Homework2

ZF1921332 林星辰

一. 实验结果

经过可视化 Gabor 调参后，指定各参数为
cv2.getGaborKernel((11, 11), 11, 2.670353755551324, 11, 1.94, -1.5707963267948966)
使用 Gabor + SIFT + FLANN 匹配方法
FLANN 匹配阈值 0.8

匹配度阈值	相同手匹配数量(3600)	相同手匹配率	不同手错误匹配数量(357600)	不同手错误匹配率
0.09	3156	0.870	20155	0.0565
0.10	3027	0.840	9234	0.0258
0.11	2870	0.797	3782	0.0158
0.12	2740	0.761	1554	0.004
0.13	2583	0.718	611	0.0017
0.14	2435	0.676	224	0.0006
0.15	2319	0.644	76	0.0002

代码库：https://github.com/SUTFutureCoder/GaborFilter

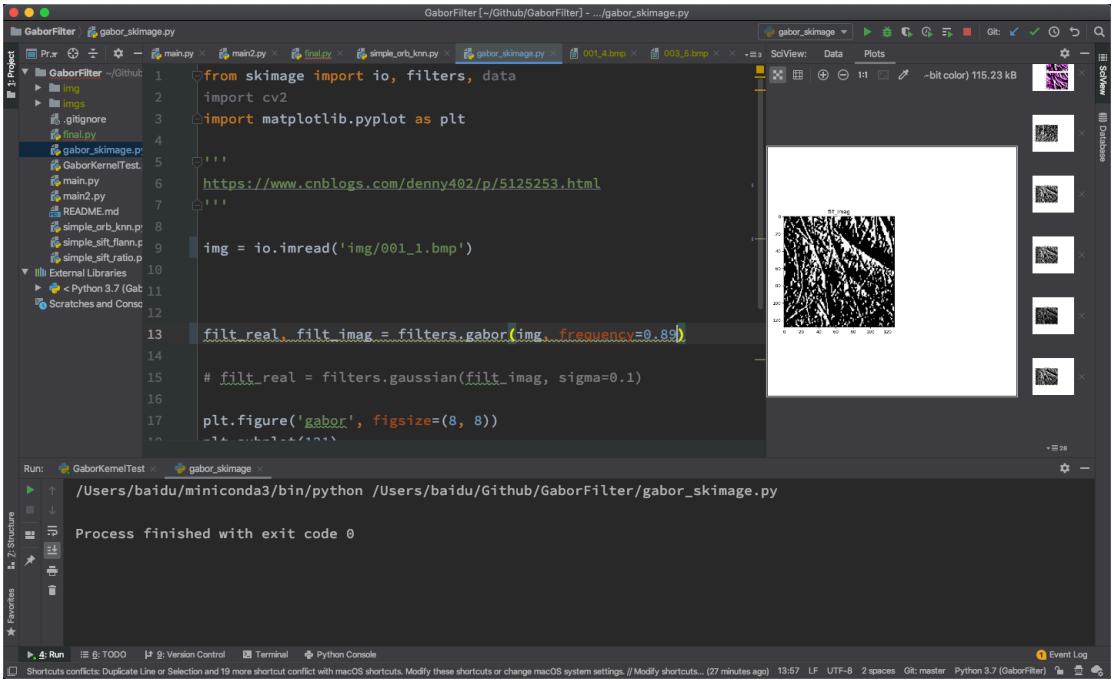
二. 实验步骤

2.1 sift + flann

因 sift + flann 无需调参，相对于具有多个参数的 gabor 更方便进行调试。因此在本实验中，先逆向实现 sift + flann 方法，再编写 gabor 相关参数逻辑

