深圳大学考试答题纸

(以论文、报告等形式考核专用)

二〇 二三 ～二〇 二四 学年度第 二 学期

课程编号 2801000023 课程名称 机器视觉技术及工业应用 主讲教师 何志权 评分

学 号 2021280463 姓名 贺潇乐 专业年级 21 级电子信息工程 05

教师评语：

题目： **工业产品的视觉测量和外观检测**

目录

1. 总体介绍

本次课程设计的内容为，运用机器视觉方法，通过 OpenCV 进行工业样品的测量和缺陷检测。具体为两个任务：

1）用亚像素边缘检测方法，测量一批样品的边缘是否为严格的矩形，并画出 data/measure 目录下全部产品的高度，宽度，矩形度的分布，均值和方差；

2）对另一批样品做三个层次的缺陷检测，主要包括图像级别（样品是否为缺陷样品）、缺陷级别（标注出缺陷的大致位置）和像素级别（对每个像素是否为缺陷做二分类，最终将缺陷图像二值化）。

数据介绍：

在“data/measure”目录下，存有任务一的样品图。样品在图像的正中心，呈现黑色；样品的摆放角度略微倾斜；图像背景为纯白，并且底部存在着无关部分。

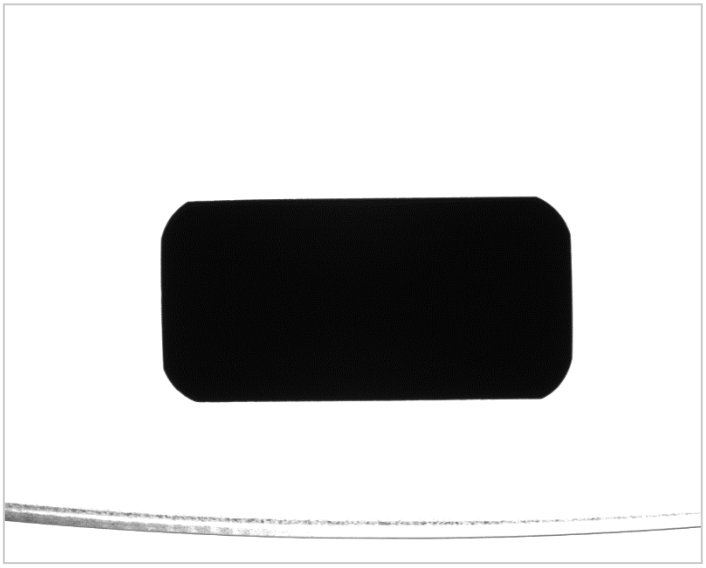


图 1-1

在“data/defect”目录下，存有任务二的样品图，包含了缺陷（NG）和无缺陷（OK）两类样品。无缺陷样品只有图片，有缺陷的样品包括了图片和缺陷标签两种数据。

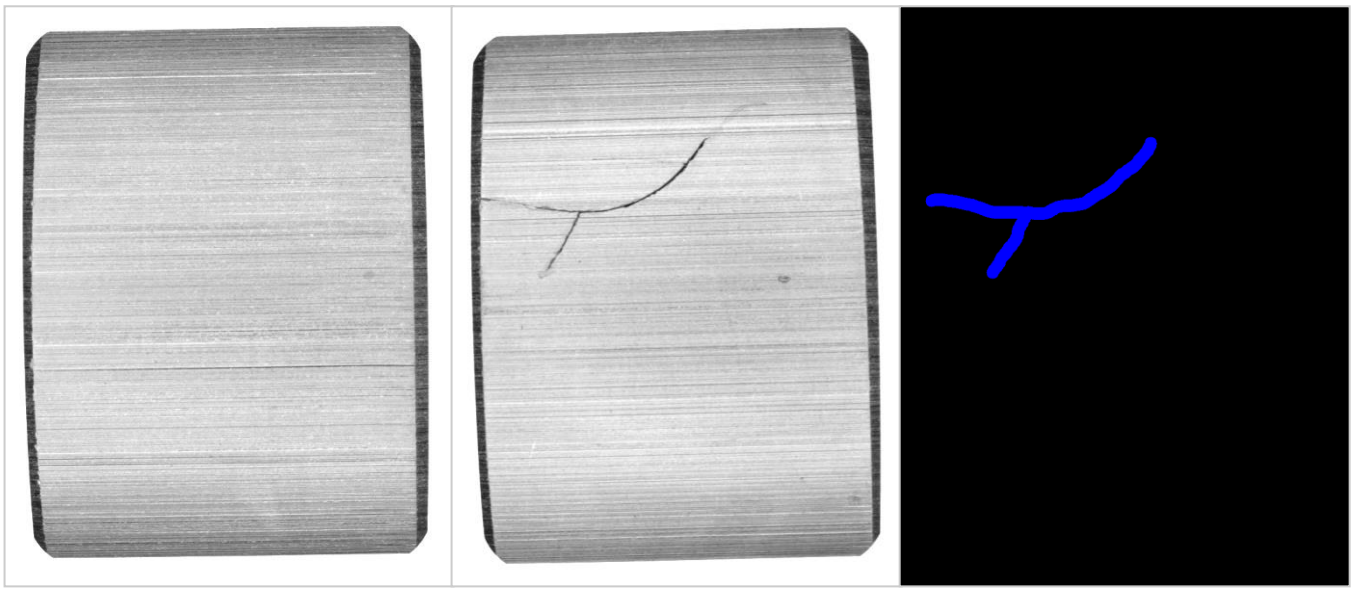


图 1-2 从左到右依次为无缺陷样品、缺陷样品和对应的标签

2. 图片的矩形度评估

对图片的矩形度评估分为三个阶段，依次为倾斜校正、边缘检测和矩形度的计算，下面做依次介绍。

2.1 图像的倾斜校正

先读取图像。为了省去图片底部的无关部分，以及缩短亚像素边缘的计算时间，直接将图片横向裁剪，留下中间部分。

然后，对图像取反色，并抑制像素值为 220 以下的所有噪声。

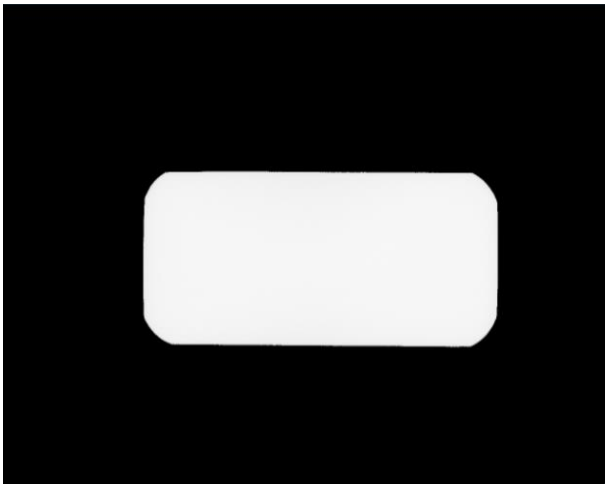


图 2.1-1 中间结果

根据图片 2.1-1 进行倾斜纠正，具体的方法为：将样品旋转 N 度后，观察样品在竖直方向上的跨度 L（图像不为 0 的部分最大索引与最小索引在竖直方向上的差），找到使跨度 L 最小的旋转角度$N = argmin_N(L)$，就是将工件调整到水平所需的旋转角。（函数 tile_correction()）

```
D:\anaconda\envs\He\python.exe D:\work\project\机器视觉技术及工业应用\course
为纠正图像0_20211109092034452_486_8_OK.bmp的倾斜，  将其旋转90.4度
为纠正图像0_20211109092035739_488_8_OK.bmp的倾斜，  将其旋转90.2度
为纠正图像0_20211109092036372_489_8_OK.bmp的倾斜，  将其旋转90.0度
为纠正图像0_20211109092037926_491_8_OK.bmp的倾斜，  将其旋转88.9度
为纠正图像0_20211109092041259_495_8_OK.bmp的倾斜，  将其旋转89.5度
```

图 2.1-2 倾斜矫正的运行效果，矫正的结果保存在“data/ measure_correctedEdge/”目录下

2.2 亚像素边缘检测和边缘提取

用亚像素的方法检测边缘，亚像素检测的思路为：求得每一个边缘点的梯度幅值和方向，然后结合自身以及邻域像素，沿着梯度方向做拟合（这里用的是二次拟合），求出真实的边缘所在的位置，从而使边缘的位置更加准确。（函数 tile_correction()中调用的函数 subpixel_edge()）

