

每周学习进展阶段汇报

---CS131 Stanford University 计算机视觉基础课程

汇报人：胡小婉

时间段：2018 年 2 月 5 日至 2018 年 8 日&&2 月 14 日至 15 日

1. Visual Bag of Words.....	3
1.1 算法原理.....	3
1.2 算法步骤.....	3
2. Spatial Pyramid Matching(空间金字塔匹配).....	5
2.1 Pyramids.....	5
2.2 Bag of Words + Pyramids.....	5
2.3 Pyramid Match Kernels.....	6
3. Naive Bayes(朴素贝叶斯).....	7
3.1 基本原理.....	7
3.2 基本步骤.....	8
3.2.1 Prior(先验).....	8
3.2.2 Posterior(后验).....	8
3.2.3 Classification(分类).....	8
4. support vector machine (SVM).....	9
4.1 基本原理.....	9
4.2 推导过程及应用.....	11
5. HOG 特征的提取--基于 scikit-image.....	11
5.1 基本思想.....	11
5.2 python 实现.....	12
Tips: 几种归一化方法 (Normalization Method) python 实现.....	13
1. (0,1)标准化 :	13
2. Z-score 标准化 :	13
3. Sigmoid 函数.....	13
6. Evaluating Object Detection.....	14
7. A Simple Sliding Window Detector.....	15
8. The Deformable Parts Model (DPM).....	15
8.1 基本原理.....	16
8.2 算法实现.....	16
8.2.1 HOG.....	16
8.2.2 DPM 模型.....	16
Tip:参考网址.....	18
9. Hw7(作业题).....	18
9.1 Hog Representation.....	18
9.2 Sliding Window.....	19
9.3 Image Pyramid.....	20
9.4 Pyramid Score.....	20
9.5 Deformable Parts Detection.....	21
9.6 Human Parts Location.....	22
9.7 Gaussian Filter.....	25
Tips:skimage.filters 的 Gaussian 函数.....	27

Lecture #14: Visual Bag of Words

Lecture 15: Detecting Objects by Parts

Lecture 16: Recognizing Objects by Parts

1. Visual Bag of Words

1.1 算法原理

Bag of words 模型最初被用在文本分类中，将文档表示成特征矢量。它的基本思想是假定对于一个文本，忽略其词序和语法、句法，仅仅将其看做是一些词汇的集合，而文本中的每个词汇都是独立的。简单说就是讲每篇文档都看成一个袋子（因为里面装的都是词汇，所以称为词袋，Bag of words 即因此而来），然后看这个袋子里装的都是些什么词汇，将其分类。如果文档中猪、马、牛、羊、山谷、土地、拖拉机这样的词汇多些，而银行、大厦、汽车、公园这样的词汇少些，我们就倾向于判断它是一篇描绘乡村的文档，而不是描述城镇的

In Computer Vision, we can consider an image to be a collection of image features. By incorporating frequency counts of these features, we can apply the "Bag of Words" model towards images and use this for prediction tasks such as image classification and face detection.

There are two main steps for the "Bag of Words" method when applied to computer vision, and these will further be explored in the Outline section below.

1. Build a "dictionary" or "vocabulary" of features across many images - what kinds of common features exist in images? We can consider, for example, color scheme of the room, parts of faces such as eyes, and different types of objects.

2. Given new images, represent them as histograms of the features we had collected - frequencies of the visual "words" in the vocabulary we have built.

1.2 算法步骤

Bag of visual word 类似于 BoW 模型，基本思想概括如下：

1) 提取特征 (Extract Features)

根据具体应用考虑，综合考虑特征的独特性、提取复杂性、效果好坏，处理是否方便等选择特征。

2) 学习视觉词袋 (Learn Visual Vocabulary)

统计图像数据库中出现的所有特征，去除冗余组成词袋。如果提取的图像特征过多，一般需要利用聚类算法先把相近的单词归为一类（类似于文档检索里的找词根），利用这些聚类结果来组成词袋。

- 3) 利用视觉词袋量化图像特征 (Quantize features using visual vocabulary)
- 4) 利用词频表示图像 (Represent images by frequencies of visual words)

1. Extracting Interesting Features
2. Learning Visual Vocabulary

To find textons, we simply cluster our features. We can use any clustering technique (K-Means is most common, but Mean Shift or HAC may also work) to cluster the features. We then use the centers of each cluster as the textons. Our set of textons is known as a visual vocabulary. An example of a visual vocabulary is given below.

3. Quantize Features
4. Represent Images by Frequencies

From clustering to vector quantization

- Clustering is a common method for learning a visual vocabulary or codebook
 - Unsupervised learning process
 - Each cluster center produced by k-means becomes a codevector
 - Codebook can be learned on separate training set
 - Provided the training set is sufficiently representative, the codebook will be “universal”
- The codebook is used for quantizing features
 - A *vector quantizer* takes a feature vector and maps it to the index of the nearest codevector in a codebook
 - Codebook = visual vocabulary
 - Codevector = visual word

1.3 Large-Scale Image Search

1.3.1 TF-IDF weighting

- Instead of computing a regular histogram distance, we'll weight each word by it's *inverse document frequency*

- inverse document frequency (IDF) of word j =

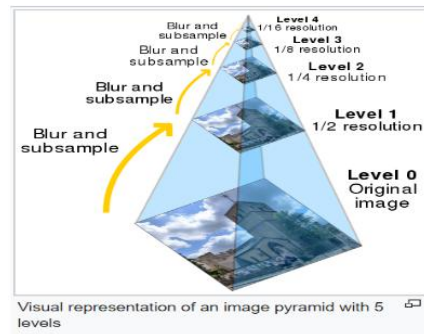
$$\log \frac{\text{number of documents}}{\text{number of documents in which } j \text{ appears}} \quad \textit{the, and, or} \quad \text{vs.} \quad \textit{cow, AT\&T, Cher}$$

- To compute the value of bin j in image l :

term frequency of j in l \times *inverse document frequency of j*

2. Spatial Pyramid Matching(空间金字塔匹配)

2.1 Pyramids

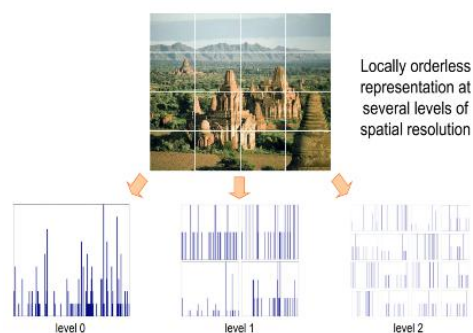


2.2 Bag of Words + Pyramids

Bag of Words alone doesn't discriminate if a patch was obtained from the top, middle or bottom of the image because it doesn't save any spatial information. Spatial pyramid matching partitions the image into increasingly fine sub-regions and allows us to compute histograms (BoW) of local features inside each sub-region. [?]

If the BoWs of the upper part of the image contain "sky visual words", the BoWs in the middle "vegetation and mountains visual words" and the BoWs at the bottom "mountains visual words", then it is very likely that the image scene category is "mountains".

SPM即Spatial Pyramid Matching，是一种利用空间金字塔进行图像匹配、识别、分类的算法。SPM是BOF(Bag Of Features)的改进，因为BOF是在整张图像中计算特征点的分布特征，进而生成全局直方图，所以会丢失图像的局部/细节信息，无法对图像进行精确地识别。为了克服BOF的固有缺点，作者提出了SPM算法，它是在不同分辨率上统计图像特征点分布，从而获取图像的局部信息。



2.3 Pyramid Match Kernels

1) 假设存在两个特征集合 X, Y , 其中每个特征 x 的维度为 d 。将特征空间划分为不同的尺度 $0, \dots, L$, 在尺度 ℓ 下把特征空间的每一维划出 2^ℓ 个bins, 那么 d 维的特征空间就能划出 $D = 2^{d\ell}$ 个bins (论文中这么描述, 但是在实际中是用K-means或BOW进行聚类, 得到的每个类中心就是一个bin)。

2) 在level(ℓ)中, 如果点 x, y 落入同一bin中就称 x, y 点Match, 每个bin中匹配的点的个数为 $\min(X_i, Y_i)$, 其中 X_i, Y_i 代表相应level下的第 i 个bin。

