Task 1: Iso-data Intensity Thresholding

Iso-data intensity thresholding is an intensity thresholding method where an initial threshold is chosen (usually the mean of all pixel values of the image) and it is updated based on the mean of white and black pixels in the image in order to convert the original image into a binary image. In this assignment the output image should be such that all the rice kernels in the image should be black (pixel value 0) and the background should be white (pixel value 255). The threshold is updated until it converges, that is, the difference between latest threshold and previous threshold is less than epsilon (which I have chosen to be 0.01).

Initially the image is converted to gray scale. Hence, only one channel was considered for every calculation. Upon threshold convergence, the final threshold is then obtained.

This threshold is displayed in the output image as shown below for input image '*rice_img6.png',*

Figure 1 Original image

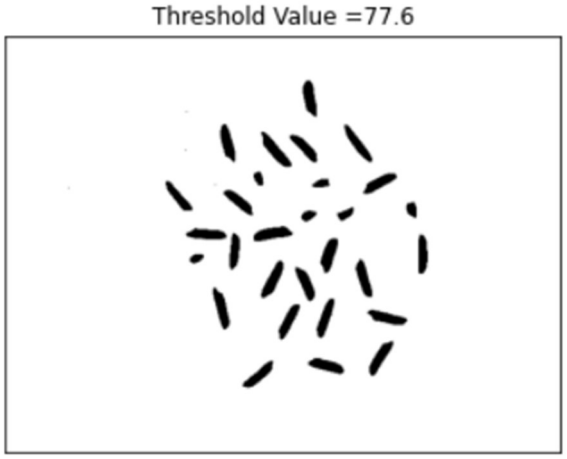


Figure 2 Binary image

Following is a graph that displays the changes in threshold value after every iteration (left image). It can be observed that the threshold keeps increasing up till iteration 4 and seems to remain steady till from iterations 4 to 8. This results in a threshold value of 77.6. An intermediate image of the rice kernels is as shown (right image):

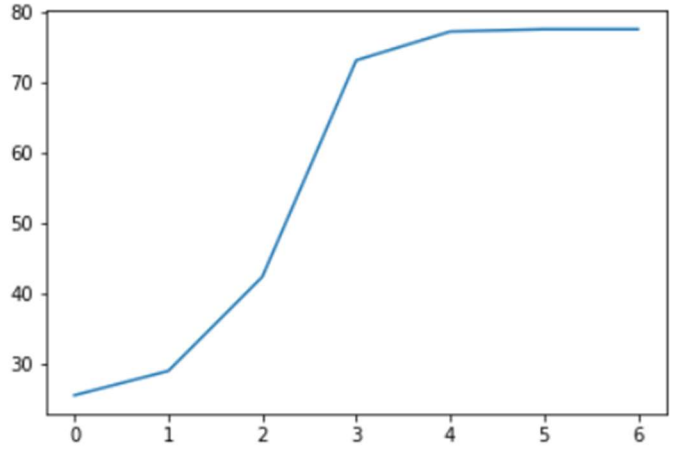
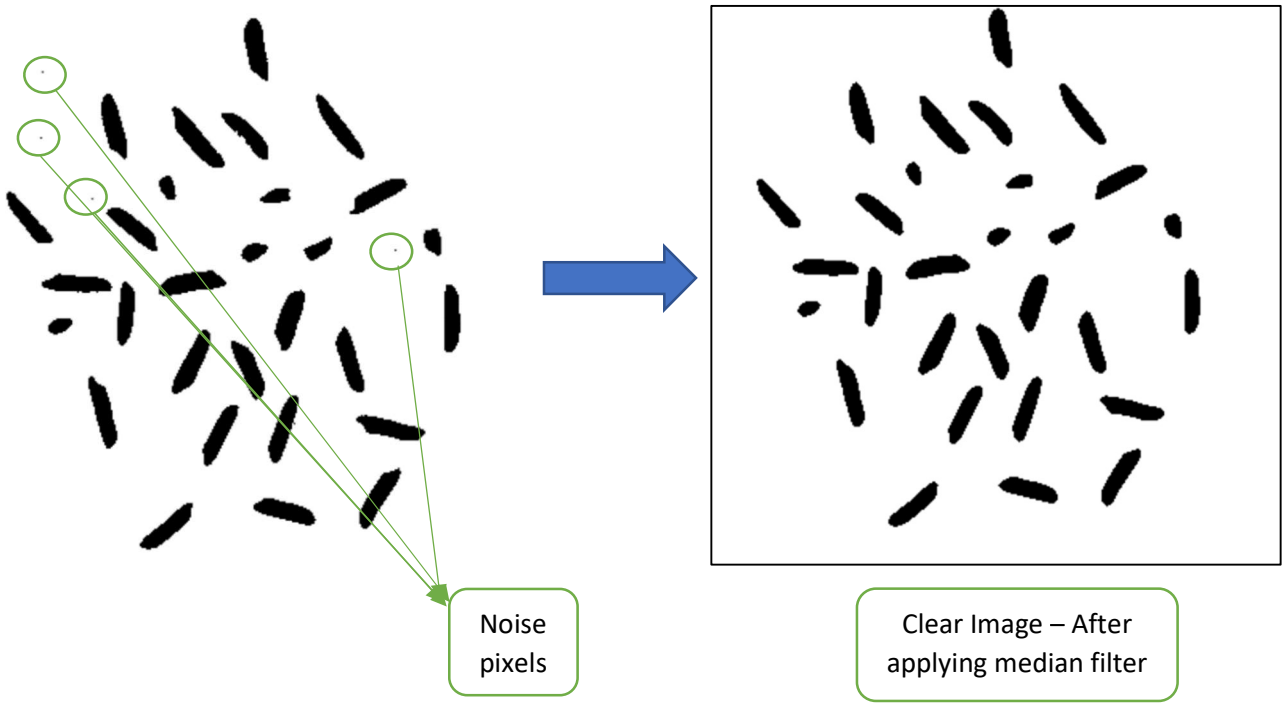


