# COMP9517 Assignment 1

## Task 1: Iso-data Intensity Thresholding

In task1, iso-data thresholding is used to find a threshold for the grayscale image input.

(1) Select an initial threshold value. I tried 127(255//2) and the mean of gray values of all pixels. It is found that the latter has more iterations than the former after experiments. Thus, 127 is chosen. Create an empty list thres_list to record the threshold value t at every iteration.
(2) Append t into the list thres_list and
compute a new threshold t' = (image[image < t].mean() + image[image >= t].mean())/2.
(3) Compare the difference between t and t' against a small epsilon value. I set epsilon value = 0.1 because the threshold will be a positive integer.
(4) If the difference go beyond the epsilon, set t = t' and go back to step(2).
(5) Set image[image < t] = 255 and image[image >= t] = 0.
(6) According to the image array and list thres_list, generate the binary image and plot the threshold value t at every iteration on a graph.

**Results on the provided images:**

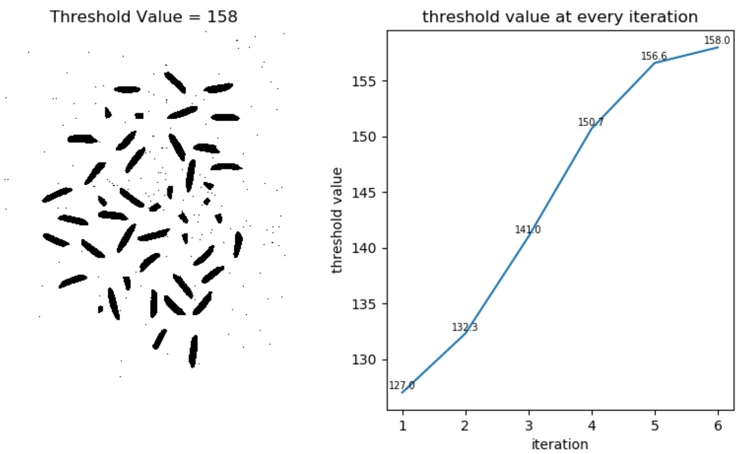


Figure1: rice_img1_Task1.png

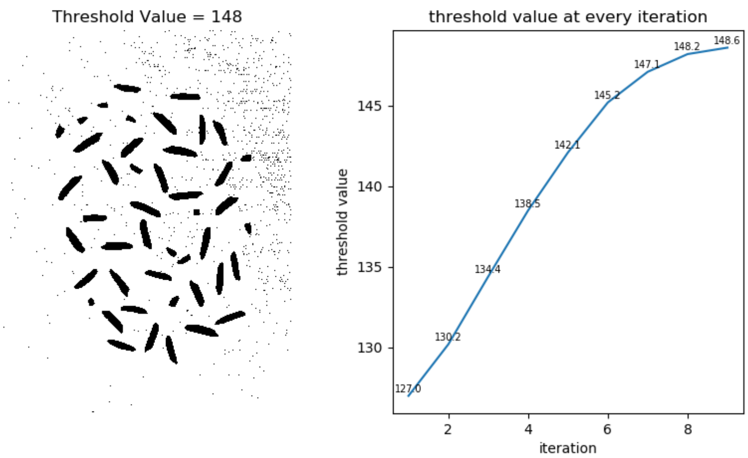


Figure2: rice_img2_Task1.png

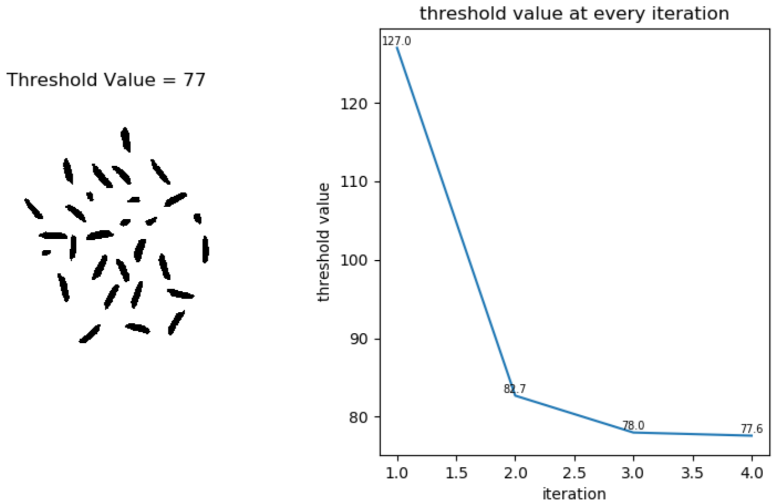


Figure3: rice_img6_Task1.png

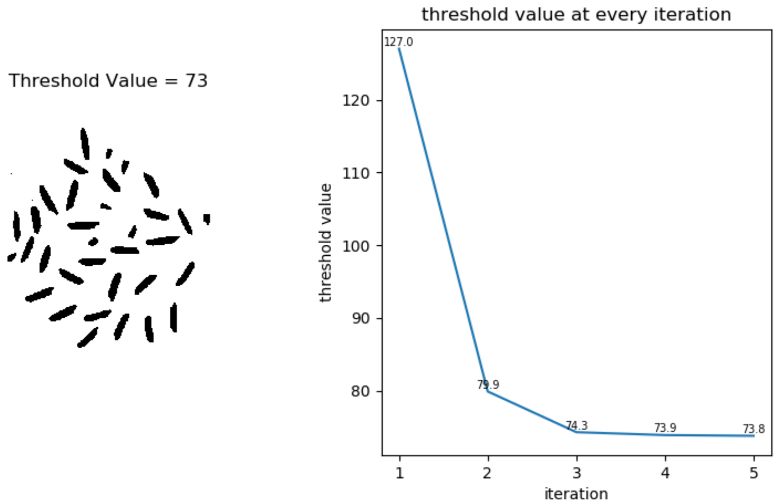


Figure4: rice_img7_Task1.png

## Task 2: Counting the rice kernels

Firstly, apply median filter to remove noisy pixels. I set the kernel size (ksize) = 5, which works better than 3 and 7. The implementation of the median filter is explained below.

(1) Compute the coordinate range of the pixels under the kernel area. For some image coordinates, illegal coordinates may be generated under the kernel area. Thus, value interval should be computed. Assume row, column = image[:2]. For all (x, y) in image,
