# 计算机视觉作业-2: 利用聚类技术实现纹理图像分割

王天锐

20120318 信号 1 班

**摘　要：**本次实验利用灰度共生矩阵提取出图像的纹理特征，然后再使用 k-均值聚类算法根据纹理特征进行聚类达到对图像进行分割的目的。利用滑窗技术对原始图像进行窗划分；再对于每个窗口利用灰度共生矩阵的统计方式提取出像素点横向、纵向、斜向的灰度值分布关系；并根据灰度共生矩阵计算出相应的特征值；最后根据非监督聚类算法最近邻方法进行分类。所用语言为 python 语言，利用 numpy 矩阵运算包实现

## 1　方法原理及实现

### 1.1　滑窗和灰度级调整

本实验采用 numpy 实现对数组的滑窗采样，然后对整个数据进行相应的灰度级调整，整个操作的代码如下:

```
W, H = img.shape
new_W = (W - (win_len - 1)) // stride[0]
new_H = (H - (win_len - 1)) // stride[1]
strides = img.itemsize * np.array([W * stride[1], stride[0], W, 1])
win_wrap_img = np.lib.stride_tricks.as_strided(img, shape=(new_W,
                                    new_H, win_len, win_len),
strides=strides)
max_gray = np.max(win_wrap_img) + 1
win_wrap_img = np.array(win_wrap_img * gray_level / max_gray, dtype="
                                    int")
```

### 1.2　灰度共生矩阵

灰度共生矩阵 (Gray-level Co-occurrence Matrix，GLCM) 能够描述具有某种空间关系的两个像素灰度的联合分布概率。其计算公式为:

$$P(i,j|\Delta x,\Delta y)=\frac{\sum_{n=1}^{H-\Delta y}\sum_{m=1}^{W-\Delta x}A}{(W-\Delta x)(H-\Delta y)}$$

$$A=\begin{cases}1, if \quad f(m,n)=i \quad and \quad f(m+\Delta x,n+\Delta y)=j\\ 0, else\end{cases}$$

我们可以设置不同的 $\Delta x$ 和 $\Delta y$ 来提取到不同方向上像素之间的联合概率情况。本实验提取了 0°、45°、90° 和 135° 的特征，对应 $\Delta x$ 和 $\Delta y$ 分别为 (0,1),(1,1),(1,0),(-1,1), 整体代码实现为:

```
def glcm(input_win, step_x, step_y, gray_level):
    w, h = input_win.shape
    ret = np.zeros([gray_level, gray_level])
    count = 0
    for index_h in range(h):
        for index_w in range(w):
            try:
                row = int(input_win[index_h][index_w])
                col = int(input_win[index_h + step_y][index_w +
                                  step_x])
                ret[row, col] += 1
                count += 1
            except Exception as e:
                continue
    return ret / count
glcm_4_direction = (glcm(input_win=win_wrap_img[w][h], step_x=1,
                                  step_y=0, gray_level=gray_level) +
glcm(input_win=win_wrap_img[w][h], step_x=0, step_y=1, gray_level=
                                  gray_level) +
glcm(input_win=win_wrap_img[w][h], step_x=1, step_y=1, gray_level=
                                  gray_level) +
glcm(input_win=win_wrap_img[w][h], step_x=-1, step_y=1, gray_level=
                                  gray_level)) / 4
glcm_4_direction = glcm_4_direction / np.sum(glcm_4_direction)
```

### 1.3 特征计算

本实验计算了熵 (entropy)、能量 (energy)、惯性 (inertia)、相关性 (correlation)、标准差 (std) 以及同质性 (homogeneity)，计算公式分别为:

$$entropy = -\sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j)\log(P(i,j))$$

$$energy = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} P(i,j)^2$$

$$inertia = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} (i-j)^2 P(i,j)$$

$$correlation = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \frac{(i-\mu_x)(j-\mu_y)P(i,j)}{\delta_x\delta_y}$$

$$std = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} ((p(i,j)-\mu)P(i,j))^{0.5}$$

$$homogeneity = \sum_{i=0}^{G-1}\sum_{j=0}^{G-1} \frac{P(i,j)}{1+(i-j)^2}$$

经过实验发现，直接利用特征值进行分类效果并不是很好，需要加入区域性平滑操作，整体特征值计算和区域性平滑代码如下:

```
    entropy = -np.sum(glcm_4_direction * np.log(glcm_4_direction + eps))
    energy = np.sum(glcm_4_direction ** 2)
    i_array = np.arange(gray_level).reshape((1, gray_level)).repeat(
                                        gray_level, 0)
    j_array = np.arange(gray_level).reshape((gray_level, 1)).repeat(
                                        gray_level, 1)
    inertia = np.sum((i_array - j_array) ** 2 * glcm_4_direction)
    mu_x = np.sum(i_array * glcm_4_direction)
    mu_y = np.sum(j_array * glcm_4_direction)
    delta_x = np.sum((i_array - mu_x) ** 2 * glcm_4_direction)
    delta_y = np.sum((j_array - mu_y) ** 2 * glcm_4_direction)
    corr = np.sum((i_array*j_array*glcm_4_direction - mu_x mu_y)/(delta_x
                                        * delta_y))
    std_ = np.std(glcm_4_direction)
    homogeneity = np.sum(glcm_4_direction / (1+(i_array-j_array)**2))
    def conv_smooth(array, ker_s=10):
        re = np.zeros_like(array)
        W, H, f_dim = array.shape
        padding = (ker_s - 1) // 2
```

```
    array = np.pad(array,((padding,padding),(padding,padding),(0,0)),
                                    "edge")
    for w in range(padding, W+padding):
        for h in range(padding, H+padding):
            re[w-padding][h-padding] = np.mean(array[w-padding:w+
                                    padding, h-padding:h+padding,:].
                                    reshape((-1,f_dim)), axis=0)
    return re
```

