

# 数字图象处理

## 小作业六

2019010485 自 91 刘祖炎\*

2021 年 12 月 19 日

## 1 算法原理

整体算法流程及部分函数已经在作业文档中给出，此处仅对笔者新编写的函数相关代码进行分析。

### 1.1 数据集处理

由于作业所给代码的 `sample_select` 函数中只获取了数据集中对应图片的路径，并未将对应图片加入数据集中，因此需要对原代码进行一定的修改，将图片读入 `train_samples`、`test_samples` 变量中。更改后的代码如下所示，由于已经得到了对应路径和文件名，因此只需要利用 `imread` 函数进行读取，并对应修改后续函数即可。

```
1 train_samples.image{i} = imread([trainDataDir, '/', files(i).name]);
2 ind = strfind(files(i).name, '_');
3 train_samples.class{i} = files(i).name(1:ind(1)-1);
```

### 1.2 制作滤波器组

考虑 PPT 中所述，制作由 8 个滤波器构成的滤波器组。对应滤波器类型及参数如表1所示，笔者采用了 6 个高斯一阶导滤波核、2 个 LoG 滤波核。

对应滤波核归一化后的可视化结果分别如图1、2、3、4、5、6、7、8 所示。可见滤波器形状与要求相符。

对应代码如下所示。完成 `makeFilterbank` 函数，利用 `fspecial` 函数构造高斯滤波核、LoG 滤波核，利用 '`prewitt`' 方式构造梯度算子，利用 `conv2` 函数对高斯滤波核进行求导，利用 `transpose` 函数对高斯滤波核应用旋转，最终将 8 个滤波核依次赋值至 `fb` 变量即可。`fb` 变量的形状 `[hsize(20), hsize(20), numOffilter(8)]`。

```
1 function fb = makeFilterbank
2     hsize = 20;
3     sigma = 3;
4     fb = zeros(hsize, hsize, 8);
5     gaussian_a = fspecial('gaussian', hsize, sigma);
6     gaussian_b = fspecial('gaussian', hsize, sigma - 1);
```

\*liuzuyan19@mails.tsinghua.edu.cn

表 1: 滤波器类型及参数

序号	类型	参数	方向
1	高斯一阶导	$(20, 20), \sigma = 3$	$0^\circ$
2	高斯一阶导	$(20, 20), \sigma = 2$	$0^\circ$
3	高斯一阶导	$(20, 20), \sigma = 1$	$0^\circ$
4	高斯一阶导	$(20, 20), \sigma = 3$	$90^\circ$
5	高斯一阶导	$(20, 20), \sigma = 2$	$90^\circ$
6	高斯一阶导	$(20, 20), \sigma = 1$	$90^\circ$
7	LoG	$(20, 20), \sigma = 3$	$0^\circ$
8	LoG	$(20, 20), \sigma = 1$	$0^\circ$