3) H_X^ℓ and H_Y^ℓ 表示 X, Y 在level ℓ 下的直方图特征, $H_X^\ell(i)$ and $H_Y^\ell(i)$ 表示level ℓ 中 X, Y 落入第 i 个bin的特征点的个数, 那么在level ℓ 下匹配的点的总数为:

$$\mathcal{I}(H_X^\ell, H_Y^\ell) = \sum_{i=1}^D \min(H_X^\ell(i), H_Y^\ell(i)).$$

在后文中, 我们把 $\mathcal{I}(H_X^\ell, H_Y^\ell)$ 简写为 \mathcal{I}^ℓ 。

4) 统计各个尺度下match的总数 \mathcal{I}^ℓ (就等于直方图相交)。由于细粒度的bin被大粒度的bin所包含, 为了不重复计算, 每个尺度的有效Match定义为match的增量 $\mathcal{I}^\ell - \mathcal{I}^{\ell+1}$;

5) 不同的尺度下的match应赋予不同权重, 显然大尺度的权重小, 而小尺度的权重大, 因此定义权重为 $\frac{1}{2^{L-\ell}}$

6) 两个点集 X, Y 的匹配程度pyramid match kernel为:

$$\begin{aligned} \kappa^L(X, Y) &= \mathcal{I}^L + \sum_{\ell=0}^{L-1} \frac{1}{2^{L-\ell}} (\mathcal{I}^\ell - \mathcal{I}^{\ell+1}) \\ &= \frac{1}{2^L} \mathcal{I}^0 + \sum_{\ell=1}^L \frac{1}{2^{L-\ell+1}} \mathcal{I}^\ell. \end{aligned} \quad (3)$$

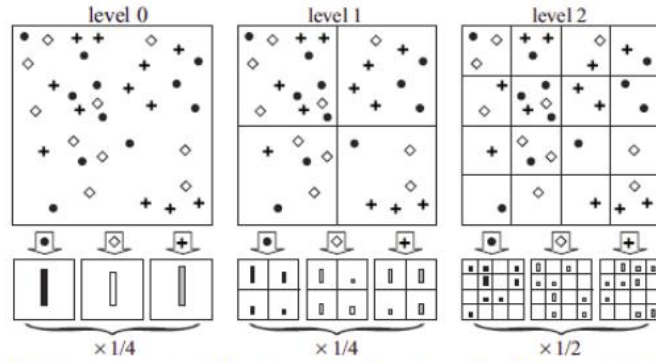


Figure 1. Toy example of constructing a three-level pyramid. The image has three feature types, indicated by circles, diamonds, and crosses. At the top, we subdivide the image at three different levels of resolution. Next, for each level of resolution and each channel, we count the features that fall in each spatial bin. Finally, we weight each spatial histogram according to eq. (3).

上面的黑圆点、方块、十字星代表一副图像上某个pitch属于kmeans后词典中的某个词；

- 1) 将图像划分为固定大小的块，如从左到右：1*1, 2*2, 4*4, 然后统计每个方块中词中的不同word的个数；
- 2) 从左到右，统计不同level中各个块内的直方图；
- 3) 最后将每个level中获得的直方图都串联起来，并且给每个level赋给相应的权重，从左到右权重依次增大；
- 4) 将spm放入svm中进行训练和预测；

论文及参考网址：

Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories

<http://blog.csdn.net/chlele0105/article/details/16972695>

3. Naive Bayes(朴素贝叶斯)

3.1 基本原理

贝叶斯分类是一类分类算法的总称，这类算法以贝叶斯定理为基础，故统称为贝叶斯分类。贝叶斯定理解决了现实生活中经常遇到的问题：已知某条件概率，如何得到事件交换后的概率，即在已知 $P(A|B)$ 的情况下求得 $P(B|A)$ 。条件概率 $P(A|B)$ 表示事件 B 已经发生的前提下，事件 A 发生的概率，叫做事件 B 条件下发生事件 A 的条件概率。其基本求解公式为： $P(A|B)=P(AB)/P(B)$ 。贝叶斯定理：

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

贝叶斯的主要思想可以概括为：先验概率+数据=后验概率。贝叶斯定理换个表达形式：

$$P(\text{类别}|\text{特征}) = \frac{P(\text{特征}|\text{类别})P(\text{类别})}{P(\text{特征})}$$

朴素贝叶斯的含义是：朴素——特征条件独立，贝叶斯——基于贝叶斯定理。朴素贝叶斯分类是一种十分简单的分类方法，这种分类的思想真的很朴素：对于给出的待分类项，求解此项条件下各个类别出现的概率，概率最大的类别就认为此项属于该类别


3.2 基本步骤

3.2.1 Prior(先验)

Using the prior equation, we can now calculate the probability than the image represented by histogram x belongs to class category c using **Bayes Theorem**

$$P(c|x) = \frac{P(c)P(x|c)}{\sum_{c'} P(c')P(x|c')}$$

Expanding the numerator and denominator, we can rewrite the previous equation as


$$P(c|x) = \frac{P(c) \prod_{i=1}^m P(x_i|c)}{\sum_{c'} P(c') \prod_{i=1}^m P(x_i|c')}$$

3.2.2 Posterior(后验)

Using the prior equation, we can now calculate the probability than the image represented by histogram x belongs to class category c using **Bayes Theorem**

$$P(c|x) = \frac{P(c)P(x|c)}{\sum_{c'} P(c')P(x|c')}$$

Expanding the numerator and denominator, we can rewrite the previous equation as

$$P(c|x) = \frac{P(c) \prod_{i=1}^m P(x_i|c)}{\sum_{c'} P(c') \prod_{i=1}^m P(x_i|c')}$$

3.2.3 Classification(分类)

In order to classify the image represented by histogram x , we simply find the class c^* that maximizes the previous equation:

$$c^* = \operatorname{argmax}_c P(c|x)$$

Since we end up multiplying together a large number of very small probabilities, we will likely run into unstable values as they approach 0. As a result, we use logs to calculate probabilities:

$$c^* = \operatorname{argmax}_c \log P(c|x)$$

Now consider two classes c_1 and c_2 :

$$P(c_1|x) = \frac{P(c_1) \prod_{i=1}^m P(x_i|c_1)}{\sum_{c'} P(c') \prod_{i=1}^m P(x_i|c')}$$

and

$$P(c_2|x) = \frac{P(c_2) \prod_{i=1}^m P(x_i|c_2)}{\sum_{c'} P(c') \prod_{i=1}^m P(x_i|c')}$$

Since the denominators are identical, we can ignore it when calculating the maximum. Thus

$$P(c_1|x) \propto P(c_1) \prod_{i=1}^m P(x_i|c_1)$$

and

$$P(c_2|x) \propto P(c_2) \prod_{i=1}^m P(x_i|c_2)$$

and for the general class c :

$$P(c|x) \propto P(c) \prod_{i=1}^m P(x_i|c)$$

and using logs:

$$\log P(c|x) \propto \log P(c) + \sum_{i=1}^m \log P(x_i|c)$$

Now, classification becomes

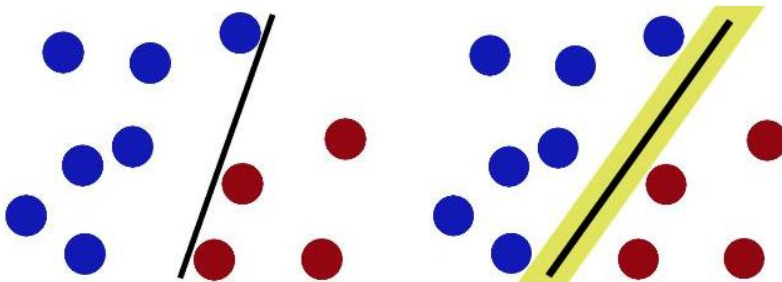
$$\begin{aligned} c^* &= \operatorname{argmax}_c P(c|x) \\ c^* &= \operatorname{argmax}_c \log P(c|x) \\ c^* &= \operatorname{argmax}_c \log P(c) + \sum_{i=1}^m \log P(x_i|c) \end{aligned}$$

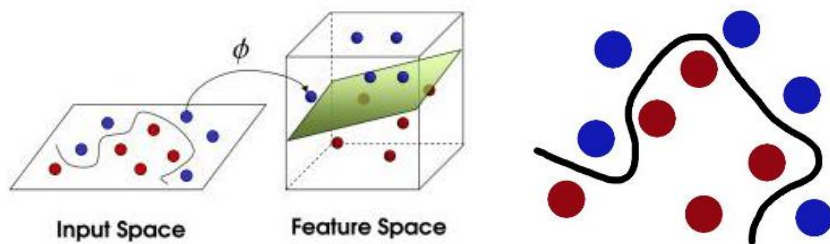
4. support vector machine (SVM)