2.2 gabor

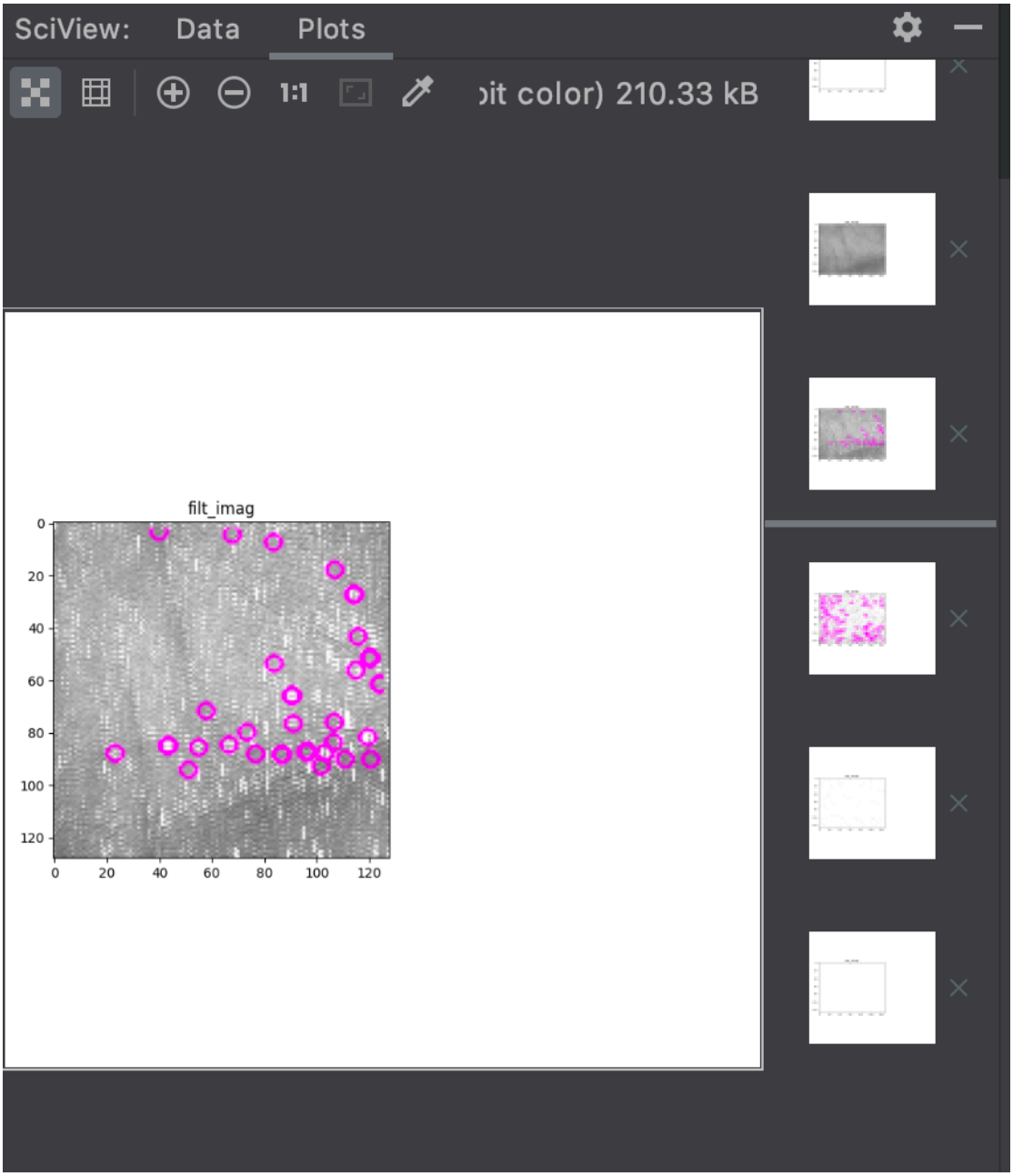
Opencv 和 skimage 都提供了 Gabor 滤波器的相关方法，其中 skimage 的 gabor 滤波器更加便捷，只需要调整频率参数即可达到不同的滤波效果。但通过调试和肉眼可见具有大量的非关键条纹噪音，所以无法采用改为 Opencv 的方法。