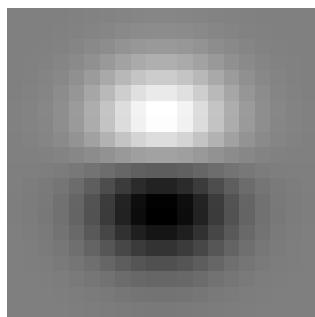


图 1: 滤波核 1

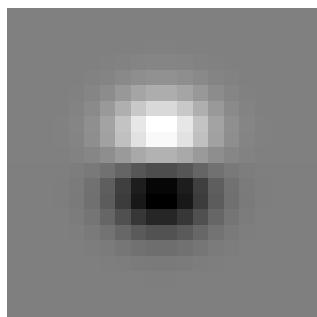


图 2: 滤波核 2

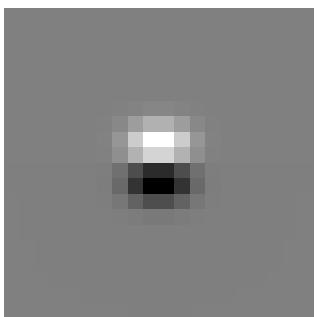


图 3: 滤波核 3

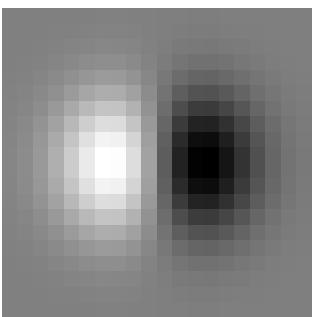


图 4: 滤波核 4

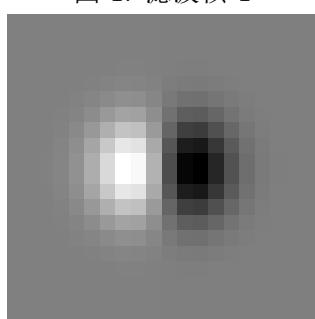


图 5: 滤波核 5

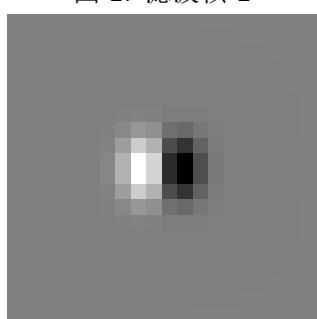


图 6: 滤波核 6

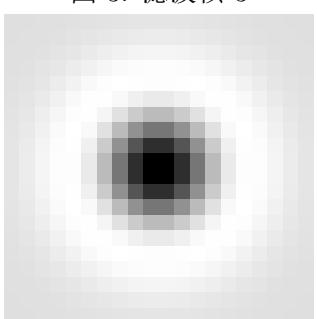


图 7: 滤波核 7

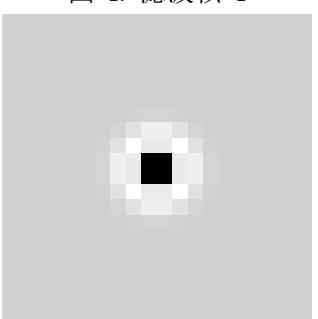


图 8: 滤波核 8

```

7 gaussian_c = fspecial('gaussian', hsize, sigma - 2);
8 prewitt_f = fspecial('prewitt');
9 gaussian_a = conv2(gaussian_a, prewitt_f, 'same');
10 gaussian_b = conv2(gaussian_b, prewitt_f, 'same');
11 gaussian_c = conv2(gaussian_c, prewitt_f, 'same');
12 fb(:, :, 1) = gaussian_a;
13 fb(:, :, 2) = gaussian_b;
14 fb(:, :, 3) = gaussian_c;
15 fb(:, :, 4) = transpose(gaussian_a);
16 fb(:, :, 5) = transpose(gaussian_b);
17 fb(:, :, 6) = transpose(gaussian_c);
18 gaussian_lap_a = fspecial('log', hsize, sigma);
19 gaussian_lap_b = fspecial('log', hsize, sigma - 2);
20 fb(:, :, 7) = gaussian_lap_a;
21 fb(:, :, 8) = gaussian_lap_b;
22 end

```

### 1.3 濾波器响应

完成 *applyFilterbank* 函数求取滤波器响应。遍历需要应用滤波器的每个图像，首先利用 *im2double* 函数将图像转换为范围 [0, 1] 的 *double* 变量，便于计算，后依次应用 *imfilter* 即可。滤波后的响应存储在 *samples\_res* 函数中，形状为 [*numOfImage*(5), *numOfFilter*(8), *Height*(480), *Width*(640)]。对应代码如下所示。

```

1 function samples_res = applyFilterbank(train_samples, fb)
2 [H, W] = size(train_samples.image{1});
3 num = length(train_samples.image);
4 samples_res = zeros(num, 8, H, W);
5 for i = 1:num
6     train_samples.image{i} = im2double(train_samples.image{i});
7     for j = 1:8
8         samples_res(i, j, :, :) = imfilter(train_samples.image{i}, fb(:, :, j));
9     end
10    end
11 end

```

### 1.4 *k-means* 聚类

完成 *applyKmeans* 函数求 *k-means* 聚类。利用 *permute* 函数将样本响应的滤波器置于最后一维，并 *reshape* 为 (-1, 8) 的向量用于求取 *kmeans*。设置 *kmeans* 最大迭代次数为 1000，将各像素类别标签、聚类中心分别储存至 *sample\_cluster*、*centers* 变量中即可。*sample\_cluster* 的形状为 [*Height*, *Width*, *numOfImage*]，*centers* 的形状为 [20, *k*]。对应代码如下所示。

```

1 function [sample_cluster, centers] = applyKmeans(samples_res, k)

```

```

2 samples_res = permute(samples_res, [3, 4, 1, 2]); % H, W, 5, 8
3 [H, W, N, fb] = size(samples_res);
4 kmeans_matrix = reshape(samples_res, [], fb); % H*W*5, 8
5 [samples_cluster, centers] = kmeans(kmeans_matrix, k,
6           'MaxIter', 1000, 'display', 'iter');
7 samples_cluster = reshape(samples_cluster, H, W, N); % H, W, 5
8 end

```

## 1.5 求 *texton* 直方图

完成 *applyHistogram* 函数求 *texton* 直方图。其原理较为简单，求取全局最大值（即聚类数），迭代每幅图的所有聚类标签，统计该标签对应的像素数量，并存入 *samples\_hist* 矩阵即可。对应代码如下所示。

```

1 function samples_hist = applyHistogram(samples_cluster)
2 N = size(samples_cluster, 3);
3 maxNumber = max(max(max(samples_cluster)));
4 samples_hist = zeros(N, maxNumber);
5 for i = 1:N
6     for j = 1:maxNumber
7         samples_hist(i, j) = sum(sum(samples_cluster(:, :, i) == j));
8     end
9 end
10 end

