

## Image Processing Lab

Batch 9:

Ajay Viswanathan [09EC3509]

Sourin Sutradhar [09EC3514]

### Experiment 8

#### Image Segmentation and Connected-Component Labelling

##### Problem Objective:

Write C++/Image modular functions to perform the following on a  $512 \times 512$  grayscale Lena image using Otsu thresholding.

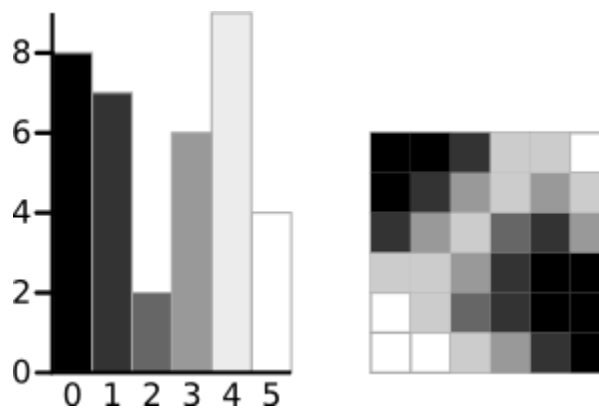
- (a) Segmentation
- (b) Connected component labeling

##### Brief Theory:

Segmentation involves partitioning an image into a set of homogeneous and meaningful regions, such that the pixels in each partitioned region possess an identical set of properties or attributes. These sets of properties of the image may include gray levels, contrast, spectral values, or textural properties. The result of segmentation is a number of homogeneous regions, each having a unique label.

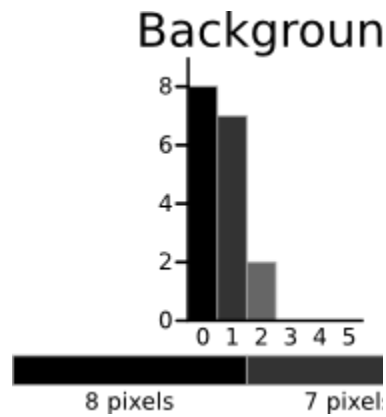
Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either fall in foreground or background. The aim is to find the threshold value where the sum of foreground and background spreads is at its minimum.

The algorithm will be demonstrated using the simple  $6 \times 6$  image shown below. The histogram for the image is shown next to it. To simplify the explanation, only 6 grayscale levels are used.



A 6-level grayscale image and its histogram

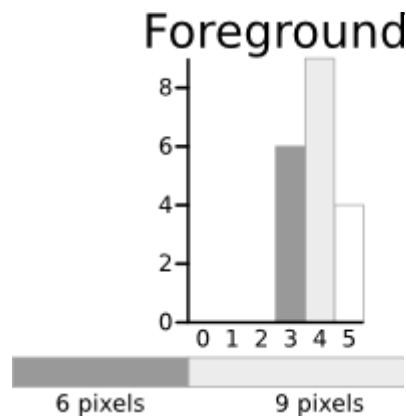
The calculations for finding the foreground and background variances (the measure of spread) for a single threshold are now shown. In this case the threshold value is 3.



$$\text{Weight } W_b = \frac{8 + 7 + 2}{36} = 0.4722$$

$$\text{Mean } \mu_b = \frac{(0 \times 8) + (1 \times 7) + (2 \times 2)}{17} = 0.6471$$

$$\begin{aligned} \text{Variance } \sigma_b^2 &= \frac{((0 - 0.6471)^2 \times 8) + ((1 - 0.6471)^2 \times 7)}{17} \\ &= \frac{(0.4187 \times 8) + (0.1246 \times 7) + (1.8304 \times 2)}{17} \\ &= 0.4637 \end{aligned}$$



$$\text{Weight } W_f = \frac{6 + 9 + 4}{36} = 0.5278$$

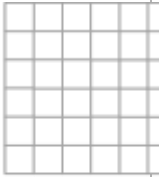
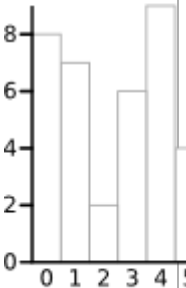

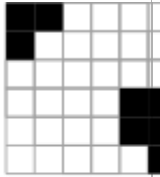
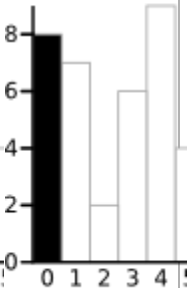


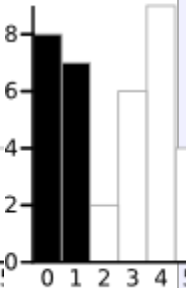


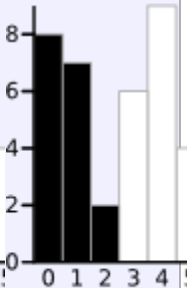

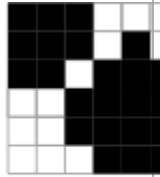
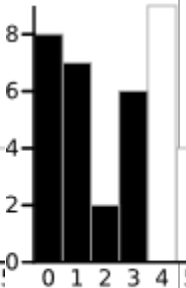