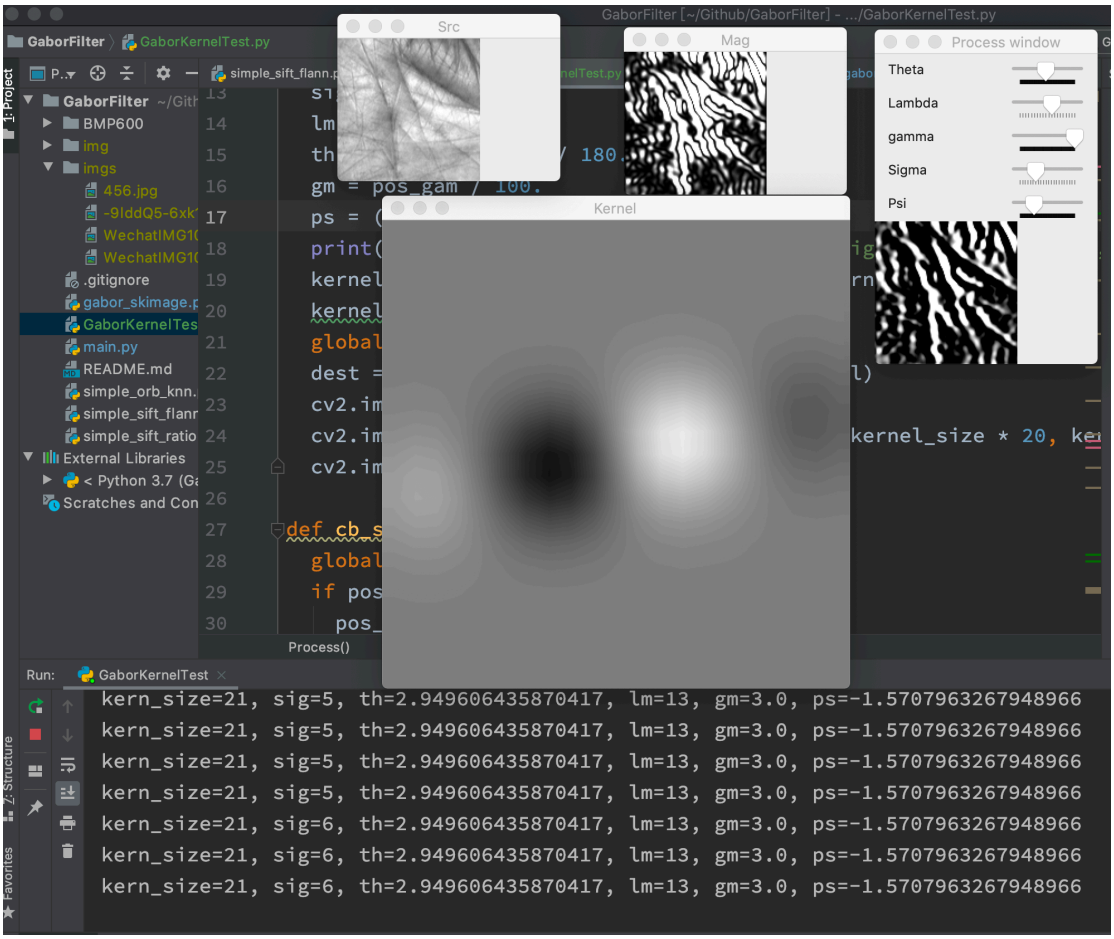
接下来，使用网络上找到的样例参数进行设置，但实际运行的效果并不理想。

(https://blog.csdn.net/hanwenhui3/article/details/48289145

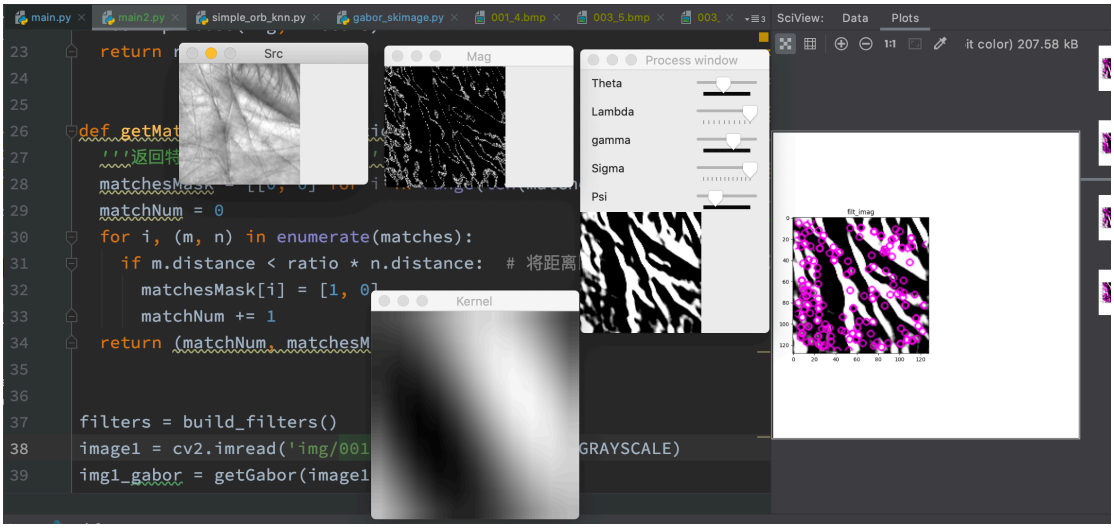
)



通过推断得出，问题可能出现在参数选取上，因此在 GaborKernelTest 文件中对各项参数进行可调参化，通过手动调参得到对应的输出滤波后图像、控制台输出各项参数的值。以此来反向调整上图中的参数。