4.1 基本原理

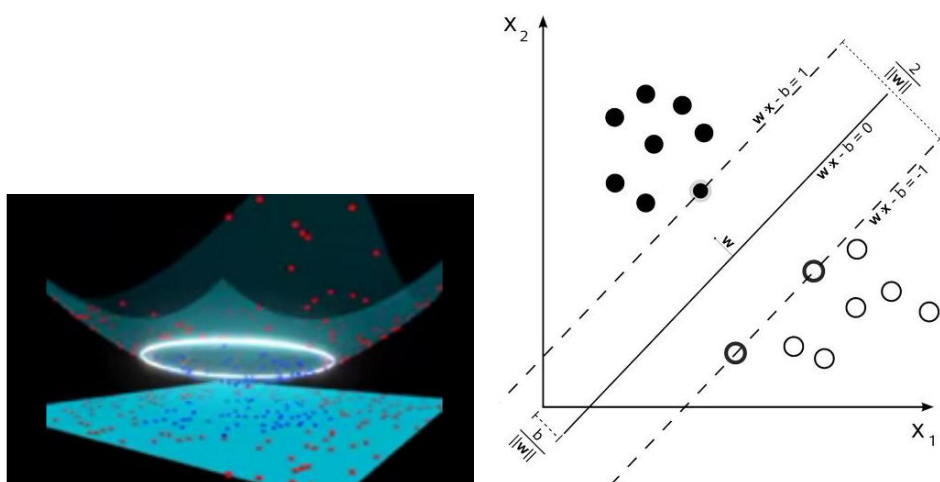
支持向量机 (Support Vector Machine ,SVM) 的主要思想是：建立一个最优决策超平面，使得该平面两侧距离该平面最近的两类样本之间的距离最大化，从而对分类问题提供良好的泛化能力。对于一个多维的样本集，系统随机产生一个超平面并不断移动，对样本进行分类，直到训练样本中属于不同类别的样本点正好位于该超平面的两侧，满足该条件的超平面可能有很多个，SVM 正式在保证分类精度的同时，寻找到这样一个超平面，使得超平面两侧的空白区域最大化，从而实现对线性可分样本的最优分类。

支持向量机中的支持向量 (Support Vector) 是指训练样本集中的某些训练点，这些点最靠近分类决策面，是最难分类的数据点。SVM 中最优分类标准就是这些点距离分类超平面的距离达到最大值；“机” (Machine) 是机器学习领域对一些算法的统称，常把算法看做一个机器，或者学习函数。SVM 是一种有监督的学习方法，主要针对小样本数据进行学习、分类和预测





把这些球叫做「data」,把棍子 叫做「classifier」,最大间隙 trick 叫做「optimization」,拍桌子叫做「kernelling」,那张纸叫做「hyperplane」。

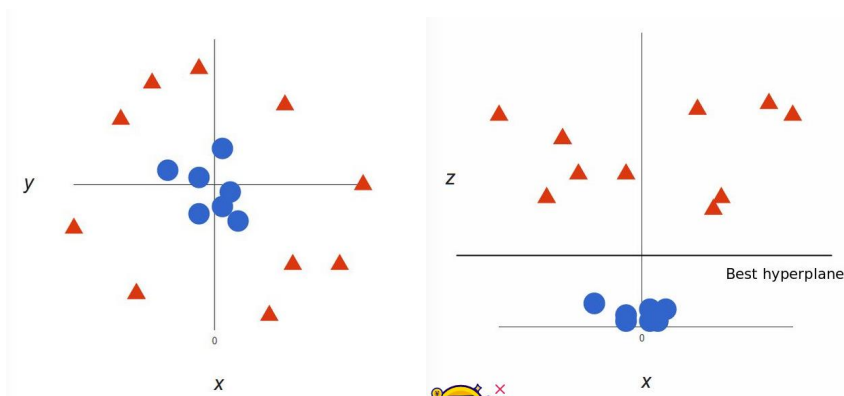


在 SVM 中,我们寻找一条最优的分界线使得它到两边的 margin 都最大,在这种情况下边缘加粗的几个数据点就叫做 support vector,这也是这个分类算法名字的来源。

$$\arg \max_{\text{boundary}} \text{margin}(\text{boundary}),$$

subject to 所有正确归类的苹果和香蕉到boundary的距离都 $\geq \text{margin}$

(可以把margin看作是boundary的函数,并且想要找到使得是使得margin最大化的boundary,而margin(*)这个函数是:输入一个boundary,计算(正确分类的)所有苹果和香蕉中,到boundary的最小距离。)



还好，我们已经找到了诀窍：SVM 其实并不需要真正的向量，它可以用它们的数量积（点积）来进行分类。这意味着我们可以避免耗费计算资源的境地了。我们需要这样做：

想象一个我们需要的新空间：

$$z = x^2 + y^2$$

找到新空间中点积的形式：

$$a \cdot b = x_a \cdot x_b + y_a \cdot y_b + z_a \cdot z_b$$

$$a \cdot b = x_a \cdot x_b + y_a \cdot y_b + (x_a^2 + y_a^2) \cdot (x_b^2 + y_b^2)$$

让 SVM 处理新的点积结果——这就是核函数

这就是核函数的技巧，它可以减少大量的计算资源需求。通常，内核是线性的，所以我们得到了一个线性分类器。但如果使用非线性内核（如上例），我们可以在完全不改变数据的情况下得到一个非线性分类器：我们只需改变点积为我们想要的空间，SVM 就会对它忠实地进行分类。

引入核函数，就是使用核函数对应高维空间内积的性质来使得线性分类器可以隐式地在高维空间建立分类面。因为在高维空间中的分类面更容易绕过一些低维空间中不可分的区域，这样可以达到更好的分类效果。

(1) 它是针对线性可分情况进行分析，对于线性不可分的情况，通过使用非线性映射算法将低维输入空间线性不可分的样本转化为高维特征空间，使其线性可分，从而使得高维特征空间采用线性算法对样本的非线性特征进行线性分析成为可能；

(2) 它基于结构风险最小化理论之上在特征空间中建构最优分割超平面，使得学习器得到全局最优，并且在整个样本空间的期望风险以某个概率满足一定上界。

4.2 推导过程及应用

blog.csdn.net/american199062/article/details/51322852

知乎：<https://www.zhihu.com/question/21094489>

5. HOG 特征的提取--基于 scikit-image

5.1 基本思想

HOG 特征, histogram of oriented gradient, 梯度方向直方图特征，作为提取基于梯度的特征，HOG 采用了统计的方式(直方图)进行提取。其基本思路是将图像局部的梯度统计特征拼接起来作为总特征。局部特征在这里指的是将图像划分为多个 Block，每个 Block 内的特征进行联合以形成最终的特征。具体来说：

将图像分块：以 Block 为单位，每个 Block 以一定的步长在图像上滑动，以此来产生新的 Block。

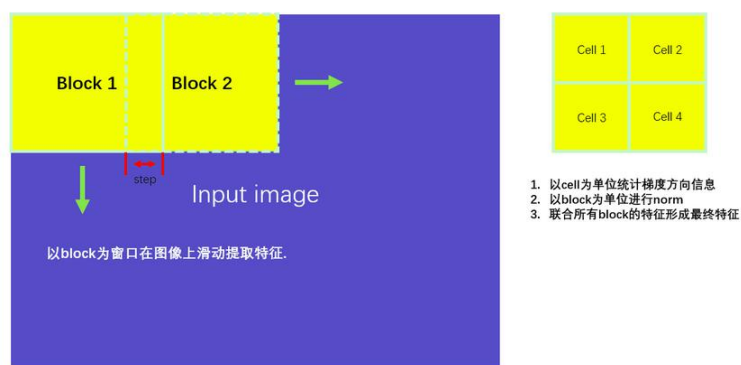
Block 作为基本的特征提取单位，在其内部再次进行细分：将 Block 划分为(一般是均匀划分) $N \times N$ 的小块，每个小块叫做 cell。

cell 是最基本的统计单元，在 cell 内部，统计每个像素的梯度方向，并将它们映射到预设的 M 个方向的 bin 里面形成直方图。

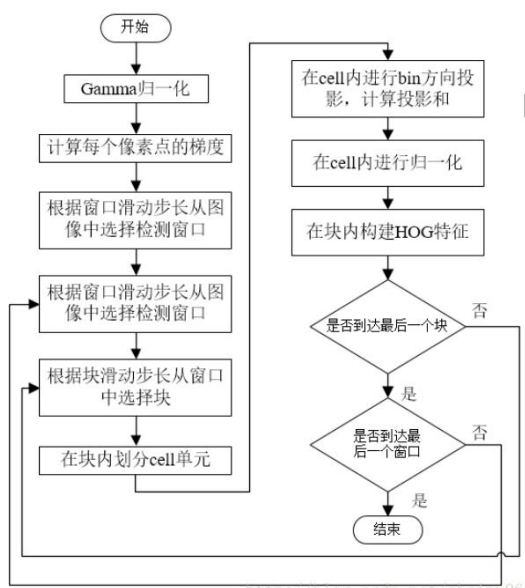
每个 Block 内部的所有 cell 的梯度直方图联合起来并进行归一化处理(L1-norm, L2-Norm, L2-hys-norm, etc), 据说这样可以使特征具有光照不变性. 光照属于加性噪声, 归一化之后会抵消掉光照变化对特征的影响.