```

## 1.6 可视化函数

完成 *visualizeSamples* 函数求聚类的可视化结果。利用 *labeloverlay* 函数将每个像素的聚类标签赋予一定的颜色，并叠加在原图上即可。对应代码如下所示。

```

1 function visualizeSamples(samples_cluster, train_samples)
2 [~, ~, N] = size(samples_cluster);
3 figure(1);
4 for i = 1:N
5     subplot(2, N, i);
6     imshow(train_samples.image{i});
7     train_cluster = squeeze(samples_cluster(:, :, i));
8     train_cluster = labeloverlay(train_samples.image{i}, train_cluster);
9     subplot(2, N, i + N);
10    imshow(train_cluster);
11 end
12 end

```

完成 *visualizeHistogram* 函数求直方图可视化结果，依次利用 *bar* 函数显示条形统计图即可。对应代码如下所示。

```

1 function visualizeHistogram(samples_hist)
2     [N, ~] = size(samples_hist);
3     figure(2);
4     for i = 1:N
5         subplot(1, N, i);
6         bar(samples_hist(i, :));
7     end
8 end

```

## 1.7 测试像素归类

完成 *predictClasses* 函数对测试像素进行归类。遍历每幅测试图像，利用 *pdist2* 函数计算每个像素与聚类中心的距离，并求上述距离的最小值作为该像素对应的聚类标签。对应代码如下所示。

```

1 function test_samples_cluster = predictClasses(test_samples_res, centers)
2     [N, fb, H, W] = size(test_samples_res);
3     test_samples_cluster = zeros(H, W, N);
4     test_samples_res = permute(test_samples_res, [1, 3, 4, 2]); % 25, H, W, 8
5     for image = 1:N
6         sample = squeeze(test_samples_res(image, :, :, :));
7         sample = reshape(sample, [], fb);
8         dist = pdist2(sample, centers);
9         [~, class] = min(dist, [], 2);
10        class = reshape(class, H, W);
11        test_samples_cluster(:, :, image) = class;
12    end
13 end

```

## 1.8 最近邻法预测类别

完成 *predictLabels* 函数对测试图像进行类别预测。对应代码如下所示。

```

1 function test_samples_labels = predictLabels(test_samples_hist, train_samples_hist)
2     test_samples_labels.classId = zeros([25, 1]);
3     distance = pdist2(test_samples_hist, train_samples_hist);
4     [~, class] = min(distance, [], 2);
5     test_samples_labels.classId = reshape(class, [], 1);
6 end

```

同样地，完成测试图像的直方图统计后，利用 *pdist2* 函数求测试图像与训练图像直方图之间的距离，并求上述距离的最小值，即直方图意义下与每幅测试图像最相近的训练图像，将其作为最终的预测结果。

## 2 运行结果

### 2.1 准确率

完成上述代码编写及调试后，尝试聚类数量  $k$  为不同值，分别观察纹理分类准确率，测试结果如表2所示。

需要注意的是，由于 k-means 聚类每次的结果并不相同，因此下表中部分准确率为相同条件下运行几次中的最大值。

表 2: 不同聚类数下分类准确率

聚类数 $k$	5	20	40
准确率	76%	80%	76%

可见， $k = 20$  时具有最佳的分类效果，当  $k$  持续增大时，效果降低，这可能是因为分类数过多，引入原图中过多细节及噪声。

### 2.2 可视化结果

上述四种情况下聚类、直方图的可视化结果如下所示。

- $k = 5$

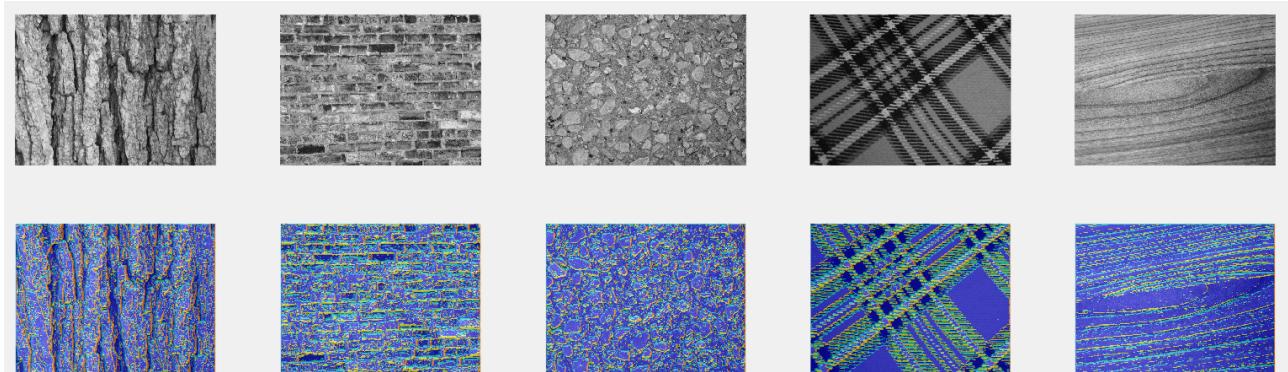


图 9: 聚类结果 ( $k=5$ )

