Image Analogies 实验报告

2015011004 鞠家兴

1. 算法原理

image analogy 是 2001的一篇 paper，用来自动提取滤镜特征然后应用到其他图片上的一种算法。算法要求输入3张图片: A, A', B。A 与 A' 作为训练数据，提取 A -> A' 的转换特征，从 B 计算得到图像 B'。

1.1 数据结构与表示

用 p 来表示 A 和 A' 中的某个像素，q 表示 B 中的某个像素。使用 A(p) 表示 p 处的特征向量，同理 A'(p), B(q), B'(q) ，算法需要跟踪 A' 中被复制到 B' 的像素 q 处的像素 p，记为 s(q) = p。

1.2 算法大概

算法的思路是对于 B 中的每一个像素q，找到在 A 中与 q 最相似的像素 p，把 p 复制到 B' 的位置 q 上。算法可以表示为：

function CREATEIMAGEANALOGY(A, A', B):
  Compute Gaussian pyramids for A, A', and B
  Compute features for A, A', and B
  Initialize the search structures (e.g., for ANN)
  for each level ℓ, from coarsest to finest, **do**:
    for each pixel $q \in B'_\ell$, in scan-line order, **do**:
      $p \leftarrow$ BESTMATCH(A, A', B, B', s, ℓ, q)
      $B'_\ell(q) \leftarrow A'_\ell(p)$
      $s_\ell(q) \leftarrow p$
  return B'_L

首先利用高斯下采样，得到起更低分辨率的不同尺度的输入图片，如果令 A 的长宽为 (Row_A, Row_B)，那么下采样得到的图片大小为 $(Row_A/2, Row_B/2), (Row_A/4, Row_B/4), \ldots$，记分辨率级别 l 下的图片为 A_l, A'_l, B_l。 计算分辨率金字塔是为了更好的衡量像素间的相似度。然后计算用来表示 feature 的向量，并初始化搜索结构。准备工作完成后，从低分辨率到高分辨率，依次计算 B'， 对于B 中的每一个像素q，找到在 A 中与 q 最相似的像素 p，把 p 复制到 B' 的位置 q 上，并记录下 p、q 的对应关系。

1.3 feature 表示与相似度计算

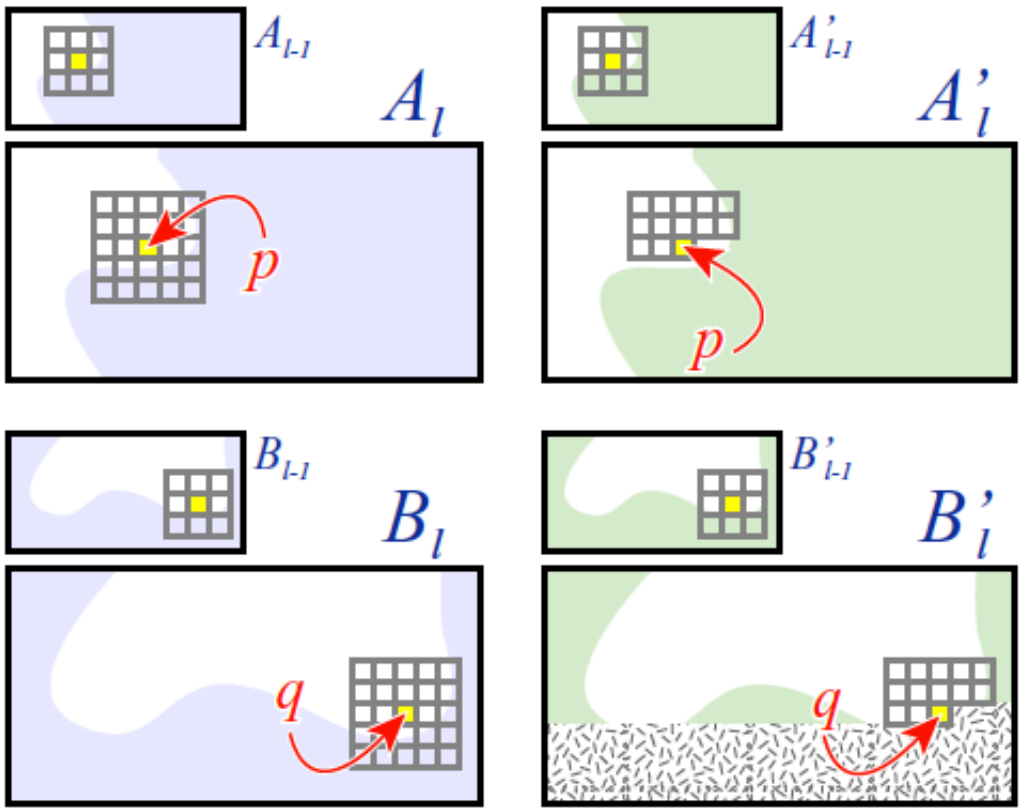


Figure 2 Neighborhood matching. In order to synthesize the pixel value at q in the filtered image B'_ℓ, we consider the set of pixels in B'_ℓ, B_ℓ, $B'_{\ell-1}$, and $B_{\ell-1}$ around q in the four images. We search for the pixel p in the A images that give the closest match. The synthesis proceeds in scan-line ordering in B'_ℓ.