所有 Block 的特征联合起来, 就是最终的 HOG 特征

这里牵扯到一些技术细节, 比如将局部的梯度方向映射到预设的方向 Bin 里面需要双线性插值或三线插值. 某点的梯度方向的映射是按照改点的梯度强度进行加权的.



5.2 python 实现



实现

基于python的scikit-image库提供了HOG特征提取的接口:

```
1 from skimage import feature as ft
2 features = ft.hog(image, # input image
3                  orientations=ori, # number of bins
4                  pixels_per_cell=ppc, # pixel per cell
5                  cells_per_block=cpb, # cells per block
6                  block_norm = 'L1', # block norm : str {'L1', 'L1-sqrt', 'L2', 'L2-sqrt'}
7                  transform_sqrt = True, # power law compression (also known as gaussian)
8                  feature_vector=True, # flatten the final vectors
9                  visualise=False) # return HOG map
```

参数说明:

- image: input image, 输入图像
- orientation: 指定bin的个数. scikit-image 实现的只有无符号方向.
(根据反正切函数的到的角度范围是在-180°~ 180°之间, 无符号是指把 -180°~0°这个范围统一加上180°转换到0°~180°范围内. 有符号是指将-180°~180°转换到0°~360°范围内.)
也就是说把所有的方向都转换为0°~180°内, 然后按照指定的orientation数量划分bins. 比如你选定的orientation= 9, 则bin一共有9个, 每20°一个:
[0°~20°, 20°~40°, 40°~60°, 60°~80°, 80°~100°, 100°~120°, 120°~140°, 140°~160°, 160°~180°]
- pixels_per_cell : 每个cell的像素数, 是一个tuple类型数据,例如(20,20)
- cell_per_block : 每个BLOCK内有多少个cell, tuple类型, 例如(2,2), 意思是将block均匀划分为2x2的块
- block_norm: block 内部采用的norm类型.
- transform_sqrt: 是否进行 power law compression, 也就是gamma correction. 是一种图像预处理操作, 可以将较暗的区域变亮, 减少阴影和光照变化对图片的影响.
- feature_vector: 将输出转换为一维向量.
- visualise: 是否输出HOG image, (应该是梯度图)
- scikit-image 版的HOG 没有进行cell级别的gaussian 平滑, 原文对cell进行了gamma= 8pix的高斯平滑操作.

Tips: 几种归一化方法 (Normalization Method) python 实现

1. (0,1)标准化:

这是最简单也是最容易想到的方法,通过遍历 feature vector 里的每一个数据,将 Max 和 Min 记录下来,并通过 Max-Min 作为基数 (即 Min=0, Max=1) 进行数据的归一化处理

$$x_{normalization} = \frac{x - Min}{Max - Min}$$

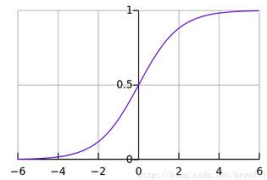
2. Z-score 标准化:

这种方法给予原始数据的均值 (mean) 和标准差 (standard deviation) 进行数据的标准化。经过处理的数据符合标准正态分布,即均值为 0,标准差为 1,这里的关键在于复合标准正态分布,个人认为在一定程度上改变了特征的分布,关于使用经验上欢迎讨论,我对这种标准化不是非常地熟悉,转化函数为:

$$x_{normalization} = \frac{x - \mu}{\sigma}$$

3. Sigmoid 函数

Sigmoid 函数是一个具有 S 形曲线的函数，是良好的阈值函数，在(0, 0.5)处中心对称，在(0, 0.5)附近有比较大的斜率，而当数据趋向于正无穷和负无穷的时候，映射出来的值就会无限趋向于 1 和 0，是个人非常喜欢的“归一化方法”，之所以打引号是因为我觉得 Sigmoid 函数在阈值分割上也有很不错的表现，根据公式的改变，就可以改变分割阈值，这里作为归一化方法，我们只考虑(0, 0.5)作为分割阈值的点的情况：



$$x_{normalization} = \frac{1}{1 + e^{-x}}$$

6. Evaluating Object Detection

	<u>Predicted 1</u>	<u>Predicted 0</u>
<u>True 1</u>	true positive	false negative
<u>True 0</u>	false positive	true negative

$$Precision = \frac{TP}{TP + FP} \quad Recall = \frac{TP}{TP + FN}$$

7. A Simple Sliding Window Detector



计算 HOG 分数
滑动窗口缩放

8. The Deformable Parts Model (DPM)

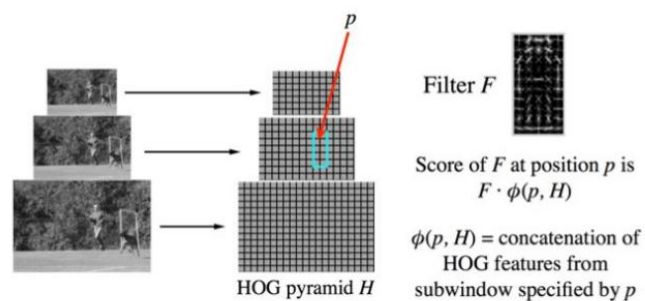
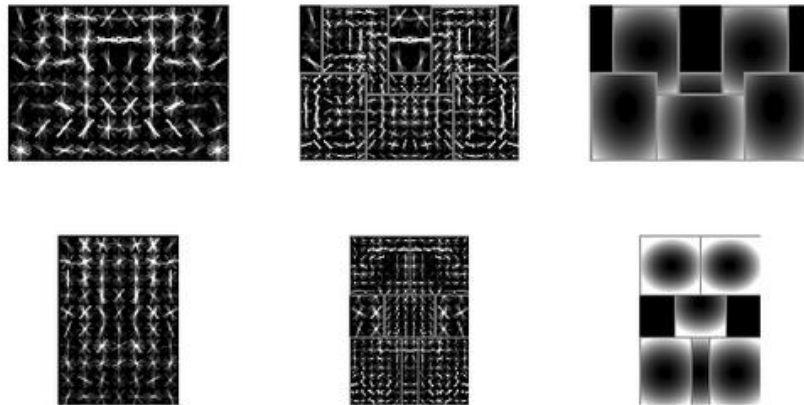


Figure 13: Using a feature pyramid of different image resizings allows the object template to match with objects that might have originally been bigger or much smaller than the template. Image source: Lecture 15, Slide 40

8.1 基本原理



Deformable Parts Model (DPM) 是一种很成功的物体检测方法。可以说这种方法是从HOG 继承而来的。传统的HOG 特征只采用一个模板表示某种物体，而DPM 把物体的模板划分成根模型和部分模型，其中的根模型等效于传统的HOG 特征，部分模型则是物体某些部分的模板。在检测的时候，根模型用来对物体可能存在的位置进行定位，部分模型用来进行进一步的确认。付出了更多的运算量使得DPM 的检测效果要优于传统的HOG。

8.2 算法实现

8.2.1 HOG

参考网址:<http://blog.csdn.net/zxpddfg/article/details/42263553>

8.2.2 DPM 模型

一个Deformable Parts Model (DPM) 模型可以表示成 $M = (F_0, P_1, P_2, \dots, P_n, b)$, 其中 F_0 是根滤波器, 或者叫根模型, 其尺寸为 $w_0 \times h_0$, P_i 是第 i 个部分模型, b 是偏置量. $P_i = (F_i, v_i, d_i)$ 由三部分组成, F_i 是第 i 部分的滤波器, 它的分辨率是根滤波器的 2 倍, 尺寸为 $w_i \times h_i$, $v_i = (v_{i,x}, v_{i,y})$ 表示 F_i 相对于 F_0 的偏移量, 以 F_0 的左上角作为原点, 坐标取 F_0 分辨率的两倍, 那么 v_i 就是 F_i 的左上角在这个坐标系下的坐标, $d_i = (d_{i,x}, d_{i,y}, d_{i,x^2}, d_{i,y^2})$ 表示形变系数, 或者叫代价系数, 惩罚系数. 如图 6 所示的是 DPM 模型结构.