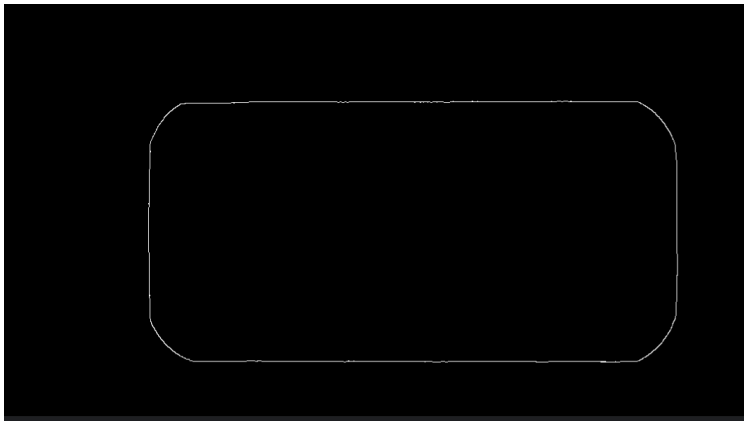


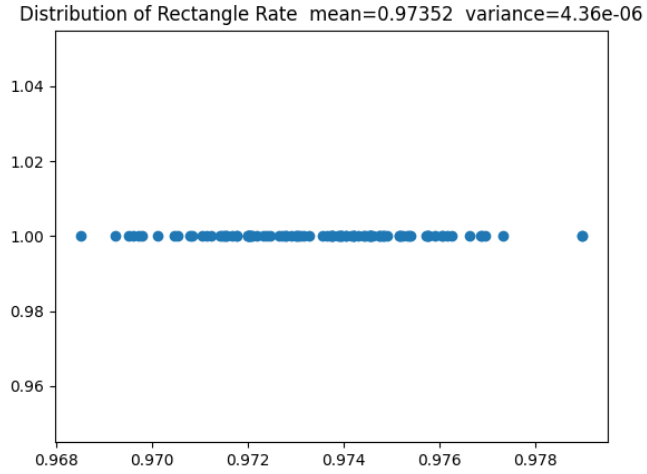
图 2.2 倾斜校正以及亚像素边缘检测，并进行闭运算后的结果

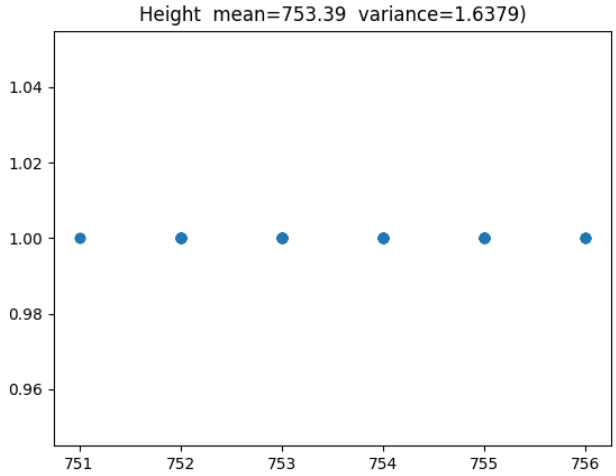
然后，进行边缘提取，按顺时针顺序获取一连串的边缘坐标。具体的边缘提取算法为：将某一个边缘点设为种子，在其顺时针的 8 领域以及 16 邻域内寻找临近的边缘像素点，若找到，就把新的边缘像素点设为种子。（函数 gen_contours()）

为了使该算法正常工作，需要保证亚像素边缘至少在 16 邻域内连续，因此，图 2.2 中的边缘为对亚像素边缘闭运算后的结果。之后再对图 2-2 做小模板匹配，去除多余的像素点（函数 thinner_edge()）。完成了以上操作，才可以应用上述边缘提取算法。

2. 3 矩形度评估

设计一个矩形度评估算法，矩形度 Rate 等于四边形自身的面积除以最小外接矩形的面积。若四边形为严格的矩形，那么经过倾斜矫正，矩形度 Rate 等于 1；若不为严格的矩形，则 Rate 小于 1。





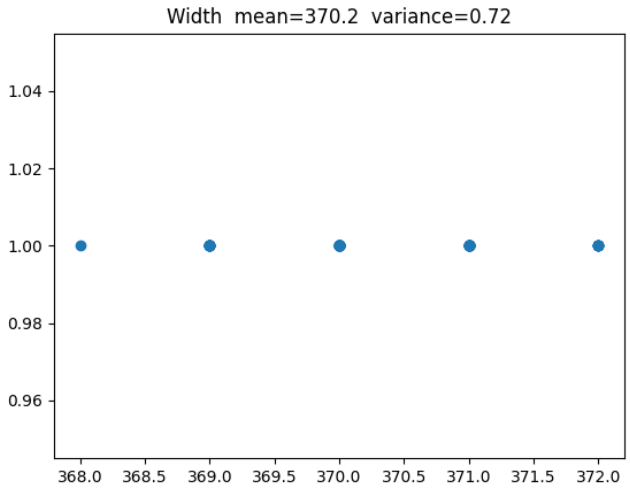


图 2.3 三张图依次为矩形度、样品高度和样品宽度的散点图，以及它们的均值和方差。由于样品本身为圆角矩形，所以矩形度在 0.97 左右，而没有预想中那么接近 1。

3. 缺陷检测

缺陷检测依次经过图像级、缺陷级和像素级三个层面，大致的思路为：先用比较粗糙的方法（3.1），识别出缺陷样品，再应用另一种算法定位缺陷（3.2），并进行像素级别的检测（3.3），得出图像级分类的准确率，以及缺陷检测的像素覆盖率。

3. 1 图像层面

将图像取反色之后，用与 2.1 相似的方法做倾斜校正。倾斜校正的结果保存在

“data/defect_correctedEdge/”目录下

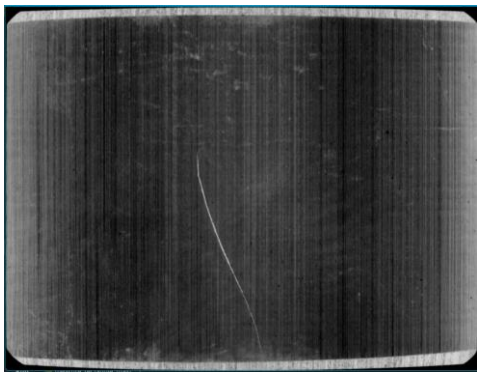


图 3.1-1 倾斜校正的结果

观察到样品中除了缺陷的条纹，还存在着大量杂乱的周期性竖直条纹。为了避免这些条纹影响后期的二值化、形态学处理等操作，需要事先将它们去除。由于这些条纹存在着一定的周期性特征，所以可以在频域上操作。具体来说，是将图像转换到频域，然后应用一个带通滤波器，将频域响应在水平方向的高频分量去除，IFFT 变换回来，就可以去除竖直方向的条纹（函数 eliminate_thin_curve()）。

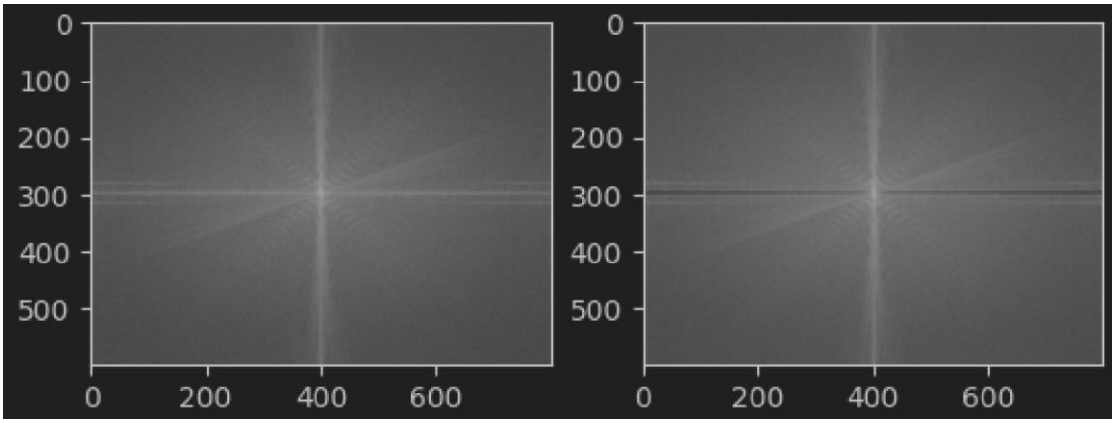


图 3.1-2 和图 3.1-3 两张图为频域操作前后的频域幅度对比图

将图 3.1-3 进行反变换，得到了去除杂乱条纹的样品图，然后通过二值化和图形学算子去除顶部和底部的白色边缘。去除条纹的图像 3.1-4 被用于缺陷和像素级别的检测。

