论文简述

Local features and kernels for classification of texture and object categories: a comprehensive study

原理简述

该论文提出了**一种基于支持向量机的物体识别方法**，并对该方法做了大量详细而全面的评估。评估的过程主要是通过改变以及组合该方法的"组件"，包括使用多种关键点检测器（keypoint detector，如Harris-Laplace detector 以及 Laplacian detector），不同级别的几何不变性（geometric invariance，如尺度不变性S、尺度旋转不变性SR、仿射不变性A），不同的特征描述符（如SPIN、RIFT、SIFT），不同的向量机核（如常用的线性核、多项式核、RBF核、文章提出的χ^2 核和EMD核）。通过在多个纹理分析以及物体检测的数据集上做分析，论文得出的结论是使用不同的组件进行组合有助于综合利用它们提取的信息，进而提高分类的效果。同时，论文还讨论了**背景图案对分类效果的影响**，它们的实验表明使用基于局部特征对图片进行表示有助于消除背景图案对识别效果的影响，使得识别方法鲁棒性更强。

算法流程

论文提出的算法具体可分为以下四步：

- 检测显著性局部区域（detect salient local regions），如上所述，这里可以使用不同的关键点检测器来实现检测。
- 计算局部区域的描述符（compute descriptor），这里可以使用不同的特征描述符。
- 比较局部特征分布，可以使用Earth Mover's Distance（EMD）距离或者χ^2 距离。
- 使用基于核的支持向量机方法进行分类。

SVM的决策函数可表示为：

$$g(x)=\sum_i \alpha_i y_i K(x_i,x)-b$$

式子中$K(x_i,x)$表示核函数，为了将EMD或者是χ^2距离与SVM方法结合起来，论文对高斯核做了修改：

$$K(S_i,S_j)=\exp\left(-\frac{1}{A}D(S_i,S_j)\right)$$

其中，$D(S_i,S_j)$表示EMD距离或者χ^2距离，两种距离的计算方式有所不同，这里不详细展开；A是一个超参数，可用交叉验证法得到。

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

背景知识

这里先对论文所涉及的重要背景知识做个简单的介绍，方便论文的理解。

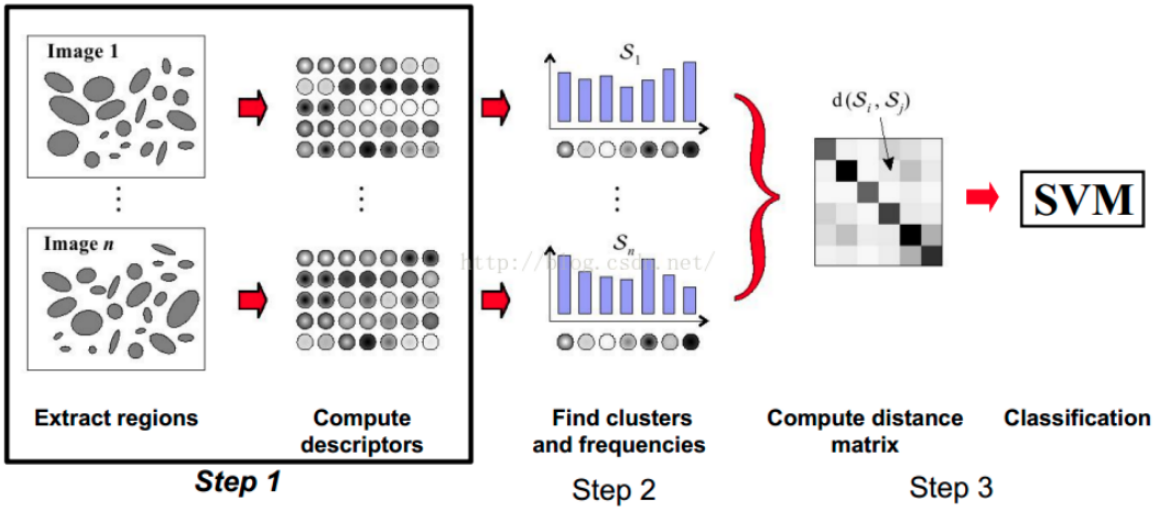
词袋模型

严格上来说，词袋模型是一种文本表示模型，它可以将一篇文档表示成一个长向量，该向量的每一维度表示一个单词，每个维度上的权重表示该单词在文档中的重要程度，这里的重要程度可以用词频或者是tf-idf值来衡量。词袋模型对文本进行向量化表示之后，就可以将其交给下流的任务了，如进行文本分类、信息检索等等。