向量长度取 56，取法如图。边缘使用 0 填充。因为 RGB 色彩空间因为维度爆炸的问题，不如灰度图包含更多信息，所以先把图像转为 YIQ 模式，提取 Y 通道的值，作为 feature 计算。

相似度的计算采用欧氏距离，需要与高斯核卷积，为了避免后续每次相乘，在feature 计算阶段即与 gauss kernel 卷积。

function BESTMATCH(A, A', B, B', s, ℓ, q):
  $p_{\text{app}} \leftarrow$ BESTAPPROXIMATEMATCH(A, A', B, B', ℓ, q)
  $p_{\text{coh}} \leftarrow$ BESTCOHERENCEMATCH(A, A', B, B', s, ℓ, q)
  $d_{\text{app}} \leftarrow \|F_\ell(p_{\text{app}}) - F_\ell(q)\|^2$
  $d_{\text{coh}} \leftarrow \|F_\ell(p_{\text{coh}}) - F_\ell(q)\|^2$
  if $d_{\text{coh}} \le d_{\text{app}}(1 + 2^{\ell-L}\kappa)$ **then**
    return p_{coh}
  else
    return p_{app}

BestMatch 算法如上图，p_{app} 是使用 ANN 计算得到的与 q 点的 feature 欧氏距离最短的，这部分使用 flann 库提供的接口完成；

$BestCoherenceMatch$ 返回 $s(r^*) + (q - r^*)$，其中，

$r^* = arg\ min_{r \in N(q)}\ ||F_l(s(r) + (q - r)) - F_l(q)||^2$

N(q) 指的是与 q 相邻的以及计算好的像素位置，这个公式简单的返回与已经计算好的邻域结果一致的最好像素。

对于部分明度差异过大的或者艺术滤镜的应用，还要使用明度映射来抹去 A 、B 使相似度的计算更有意义。计算公式为

$$Y(p) \leftarrow \frac{\sigma_B}{\sigma_A}(Y(p) - \mu_A) + \mu_B$$

经过测试发现对于一些笔刷滤镜，只有映射才能得到好的效果。

2. 效果

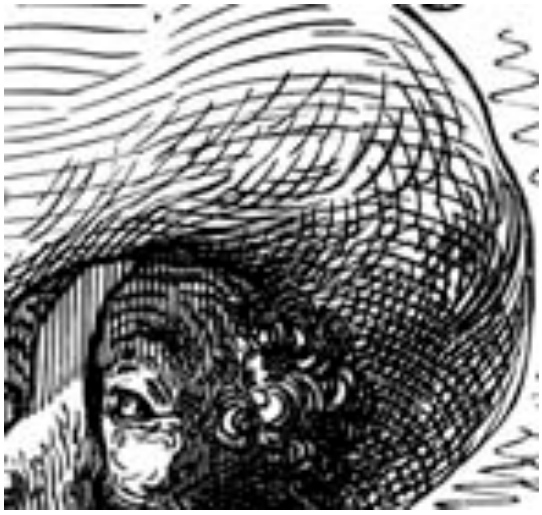
2.1 模糊

B B'

2.2 上色


2.3 各种笔刷


A A'

A A'

3. 思考与改进

算法有参数 κ，不同的参数对效果影响很大。对于上色和模糊来说，κ 取 0；对于后面的笔刷滤镜，κ 要去的稍微大一些，这从直观上可以理解：笔刷注重区域间的连续性，要模拟落笔的操作，所以需要加大 coherence 匹配的权重。另外为了加速匹配，与其计算时在线与高斯核卷积，不如提前计算feature的时候卷积，这样稍微增加了准备阶段的时间，但是减少了运行时的时间。

这个算法对所有输入并不是一视同仁的，应该根据输入的类别选择不同的参数，或是选择是否需要上色，或是选择是否需要明度 remapping，这需要经验来判断，不过确实提供了一个很好的角度与想法，为后来的 Deep Image Analogy 提供了思路。