔高一丈”的游戏：判别器要练就“火眼金睛”，尽量区分出真实的样本（如真实的图片）和由生成器生成的假样本；生成器要学着“以假乱真”，生成出使判别器判别为真实的“假样本”。

竞争的理想怦是双方都不断进步——（理想情况下）判别器的眼睛越发“雪亮”，生成器的欺骗能力也不断提高。**对抗的胜负无关紧要，重要的是，最后生成器的欺骗能力足够好，能够生成与真实样本足够相似的样本**——直观而言，生成的样本看起来像是训练集（如图片）的样本；形式化的，生成器生成样本的分布，应该与训练集样本分布接近。

理论上可以，在理想条件下，生成器是可以通过这种对抗得到目标分布的（即生成足够真实的样本）。

判别器

$$L_D = -\Sigma_i \log(D(\mathbf{x}_i)) - \Sigma_i \log(1 - D(G(\mathbf{z}_i)))$$

判别器的训练的目标为：对于真实样本，输出尽量接近1；对于生成器生成的假样本，输出尽量接近0。
也即训练判别器时，真实样本的标签为1，生成样本的标签为0。

生成器

$$L_G = \Sigma_i \log(1 - D(G(\mathbf{z}_i)))$$

判别器的训练的目标为生成的假样本，使判别器的输出尽量接近1，即尽量以假乱真。
为了解决训练过程中，梯度消失的问题，一般使用如下损失函数 (Trick 2)：

$$L_G = -\Sigma_i \log(D(G(\mathbf{z}_i)))$$

为使用这个损失函数，只需要将生成样本的标签为1，同时使用变通的交叉熵损失函数。

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

3. AlexNet 和 VGGNet
http://blog.csdn.net/sinat_25434937/article/details/51254064

4. 梯度消失和爆炸
http://blog.csdn.net/qq_29133371/article/details/51867856
http://blog.csdn.net/cppjava_/article/details/68941436
是神经网路算法用 gradient 之类不稳定性的体现