‘up’ = max(0, x-ksize//2), ‘down’ = min(row-1, y+ksize//2),
‘left’ = max(0, y-ksize//2), ‘right’ = min(column-1, y+ksize//2).
(2) Replace elements with the median value of image[up: down+1, left: right+1].

Secondly, apply two-pass connected components algorithm to count the rice kernels on the filtered binary image. The implementation is explained below.

```
algorithm TwoPass(image)
    labels = structure with dimensions of image, initialized with 0
    equivalence = []    # record the equivalence between the index+1 and equivalence[index]
    label = 0
    # First Pass
    for row in image:
        for column in row:
            if image[row][column] is not background
                neighbour_label = [connected elements which are not equal 0]
                if neighbour_label is empty
                    label += 1, equivalence.append(label), labels[h][w] = label
                else
                    labels[h][w] = min(neighbour_label)
                    for l in neighbour_label:
                        equivalence[l-1] = equivalence[min(neighbour_label)-1]
    # Second Pass
    lowest_label = []    # record the smallest equivalent labels
    for row in image:
        for column in row:
        if image[row][column] is not background
            # find the smallest label in equivalent labels
            while equivalence[labels[row][column]-1] not equal labels[row][column]:
                labels[row][column] = equivalence[labels[row][column]-1]
            if labels[row][column] not in lowest_label
                lowest_label.append(labels[row][column])
    rice_kernel_num = length of lowest_label
    return labels, rice_kernel_num
```