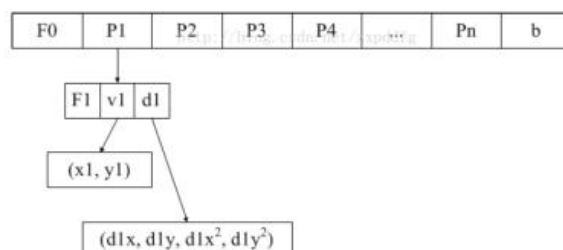


图 6: DPM 模型结构

这么理解根模型和部分模型的作用: 根模型用来对物体的位置进行大致的定位, 部分模型用来确认物体是否真实存在. 由于部分模型的分辨率是根模型的两倍, 所以部分模型具有比根模型更强的细节辨识能力, 这也就保证了确认工作的准确性. 另外, 在进行检测时, 部分模型的位置是可以移动的, 因为实际的物体的各个部分的位置可能和训练出的模型的位置不完全一致. 为了使各个部分模型和图片特征进行运算时取得最大值, 允许部分模型移动是合理的. 但是, 如果部分模型在一个偏离很远的地方和图片特征的响应值很大, 那么这个位置很可能就不是物体真实部分的位置, 为了解决这一问题, 引入了形变系数, 如果部分模型的位置和规定的位置发生偏移, 那么就要根据偏移量对运算结果进行惩罚. DPM 的设计思想将在后面介绍检测时得到更具体的体现.

DPM 模型的尺寸是固定的, 但是图片中物体的尺寸是变化的. 为了检测不同尺寸的物体, 我们需要进行多尺度分析. 我们把图片锁定纵横比进行伸缩, 得到不同尺度的图片, 然后进行 HOG 特征的计算.

给定一个原始图片的尺度, 对图片进行缩小, 对一系列图片计算 HOG 特征, 得到 HOG 特征金字塔. 原图位于金字塔第 0 层, 尺寸为 $W_0 \times H_0$, 随后的第 i 层的宽和高分别为

$$W_i = W_0 \times 2^{(-i/\lambda)}$$

$$H_i = H_0 \times 2^{(-i/\lambda)}$$

令 F 表示一个尺寸为 $w \times h$ 的滤波器. 令 H 表示一个 HOG 特征金字塔, $p = (x, y, l)$ 表示金字塔第 l 层坐标为 (x, y) 的位置. 令 $\phi(H, p, w, h)$ 表示 H 中第 l 层左上角坐标为 (x, y) , 尺寸为 $w \times h$ 的矩形区域中的 HOG 特征. 那么位置 p 的 score 等于 $F \cdot \phi(H, p, w, h)$. 因为 F 已经指定了矩形区域的大小, 所以 score 的表达式可以简化成 $F \cdot \phi(H, p)$.

一个物体假设 (Object Hypothesis) 指定每个根模型和部分模型的位置 $z = (p_0, p_1, \dots, p_n)$, 其中 $p_i = (x_i, y_i, l_i)$ 指定了第 i 个滤波器放置的位置, l_i 是 HOG 金字塔的层编号, (x_i, y_i) 是滤波器 F_i 左上角的位置. 部分模型的分辨率是根模型的 2 倍, 对于任意的 $i > 0$, $l_i = l_0 - \lambda$. 位置 z 的得分是这几个部分的和: 第一, 根滤波器和 HOG 特征的内积, 第二, 每个部分滤波器和 HOG 特征的内积, 减去部分滤波器位置偏移造成的代价, 第三部分是偏置量. 需要注意的是, 虽然 DPM 模型中规定了部分模型的相对于根模型的偏移量, 但是不规定部分模型的位置在检测时一定要处于这些规定的位置, 而是允许部分模型任意移动, 部分模型位置偏移后, 要结合形变系数对内积运算得到的结果进行惩罚. 允许部分模型移动, 这正是该模型命名为 Deformable 的原因.

$$\text{score}(p_0, p_1, p_2, \dots, p_n) = \beta \cdot \phi(H, z),$$

$$\beta = (F_0, F_1, \dots, F_n, d_1, d_2, \dots, d_n, b),$$

$$\psi(H, z) = (\phi(H, p_0), \phi(H, p_1), \dots, \phi(H, p_n),$$

$$-\phi_d(\Delta x_1, \Delta y_1), -\phi_d(\Delta x_2, \Delta y_2), \dots, -\phi_d(\Delta x_n, \Delta y_n), 1)$$

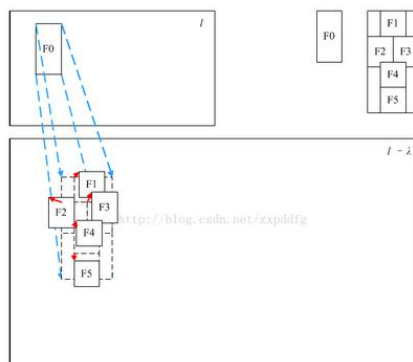


图 9: Object Hypothesis 示意图

给定根模型的位置 $p_0 = (x_0, y_0, l_0)$ 后, 通过在 $l_0 - \lambda$ 层调节部分模型的位置, 总能够找到部分模型的位置组合 (p_1, p_2, \dots, p_n) , 使得 $\text{score}(p_0, p_1, p_2, \dots, p_n)$ 达到最大, 记这个最大值为 $\text{score}(p_0)$. 由于各个部分模型相互独立, 各个部分模型的运算结果都达到最大时, $\text{score}(p_0, p_1, p_2, \dots, p_n)$ 达到最大.

<http://blog.csdn.net/zxpddfg>

令

$$R_{i,l}(x, y) = F_i \cdot \phi(H, (x, y, l))$$

表示第 i 个部分滤波器和第 l 层 HOG 金字塔进行内积运算的结果. 设 HOG 金字塔中第 i 层的尺寸是 $W_i \times H_i$, DPM 中 F_i 的尺寸是 $w_i \times h_i$, 在进行物体检测时, 不考虑物体只有一部分在 HOG

特征中的情况, 那么在内积运算时, 滤波器不会跨越 HOG 特征的边界. 所以 $R_{i,l}(x, y)$ 尺寸是 $(W_l - w_i + 1) \times (H_l - h_i + 1)$.

给定根模型的位置 $p_0 = (x_0, y_0, l_0)$ 后, 第 i 个部分模型的规定位置是 $p_i = (x_i, y_i, l_i) = (2(x_0, y_0) + v_i, l_i) = (2x_0 + v_{i,x}, 2y_0 + v_{i,y}, l_0 - \lambda)$. 如果第 i 个部分模型在第 $l_i = l_0 - \lambda$ 层 HOG 相对规定位置发生偏移 $(\Delta x, \Delta y)$, 到达 $(x_i + \Delta x, y_i + \Delta y, l_0 - \lambda)$, 那个该部分模型对最终得分的贡献是

$$T_{i,l_0-\lambda}(x_i + \Delta x, y_i + \Delta y) = R_{i,l_0-\lambda}(x_i + \Delta x, y_i + \Delta y) - d_i \cdot \phi_d(\Delta x, \Delta y)$$

令

$$D_{i,l_0-\lambda}(x_i, y_i) = \max_{\Delta x, \Delta y} T_{i,l_0-\lambda}(x_i + \Delta x, y_i + \Delta y)$$

这个值就是给定了根模型的位置 $p_0 = (x_0, y_0, l_0)$ 后, 第 i 个部分模型能够得到的最大值. 求取这个最大值是采用距离变换 (Distance Transform) 实现的. 把 R_{0,l_0} 值, n 个部分模型的 $D_{i,l_0-\lambda}(x_i, y_i)$ 值和 b 值相加得到 $\text{score}(p_0)$.

