COMP9517-Computer Vision: Exploration of the Image Processing techniques to grade rice based on the percentage of damaged kernel

---

2020

Asha Karki

Task1:

Algorithm used for the Task 1 is as advised **Iso-data Intensity Thresholding**. The below is explanation of approach that had taken:

1) Open and read the image using *imread* using zero, as grayscale.
2) Histogram is plotted for all the given images to check the pixel intensities. It is used to check if the histogram has a **bi-modal pixel intensity** or not. If it is not, then the prescribed algorithm of intensity thresholding does not work. The following figures are the histogram obtained for rice_img1.png, rice_img2.png, rice_img6.png, and rice_img7.png. The first group of spectrums in the histogram belong to background of the images and the second group of spectrums belong to foreground of the images.
For rice_img1.png and rice_img2.png, the histograms are clearly separated into two clusters (the visible amount of the bimodal pixel intensities). However, for rice_img6.png and rice_img7.png, the histogram is bimodal, but the foreground intensities are very low compare to background and difficult to see in the histogram.

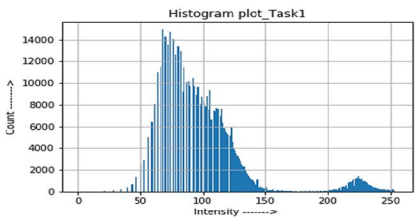

Figure 1: Histogram of rice_img1.png

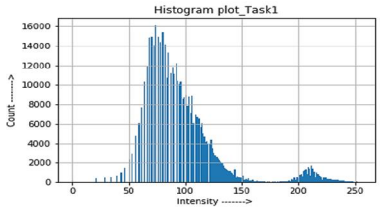

Figure 2: Histogram for rice_img2.png

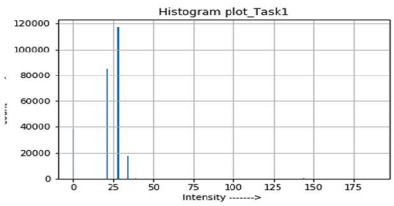

Figure 3: Histogram of rice_img6.png

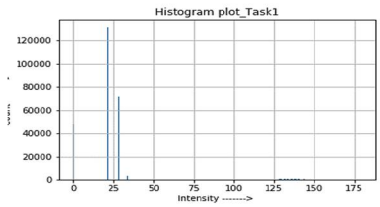

Figure 4: Histogram of rice_img7.png

3) From the histograms, we are sure that we can apply Iso-data Intensity thresholding to the given images. Now the task is finding an appropriate threshold value from the following ways.

a) **POINT1:** Setting the appropriate initial threshold value:
Calculate the average intensity value of the image and then assign it as the initial threshold value.

$$th = (np.\min(image), np.\max(image))/2$$

Reason using the above initial threshold: *From the observations, if the foreground intensities are very low compare to the background, then the given algorithm of getting the threshold value cannot properly separate foreground and background as it stuck in the range of background intensity -- if we start from lower values of the initial threshold.*

b) **POINT2:** We calculate $\mu_0 < th$ and $\mu_1 \geq th$, and update the th_{new} with the average of μ_0 and μ_1.

c) **POINT3:** We check the value $abs(th_{new} - th)$, if this value is less than ϵ, the threshold value is saturated (mid-way between the two class means) and store it in a list. We performed POINT3 for $\epsilon = [0.001, 0.01, 0.05, 0.1, 0.5]$. We picked the most appeared threshold value that is obtained for different ϵ: $th = max(set(final\_threshold),\ key = final\_threshold.count)$

d) Now with the selected threshold value, we convert the image into binary, where background is assigned 255 (white) and foreground is assigned 0 (black), if the threshold of the pixel is greater than th, and less than or equal to th, respectively.

e) We plot binary image (**Figure 5, 6, 7, 8**), display threshold value and plot threshold value th at every iteration on graph (**Figure 9, 10, 11, 12**).

Figure 5: Binary image of rice_img1.png

Figure 6: Binary image of rice_img2.png

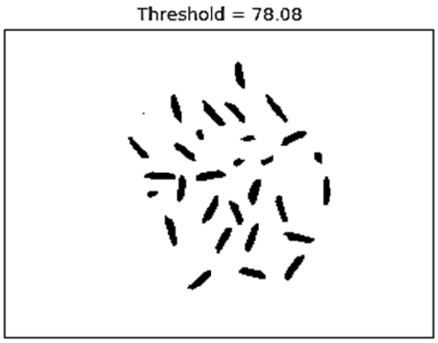

Figure 7: Binary image of rice_img6.png

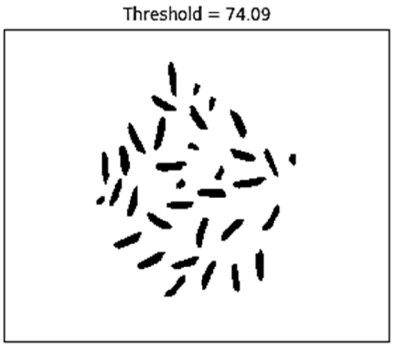

Figure 8: Binary image of rice_img7.png

In the below figures, Figures 9, 10, 11, and 12, the red dots indicate the threshold values for each iteration. The saturated threshold value is the final one (e.g., Figure 11, saturated threshold value is 78.08).