平滑之后对数据进行标准化处理，处理方式如下:

$$norm = \frac{x - mean}{\delta}$$

代码实现为:

```
means = np.mean(glcm_result.reshape((-1,f_dim)),axis=0).reshape((1,1,
                                    f_dim)).repeat(new_W,0).repeat(new_H
                                    ,1)
stds = np.std(glcm_result.reshape((-1,f_dim)),axis=0).reshape((1,1,
                                    f_dim)).repeat(new_W,0).repeat(new_H
                                    ,1)
glcm_result = (glcm_result-means)/stds
```

### 1.4 K-均值聚类算法

由于前面已经将特征值进行了标准化，所以这里采用最简答的欧式距离来对距离进行度量。整个算法流程为:

(1) 输入数据和类别数 K;
(2) 随机取目标类别数个中心点;
(3) 将每个点标定为离它最近的中心点的类别;
(4) 计算每个类别的均值来更新中心点;
(5) 判断中心点是否更新，否则回到第 (3) 步。

整体代码实现如下:

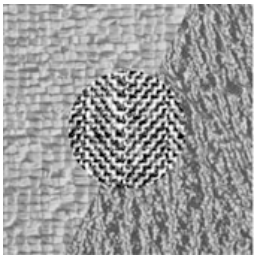
```
def dist(a, b, img):
    now_win = img[a[0]][a[1]]
```

```
    distance = []
    for index in range(len(b)):
        diff = (now_win - b[index]) ** 2
        distance.append(np.sum(diff))
    return np.array(distance)

def point_change(old, new):
    return np.sum((old - new) ** 2)

def k_means(img, K):
    W, H, f_dim = img.shape
    C = np.array([
        img[np.random.randint(low=0, high=W)][np.random.randint(low=
                                    0, high=H)][0]
        for i in range(K)
      ])
    old = np.zeros_like(C)
    clusters = np.zeros((W, H))
    while point_change(C, old) != 0:
        for i in range(W):
            for j in range(H):
                distance = dist(a=[i, j], b=C, img=img)
                cluster = np.argmin(distance)
                clusters[i][j] = cluster
        old = copy.deepcopy(C)
        for k in range(K):
            points = []
            for i in range(W):
                for j in range(H):
                    if clusters[i][j] == k:
                        points.append(img[i][j])
        C[k] = np.mean(np.array(points), axis=0)
    return clusters
```

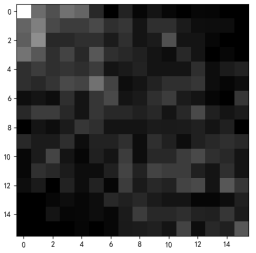
# 2 实验分析

## 2.1 灰度共生阵的计算

本实验窗长设置为 19，滑窗步长为 1，平滑窗宽为 10，灰度级为 16，共生阵计算角度为 0°、45°、90° 和 135°。下面分别是原始图片和坐标为 (79,79) 的点对应的灰度共生矩阵。



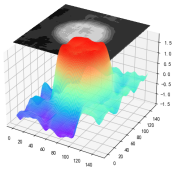
(a) 原始图像



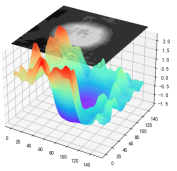
(b) (79,79) 点的灰度共生阵

## 2.2 特征计算

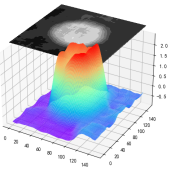
计算得到原图像素点数量的共生矩阵后，对其进行平滑以及熵 (entropy)、能量 (energy)、惯性 (inertia)、相关性 (correlation)、标准差 (std) 和同质性 (homogeneity) 的特征值计算。以下是每个特征在原图中的分布情况可视化:



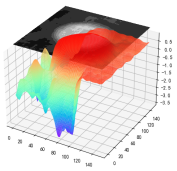
(c) 熵 (entropy)



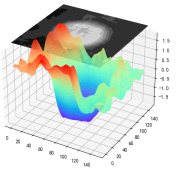
(d) 能量 (energy)



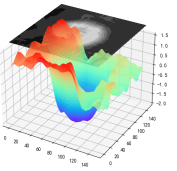
(e) 惯性 (inertia)



(f) 相关性 (correlation)



(g) 标准差 (std)

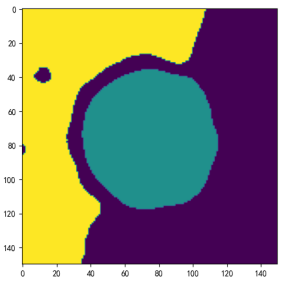


(h) 同质性 (homogeneity)

从图中可以看出不同的特征对于纹理的拟合效果不同，并且实验发现，针对不同的纹理和类别之间的关系需要选择不同的特征来进行计算。

### 2.3 聚类分割

通过算法可以将每个像素点根据其特征数值的大小进行聚类。最终结果如下所示:




(i) 分割结果

从结果来看，中间类的边缘有一半都被分错了，分析其原因是因为滑窗长度较大，导致边缘处的窗下检测到的特征较多，并且由于我们灰度共生矩阵的求取步长都为 1 所以容易将其分类错误。

## 3 总结

经过此次实验，让我更加深入了解了传统的图像分割算法。让我了解到图像纹理的重要性与技巧性。在选择构造特征时还发现，不同特征所想要表达的几何意义也是不同的，我们需要根据实际需求和纹理构造来对特征进行组合。也再次让我意识到平滑降噪的重要性。该实验不但锻炼了我的数学分析能力还提升了我的代码写作能力。非常感谢朱老师给我这次实验的机会。