Tip: 参考网址

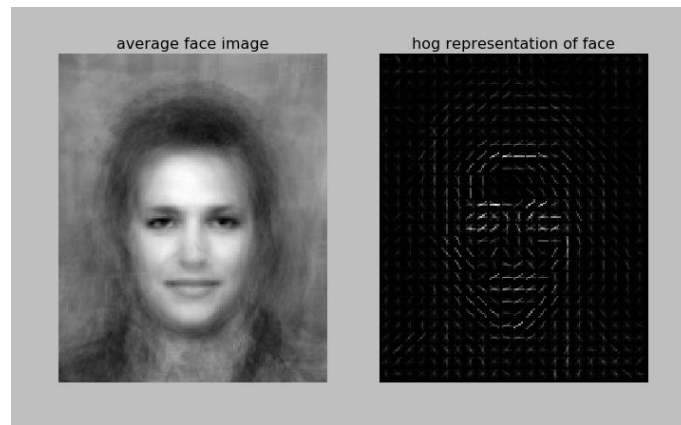
<http://blog.csdn.net/zxpddfg/article/details/42263553>

9. Hw7(作业题)

In this homework, we will implement a simplified version of object detection process.

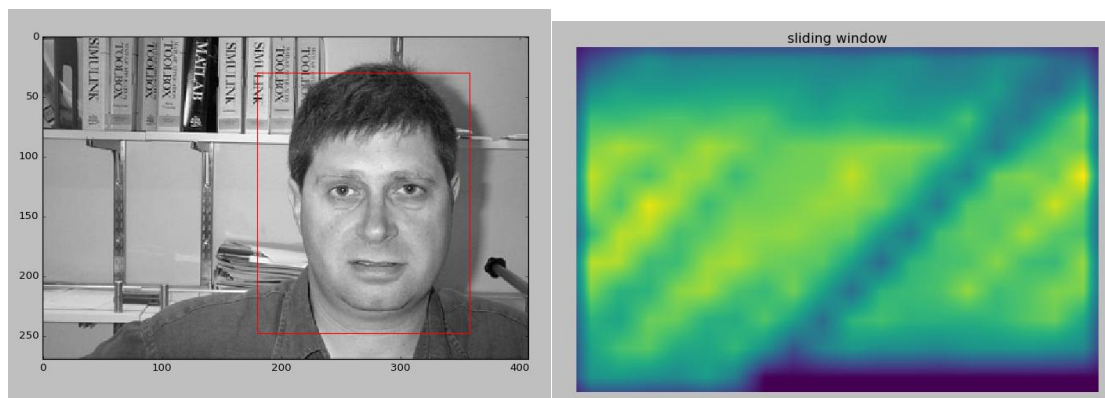
9.1 Hog Representation

In this section, we will compute the average hog representation of human faces. There are 31 aligned face images provided in the \face folder. They are all aligned and have the same size. We will get an average face from these images and compute a hog feature representation for the averaged face.

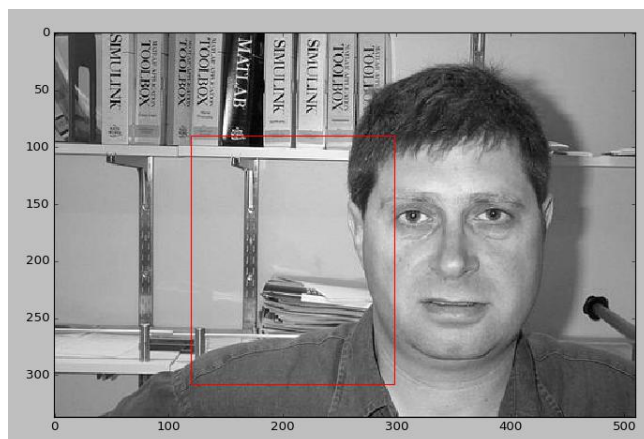


9.2 Sliding Window

Implement `sliding_window` function to have windows slide across an image with a specific window size. The window slides through the image and check if an object is detected with a high score at every location. These scores will generate a response map and you will be able to find the location of the window with the highest hog score.



Sliding window successfully found the human face in the above example. However, in the cell below, we are only changing the scale of the image, and you can see that sliding window does not work once the scale of the image is changed.



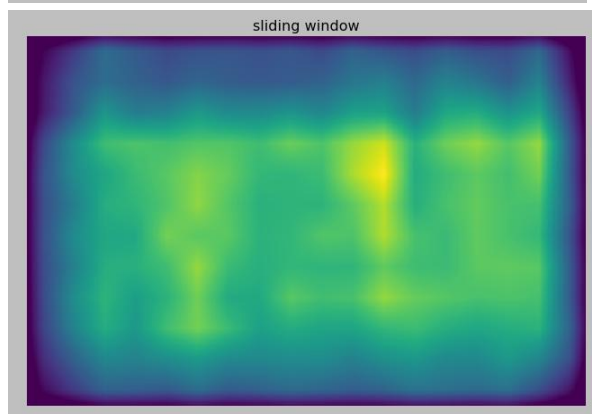
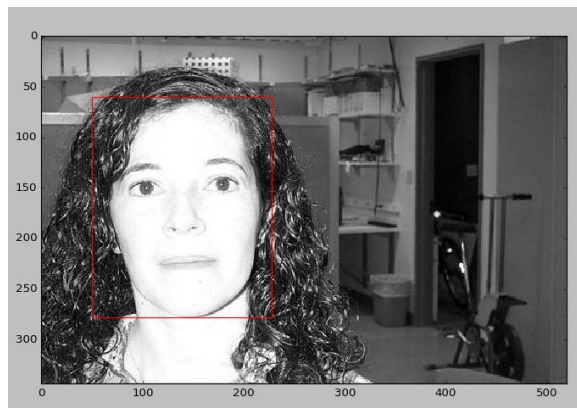
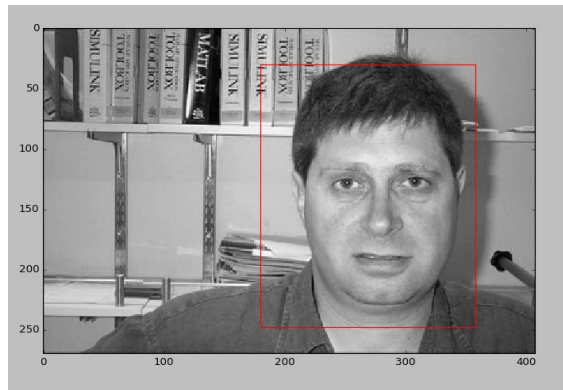
9.3 Image Pyramid

Implement `pyramid` function in `detection.py`, this will create pyramid of images at different scales. Run the following code, and you will see the shape of the original image gets smaller until it reaches a minimum size.



9.4 Pyramid Score

After getting the image pyramid, we will run sliding window on all the images to find a place that gets the highest score. Implement `pyramid_score` function in `detection.py`. It will return the highest score and its related information in the image pyramids.