***Note point**: The limitations of the above algorithm are: (1) image must have two classes of pixels (i.e., it has bi-modal pixel intensity histogram, (2) if the initial threshold value is too low, then it may not be the appropriate threshold value*

for the binary conversion. If the intensity distribution is too extreme (e.g., image_rice4.png), then the algorithm does not work, as the threshold value lies in the background pixel values.

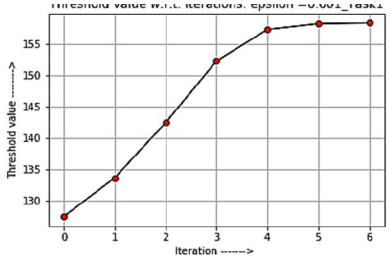

Figure 9: Plot for threshold value for the rice_img1.png

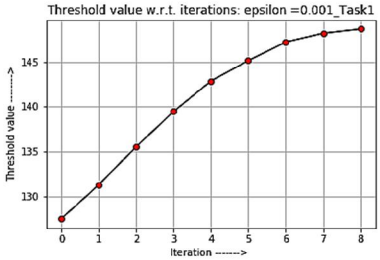

Figure 10: Plot for threshold value for the rice_img2.png

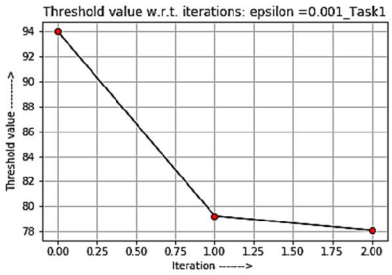

Figure 11: Plot for threshold value for the rice_img6.png

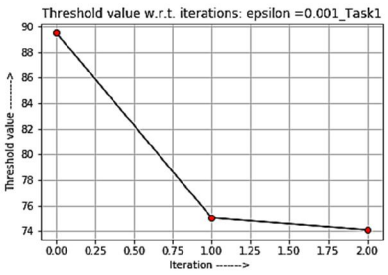

Figure 12: Plot for threshold value for the rice_img7.png

Task 2:

Algorithm used for Task 2 is as advised in the **two-pass connected components labelling algorithm**.

1) **POINT1:** Applying the median filter to remove the noisy pixels if any in the image from the task1.
   Observation: *Median filter kernel size 3x3 provides good result for the images of task1 with respect to rice_img6.png and rice_img7.png. However, it does not remove noise in the images of task1 with respect to rice_img1.png and rice_img2.png, where kernel size 7x7 provides good result. Thus, we use median filter kernel size 7x7 for all.*

2) **POINT2:** First pass considering eight-neighborhood. Temporary labels are assigned to each data points and the equivalence are stored.
   a) Step1: Each pixel from the top left to the end is checked: If the pixel is at (i, j), then its neighbors are located at:
   $(north, west), (north, j), (north, east), (i, west), (i, east), (south, west), (south, j), (south, east)$,
   where $north = i - 1$, $south = i + 1$, $west = j - 1$, $east = j + 1$. We consider only those neighboring pixels which are not background (255) and has some labels, we call this *eight_conn_neighbors.*

   b) Step2: For the *eight_conn_neighbors,* if its length is zero, then all my neighbours are connected components – assign the same label as the current pixel's label. Increase the label_count by one. Otherwise, collect all the labels of the pixels in *eight_conn_neighbors,* and assign the minimum collected labels to the current pixel.
   c) Step3: Equivalence check for all connected components found in Step2
      (i) For each connected component: We pick the first element (set), then check whether there is any other pixel with the same value in the remaining set, if yes, then we union those two in to the first element; Otherwise, the element in the remaining set will be kept in the remaining set. We repeat this process till we check for all elements in the connected component.

(ii) Merge the all equivalent elements

3) **POINT3:** Second pass. The temporary labels are replaced by the smallest label from its equivalence class. If the pixel is not in the background, then we relabel the element with the lowest equivalent label.
4) **POINT4:** The output of the POINT3 provides us the image *img_with_lables*, and the maximum of the image is the number of rice kernels in the image; $total\_grain = np.max(img\_with\_labels)$. We plot the filtered binary image (**Figure 13, 14, 15, 16**) and display the total number of rice kernels (Figure 17).