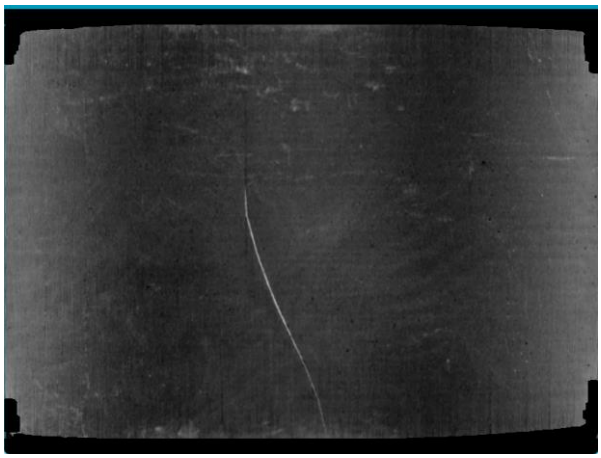


图 3.1-4 去除条纹、图像顶边和底边的图像

然后再应用一个带通滤波器，将频域响应在竖直方向的高频分量去除，然后 IFFT 变换回来得到图像 I，经过二值化、顶帽运算和开运算，得到 I_thres. 计算 I_thres 中非 0 像素点的个数 count，作为图像层面缺陷检测的依据。count 越大，越可能是缺陷样品。

分别画出缺陷和无缺陷样品 count 的散点图，设定一个阈值 th= 580，count 大于 th 就认为是缺陷样品，否则就没有缺陷。

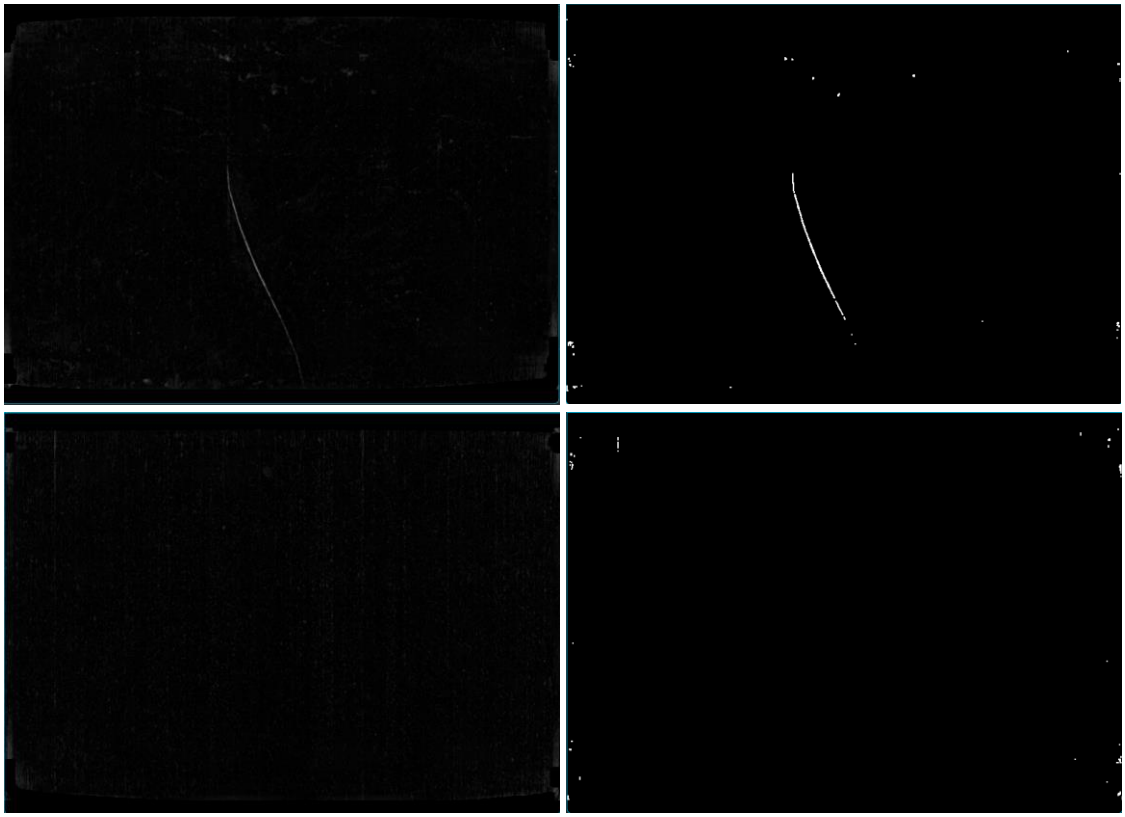


图 3.1-5 从左到右，从上到下依次为有缺陷样品的 I 和 I_thres，以及无缺陷样品的 I 和 I_thres

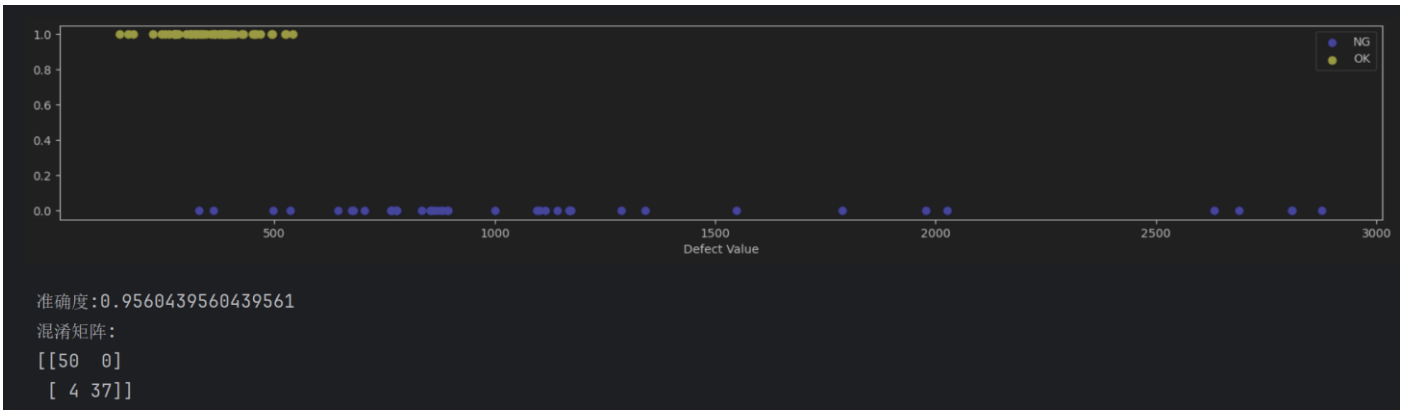


图 3.1-6 以上为缺陷样品和无缺陷样品 count 的散点图。th=580 时，检测准确率在 0.96，有 4 个缺陷样品没有检测出来，不过无缺陷的样品没有被误检。

3. 2 缺陷层面

图像层面中，带通滤波和形态学的操作能够大致判断样品是否存在缺陷，但是样品的噪声比较复杂，样品间光照强度不同，样品内光照也并不均匀，这使得二值化图像 I_thres 上存在较多的噪点，难以继续通过形态学的方法定位缺陷，这在很多样本上都能得到体现。因此，想要比较准确地定位缺陷，需要对图 3.1-4（去除条纹的图像）做其它的处理。

灰度共生矩阵

灰度共生矩阵统计给定方向和距离下不同灰度级像素对的出现频率或概率，描述图像中像素与其邻近像素的空间关系。想要生成灰度共生矩阵，首先需要确定的方向和距离，然后计算图像中每个像素与指定方向和指定距离像素的灰度对，并填充到矩阵中。在灰度缺陷检测中，灰度共生矩阵能够揭示纹理特征，如粗糙度和方向性，这些特征有利于识别材料表面的缺陷。通过计算矩阵的统计特征，如对比度和均匀性，可以提取有助于缺陷检测的特征。

本次缺陷层面的检测，就利用了灰度共生矩阵。将图像分成 15*20 个不重叠的小块，计算每个小块的在

不同方向的灰度共生矩阵 GLCM，再得到 GLCM 的对比度 Contrast，为对比度设定一个阈值 α，某小块的 Contrast 大于 α，就认为其中存在着缺陷。通过这种方式，可以提取图像中感兴趣的小块作为掩膜 mask，这个 mask 就是缺陷层面的检测结果。组成 mask 的每一个小块的对角线坐标保存在 data/detected_rectangle 路径下。（函数 GLCM(), calculate_glcm()和 extract_texture_features()）