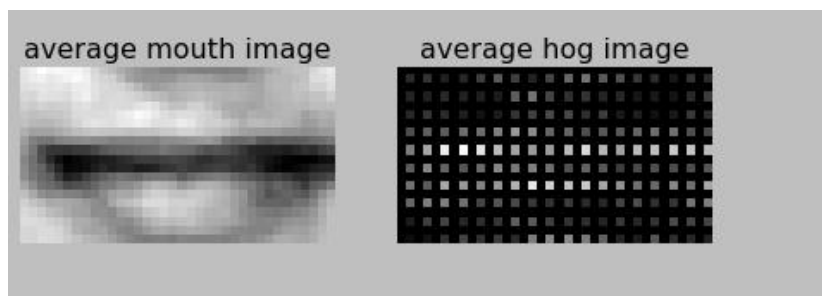
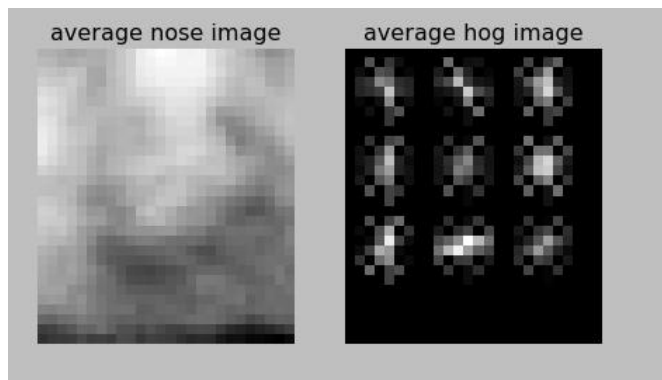
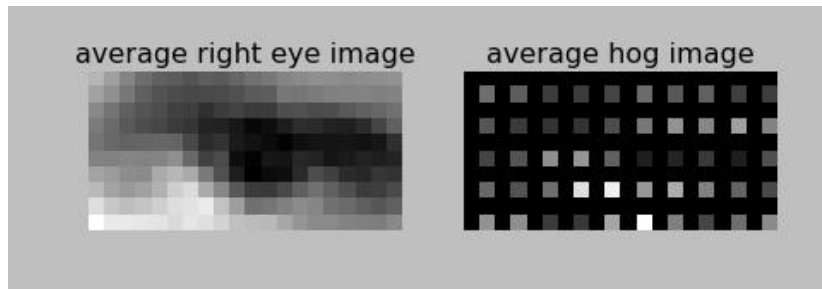
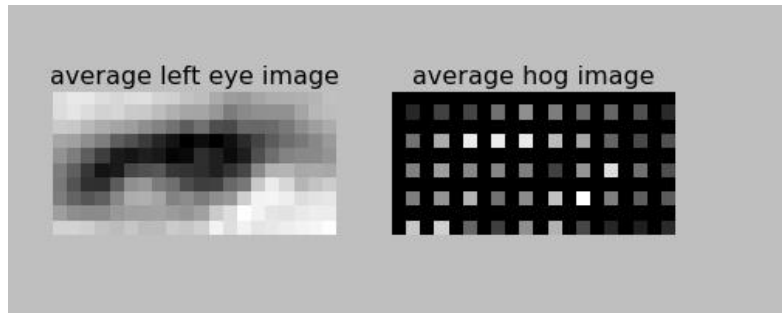


9.5 Deformable Parts Detection

In order to solve the problem above, you will implement deformable parts model in this section, and apply it on human faces.

The first step is to get a detector for each part of the face, including left eye, right eye, nose and mouth.

For example for the left eye, we have provided the groundtruth location of left eyes for each image in the `\face` directory. This is stored in the `lefteyes` array with shape $(n, 2)$, each row is the (r, c) location of the center of left eye. You will then find the average hog representation of the left eyes in the images.



9.6 Human Parts Location

Implement `compute_displacement` to get an average shift vector μ and standard deviation σ for each part of the face. The vector μ is the distance from the main center, i.e the center of the face, to the center of the part.


```

391 # test for compute_displacement
392 test_array = np.array([[0, 1], [1, 2], [2, 3], [3, 4]])
393 test_shape = (6, 6)
394 mu, std = compute_displacement(test_array, test_shape)
395 assert(np.all(mu == [1, 0]))
396 assert(np.sum(std - [1.11803399, 1.11803399]) < 1e-5)
397 print("Your implementation is correct!")
398
399

```

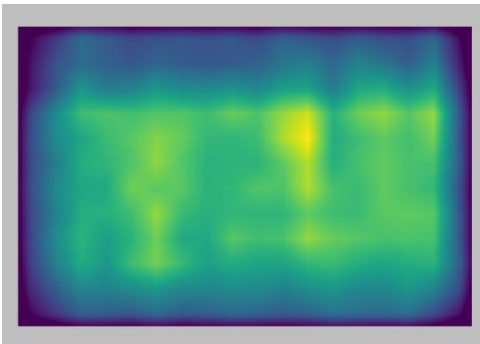
Your implementation is correct!
[Finished in 1.3s]

After getting the shift vectors, we can run our detector on a test image. We will first run the following code to detect each part of left eye, right eye, nose and mouth in the image. You will see a response map for each of them.

```

(face_H, face_W) = face_shape
max_score, face_r, face_c, face_scale, face_response_map = pyramid_score(
    image, face_feature, face_shape, stepSize=30, scale=0.8)

```

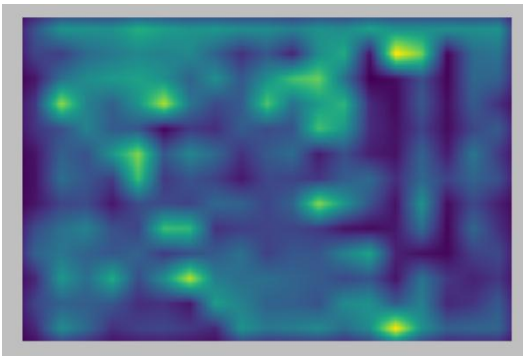


```

max_score, lefteye_r, lefteye_c, lefteye_scale, lefteye_response_map = \
    pyramid_score(image, lefteye_feature, lefteye_shape,
        stepSize=20, scale=0.9, pixel_per_cell=2)

lefteye_response_map = resize(lefteye_response_map, face_response_map.shape)

```

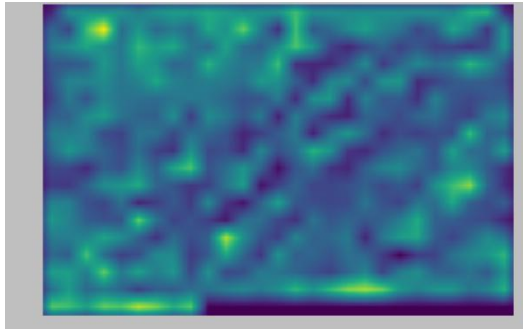


```

max_score, righteye_r, righteye_c, righteye_scale, righteye_response_map = \
    pyramid_score(image, righteye_feature, righteye_shape,
        stepSize=20, scale=0.9, pixel_per_cell=2)

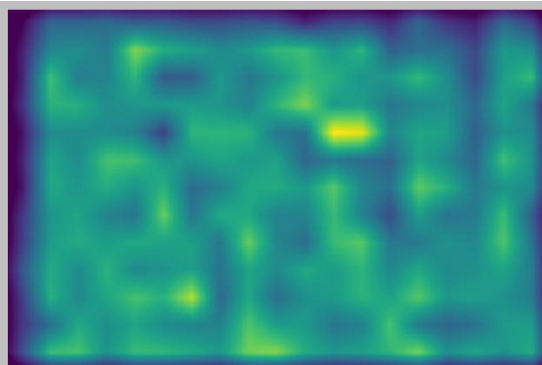
righteye_response_map = resize(righteye_response_map, face_response_map.shape)

```



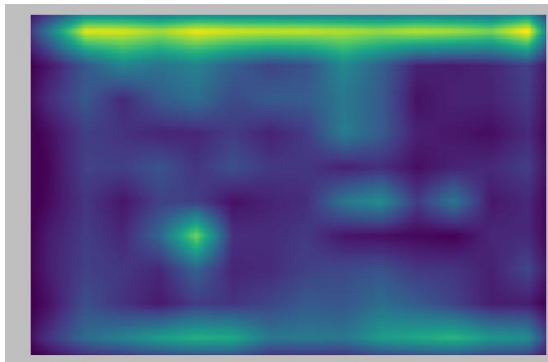
```
max_score, nose_r, nose_c, nose_scale, nose_response_map = \
    pyramid_score(image, nose_feature, nose_shape,
                  stepSize=20, scale=0.9, pixel_per_cell=2)

nose_response_map = resize(nose_response_map, face_response_map.shape)
```



```
max_score, mouth_r, mouth_c, mouth_scale, mouth_response_map = \
    pyramid_score(image, mouth_feature, mouth_shape,
                  stepSize=20, scale=0.9, pixel_per_cell=2)

mouth_response_map = resize(mouth_response_map, face_response_map.shape)
```