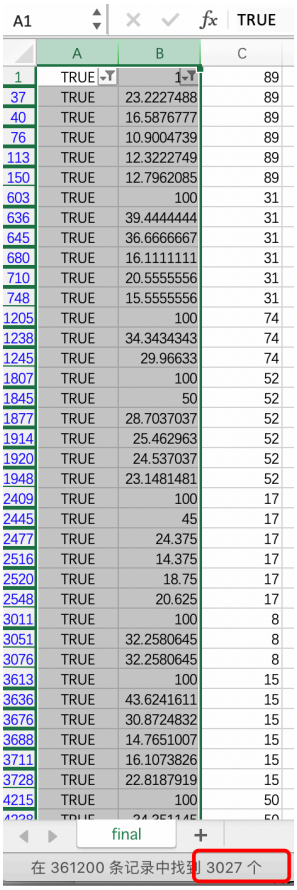


2.3 结合并参数优化后的结果



2.4 统计结果

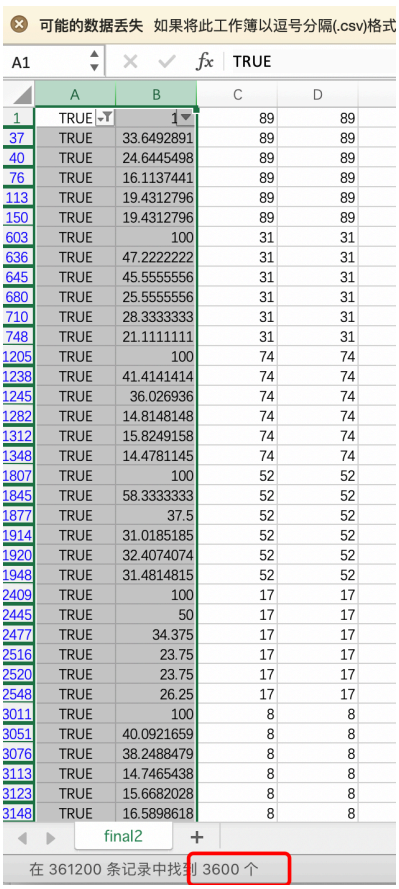
通过对输出的 csv 文件使用 excel 进行快速筛选，可得出不同匹配阈值下正确和错误匹配的数量，最终结果请看报告头部。



	A	B	C
1	TRUE	1	89
37	TRUE	23.2227488	89
40	TRUE	16.5876777	89
76	TRUE	10.9004739	89
113	TRUE	12.3222749	89
150	TRUE	12.7962085	89
603	TRUE	100	31
636	TRUE	39.4444444	31
645	TRUE	36.6666667	31
680	TRUE	16.1111111	31
710	TRUE	20.5555556	31
748	TRUE	15.5555556	31
1205	TRUE	100	74
1238	TRUE	34.3434343	74
1245	TRUE	29.96633	74
1807	TRUE	100	52
1845	TRUE	50	52
1877	TRUE	28.7037037	52
1914	TRUE	25.462963	52
1920	TRUE	24.537037	52
1948	TRUE	23.1481481	52
2409	TRUE	100	17
2445	TRUE	45	17
2477	TRUE	24.375	17
2516	TRUE	14.375	17
2520	TRUE	18.75	17
2548	TRUE	20.625	17
3011	TRUE	100	8
3051	TRUE	32.2580645	8
3076	TRUE	32.2580645	8
3613	TRUE	100	15
3636	TRUE	43.6241611	15
3676	TRUE	30.8724832	15
3688	TRUE	14.7651007	15
3711	TRUE	16.1073826	15
3728	TRUE	22.8187919	15
4215	TRUE	100	50