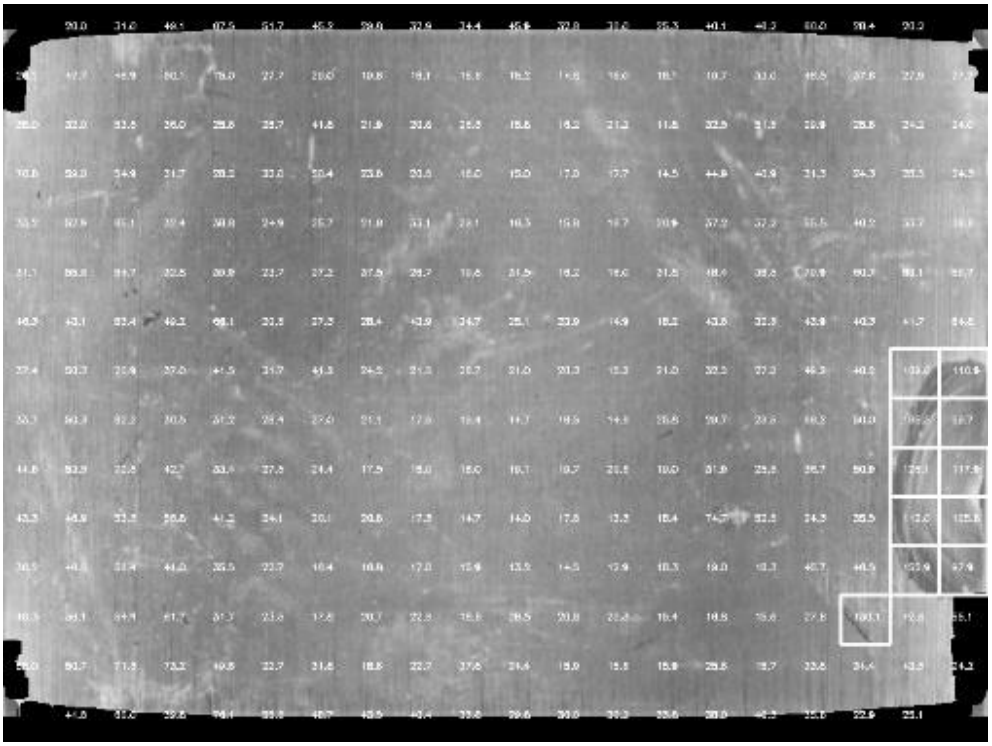


图 3.2-1 对于某个样品，分成 15*20 的小块，为每个小块计算 GLCM 的 Contrast 并显示出来，然后框出 Contrast 比较大的小块。该图片保存在 data/detected_rectangle/glcm_img 路径下

图 3.2-2 图 3.2-1 对应的掩膜 mask；

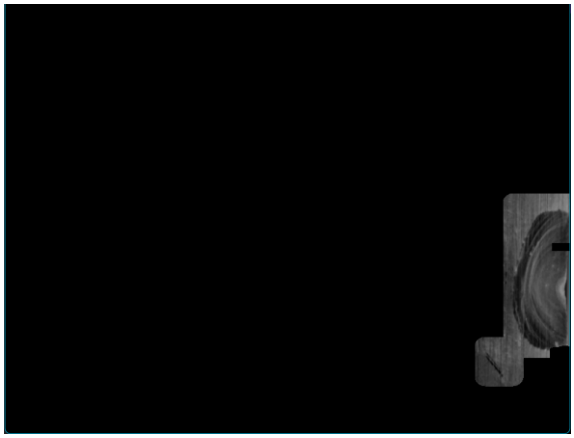


图 3.2-3 将原图用 mask 过滤，得到关注部分

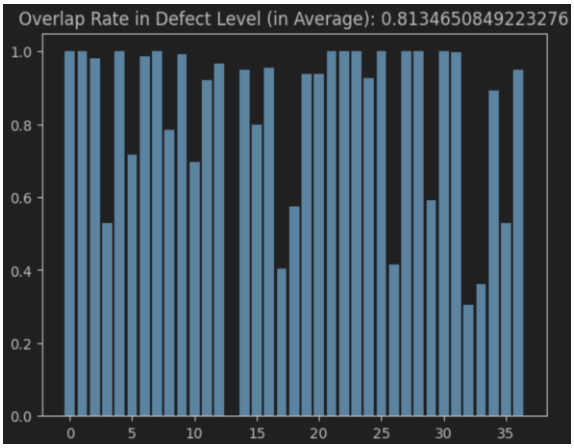


图 3.2-4 为每张图片计算 mask 与标签重叠部分占标签面积的比例，也就是缺陷被覆盖到的比例 recall。可以发现，在图像层面检测缺陷样本的基础之上，除了一个样本以外，其余样本的缺陷部分均能被或多或少地定位到。另外，覆盖率超过 0.5 的样本数为 32/37

总结 3.1 和 3.2：先通过图像层面的分析将缺陷样本和无缺陷样本分开，能够正确找出所有的无缺陷样本，同时缺陷样本的判断准确率为 37/41。对于判断出来的那部分缺陷样本，应用缺陷层面的算法，定位缺陷的大体位置。若以 0.5 作为覆盖率的阈值，则准确率为 32/41；若认为只要覆盖到缺陷即可，那么准确率为 36/41.

3. 3 像素层面

3.2 得出的 mask 使得我们只需要关注样品的一小部分，因此为我们过滤掉了很多噪声。我们可以直接求图像梯度，在经过二值化和形态学处理后，用 mask 过滤掉不关注的部分，得到像素级别的缺陷。

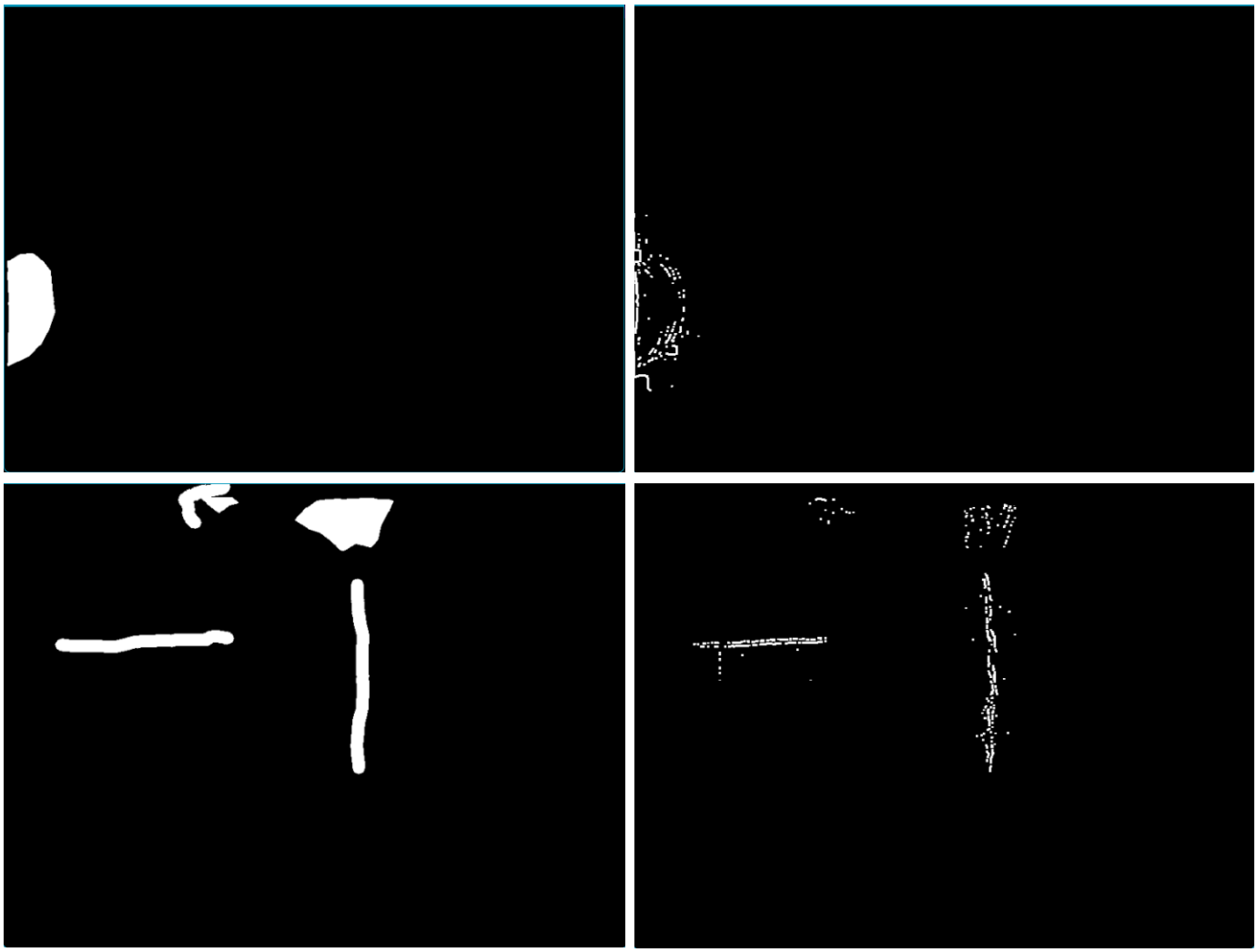


图 3.3-1 标签与像素级缺陷检测结果的对比

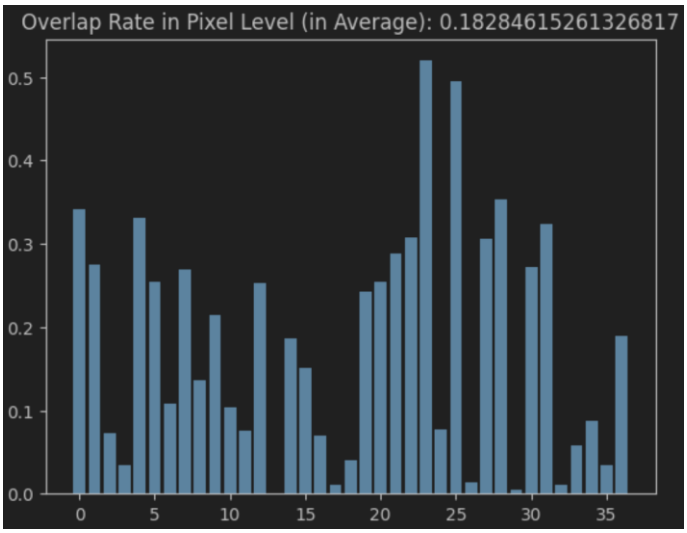


图 3.3-2 像素级缺陷检测结果与标签的覆盖率 recall，平均覆盖率为 0.18，由于像素级检测基于图像的梯度，所以很难把块状缺陷整块标注出来，这是覆盖率比较低的原因。但是检测结果能够表征缺陷的位置和部分形状。

