图像视频实验二

陈嘉杰 2017011484 计72

实验目标

给定若干个分类的不同图片，要求提取这些图片的颜色分布直方图，存下来后，按照特定的目标图片，查询出最接近的30张图片，如果在同一分类中，就计入精确度中。在颜色分布直方图的计算和相似度衡量上有不同的做法，不同的做法也得到了不同的结果。

实验过程

特征提取

特征提取在这里就是计算出颜色分布直方图并且保存下来，下面是相关的代码（见 `histo.py`）：

```
def histo(image):
    bins = [0] * bins_count
    h, w, _ = image.shape
    for i in range(h):
        for j in range(w):
            r, g, b = image[i][j]
            if bins_count == 16:
                # 2 : 4 : 2
                bins[(r // 128) + (g // 64) * 2 + (b // 128) * 8] += 1
            elif bins_count == 128:
                # 4 : 8 : 4
                bins[(r // 64) + (g // 32) * 4 + (b // 64) * 32] += 1
            else:
                exit()
    return np.array(bins, dtype=np.float) / h / w
```

对于切分为16段的直方图，每一个位置对应RGB三种颜色的一个范围，其中R切分为[0,128)和[128,256)，G切分为[0,64),[64,128),[128,192),[192,256)，B切分为[0,128),[128,256)。得到这些数据以后，最后除以像素个数，保存在目录 `histo-16` 和 `histo-128` 下。

由于这些数据对于同样的图片，结果都是一样的，所以我们做一次预处理之后就不需要再对原图片进行操作了，会加快整体的运算速度。

图片查询

在上一步中，已经得到了每一张图对应的向量，接下来就需要计算每一张图和目标图片之间的距离，我编写了以下的距离函数（见 query.py ）：

```
def l2(a, b):
    return np.linalg.norm(a - b, ord=2)

def hi1(a, b):
    return np.sum(np.minimum(a, b))

def hi2(a, b):
    return np.sum(np.minimum(a, b)) / np.sum(b)

def bh(a, b):
    return np.sqrt(1 - np.sum(np.sqrt(a * b)))

def l1(a, b):
    return np.linalg.norm(a - b, ord=1)

def l3(a, b):
    return np.linalg.norm(a - b, ord=3)

def li(a, b):
    return np.linalg.norm(a - b, ord=np.inf)

def ws(a, b):
    return wasserstein_distance(a, b)
```

其中前四个为课件上提供的距离，后面依次是L1距离，L3距离，L无穷距离和Wasserstein距离。

按照算法对每一张图片进行处理：

```
with open('QueryImages.txt', 'r') as f:
    for line in f:
        if len(line.strip()):
            file = line.split(' ')[0]
            cate = file.split('/')[0]
            current = np.load('%s/%s.histo.npy' % (root, file))
            file = file.split('.')[0]
            dists = []
            for k in data:
                dists.append((k, dist(current, data[k])))
            dists = list(sorted(dists, key=lambda kv: kv[1]))[1:31]
            correct = 0
```

```
            with open('ans-%d-%s/res_%s.txt' % (bins_count, dist.__name__,
    file.replace('/', '_')), 'w') as output:
                for k, v in dists:
                    if cate == k.split('/')[0]:
                        correct += 1
                    output.write('%s %.3f\n' % (k, v))
            precision.append(correct / 30.0)
            overalls.append('%s %.3f' % (file, correct / 30.0))
```

得到距离最小的30个以后（去掉自己），按照相应的格式输出结果。

实验结果

按照不同的组合，得到了下列总体准确度的结果：

准确度	16 bins	128 bins
L2	0.270	0.354
HI1	0.023	0.025
HI2	0.023	0.025
BH	0.360	0.468
L1	0.309	0.430
L3	0.268	0.331
Linf	0.265	0.311
Wasserstein	0.151	0.175

纵向来看，各个距离的效果 BH>L1>L2>L3>Linf>Wasserstein>HI 。可以看到，比较适合的距离量度在精确度上会比差的好很多。各种Ln范式之间效果差距也不大，但大概可以看出，n越小效果越好。

横向来看，128 bins 总是比 16 bins 会好，毕竟提取到了更精确的统计信息，只要数据集本身没问题，这种做法总是可以得到更好的效果的。

从具体的查询结果来看，其实数据集中，同一类的图片的相似度可能也不是很高。例如beach和mountains就经常得不到高的准确度，因为虽然都是沙滩和山，但是还有很多其他的东西（人、房子、船等等）在图中影响了颜色的分布；elepants经常被认成horses，毕竟背景都是草原。而且每一类里面图片的数量也不一样，有的类数量少，所以精确度可能会因此下降一些。

如果要提高精确度，可能最开始就不采用颜色分布直方图的特征，这个特征并不适合这种问题。另一方面，需要提高数据集本身的质量，保证数据集内图片之间有足够的相似性，这样聚类的时候才不容易跑飞。