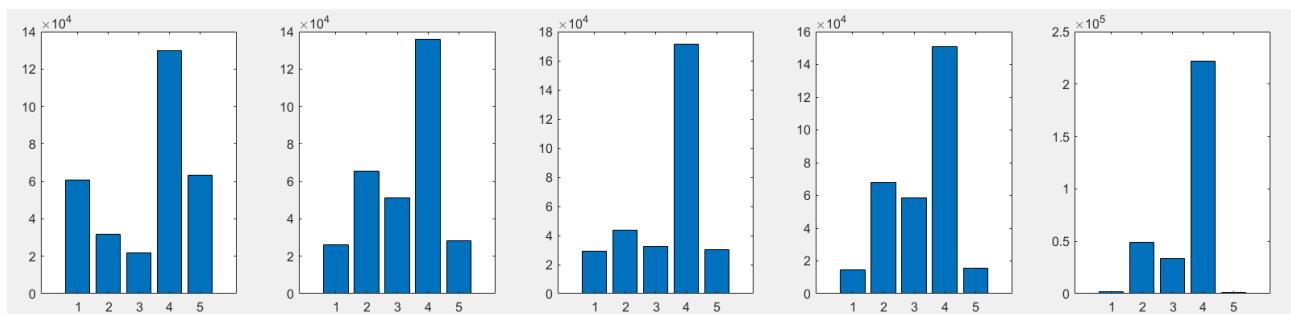


图 10: 直方图 ( $k=5$ )

- $k = 20$

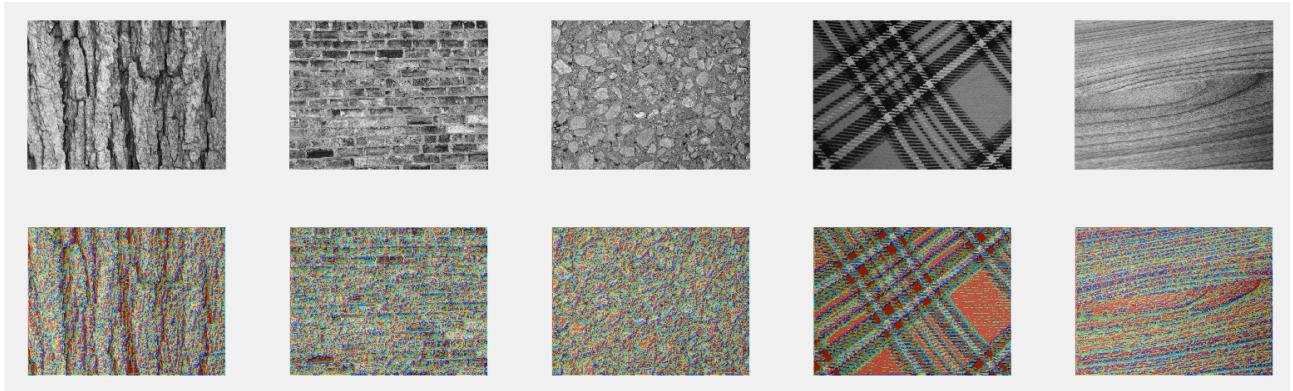


图 11: 聚类结果 ( $k=20$ )

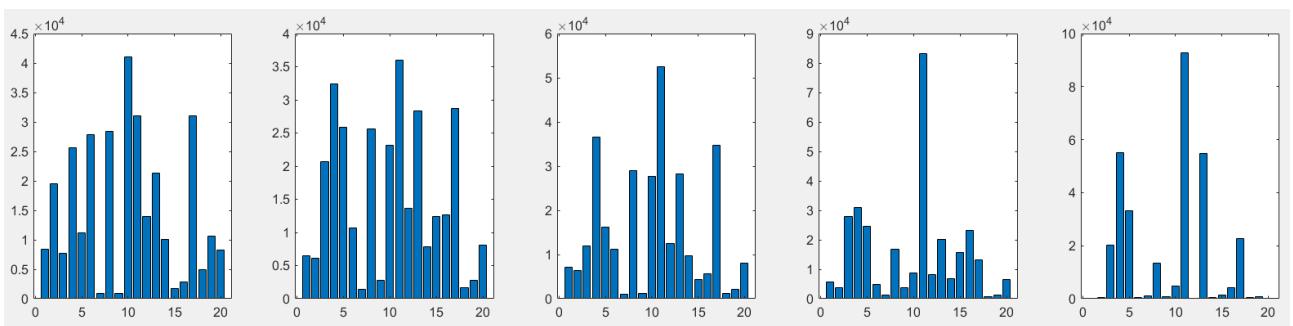


图 12: 直方图 ( $k=20$ )