Figure 3 Iterations vs Threshold graph

Figure 4 Intermediate image

Task 2: Counting the rice kernels

In order to count the number of rice kernels in the image, it is vital to clear the noise present in the binary image obtained as a result of Task1. In order to do this, median filter has been applied (with window size 3). As a result of this filter, the following image has been obtained,



Once the noise pixels are removed, the binary image can now be used for counting the number of rice kernels. *Two pass connected components algorithm* has been used for this task:

First pass-

1) Initially iterate through the filtered image and look for a non-background pixel (value 0, rice kernel).
2) Look for its non-background neighboring pixels (black pixels with value 0). Find the neighbor with the smallest label and assign the same label to the current pixel. If none of the neighbors are labeled, then assign a unique label (variable 'cls' starting with value 1) to this pixel.

Second pass-

1) Iterate through each pixel of the filtered image and look for non-background pixels.
2) For every such pixel found, find all the connected pixels and relabel them with the minimum pixel value among its 8 neighbors. Continue this till there are no more changes made to any pixel value, that is, when converged become True.
3) Then count the number of connected components. A component consists of pixels having the same label. For the above example, the total number of rice kernels = 29.

The output count includes whole as well as damaged rice kernels. Output is displayed as follows (for image '*rice_img6.png*'):

```
Number of rice kernels =  29
```

Task 3: Percentage of damaged rice kernels

A damaged kernel is a rice kernel that occupies a smaller area in comparison to whole kernels. Given a minimum pixel area 'min_area', using the connected component labels generated in Task 2, the number of damaged kernels (components occupying a pixel area < min_area) and the number of whole kernels (components occupying a pixel area > = min_area) are to be counted. The percentage of damaged kernels is the total number of damaged kernels divided by the total number of rice kernels in the image.

```
Kernel areas  [6, 3, 798, 713, 831, 751, 214, 622, 778, 855, 703, 231, 743, 727, 219, 808, 801, 787, 738, 334, 908, 899, 805,
882, 726, 846, 830, 796, 148, 606, 881, 961, 804, 829, 233, 301, 826, 787, 251, 148, 316, 862, 292, 250, 857, 906, 894]

Kernel areas  [31, 152, 662, 6, 623, 857, 776, 765, 212, 930, 771, 219, 239, 841, 736, 323, 988, 1020, 773, 714, 845, 959, 25
8, 816, 785, 314, 868, 922, 978, 841, 943, 142, 631, 914, 7, 226, 852, 908, 946, 253, 739, 332, 819, 766, 848, 684, 834, 337]

Kernel areas  [4, 394, 390, 379, 453, 343, 111, 358, 126, 328, 343, 142, 140, 131, 433, 383, 320, 359, 411, 109, 383, 422, 37
1, 414, 383, 367, 387, 392, 374, 355]

Kernel areas  [4, 451, 114, 403, 147, 470, 456, 489, 498, 114, 470, 425, 359, 407, 151, 483, 162, 517, 369, 148, 429, 449, 12
4, 407, 408, 433, 476, 435, 487, 419, 494, 412, 433, 401, 388]
```