同样的原理，词袋模型也可以用来对图片进行向量化表示。使用词袋模型对图片进行分类可以分为以下三步：

1. 计算图像的特征描述符，其实提取的是图像的**浅层特征**；
2. 对所有图像中计算的描述符进行聚类（**字典学习**），然后统计图片在每个簇中的特征数量，将图片表示成向量；
3. 使用分类器如SVM、逻辑回归进行分类。

以上流程如下图所示：



词袋模型用于图片分类的效果不错，但有一个缺点是，它丢弃了特征的空间信息，这个缺点严重限制了词袋模型的表达能力，因此这篇论文的动机就是对词袋模型做改进，以提升它的分类效果。

Pyramid Match算法

Pyramid Match是用于计算两个特征集合X和Y的相似性的算法，Paramid Match的流程如下，假设存在两个d维特征空间中的集合X和Y，Paramid Match首先将特征空间划分为不同的尺度$0, \dots, L$，那么在每一个尺度l下面，每一维度可以划分为2^l个区域（cell），所以d维特征空间就能划分出$D = 2^{dl}$个区域。令$H_X^l(i)$以及$H_Y^l(i)$分别表示X和Y特征集合中落入第i个区域的特征点的数量，那么尺度l下集合X和Y的匹配数量可以用以下式子计算：

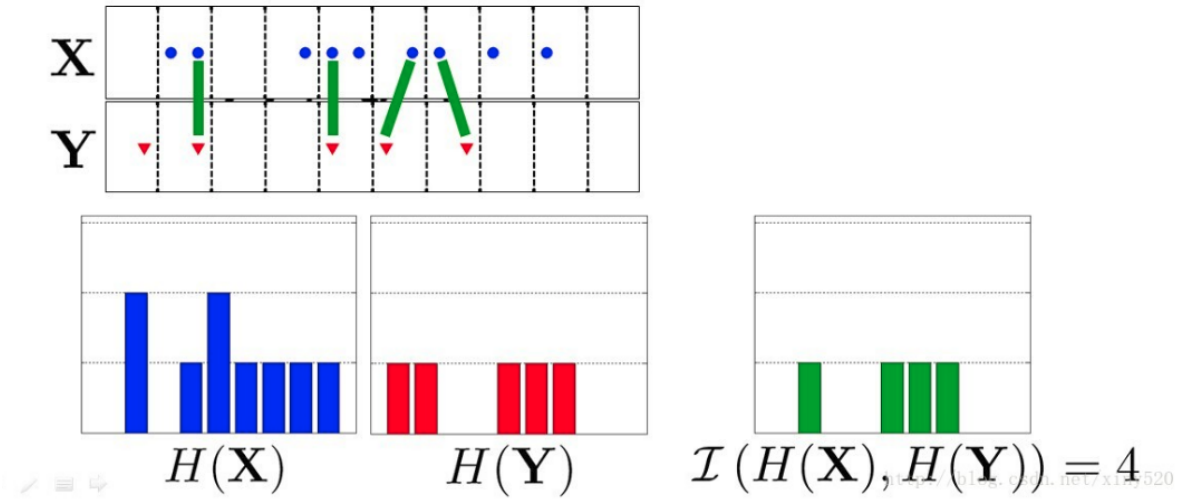
$$\mathcal{I}\left(H_X^\ell, H_Y^\ell\right) = \sum_{i=1}^{D} \min\left(H_X^\ell(i), H_Y^\ell(i)\right)$$