4. 总结

实现了工业样品的倾斜矫正、矩形度计算，以及三个层面的缺陷检测。

在缺陷检测任务中，图像层面的分类准确率为 96%，其中无缺陷样品全部分类正确，缺陷样品准确率在 37/41；在缺陷层面，只对被上一阶段识别出来的缺陷样品做处理，成功定位到缺陷的比率为 36/41，覆盖率高于 0.5 的比率为 32/41，标签的平均被覆盖率为 81%；在像素层面，标签的平均被覆盖率为 18%。

改进想法：

倾斜矫正的算法，在运行时间上可以提升；

在图像层面，灰度共生矩阵的计算太慢，可以探究 15*20 的分法是否是最佳的；而且不是每一小块都需要做灰度共生矩阵分析，可以在计算灰度共生矩阵前，先估计小块的平坦度，再决定是否跳过这一小块，这样能够缩短计算时间；

5. 代码

倾斜校正的代码 measure_rectangle_main.py

```
from utils import *

if not os.path.exists('./data/measure_correctedEdge'):
    os.mkdir('./data/measure_correctedEdge')
    imgFileNames = os.listdir('./data/measure/')
    for name in imgFileNames:
        path = './data/measure/' + name
        img = cv2.imread(path, 0)
        img = 255 - img
        img[:200, :] = 0
        img[800:, :] = 0
        img[img < 220] = 0    #  去噪
        imgCopy = copy.deepcopy(img)
        # cv2.imshow('', cv2.resize(img, dsize=(640, 512)))
        # cv2.waitKey(-1)
        # cv2.destroyAllWindows()
        imgCorrect, rot_angle = tile_correction(img, imgCopy)    #  纠正倾斜，然后用亚像素检测边缘
        cv2.imwrite(f'data/measure_correctedEdge/{name}', imgCorrect)
        print(f'为纠正图像{name}的倾斜,\t 将其旋转{rot_angle}度')

imgFileNames = os.listdir('./data/measure_correctedEdge/')
rectRateList = []
heightList = []
widthList = []
# bad_list = [21, 43]
for i, name in enumerate(imgFileNames):
    path = './data/measure_correctedEdge/' + name
    imgEdge = cv2.imread(path, 0)
    inds = np.where(imgEdge)    #  所有边缘的坐标
    inds = np.stack([inds[0], inds[1]], axis=1)
    inds, imgEdge = thinner_edge(inds, imgEdge)
    inds = gen_contours(inds, imgEdge)
    rMin = np.min(inds[:, 0])    #  最小外接矩形的顶点坐标
    rMax = np.max(inds[:, 0])
    cMin = np.min(inds[:, 1])
    cMax = np.max(inds[:, 1])
    imgEdge = cv2.drawContours(imgEdge, (np.expand_dims(inds[:, ::-1], 1)), -1, color=(128, 128, 128), thickness=1)
    heightList.append(rMax - rMin)
    widthList.append(cMax - cMin)
    tempArea = (rMax - rMin) * (cMax - cMin)    #  最大外接矩形的面具
    ctArea = cv2.contourArea(np.expand_dims(inds, 1))    #  提取到的边缘的面积
    rectangleRate = ctArea / tempArea    #  矩形度计算
    print(f"图像{i}\t{name}\t 的矩形度:\t{rectangleRate}")
    rectRateList.append(rectangleRate)
mean = np.mean(rectRateList)
variance = np.var(rectRateList)

plt.figure()
plt.scatter(rectRateList, np.ones_like(rectRateList))
plt.title(
    f'Distribution of Rectangle Rate    mean={np.round(np.mean(rectRateList), 5)}    variance={np.round(np.var(rectRateList), 8)}')
```

```
plt.figure()
plt.scatter(heightList, np.ones_like(heightList))
plt.title(f'Height    mean={np.round(np.mean(heightList), 5)}    variance={np.round(np.var(heightList), 5)}})')
plt.figure()
plt.scatter(widthList, np.ones_like(widthList))
plt.title(f'Width    mean={np.round(np.mean(widthList), 5)}    variance={np.round(np.var(widthList), 5)}')
plt.show()
print(f'矩形度的均值:{mean}\n 矩形度的方差:{variance}')
```