final

在 361200 条记录中找到 3027 个



可能的数据丢失 如果将此工作簿以逗号分隔(.csv)格式

	A	B	C	D
1	TRUE	1	89	89
37	TRUE	33.6492891	89	89
40	TRUE	24.6445498	89	89
76	TRUE	16.1137441	89	89
113	TRUE	19.4312796	89	89
150	TRUE	19.4312796	89	89
603	TRUE	100	31	31
636	TRUE	47.2222222	31	31
645	TRUE	45.5555556	31	31
680	TRUE	25.5555556	31	31
710	TRUE	28.3333333	31	31
748	TRUE	21.1111111	31	31
1205	TRUE	100	74	74
1238	TRUE	41.4141414	74	74
1245	TRUE	36.026936	74	74
1282	TRUE	14.8148148	74	74
1312	TRUE	15.8249158	74	74
1348	TRUE	14.4781145	74	74
1807	TRUE	100	52	52
1845	TRUE	58.3333333	52	52
1877	TRUE	37.5	52	52
1914	TRUE	31.0185185	52	52
1920	TRUE	32.4074074	52	52
1948	TRUE	31.4814815	52	52
2409	TRUE	100	17	17
2445	TRUE	50	17	17
2477	TRUE	34.375	17	17
2516	TRUE	23.75	17	17
2520	TRUE	23.75	17	17
2548	TRUE	26.25	17	17
3011	TRUE	100	8	8
3051	TRUE	40.0921659	8	8
3076	TRUE	38.2488479	8	8
3113	TRUE	14.7465438	8	8
3123	TRUE	15.6682028	8	8
3148	TRUE	16.5898618	8	8

final2

在 361200 条记录中找到 3600 个

三.文件目录

final.py —— 最终代码
final.csv—— 最终结果输出文件，以 10 作为阈值
main2.py —— 通过可视化 Gabor 调参之后的代码
main.py —— 循环取 theta 效果较差
GaborKernelTest —— 可视化 Gabor 调参器
gabor_skimage —— 使用 skimage 库调用 gabor 滤波相关方法，噪音过大
simple_orb_knn —— 使用 orb+knn 匹配方法，噪音过大
simple_sift_flann —— sift + flann 方法
simple_sift_ratio —— 能够量化匹配度的 sift + flann 方法

四.疑问

1. Theta 值在不同图片光照和方向情况下最优值各不相同，如何确定合适的 Theta 值，循环遍历各种方向的效率过低且掺杂错误值。
2. SIFT + FLANN 方案可行性
3. 如何（固定/动态）选取合适的值，因为参数较多而且难以控制变量调试，我写了个可视化调参器，但对于多个图片的效果也不满意。

五.最终版代码

```
import cv2
import os

def getGabor(img, filters):
    res = cv2.filter2D(img, cv2.CV_8UC4, filters) # 2D 滤波函数  kern 为滤波模板
    return res

def getMatchNum(matches, ratio):
    '''返回特征点匹配数量和匹配掩码'''
    matchesMask = [[0, 0] for i in range(len(matches))]
    matchNum = 0
    for i, (m, n) in enumerate(matches):
        if m.distance < ratio * n.distance:    #  将距离比率小于 ratio 的匹配点筛选出来
            matchesMask[i] = [1, 0]
            matchNum += 1
    return (matchNum, matchesMask)


filters = cv2.getGaborKernel((11, 11), 11, 2.670353755551324, 11, 1.94, -
1.5707963267948966)

#  读取文件
imgfiles = []
for root, dirs, files in os.walk("./img"):
    imgfiles = files

#  创建 sift 及 flann 初始化
#  创建 SIFT 特征提取器
sift = cv2.xfeatures2d.SIFT_create()
#  创建 FLANN 匹配对象
FLANN_INDEX_KDTREE = 0
indexParams = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
searchParams = dict(checks=50)
flann = cv2.FlannBasedMatcher(indexParams, searchParams)

#  遍历并缓存处理
cache = {}
for file in imgfiles:
    image = cv2.imread('./img/' + file, cv2.IMREAD_GRAYSCALE)
    cache[file] = {}
    cache[file]["kp"], cache[file]["des"] = sift.detectAndCompute(getGabor(image, filters), None)
```

```
# 遍历
round = 0 # 轮次
right = 0 # 成功次数
for file1 in imgfiles:
    split_file_name1 = file1.split("_")
    img1kp = cache[file1]["kp"]
    img1des = cache[file1]["des"]

    for file2 in imgfiles:
        round += 1
        split_file_name2 = file2.split("_")
        kp2 = cache[file2]["kp"]
        des2 = cache[file2]["des"]
        matches = flann.knnMatch(img1des, des2, k=2)  # 匹配特征点，为了删除匹配点，指定 k=2，对样本图每个特征点，返回两个匹配
        (matchNum, matchesMask) = getMatchNum(matches, 0.8)  # 通过比率条件，计算匹配度
        matchRatio = matchNum * 100 / len(matches)

        if matchRatio > 10.:
            # 假定通过，观察是否正确
            if split_file_name1[0] == split_file_name2[0]:
                right += 1
        # print(str(split_file_name1[0] == split_file_name2[0]) + "," + str(matchRatio) + "," + split_file_name1[0] + "," + split_file_name2[0])

print("#####")
print("rate:" + right/round)
print("#####")
print("right:" + right)
print("#####")
print("round:" + right/round)
```