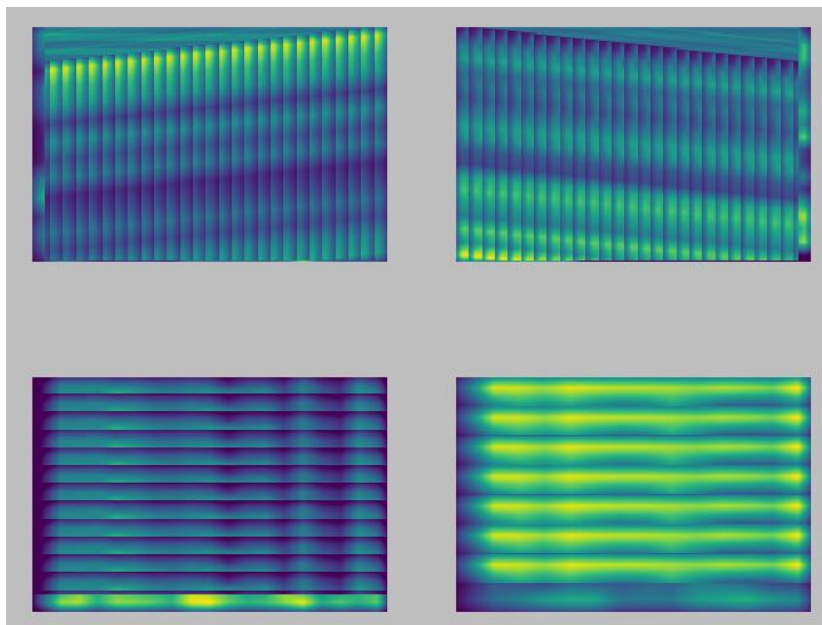
After getting the response maps for each part of the face, we will shift these maps so that they all have the same center as the face. We have calculated the shift vector μ in `compute_displacement`, so we are shifting based on vector μ . Implement `shift_heatmap` function in `detection.py`.

```

face_heatmap_shifted = shift_heatmap(face_response_map, [0, 0])
lefteye_heatmap_shifted = shift_heatmap(lefteye_response_map, lefteye_mu)
righteye_heatmap_shifted = shift_heatmap(righteye_response_map, righteye_mu)
nose_heatmap_shifted = shift_heatmap(nose_response_map, nose_mu)
mouth_heatmap_shifted = shift_heatmap(mouth_response_map, mouth_mu)

f, axarr = plt.subplots(2, 2)
axarr[0, 0].axis('off')
axarr[0, 1].axis('off')
axarr[1, 0].axis('off')
axarr[1, 1].axis('off')
axarr[0, 0].imshow(lefteye_heatmap_shifted,
                    cmap='viridis', interpolation='nearest')
axarr[0, 1].imshow(righteye_heatmap_shifted,
                    cmap='viridis', interpolation='nearest')
axarr[1, 0].imshow(nose_heatmap_shifted, cmap='viridis',
                    interpolation='nearest')
axarr[1, 1].imshow(mouth_heatmap_shifted, cmap='viridis',
                    interpolation='nearest')
plt.show()

```



9.7 Gaussian Filter

Part 6.1 Gaussian Filter

In this part, apply gaussian filter convolution to each heatmap. Blur by kernel of standard deviation sigma, and then add the heatmaps of the parts with the heatmap of the face. On the combined heatmap, find the maximum value and its location. You can use function provided by skimage to implement `gaussian_heatmap`.


```
def gaussian_heatmap(heatmap_face, heatmaps, sigmas):
    """
    Apply gaussian filter with the given sigmas to the corresponding heatmap.
    Then add the filtered heatmaps together with the face heatmap.
    Find the index where the maximum value in the heatmap is found.

    Hint: use gaussian function provided by skimage

    Args:
        image: np array of (h,w)
        sigma: sigma for the gaussian filter
    Return:
        new_image: an image np array of (h,w) after gaussian convoluted
    """
    # YOUR CODE HERE
    O, P = heatmap_face.shape
    heatmap = np.zeros((O, P))
    for i in range(4):
        heatmap += gaussian(heatmaps[i], sigmas[i])
    heatmap += heatmap_face
    H, W = heatmap.shape
    num = np.argmax(heatmap)
    r = (num // W).astype(int)
    c = num % W
    # END YOUR CODE
    return heatmap, r, c

heatmap_face = face_heatmap_shifted

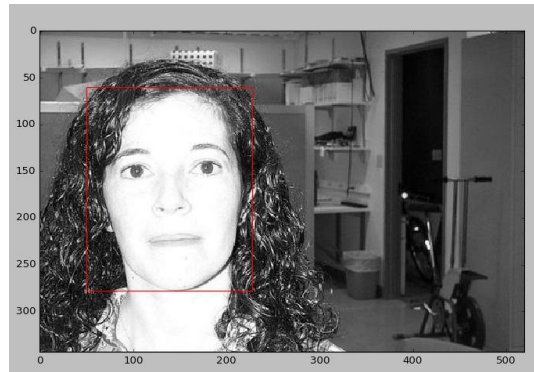
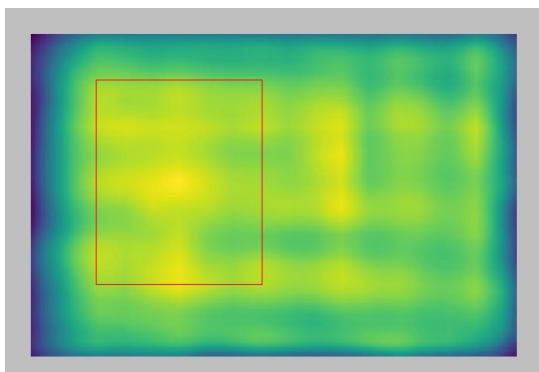
heatmaps = [lefteye_heatmap_shifted,
            righteye_heatmap_shifted,
            nose_heatmap_shifted,
            mouth_heatmap_shifted]
sigmas = [lefteye_std, righteye_std, nose_std, mouth_std]

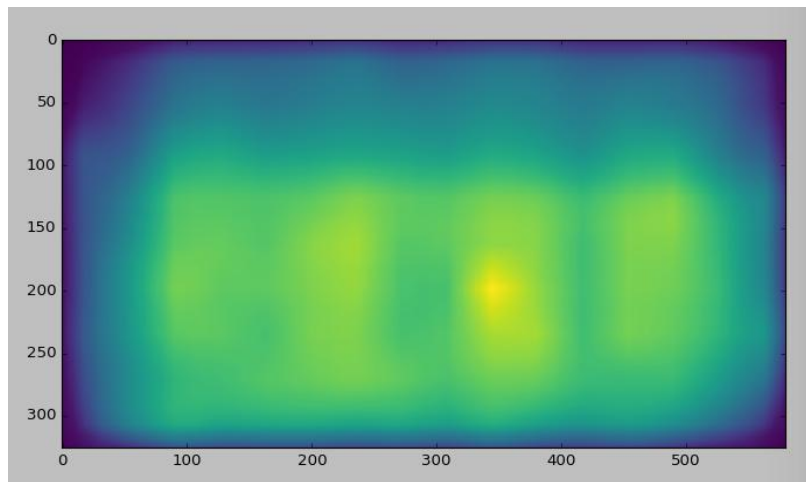
heatmap, i, j = gaussian_heatmap(heatmap_face, heatmaps, sigmas)

fig, ax = plt.subplots(1)
rect = patches.Rectangle((j - winW // 2, i - winH // 2),
                        winW, winH, linewidth=1, edgecolor='r', facecolor='none')
ax.add_patch(rect)

plt.imshow(heatmap, cmap='viridis', interpolation='nearest')
plt.axis('off')
plt.show()

fig, ax = plt.subplots(1)
rect = patches.Rectangle((j - winW // 2, i - winH // 2),
                        winW, winH, linewidth=1, edgecolor='r', facecolor='none')
ax.add_patch(rect)
```





Tips:skimage.filters 的 Gaussian 函数

gaussian

```
skimage.filters.gaussian(image, sigma=1, output=None, mode='nearest', cval=0,
                        multichannel=None, preserve_range=False, truncate=4.0)
```

[\[source\]](#)

Multi-dimensional Gaussian filter.

Parameters:

image : array-like

Input image (grayscale or color) to filter.

sigma : scalar or sequence of scalars, optional

Standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.

output : array, optional

The **output** parameter passes an array in which to store the filter output.

mode : {'reflect', 'constant', 'nearest', 'mirror', 'wrap'}, optional

The mode parameter determines how the array borders are handled, where cval is the value when mode is equal to 'constant'. Default is 'nearest'.

cval : scalar, optional

Value to fill past edges of input if mode is 'constant'. Default is 0.0

multichannel : bool, optional (default: None)

Whether the last axis of the image is to be interpreted as multiple channels. If True, each channel is filtered separately (channels are not mixed together). Only 3 channels are supported. If None, the function will attempt to guess this, and raise a warning if ambiguous, when the array has shape (M, N, 3).

preserve_range : bool, optional

Whether to keep the original range of values. Otherwise, the input image is converted according to the conventions of `img_as_float`.

truncate : float, optional

Truncate the filter at this many standard deviations.

Returns:

filtered_image : ndarray

the filtered array