- $k = 40$

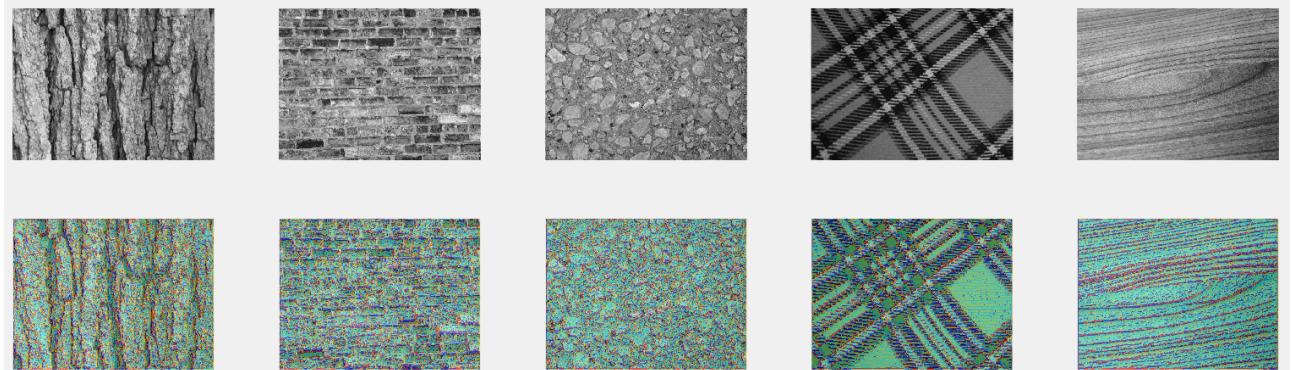


图 13: 聚类结果 ( $k=40$ )

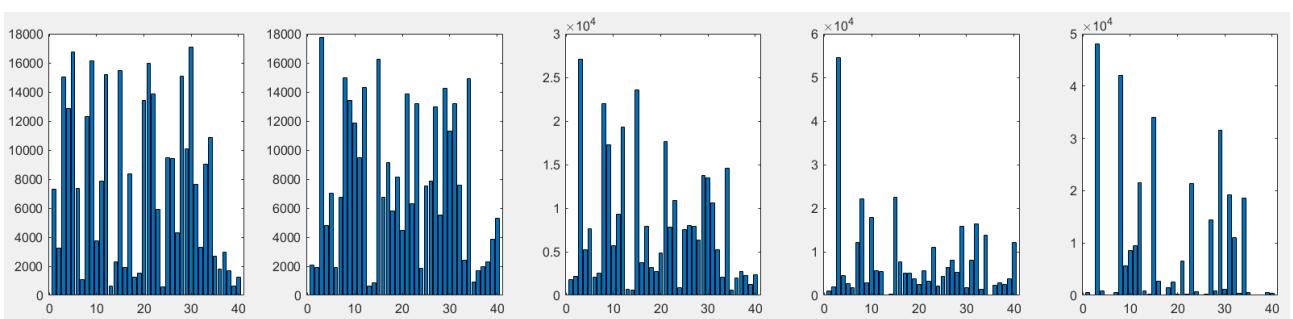


图 14: 直方图 ( $k=40$ )