Paramid Match给尺度l所赋予的权重是$\frac{1}{2^{L-\ell}}$，这样就能做到给大尺度下的匹配赋予较高的权重，给小尺度下的匹配赋予较低的权重。最终，两点集X和Y的匹配程度可以定义为：

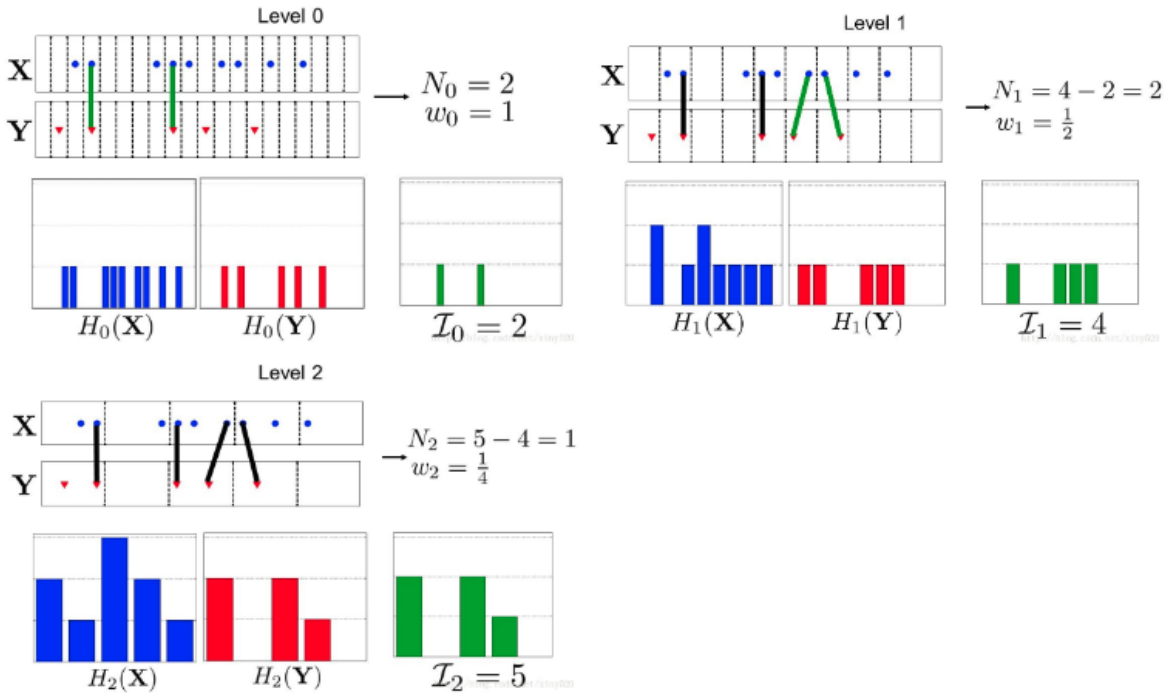
$$\begin{aligned}\kappa^L(X,Y) &= \mathcal{I}^L + \sum_{\ell=0}^{L-1} \frac{1}{2^{L-\ell}}\left(\mathcal{I}^\ell - \mathcal{I}^{\ell+1}\right) \\ &= \frac{1}{2^L}\mathcal{I}^0 + \sum_{\ell=1}^{L} \frac{1}{2^{L-\ell+1}}\mathcal{I}^\ell\end{aligned}$$