缺陷检测的代码 defect_detection_main.ipynb

```
#%%
from utils import *
from sklearn.metrics import confusion_matrix, accuracy_score
import cv2
#%%
OK_img, NG_img, NG_labels = gen_dataset()
print('缺陷图倾斜校正:')  # 做倾斜校正
for i, img in enumerate(NG_img):  # 判断 NG 图
    # 倾斜矫正
    label = NG_labels[i]
    img = 255 - img
    img_thres = cv2.Canny(img, 100, 200)
    ok_img, label, angle = tile_correction_2(img_thres, img, label)  # 倾斜校正函数
    # cv2.imshow('',ok_img)
    # cv2.waitKey(-1)
    # cv2.destroyAllWindows()
    label[label > 0] = 255
    ok_img = cv2.resize(ok_img, (800, 600))
    label = cv2.resize(label, (800, 600))
    NG_img[i] = ok_img
    NG_labels[i] = label
    print(f'{i}\t 旋转{angle}度')

print('无缺陷图倾斜校正:')
for i, img in enumerate(OK_img):  # 处理 OK 图
    # 倾斜矫正
    img = 255 - img
    img_thres = cv2.Canny(img, 100, 200)
    ok_img, angle = tile_correction_2(img_thres, img)
    ok_img = cv2.resize(ok_img, (800, 600))
    OK_img[i] = ok_img
    print(f'{i}\t 旋转{angle}度')

save('data/defect_correctedEdge/OK_img.pickle', OK_img)
save('data/defect_correctedEdge/NG_img.pickle', NG_img)
save('data/defect_correctedEdge/NG_labels.pickle', NG_labels)
#%%
# 图像级别的分类
OK_img = load('data/defect_correctedEdge/OK_img.pickle')
NG_img = load('data/defect_correctedEdge/NG_img.pickle')
NG_labels = load('data/defect_correctedEdge/NG_labels.pickle')
print('处理缺陷图...')
thres = 580
bad_assess_list = []
pred = []
l = [1] * 41 + [0] * 50
detect_list = []
mask_list = []
for i, img in enumerate(NG_img):  # 判断 NG 图
    fft1, fft2, mask = eliminate_thin_curve(img)
    # cv2.imshow('fft2',fft2)
    fft1[mask > 0] = 0
    fft2[fft2 < 27] = 0
    k = np.ones((4, 4))
    th, fft2_thres = cv2.threshold(cv2.equalizeHist(fft2), 135, 255, cv2.THRESH_BINARY)
    temp = cv2.morphologyEx(fft2_thres, cv2.MORPH_OPEN, k)
    fft2_thres = fft2_thres - temp
    fft2_thres[mask > 0] = 0
    fft2_thres = cv2.morphologyEx(fft2_thres, cv2.MORPH_OPEN, np.ones((2, 2)))
    assess = fft2_thres.sum() / 255
    bad_assess_list.append(assess)
    mask_list.append(mask)
    # cv2.imshow('fft2_thres', fft2_thres)
    # cv2.waitKey(-1)
```

```
        # cv2.destroyAllWindows()
        if assess > thres:
            pred.append(1)
            detect_list.append((fft1, NG_labels[i]))
        else:
            pred.append(0)
            detect_list.append((fft1, 0))

print('处理无缺陷图...')
good_assess_list = []
# OK_mask_list = []
for i, img in enumerate(OK_img):   # 处理 OK 图
        fft1, fft2, mask = eliminate_thin_curve(img)
        # cv2.imshow('fft2',fft2)
        fft1[mask > 0] = 0
        fft2[fft2 < 27] = 0
        k = np.ones((4, 4))
        th, fft2_thres = cv2.threshold(cv2.equalizeHist(fft2), 135, 255, cv2.THRESH_BINARY)
        temp = cv2.morphologyEx(fft2_thres, cv2.MORPH_OPEN, k)
        fft2_thres = fft2_thres - temp
        fft2_thres[mask > 0] = 0
        fft2_thres = cv2.morphologyEx(fft2_thres, cv2.MORPH_OPEN, np.ones((2, 2)))
        assess = fft2_thres.sum() / 255
        good_assess_list.append(assess)
        mask_list.append(mask)
        # cv2.imshow('fft2_thres', fft2_thres)
        # cv2.waitKey(-1)
        if assess > thres:
            pred.append(1)
            detect_list.append((fft1, NG_labels[i]))
        else:
            pred.append(0)
            detect_list.append((fft1, 0))

plt.figure(figsize=(20, 3))
plt.scatter(bad_assess_list, np.zeros_like(bad_assess_list), s=40, alpha=1, c='blue')
plt.scatter(good_assess_list, np.zeros_like(good_assess_list) + 1, s=40, alpha=1, c='yellow')
plt.legend(['NG', 'OK'])
plt.xlabel('Defect Value')
plt.show()

acc = accuracy_score(l, pred)
conf = confusion_matrix(l, pred)
print(f'准确度:{acc}\n 混淆矩阵:\n{conf}')
#%%
rectangles_list = []
labels_list = []
img_list = []
for i, (img, label) in enumerate(detect_list):
        if type(label) == np.ndarray:   # 在图像层面是否被标识为负样本，是则继续处理
            mask = mask_list[i]
            bn_r = 15
            bn_c = 20
            img_balance, _ = light_balance(img, 80, 60, 150)
            img_balance[mask > 0] = 0
            th, img_thres = cv2.threshold(img_balance, 140, 255, cv2.THRESH_BINARY)
            contrast_threshold = 95
            img_glcm, rectangles = GLCM(img_balance, bn_r, bn_c, contrast_threshold)
            cv2.imwrite(f'data/glcm/glcm_img/{i}.png', img_glcm)
            rectangles_list.append(rectangles)
            labels_list.append(label)
            img_list.append(img)
            # cv2.imshow(f'认为图像{i}存在缺陷，灰度共生矩阵的对比度', img_glcm)
            # cv2.imshow(f'缺陷的标签', label)
        else:
            # cv2.imshow(f'认为图像{i}不存在缺陷，图像样例', img)
            pass
        print(f'\r 缺陷级别的检测{i + 1}/91', end='\r')
        # cv2.waitKey(-1)
        # cv2.destroyAllWindows()
save('data/detected_rectangle/rectangles_list.pickle', rectangles_list)
save('data/detected_rectangle/labels_list.pickle', labels_list)
save('data/detected_rectangle/img_list.pickle', img_list)
#%%
#  像素层面的检测
```

```
rectangles_list = load('data/detected_rectangle/rectangles_list.pickle')  # 加载缺陷层面的检测结果
labels_list = load('data/detected_rectangle/labels_list.pickle')
img_list = load('data/detected_rectangle/img_list.pickle')
overlap_rate_list = []
pixel_overlap_rate_list = []
for i, label in enumerate(labels_list):
    rectangles = rectangles_list[i]
    img = img_list[i]
    ol_rate, pol_rate = detect_pixel(img, rectangles, label)
    overlap_rate_list.append(ol_rate)
    pixel_overlap_rate_list.append(pol_rate)
plt.figure()
plt.bar(np.arange(len(overlap_rate_list)), overlap_rate_list)
plt.title(f'Overlap Rate in Defect Level (in Average): {np.mean(overlap_rate_list)}')
plt.figure()
plt.bar(np.arange(len(pixel_overlap_rate_list)), pixel_overlap_rate_list)
plt.title(f'Overlap Rate in Pixel Level (in Average): {np.mean(pixel_overlap_rate_list)}')
print(f'在缺陷层面，检测覆盖率高于 0.5 的样本数:{np.sum(np.array(overlap_rate_list) > 0.5)}/{len(overlap_rate_list)}')
print(f'在缺陷层面，检测覆盖率高于 0 的样本数:{np.sum(np.array(overlap_rate_list) > 0)}/{len(overlap_rate_list)}')
print(f'在像素层面，检测的像素平均覆盖率:{np.mean(pixel_overlap_rate_list)}')
```