After observing the above areas it is clear that the size of grain(undamaged) is ranging from 300 to 1000 (approx.). Hence it is difficult to estimate a common 'min_area' to find damaged kernels for all four images. Upon observing the above data the area of a damaged kernel can be approximately taken as 200. But this condition may not work and may give damaged kernels as whole for images 1 and 2. The best solution can be that the min_area must be a relative number in comparison to min and max kernel areas for a specific image. Due to this even though an input is accepted as min_area, internally I have considered a logic to calculate approximate min_area based on area range of kernels.

My approach is that the 'min_area' is half of the maximum kernel's area. Any kernel that occupies an area less than half of the maximum kernel's area is considered as a damaged kernel. This approach seemed reasonable by looking at the 4 images and its damaged kernel sizes.

The area of a rice kernel is calculated by adding the number of pixels of that rice kernel. This can be achieved since in Task 2, we have labelled every pixel of a kernel in the image with a number and no two kernels have the same number. 'kernel_areas' is a list consisting of the pixels count (area) in every kernel and 'unique_kernels' is a list containing all the labels given to kernels in the image.

The areas of damaged kernels are stored in a list (l4) and after calculating the percentage of damaged pixels, these areas are removed from the binary image (the pixel values made white(255)). All the remaining whole kernels pixels are then darkened (pixel value set to 0) to get the original binary image without damaged kernels. The final binary image with all the whole kernels looks as follows:

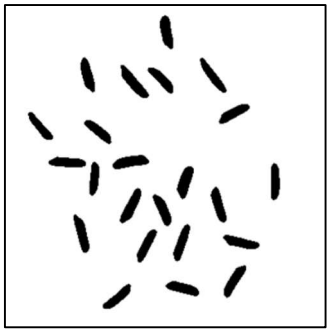


Figure 5 Rice image after damaged kernels removed

Output for other rice samples:

Input - *rice_img1.png*

Figure 6 Task 1 output

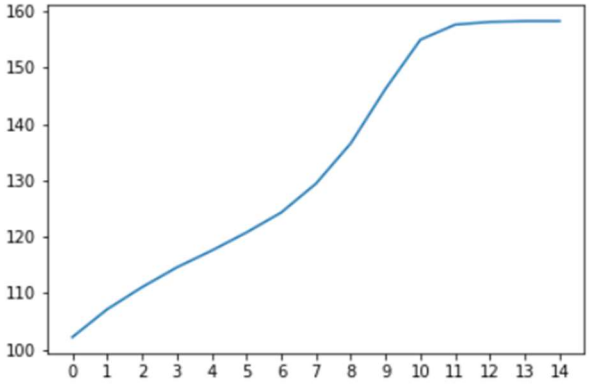


Figure 7 Task 1 output

Figure 8 Task 2 output

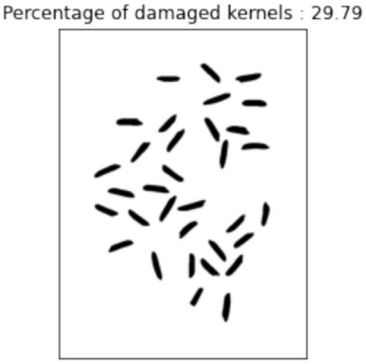


Figure 9 Task 3 output

Input – *rice_img2.png*

Figure 10 Task 1 output

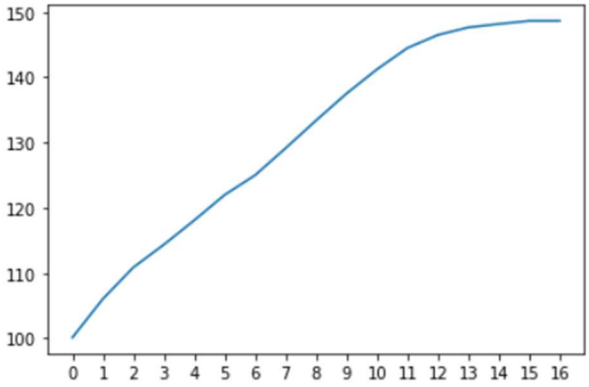


Figure 11 Task 1 output

Figure 12 Task 2 output

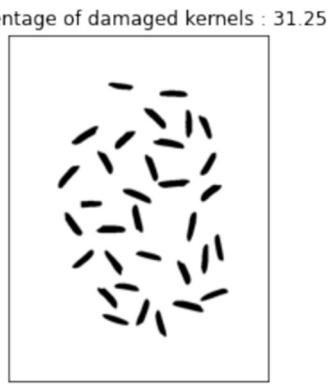


Figure 13 Task 3 output

Input – *rice_img7.png*

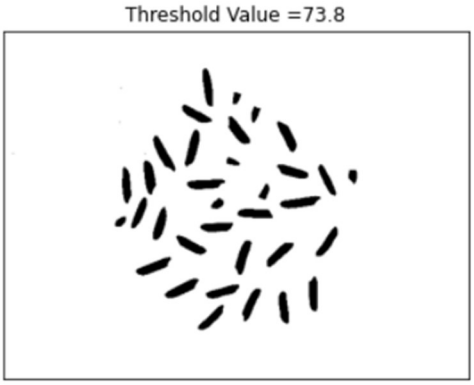


Figure 14 Task 1 output

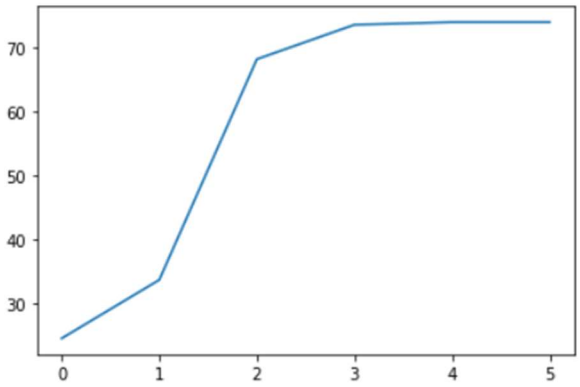


Figure 15 Task 1 output

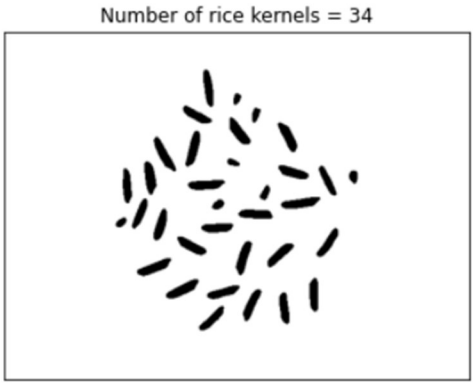


Figure 15 Task 2 output

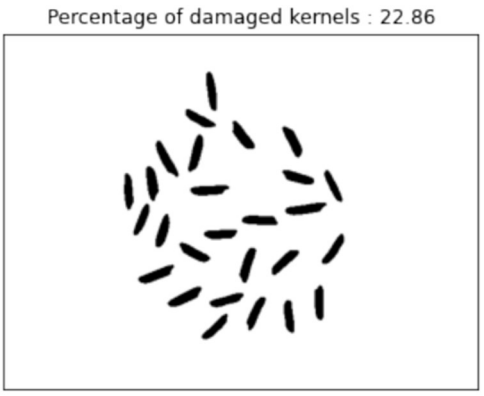


Figure 16 Task 3 output