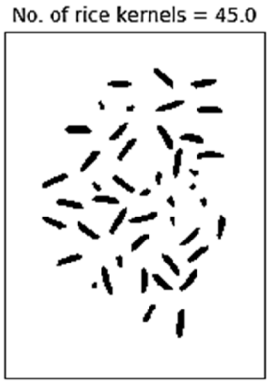

Figure 13: Filtered binary image of rice_img1.png

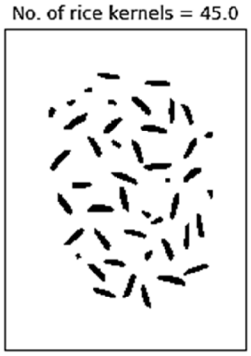

Figure 14: Filtered binary image of rice_img2.png

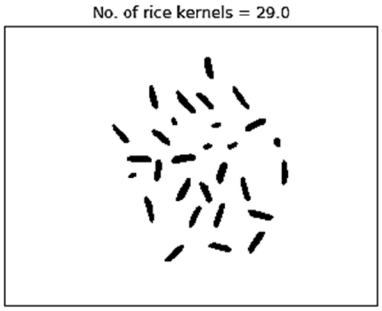

Figure 15: Filtered binary image of rice_img6.png

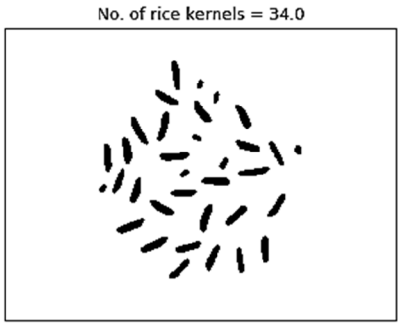

Figure 16: Filtered binary image of rice_img7.png

```
Number of rice kernels in the image_rice_img1.png = 45.0
Number of rice kernels in the image_rice_img2.png = 45.0
Number of rice kernels in the image_rice_img6.png = 29.0
Number of rice kernels in the image_rice_img7.png = 34.0
```

Figure 17: Filtered binary image of rice_img6.png. The outputs are obtained while running separately for the respective input image.

Task 3:

Method used in the task 3 to calculate the **percentage of damaged rice kernels** and the **binary image after excluding the broken grains**:

1) **POINT1:** From the image from task 2, which we call *image_with_labels*, we construct a dictionary with labels as keys and pixel counts (i.e., area) as values. If the *min_area* is not given, then we choose the *min_area* as the half of the maximum of the pixel counts among all in the dictionary. This means the 50% of the size of the

largest undamaged kernels is the threshold for the separation of broken and unbroken kernels. 50% is chosen based on the observation.

2) **POINT2:** Each pixel more than greater than or equal to *min_area* is accumulated to a list *area_with_max*, later it is used to provide the labels to connected components in the image obtained from task2. The resulting image is *img_with_fine_kernels*. Now, the damaged rice kernels percentage is calculated as $(int(np.\max(img\_with\_labels)) - int(np.max(img\_with\_fine\_kernels))/2$ (**Figure 18, 19, 20, 21**). The broken kernels are removed from the image and the resulting binary image is saved (**Figure 22, 23, 24, 25**).

```
No. of undamaged rice kernels =  33
No. of damaged rice kernels =  12
Percentage of damaged rice kernels = 26.67 %
```

Figure 18: Count output for rice_img1.png

```
No. of undamaged rice kernels =  33
No. of damaged rice kernels =  12
Percentage of damaged rice kernels = 26.67 %
```

Figure 19: Count output for rice_img2.png

```
No. of undamaged rice kernels =  23
No. of damaged rice kernels =  6
Percentage of damaged rice kernels = 20.69 %
```

Figure 20: Count output for rice_img6.png

```
No. of undamaged rice kernels =  27
No. of damaged rice kernels =  7
Percentage of damaged rice kernels = 20.59 %
```

Figure 21: Count output for rice_img7.png

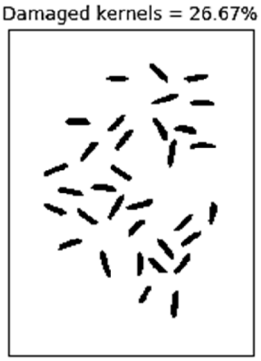

Figure 22: Binary image excluding all damaged kernels of rice_img1.png

Figure 23: Binary image excluding all damaged kernels of rice_img2.png

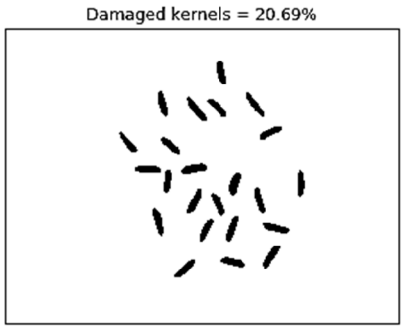

Figure 24: Binary image excluding all damaged kernels of rice_img6.png

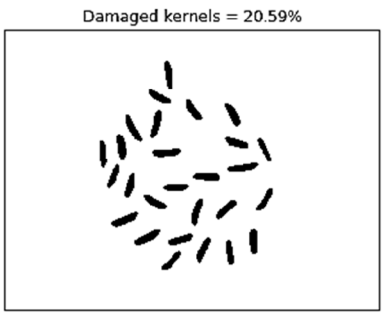

Figure 25: Binary image excluding all damaged kernels of rice_img7.png