可以用一个例子来理解上述过程。假设X和Y是两个一维的点集，手动计算它们的最优匹配数为：

Histogram intersection $\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$



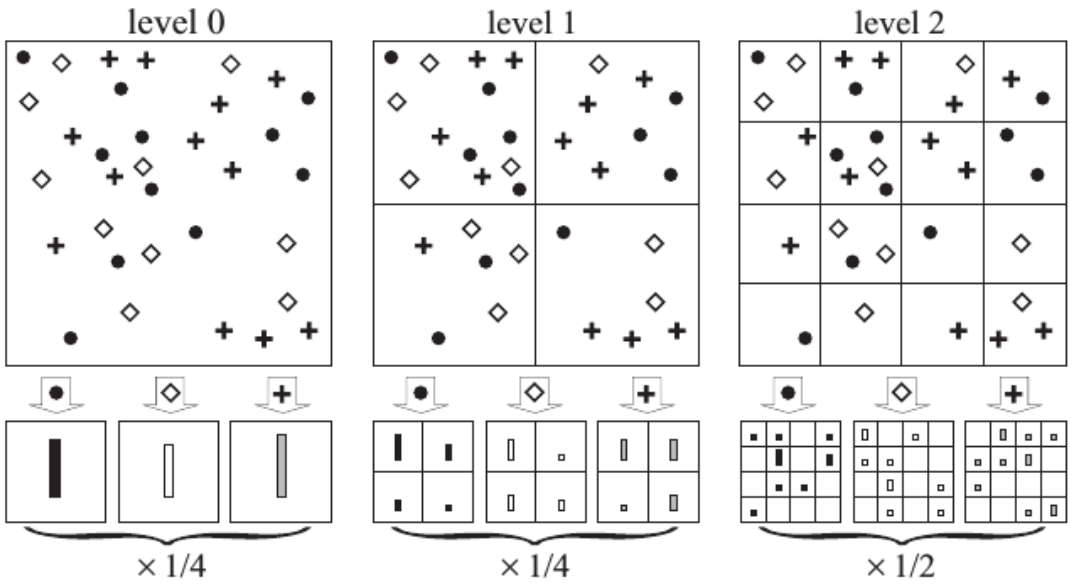
使用Pyramid Match算法可以实现上述最优匹配的近似结果：



其近似结果为$1 \times 2 + \frac{1}{2} \times 4 + \frac{1}{4} \times 5$，与最优匹配数4近似。

原理简述

论文提出的Spatial Pyramid Matching（下面简称SPM）借鉴了Pyramid Match的思想，只是这里不再是对图像特征的每一维进行划分，而是对特征在图像中的坐标进行划分，进而实现对特征的空间信息进行保留。论文中举的例子如下：



这个例子中假设我们从图片中抽取出来的特征总共有三类，分别用圆形、菱形、加号三种符号表示，然后在三个不同的尺度下对图片进行划分，即对应上图的三个level，然后和词袋模型类似，统计每个尺度下三个特征出现的次数，乘以该尺度下对应的权重，就可以作为该尺度下的特征表示。需要注意，不同的尺度对应着该尺度下的权重，对图片划分越细，权重就越大。