### 3 选做：增加滤波器个数

笔者考虑到部分图片以斜向的边缘作为主要特征(例如布料、木纹图片)，而部分图片以横向、竖向的边缘作为主要特征(例如砖块图片)，如图15、16所示。

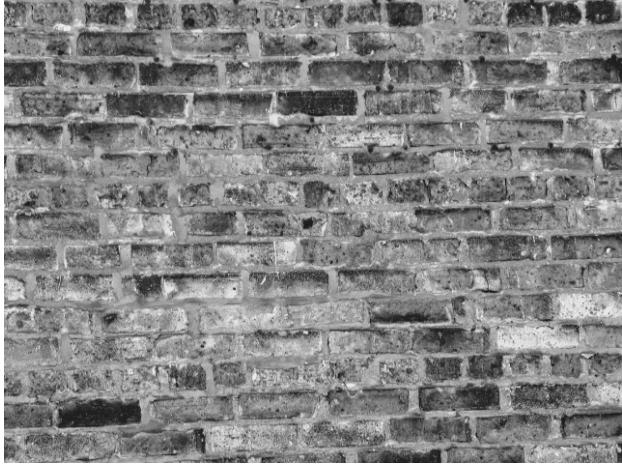


图 15: 横向边缘

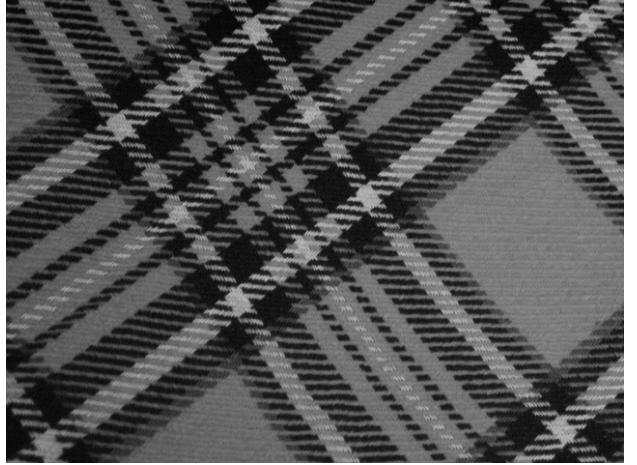


图 16: 斜向边缘

因此，笔者考虑增加角度为 $45^\circ$ 、 $-45^\circ$ 的高斯滤波器以更好地提取斜向边缘特征。增加的滤波器方向分别为 $45^\circ$ 、 $-45^\circ$ ， $\sigma$ 参数不变，对应参数如表3所示。

归一化后可视化结果分别如图17、18、19、20、21、22所示。

表 3: 滤波器类型及参数

序号	类型	参数	方向
9	高斯一阶导	$(20, 20), \sigma = 3$	$45^\circ$
10	高斯一阶导	$(20, 20), \sigma = 2$	$45^\circ$
11	高斯一阶导	$(20, 20), \sigma = 1$	$45^\circ$
12	高斯一阶导	$(20, 20), \sigma = 3$	$-45^\circ$
13	高斯一阶导	$(20, 20), \sigma = 2$	$-45^\circ$
14	高斯一阶导	$(20, 20), \sigma = 1$	$-45^\circ$

准确率测试结果如表4所示。可见，预测准确率有明确的提升，且当 $k = 40$ 时准确率最高，这表明增加滤波器后分类的特征增多，需要更多的聚类数。测试结果表明该方法有效。

表 4: 增加滤波器后分类准确率

聚类数 $k$	20	40
准确率 (8 个滤波器)	80%	76%
准确率 (14 个滤波器)	<b>84%</b>	<b>88%</b>

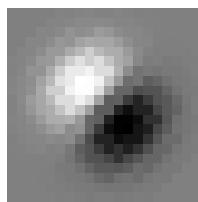


图 17: 滤波核 9

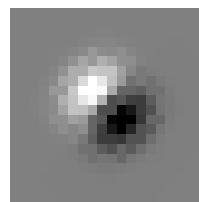


图 18: 滤波核 10

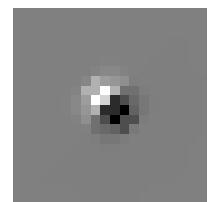


图 19: 滤波核 11

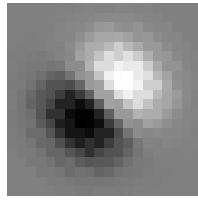


图 20: 滤波核 12

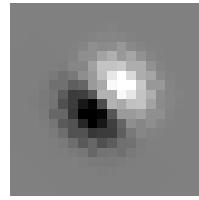


图 21: 滤波核 13

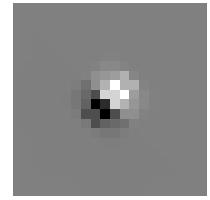


图 22: 滤波核 14

增加滤波器后的可视化结果如下所示。

- $k = 20$ (增加滤波器后)