**Results on the provided images (rice kernel numbers shown as the title of the images):**

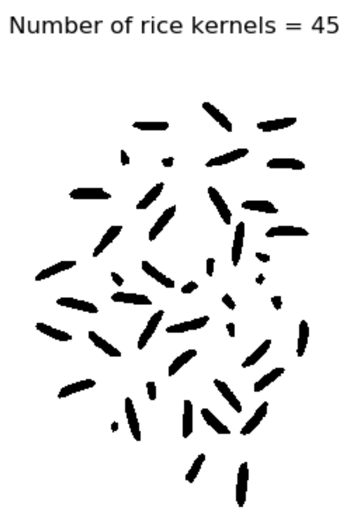


Figure5: rice_img1_Task2.png

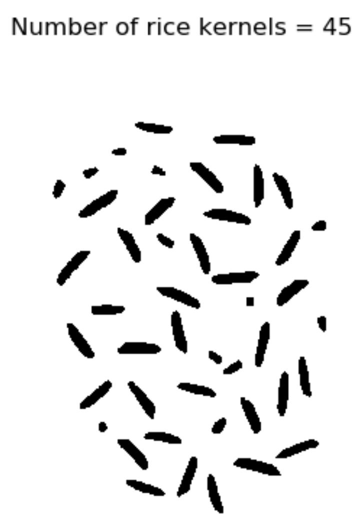


Figure6: rice_img2_Task2.png


Figure7: rice_img6_Task2.png


Figure8: rice_img7_Task2.png

## Task 3: Percentage of damaged rice kernels

To implement the task3, threshold value 'min_area' should be chosen. In my code, the average area of connected components labelled from task2 is chosen to be the 'min_area' of each image(the value can be computed in task2 and this implementation code is annotated in my code). This average value is midway between the area of damaged kernels and whole kernels. **Thus, min_area in rice_img1.png = 649, in rice_img2.png = 669, in rice_img6 = 207, in rice_img7 = 280.**

```
Algorithm task3(labelled_image, min_area)
    label_area = a dictionary recording the pixel area of each label
    damaged_label = a list storing label value of components of damaged kernels
    for row in labelled_image:
        for column in labelled_image:
            if label_area[labelled_image[row][column]] <= min_area
                if labelled_image[row][column] not equal 0 and not in damaged_label
                    damaged_label.append(labelled_image[row][column])
                labelled_image[row][column] = 255
            else
                labelled_image[row][column] = 0
    percentage = length of damaged_label / length of label_area.keys() * 100
    return labelled_image, percentage
```

**Results on the provided images (percentage shown as the title of the images):**

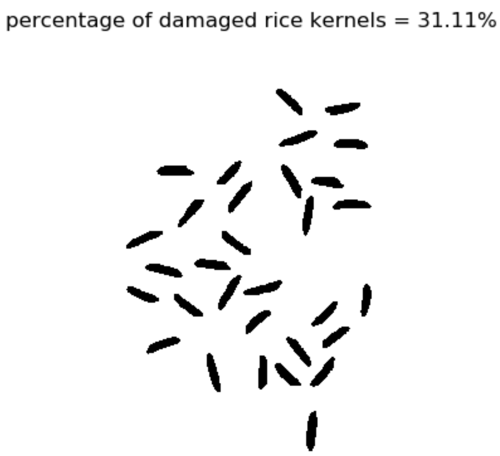


Figure9: rice_img1_Task3.png

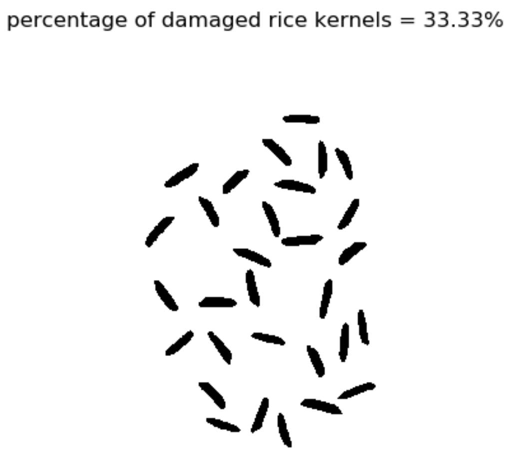


Figure10: rice_img2_Task3.png

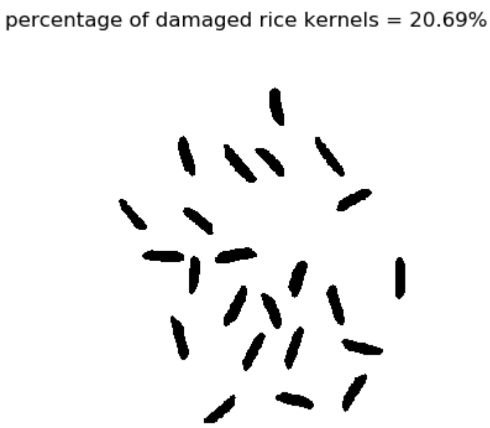


Figure11: rice_img6_Task3.png

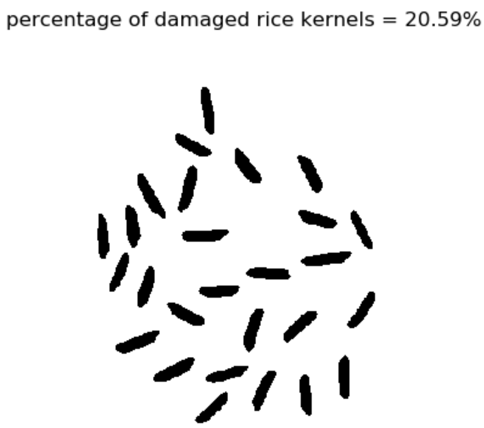


Figure12: rice_img7_Task3.png