自定义函数 utils.py

```
import copy
import os
import cv2
import matplotlib.pyplot as plt
import numpy as np
import numpy.fft as fft
import pickle
from skimage.feature import graycomatrix, graycoprops


def detect_pixel(img, rect_list, label):  # 像素级别的检测
    mask = np.zeros_like(img)
    for rect in rect_list:  # 生成掩膜 mask
        r1, r2, c1, c2 = rect[0][1], rect[1][1], rect[0][0], rect[1][0]
        mask[r1:r2, c1:c2] = 255
    mask = cv2.dilate(mask, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (30, 30)))

    overlap = np.bitwise_and(mask, label)
    overlap_rate = np.sum(overlap) / np.sum(label)
    img_masked = copy.deepcopy(img)
    img_masked[mask == 0] = 0
    img = cv2.GaussianBlur(img, (3, 3), 0.5)
    img_grad = cv2.Laplacian(img, 0)
    th, img_grad = cv2.threshold(img_grad, 10, 255, cv2.THRESH_BINARY)
    k1 = np.array([[1, 1]])
    k2 = np.array([[1], [1]])
    img_grad = cv2.morphologyEx(img_grad, cv2.MORPH_OPEN, k2)
    img_grad = cv2.morphologyEx(img_grad, cv2.MORPH_OPEN, k1)
    img_grad = cv2.dilate(img_grad, k2)
    img_grad = cv2.dilate(img_grad, k1)
    img_grad[mask == 0] = 0
    pixel_overlap = np.bitwise_and(img_grad, label)
    pixel_overlap_rate = np.sum(pixel_overlap) / np.sum(label)
    # cv2.imshow('原图', img)
    # cv2.imshow('真实的标签', label)
    # cv2.imshow('套上掩膜的图像', img_masked)
    # cv2.imshow('检测出来的缺陷', img_grad)
    # cv2.waitKey(-1)
    # cv2.destroyAllWindows()
    return overlap_rate, pixel_overlap_rate


def gen_dataset():  # 加载并整理测数据
    NG_img = []
    NG_labels = []
    NG_img_path = 'data/defect/NG/imagenormal/'
    label_path = 'data/defect/NG/imagedrawn/'
    label_names = os.listdir(label_path)
    for i in range(len(label_names)):
        NG_img.append(cv2.imread(f'{NG_img_path}{i}.bmp', 0))
        NG_labels.append(cv2.imread(f'{label_path}{i}.png', 0))
```

```
    OK_path = 'data/defect/OK/'
    OK_img = []
    OK_img_names = os.listdir(OK_path)
    for i in range(len(OK_img_names)):
        OK_img.append(cv2.imread(f'{OK_path}{i}.bmp', 0))

    return OK_img, NG_img, NG_labels


def light_balance(image, num_blocks_c, num_blocks_r, dst_avg=None):  # 对两个方向分别做光照均衡，稍微有点用
    mask = (image != 0).astype(np.int32)
    image = image.astype(np.double)
    if dst_avg is None:
        avg = np.mean(image)
    else:
        avg = dst_avg
    r, c = image.shape[:2]
    block_size_c = int(c / num_blocks_c)
    block_size_r = int(r / num_blocks_r)

    l2 = []
    for j in range(num_blocks_c):
        mask_2 = np.zeros_like(image, np.int32)
        c1 = j * block_size_c
        c2 = c1 + block_size_c
        mask_2[:, c1:c2] = 1
        mask_2[mask == 0] = 0
        if np.sum(mask_2) == 0:
            l2.append(0)
            continue
        block_image = image[mask_2 > 0]
        l2.append(np.mean(block_image))
    light = np.array(l2) - avg
    light = cv2.resize(light, (c, r), interpolation=cv2.INTER_CUBIC)
    res = image - light
    res[res < 0] = 0
    res[mask == 0] = 0

    l2 = []
    for j in range(num_blocks_r):
        mask_2 = np.zeros_like(image, np.int32)
        r1 = j * block_size_r
        r2 = r1 + block_size_r
        mask_2[r1:r2, :] = 1
        mask_2[mask == 0] = 0
        if np.sum(mask_2) == 0:
            l2.append(0)
            continue
        block_image = image[mask_2 > 0]
        l2.append(np.mean(block_image))
    light = np.array(l2) - avg
    light = cv2.resize(light, (c, r), interpolation=cv2.INTER_CUBIC)
    res = image - light
    res[res < 0] = 0
    res[mask == 0] = 0
    res = res.astype(np.uint8)
    return res, avg


def calculate_glcm(image, distances, angles):  # 灰度共生矩阵计算
    image[image < 0] = 0
    glcm = graycomatrix(image, distances, angles, levels=256, symmetric=True, normed=True)
    return glcm


def extract_texture_features(glcm):
    contrast = graycoprops(glcm, 'contrast')
    energy = graycoprops(glcm, 'energy')
    correlation = graycoprops(glcm, 'correlation')
    homogeneity = graycoprops(glcm, 'homogeneity')
    return contrast, energy, correlation, homogeneity


def GLCM(image, num_blocks_r, num_blocks_c, threshold):  # 分块做灰度共生矩阵分析，并将感兴趣的区域筛选出来
    img_copy = copy.deepcopy(image).copy()
```

```
    image = image.astype(np.int32)
    r, c = image.shape[:2]
    block_size_c = int(c / num_blocks_c)
    block_size_r = int(r / num_blocks_r)
    res_rect_list = []
    for i in np.arange(0, num_blocks_r):
        for j in np.arange(0, num_blocks_c):
            if i == 0 and j == 0 or i == 0 and j == num_blocks_c - 1 or i == num_blocks_r - 1 and j == 0 or i == num_blocks_r - 1 and j ==
num_blocks_c - 1:
                continue
            r1 = i * block_size_r
            r2 = r1 + block_size_r
            c1 = j * block_size_c
            c2 = c1 + block_size_c
            block_image = image[r1:r2, c1:c2]
            if not np.any(block_image):
                continue
            block_image[block_image == 0] = np.mean(block_image[block_image != 0])
            g = calculate_glcm(block_image, [1], [0, np.pi / 2, np.pi / 4, np.pi * 3 / 4])
            contrast, energy, correlation, homogeneity = extract_texture_features(g)
            if np.max(contrast) > threshold:
                img_copy = cv2.rectangle(img_copy, (c1, r1), (c2, r2), (255, 255, 255), 2)
                res_rect_list.append([[c1, r1], [c2, r2]])
            cv2.putText(img_copy,
                        f'{np.round(np.max(contrast), 1)}',
                        (int(0.75 * c1 + 0.25 * c2), int(0.5 * r1 + 0.5 * r2)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.25, (255, 255, 255), 1)
    return img_copy, res_rect_list


def subpixel_edge(grayImage):   # 亚像素边缘检测
    kernels_Num = 8
    kernels = ['_' for i in range(kernels_Num)]
    kernels[0] = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], dtype=int)
    kernels[1] = np.array([[2, 1, 0], [1, 0, -1], [0, -1, -2]], dtype=int)
    kernels[2] = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]], dtype=int)
    kernels[3] = np.array([[0, -1, -2], [1, 0, -1], [2, 1, 0]], dtype=int)
    kernels[4] = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], dtype=int)
    kernels[5] = np.array([[-2, -1, 0], [-1, 0, 1], [0, 1, 2]], dtype=int)
    kernels[6] = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], dtype=int)
    kernels[7] = np.array([[0, 1, 2], [-1, 0, 1], [-2, -1, 0]], dtype=int)

    gradients = ['_' for i in range(kernels_Num)]
    for i in range(kernels_Num):
        gradients[i] = cv2.filter2D(grayImage, cv2.CV_16S, kernels[i])

    angle_list = [270, 315, 0, 45, 90, 135, 180, 225]
    amplitude = np.full(grayImage.shape, 0)
    angle = np.full(grayImage.shape, -64)

    for r in range(grayImage.shape[0]):
        pAmp = amplitude[r]
        pAng = angle[r]

        pGrad = ['_' for i in range(kernels_Num)]
        for i in range(kernels_Num):
            pGrad[i] = gradients[i][r]   # 不同方向（8 个方向）上同一行的梯度
        for c in range(grayImage.shape[1]):
            for i in range(kernels_Num):   # 每个方向的梯度比较
                if (pAmp[c] < pGrad[i][c]):
                    pAmp[c] = pGrad[i][c]
                    pAng[c] = angle_list[i]

    edge = np.full(grayImage.shape, 0)
    edge.astype('uint8')
    thres = 100   # 阈值   设置最小幅度值
    for r in range(1, grayImage.shape[0] - 1):
        pAmp1 = amplitude[r - 1]
        pAmp2 = amplitude[r]
        pAmp3 = amplitude[r + 1]

        pAng = angle[r]
        pEdge = edge[r]
        for c in range(1, grayImage.shape[1] - 1):
```

```
            if (pAmp2[c] < thres):
                continue
            if pAng[c] == 270:
                if pAmp2[c] > pAmp1[c] and pAmp2[c] >= pAmp3[c]:
                    pEdge[c] = 255
            elif pAng[c] == 90:
                if pAmp2[c] >= pAmp1[c] and pAmp2[c] > pAmp3[c]:
                    pEdge[c] = 255
            elif pAng[c] == 315:
                if pAmp2[c] > pAmp1[c - 1] and pAmp2[c] >= pAmp3[c + 1]:
                    pEdge[c] = 255
            elif pAng[c] == 135:
                if pAmp2[c] >= pAmp1[c - 1] and pAmp2[c] > pAmp3[c + 1]:
                    pEdge[c] = 255
            elif pAng[c] == 0:
                if pAmp2[c] > pAmp2[c - 1] and pAmp2[c] >= pAmp2[c + 1]:
                    pEdge[c] = 255
            elif pAng[c] == 180:
                if pAmp2[c] >= pAmp2[c - 1] and pAmp2[c] > pAmp2[c + 1]:
                    pEdge[c] = 255
            elif pAng[c] == 45:
                if pAmp2[c] >= pAmp1[c + 1] and pAmp2[c] > pAmp3[c - 1]:
                    pEdge[c] = 255
            elif pAng[c] == 225:
                if pAmp2[c] > pAmp1[c + 1] and pAmp2[c] >= pAmp3[c - 1]:
                    pEdge[c] = 255

edge = cv2.convertScaleAbs(edge)
root2 = np.sqrt(2.0)
tri_list = [[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0]]
for i in range(kernels_Num):
    tri_list[0][i] = np.cos(angle_list[i] * np.pi / 180.0)
    tri_list[1][i] = -np.sin(angle_list[i] * np.pi / 180.0)
vPts = []

for r in range(1, grayImage.shape[0] - 1):
    pAmp1 = amplitude[r - 1]
    pAmp2 = amplitude[r]
    pAmp3 = amplitude[r + 1]

    pAng = angle[r]
    pEdge = edge[r]