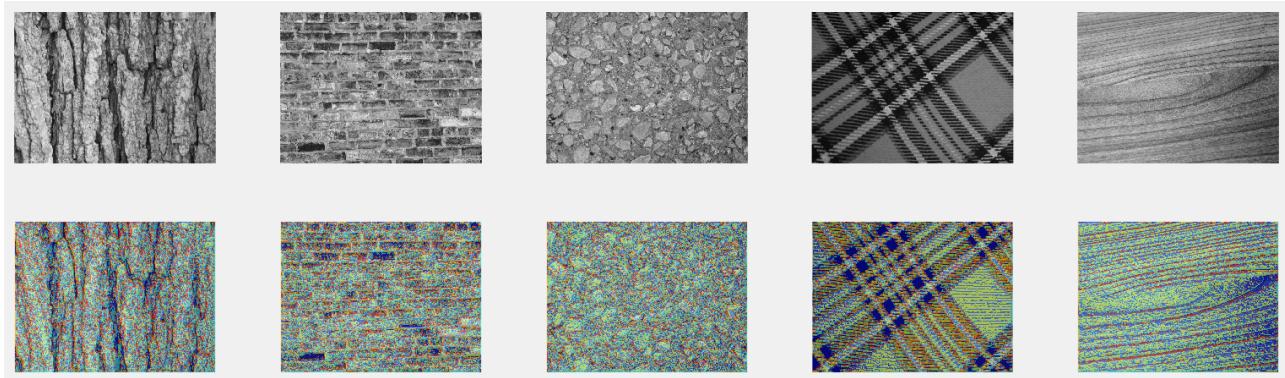


图 23: 聚类结果 ( $k=20$ )

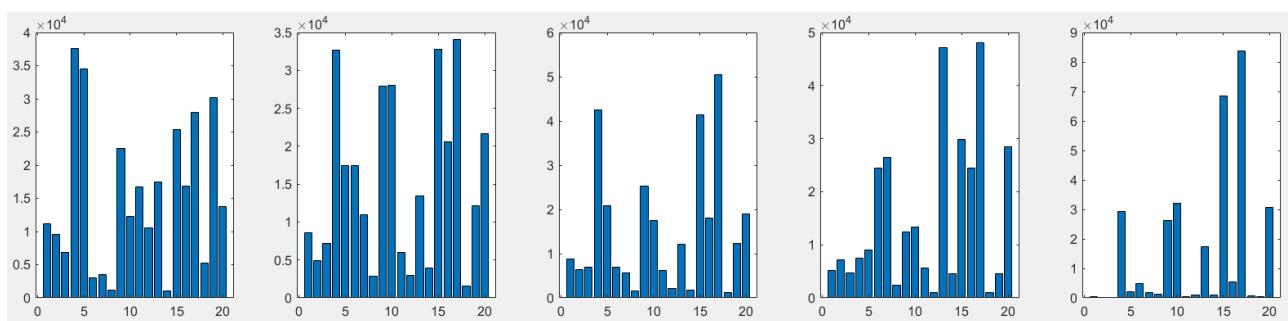


图 24: 直方图 ( $k=20$ )

- $k = 40$ (增加滤波器后)

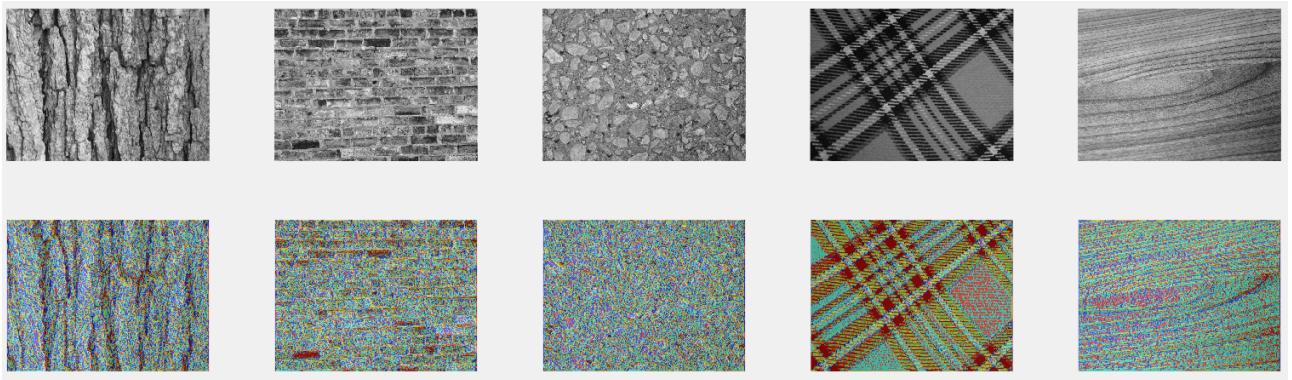


图 25: 聚类结果 ( $k=40$ )