那么容易理解，在ℓ个不同的尺度，M个特征类型下，SPM算法产生的特征向量的维数是：

$$M\sum_{\ell=0}^{L}4^{\ell}=M\frac{1}{3}\left(4^{L+1}-1\right)$$

算法流程

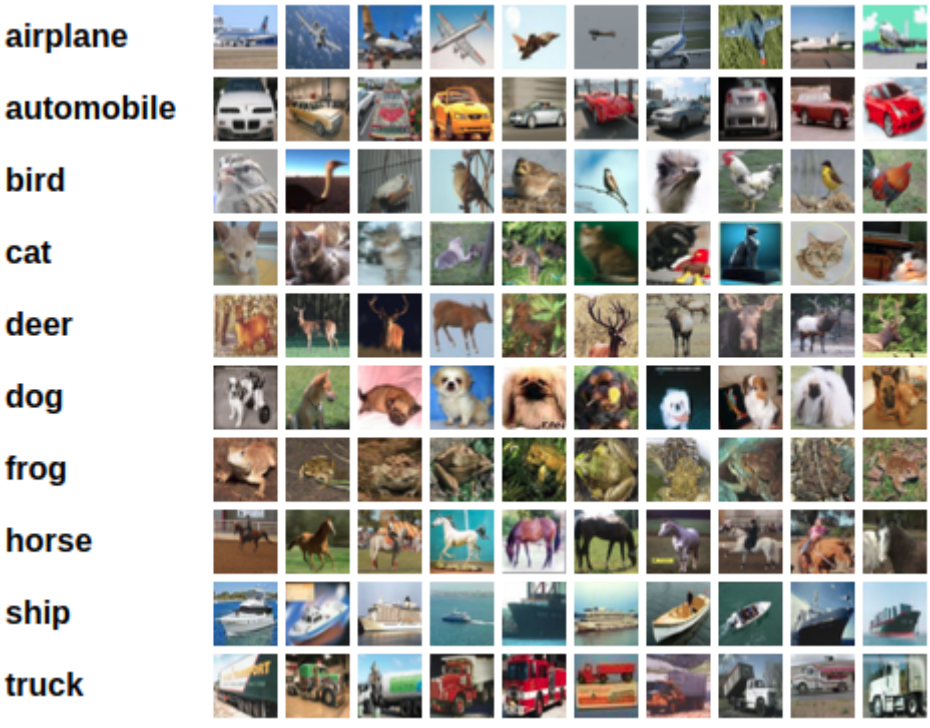
总的来说，算法的流程与使用词袋模型的算法大致相同。可以将整个流程分为以下四步：

- 特征提取
- 特征聚类，论文中使用的聚类算法是K-means；
- 使用SPM算法对图片进行表示，如前一部分所述；
- 使用分类器如SVM进行分类。

需要特别说明的是，论文中将提取的特征分为两大类，一类是“弱特征（weak features）”，即定向边缘点（oriented edge points），它们的特点是，在某个特点方向是的梯度大小超过最小的阈值。另一类是“强特征（strong features）”，“强特征”用SIFT描述符进行提取。

实现

我复现是Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories中提出的**SPM算法**。由于SPM算法是对BOW模型的改进，因此我同样也实现了BOW算法，为了比较，我将这两个算法应用于Cifar-10数据集，测试其分类效果。

> Cifar-10 由60000张32*32的 RGB 彩色图片构成，共10个分类。50000张用于训练，10000张用于测试。这个数据集最大的特点在于将识别迁移到了普适物体，而且应用于多分类。下面是Cifar-10数据集中的一些例子：
>
> 
>

下面对这两个算法的具体实现分别进行描述。

BOW实现

BOW的算法实现很简单，具体来说，可以分为四步：抽取特征、特征聚类、图片向量化、使用分类器进行分类。

代码如下所示：

```
# 抽取sift特征
print("抽取SIFT特征...")
x_train = [extract_sift_descriptors(img) for img in x_train]
x_test = [extract_sift_descriptors(img) for img in x_test]

# 使用Kmeans聚类算法对抽取出来的特征进行聚类，VOC_SIZE表示k-means中的k
print("使用Kmeans聚类算法对抽取出来的特征进行聚类...")
vocab = build_vocab(x_train, voc_size=VOC_SIZE)

# 将图片转成它的向量化表示
print("图片向量化...")
x_train = [input_vector_encoder(x, vocab) for x in x_train]
x_train = np.asarray(x_train)
x_test = [input_vector_encoder(each, vocab) for each in x_test]
x_test = np.asarray(x_test)

# 输入到分类器进行分类
print("分类器训练、测试中....")
svm_classifier(x_train, y_train, x_test, y_test)
```

SPM实现

SPM算法与BOW算法的流程大致相同，只不过**在图片向量化**这一步中，SPM借鉴了Pyramid Match的思想，对特征在图像中的坐标进行划分（具体可以查看上面的原理简述），从而保留了特征的空间信息。下面的代码就是这一步的具体实现：

```
def spatial_pyramid_matching(image, descriptor, codebook, level):
    pyramid = []
    if level == 0:
        pyramid += build_spatial_pyramid(image, descriptor, level=0)
        code = [input_vector_encoder(crop, codebook) for crop in pyramid]
        return np.asarray(code).flatten()
    if level == 1:
        pyramid += build_spatial_pyramid(image, descriptor, level=0)
        pyramid += build_spatial_pyramid(image, descriptor, level=1)
        code = [input_vector_encoder(crop, codebook) for crop in pyramid]
        code_level_0 = 0.5 * np.asarray(code[0]).flatten()
        code_level_1 = 0.5 * np.asarray(code[1:]).flatten()
        return np.concatenate((code_level_0, code_level_1))
    if level == 2:
        pyramid += build_spatial_pyramid(image, descriptor, level=0)
        pyramid += build_spatial_pyramid(image, descriptor, level=1)
        pyramid += build_spatial_pyramid(image, descriptor, level=2)
        code = [input_vector_encoder(crop, codebook) for crop in pyramid]
        code_level_0 = 0.25 * np.asarray(code[0]).flatten()
        code_level_1 = 0.25 * np.asarray(code[1:5]).flatten()
        code_level_2 = 0.5 * np.asarray(code[5:]).flatten()
        return np.concatenate((code_level_0, code_level_1, code_level_2)
```