    for c in range(1, grayImage.shape[1] - 1):
        if (pEdge[c]):
            nAngTmp = 0
            dTmp = 0
            if pAng[c] == 270:
                nAngTmp = 0
                dTmp = (pAmp1[c] - pAmp3[c]) / (pAmp1[c] + pAmp3[c] - 2 * pAmp2[c]) * 0.5
            elif pAng[c] == 90:
                nAngTmp = 4
                dTmp = -(pAmp1[c] - pAmp3[c]) / (pAmp1[c] + pAmp3[c] - 2 * pAmp2[c]) * 0.5
            elif pAng[c] == 315:
                nAngTmp = 1
                dTmp = (pAmp1[c - 1] - pAmp3[c + 1]) / (pAmp1[c - 1] + pAmp3[c + 1] - 2 * pAmp2[c]) * root2 * 0.5
            elif pAng[c] == 135:
                nAngTmp = 5
                dTmp = -(pAmp1[c - 1] - pAmp3[c + 1]) / (pAmp1[c - 1] + pAmp3[c + 1] - 2 * pAmp2[c]) * root2 * 0.5
            elif pAng[c] == 0:
                nAngTmp = 2
                dTmp = (pAmp2[c - 1] - pAmp2[c + 1]) / (pAmp2[c - 1] + pAmp2[c + 1] - 2 * pAmp2[c]) * 0.5
            elif pAng[c] == 180:
                nAngTmp = 6
                dTmp = -(pAmp2[c - 1] - pAmp2[c + 1]) / (pAmp2[c - 1] + pAmp2[c + 1] - 2 * pAmp2[c]) * 0.5
            elif pAng[c] == 45:
                nAngTmp = 3
                dTmp = (pAmp3[c - 1] - pAmp1[c + 1]) / (pAmp1[c + 1] + pAmp3[c - 1] - 2 * pAmp2[c]) * root2 * 0.5
            elif pAng[c] == 225:
                nAngTmp = 7
                dTmp = -(pAmp3[c - 1] - pAmp1[c + 1]) / (pAmp1[c + 1] + pAmp3[c - 1] - 2 * pAmp2[c]) * root2 * 0.5

            x = c + dTmp * tri_list[0][nAngTmp]
            y = r + dTmp * tri_list[1][nAngTmp]
            vPts.append([x, y])
tmpImg = np.zeros(grayImage.shape, dtype=np.uint8)
```

```
    for x, y in vPts:
        tmpImg[int(y), int(x)] = 255
    return tmpImg


def tile_correction(img, imgRaw):  # 倾斜矫正 1
    img = np.pad(img, pad_width=((0, 0), (128, 128)))
    imgRaw = np.pad(imgRaw, pad_width=((0, 0), (128, 128)))
    angleList = np.round(np.arange(80, 100, 0.1), 1)
    resList = []
    for angle in angleList:
        mat = cv2.getRotationMatrix2D((640, 640), angle, 1)
        imgRot = cv2.warpAffine(img, mat, (1280, 1280))
        imgSum = np.any(imgRot, axis=0)
        resList.append(np.sum(imgSum > 0))
    expectedAngle = angleList[np.argmin(resList)]
    mat = cv2.getRotationMatrix2D((640, 640), expectedAngle, 1)
    imgCorrect = cv2.warpAffine(imgRaw, mat, (1280, 1280))
    imgCorrect = subpixel_edge(imgCorrect[:, 200:800])
    imgCorrect = cv2.morphologyEx(imgCorrect, cv2.MORPH_CLOSE, np.ones((3, 3)))
    return imgCorrect, expectedAngle


def tile_correction_2(thres, imgRaw, label=None):  # 倾斜校正 2，给缺陷检测任务用的，和上面的大致差不多
    r, c = thres.shape[0], thres.shape[1]
    dst_size = 1100
    pad_width_c = int((dst_size - c) / 2)
    pad_width_r = int((dst_size - r) / 2)
    thres = np.pad(thres, pad_width=((pad_width_r, pad_width_r), (pad_width_c, pad_width_c)))
    label = np.pad(label,
                   pad_width=((pad_width_r, pad_width_r), (pad_width_c, pad_width_c))) if label is not None else None
    imgRaw = np.pad(imgRaw, pad_width=((pad_width_r, pad_width_r), (pad_width_c, pad_width_c)))
    angleList = np.round(np.arange(80, 100, 0.1), 1)
    resList = []
    for angle in angleList:
        mat = cv2.getRotationMatrix2D((481, 372), angle, 1)
        imgRot = cv2.warpAffine(thres, mat, (1100, 1100))
        imgAny = np.any(imgRot, axis=0)
        resList.append(np.sum(imgAny))
    expectedAngle = angleList[np.argmin(resList)]
    mat = cv2.getRotationMatrix2D((550, 550), expectedAngle, 1)
    thresCorrect = cv2.warpAffine(thres, mat, (1100, 1100))
    th, thresCorrect = cv2.threshold(thresCorrect, 127, 255, cv2.THRESH_BINARY)
    thresCorrect = thresCorrect[pad_width_c:-pad_width_c, pad_width_r:-pad_width_r]
    inds = np.where(thresCorrect)
    inds = np.stack([inds[0], inds[1]]).T
    # a = 35
    r_top_clip = inds[:, 0].min()
    r_button_clip = inds[:, 0].max()
    c_left_clip = inds[:, 1].min()
    c_right_clip = inds[:, 1].max()

    imgCorrect = cv2.warpAffine(imgRaw, mat, (1100, 1100))
    imgCorrect = imgCorrect[pad_width_c:-pad_width_c, pad_width_r:-pad_width_r]
    labelCorrect = cv2.warpAffine(label, mat, (1100, 1100)) if label is not None else None
    labelCorrect = labelCorrect[pad_width_c:-pad_width_c, pad_width_r:-pad_width_r] if label is not None else None
    # cv2.imshow('裁剪之前的图', imgCorrect)
    imgCorrect = imgCorrect[:r_button_clip]
    imgCorrect = imgCorrect[r_top_clip:]
    imgCorrect = imgCorrect[:, :c_right_clip]
    imgCorrect = imgCorrect[:, c_left_clip:]
    # cv2.imshow('label', labelCorrect)
    # cv2.imshow('correctimg', imgCorrect)
    # cv2.waitKey(-1)
    # cv2.destroyAllWindows()
    if label is not None:
        labelCorrect = labelCorrect[:r_button_clip]
        labelCorrect = labelCorrect[r_top_clip:]
        labelCorrect = labelCorrect[:, :c_right_clip]
        labelCorrect = labelCorrect[:, c_left_clip:]
        # cv2.imshow('label', labelCorrect)
        # cv2.imshow('correctimg', imgCorrect)
        # cv2.waitKey(-1)
        # cv2.destroyAllWindows()
        return imgCorrect, labelCorrect, expectedAngle
```

```
    return imgCorrect, expectedAngle


def gen_contours(inds, img):  # 顺时针生成正确顺序的边缘
    first_ind = copy.deepcopy(inds[0, :])
    res_list = [first_ind]
    present_ind = copy.deepcopy(first_ind)
    inds = inds[1:, :]
    operate_list = np.array(
        [[0, -1], [1, 1], [0, 1], [0, 1], [1, -1], [1, -1], [0, -1], [0, -1], [0, -1], [1, 1], [1, 1],
         [1, 1], [0, 1], [0, 1], [0, 1], [0, 1], [1, -1], [1, -1], [1, -1], [1, -1], [0, -1], [0, -1],
         [0, -1], [0, -1]])
    while True:  # 按顺时针往 8+16 个方向找下一个邻接的像素边缘
        sign = 0
        for p in operate_list:
            present_ind[p[0]] += p[1]
            n = np.argwhere(np.all(present_ind == inds, axis=1))
            if len(n):
                if np.all(present_ind == first_ind):
                    break
                inds = np.delete(inds, n, axis=0)
                res_list.append(copy.deepcopy(present_ind))
                sign = 1
                break
        if not sign:
            break  # 如果 8 个邻域都没有找到像素点，就提前结束
    res_list.append(first_ind)
    res_list = np.array(res_list)
    return res_list


def thinner_edge(inds, img):  # 为了尽量保证 subpixel 边缘的连续性，对边缘做了闭运算，这一定程度上加粗了边缘，所以需要重新细化
    j = 0
    for i in range(inds.shape[0]):
        ind = inds[j]
        r = ind[0]
        c = ind[1]
        piece = [img[r, c], img[r - 1, c], img[r, c + 1]]
        if np.all(piece):
            inds = np.delete(inds, j, axis=0)
            img[r, c] = 0
            j -= 1
            continue
        piece = [img[r, c], img[r - 1, c], img[r, c - 1]]
        if np.all(piece):
            inds = np.delete(inds, j, axis=0)
            img[r, c] = 0
            j -= 1
            continue
        piece = [img[r, c], img[r + 1, c], img[r, c + 1]]
        if np.all(piece):
            inds = np.delete(inds, j, axis=0)
            img[r, c] = 0
            j -= 1
            continue
        piece = [img[r, c], img[r + 1, c], img[r, c - 1]]
        if np.all(piece):
            inds = np.delete(inds, j, axis=0)
            img[r, c] = 0
            j -= 1
            continue
        j += 1
    return inds, img


def eliminate_thin_curve(ok_img):  # 在频域中，对于水平和垂直的分量，只保留低频部分
    img_f = fft.fftshift(fft.fft2(ok_img))
    cr, cc = int(img_f.shape[0] / 2), int(img_f.shape[1] / 2)
    k = 2
    j = 1
    # plt.figure()
    # plt.subplot(121)
    # plt.imshow(-np.log10(np.abs(img_f)+1),cmap='gray')
    img_f[cr - k:cr + k, :cc - j] /= np.arange(1, img_f[cr - k:cr + k, :cc - j].shape[1] + 1)[::-1]
```

```
    img_f[cr - k:cr + k, cc + j:] /= np.arange(1, img_f[cr - k:cr + k, cc + j:].shape[1] + 1)
    # img_f[cr - k:cr + k, cc + j:] /= np.arange(1, img_f[cr - k:cr + k, cc + j:].shape[1] + 1)
    # plt.subplot(122)
    # plt.imshow(-np.log10(np.abs(img_f) + 1),cmap='gray')
    # plt.show()
    img_recon_1 = np.abs(fft.ifft2(fft.ifftshift(img_f))).astype(np.uint8)
    th, img_recon_1_thres = cv2.threshold(img_recon_1, 125, 255, cv2.THRESH_BINARY)
    img_recon_1_thres[:17] = 255
    img_recon_1_thres[583:] = 255
    kernel1 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 5))
    kernel2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))
    img_recon_1_thres = cv2.morphologyEx(img_recon_1_thres, cv2.MORPH_OPEN, kernel1)
    img_recon_1_thres = cv2.morphologyEx(img_recon_1_thres, cv2.MORPH_CLOSE, kernel2)
    img_recon_1[img_recon_1_thres > 0] = 0
    img_f = fft.fftshift(fft.fft2(img_recon_1))
    img_f[cr - k:cr + k, :cc - j] = 0
    img_f[cr - k:cr + k, cc + j:] = 0
    img_f[:cr - j, cc - k:cc + k] = 0
    img_f[cr - j:, cc - k:cc + k] = 0
    img_recon_2 = np.abs(fft.ifft2(fft.ifftshift(img_f))).astype(np.uint8)
    img_recon_1_thres = cv2.dilate(img_recon_1_thres, np.ones((9, 9)))
    img_recon_2[img_recon_1_thres > 0] = 0
    return img_recon_1, img_recon_2, img_recon_1_thres


def save(file_path, lst):
    with open(file_path, 'wb') as file:
        pickle.dump(lst, file)


def load(file_path):
    with open(file_path, 'rb') as f:
        data = pickle.load(f)
    return data
```