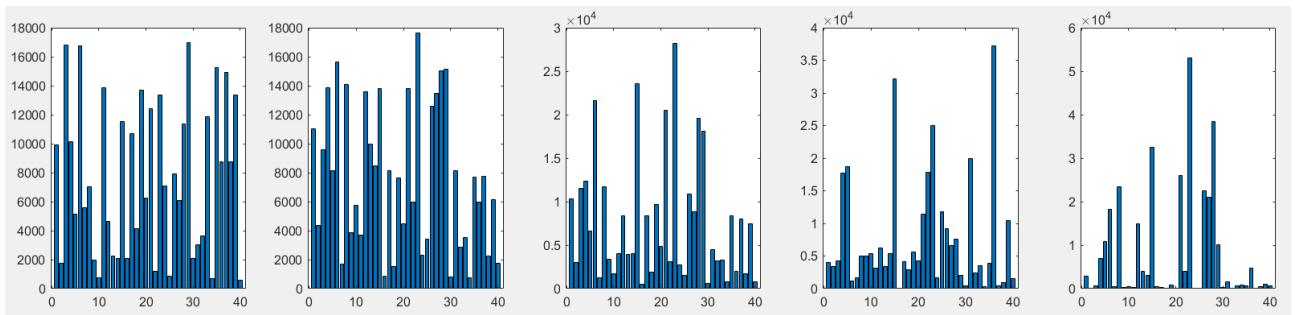


图 26: 直方图 ( $k=40$ )

## 4 遇到的困难及解决方案

由于此次作业提供了完善的代码框架及流程，故在正确运用 Matlab 工具箱函数的基础上，笔者并未遇到太大的困难。概括而言，主要遇到了以下几个问题。

- 程序运行效率低

在一开始运行代码时，笔者发现程序运行的效率较低，需要大约 5 分钟才能完成整个分类过程。在仔细检查代码阻塞的函数后，笔者意识到这是因为，对测试集求聚类的步骤是逐像素进行的。考虑将其修改为以矩阵形式进行运算的方式后，上述问题得到解决，基本上所有程序的运行时间都耗费于 k-means 聚类步骤中。

- 对工具箱函数的运用

本次作业涉及到 *pdist2*、*kmeans* 等 Matlab 工具箱函数，因此，正确设置函数参数较为重要。一开始笔者发现 k-means 聚类以默认参数进行运算时会提示“未能在 100 次迭代后收敛”，查看帮助手册后笔者修改了参数，设置停止条件由 100 次迭代改为 1000 次迭代，并将每次迭代的数值打印出来，上述问题得到解决。

- 增加滤波器的方式

由于在做此次小作业的同时笔者正在复习数图，受形态学运算一章“检测各方向的线”(图27)启发，笔者很自然地想到了增加 45 度滤波器的方式提取斜向特征，最终检验可知该方式起到了较好的效果。

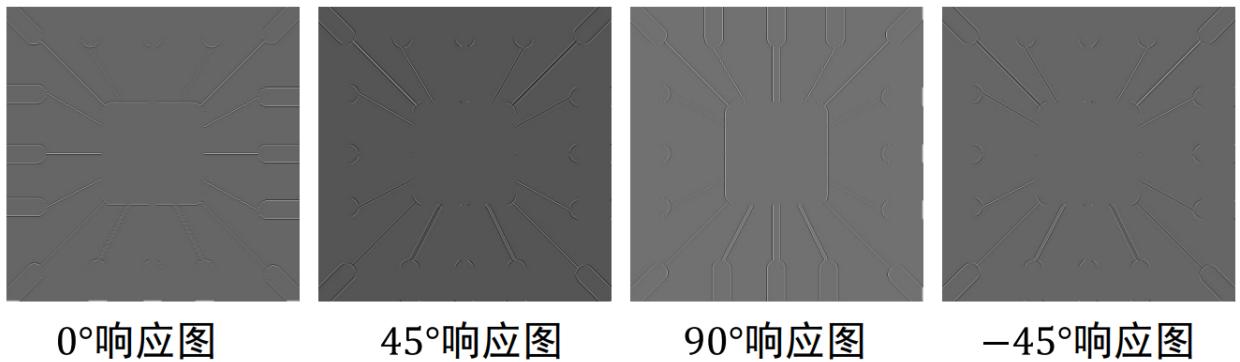


图 27: 提取各方向特征 (检测各方向的线)

## 5 收获与心得体会

- 代码风格的重要性

尽管一直知道在矩阵运算中应当尽量避免循环，但是由于此前的数图作业基本上不会有太大的计算量，因此笔者逐渐忽略了这一要求，而是使用更不容易出错的循环写法。此次作业进一步提示笔者，在矩阵运算中应当尽量应用矩阵的方式进行计算。此外，此次作业中，对部分参数应当避免赋予定值，而应当根据矩阵形状动态设置，这样可以很大程度上提升调试的效率。

- 传统算法的可解释性

传统算法的一个重要优势是具有可解释性。基于此，此次作业增加滤波器提升准确率并非随手尝试，而是应当从算法原理的角度出发设计，这样能够更好地找到正确的方法。

## 6 可能的改进方向

- 可以进一步应用所学知识，对原图进行图像增强，减少噪声的干扰，提升准确率。
- 可以设计更加合理的滤波器，更加准确地提取图像信息。