其中`build_spatial_pyramid`会根据描述符的位置对其进行分类，假设level大小为ℓ，特征类型数量为M，那么`spatial_pyramid_matching`函数会返回一个维数为$M\sum_{\ell=0}^{L}4^{\ell}$的长向量作为图片的表示。

结果分析

BOW运行结果如下：

```
              precision    recall  f1-score   support

    airplane       0.48      0.10      0.17       993
  automobile       0.25      0.16      0.20      1000
        bird       0.41      0.02      0.03       998
         cat       0.00      0.00      0.00       999
        deer       0.00      0.00      0.00       999
         dog       0.17      0.53      0.25      1000
        frog       0.00      0.00      0.00      1000
       horse       0.15      0.66      0.24      1000
        ship       0.48      0.09      0.15       997
       truck       0.23      0.30      0.26      1000

   micro avg       0.19      0.19      0.19      9986
   macro avg       0.22      0.19      0.13      9986
weighted avg       0.22      0.19      0.13      9986
```

SPM算法运行结果如下：

```
分类结果如下:

              precision    recall  f1-score   support

    airplane       0.41      0.22      0.29      1000
  automobile       0.28      0.41      0.33      1000
        bird       0.27      0.31      0.29      1000
         cat       0.23      0.16      0.19      1000
        deer       0.23      0.32      0.27      1000
         dog       0.35      0.26      0.30      1000
        frog       0.31      0.24      0.27      1000
       horse       0.31      0.32      0.32      1000
        ship       0.48      0.29      0.36      1000
       truck       0.32      0.51      0.39      1000

   micro avg       0.30      0.30      0.30     10000
   macro avg       0.32      0.30      0.30     10000
weighted avg       0.32      0.30      0.30     10000
```

容易看到，SPM算法的效果要比BOW算法好很多，这说明论文的改进是有效的。

分类器与超参数的设置：这里的分类器使用的是sklearn中svc，超参数用的都是默认的值，字典的大小取100，SPM的Level为2。由于机器性能的限制，如果要进行超参数调优或者扩大字典的大小，都会显著增加实验时间，因此这里并没有进行超参数调优，并且将字典设置为一个比较小的数值（源论文字典的大小设置为400），这可能也是整体效果不够好的原因。

运行代码

最后提交完整目录如下：

```
.
├── code                                # 代码目录
│   ├── bow.py                          # bow算法实现
│   ├── classifier.py                   # 分类器
│   ├── requirement.txt                 # 依赖的第三方库
│   ├── spm.py                          # SPM算法实现
│   └── utils.py                        # 一些工具函数
├── imgs                                # 存放markdown文件使用的图片
├── papers                              # 两篇原论文
└── 实验报告.md                          # 本文件
└── 实验报告.pdf                         # Markdown导出的pdf文件(便于查看)
```

由于数据集比较大，因此我没有将它放到提交文件里面，而是将它上传到Github。如果想要运行代码，可以从Github上下载：

```
git clone https://github.com/luopeixiang/CV_Project.git
```

然后安装依赖：

```
cd CV_Project/code
pip install -r requirement.txt
```

运行bow算法：

```
python bow.py
```

运行spm算法：

```
python spm.py
```

运行结束时会打印出前一部分给出的分类结果，不过整个过程需要很长时间，请耐心等候。

参考链接

Spatial Pyramid Matching

Spatial Pyramid Matching 小结

libpmk

Image-Recognition

SPM