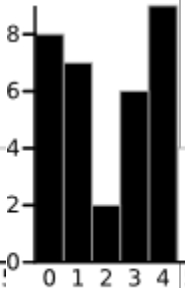

$$\text{Mean } \mu_f = \frac{(3 \times 6) + (4 \times 9) + (5 \times 4)}{19} = 3.8947$$

$$\begin{aligned} \text{Variance } \sigma_f^2 &= \frac{((3 - 3.8947)^2 \times 6) + ((4 - 3.8947)^2 \times 9)}{19} \\ &= \frac{(4.8033 \times 6) + (0.0997 \times 9) + (4.8864 \times 9)}{19} \\ &= 0.5152 \end{aligned}$$

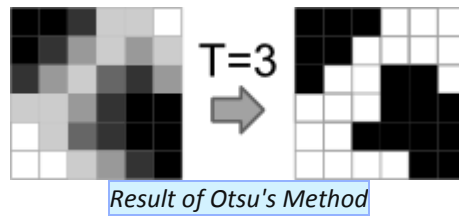
The next step is to calculate the 'Within-Class Variance'. This is simply the sum of the two variances multiplied by their associated weights.

$$\begin{aligned} \text{Within Class Variance } \sigma_W^2 &= W_b \sigma_b^2 + W_f \sigma_f^2 = 0.4722 \times 0.4637 + 0.5278 \times 0.5152 \\ &= 0.4909 \end{aligned}$$

This final value is the 'sum of weighted variances' for the threshold value 3. This same calculation needs to be performed for all the possible threshold values 0 to 5. The table below shows the results for these calculations. The highlighted column shows the values for the threshold calculated above.

Threshold	T=0	T=1	T=2	T=3	T=4	T=5
	  	  	  	  	  	  
Weight, Background	$W_b = 0$	$W_b = 0.222$	$W_b = 0.4167$	$W_b = 0.4722$	$W_b = 0.6389$	$W_b = 0.8889$
Mean, Background	$M_b = 0$	$M_b = 0$	$M_b = 0.4667$	$M_b = 0.6471$	$M_b = 1.2609$	$M_b = 2.0313$
Variance, Background	$\sigma_b^2 = 0$	$\sigma_b^2 = 0$	$\sigma_b^2 = 0.2489$	$\sigma_b^2 = 0.4637$	$\sigma_b^2 = 1.4102$	$\sigma_b^2 = 2.5303$
Weight, Foreground	$W_f = 1$	$W_f = 0.7778$	$W_f = 0.5833$	$W_f = 0.5278$	$W_f = 0.3611$	$W_f = 0.1111$
Mean, Foreground	$M_f = 2.3611$	$M_f = 3.0357$	$M_f = 3.7143$	$M_f = 3.8947$	$M_f = 4.3077$	$M_f = 5.000$
Variance, Foreground	$\sigma_f^2 = 3.1196$	$\sigma_f^2 = 1.9639$	$\sigma_f^2 = 0.7755$	$\sigma_f^2 = 0.5152$	$\sigma_f^2 = 0.2130$	$\sigma_f^2 = 0$
Within Class Variance	$\sigma_w^2 = 3.1196$	$\sigma_w^2 = 1.5268$	$\sigma_w^2 = 0.5561$	$\sigma_w^2 = 0.4909$	$\sigma_w^2 = 0.9779$	$\sigma_w^2 = 2.2491$

It can be seen that for the threshold equal to 3, as well as being used for the example, also has the lowest sum of weighted variances. Therefore, this is the final selected threshold. All pixels with a level less than 3 are background, all those with a level equal to or greater than 3 are foreground. As the images in the table show, this threshold works well.



This approach for calculating Otsu's threshold is useful for explaining the theory, but it is computationally intensive, especially if you have a full 8-bit greyscale. The next section shows a faster method of performing the calculations which is much more appropriate for implementations.

### A Faster Approach

By a bit of manipulation, you can calculate what is called the *between class* variance, which is far quicker to calculate. Luckily, the threshold with the maximum *between class* variance also has the minimum *within class* variance. So it can also be used for finding the best threshold and therefore due to being simpler is a much better approach to use.

$$\text{Within Class Variance } \sigma_W^2 = W_b \sigma_b^2 + W_f \sigma_f^2 \quad (\text{as seen above})$$

$$\begin{aligned} \text{Between Class Variance } \sigma_B^2 &= \sigma^2 - \sigma_W^2 \\ &= W_b(\mu_b - \mu)^2 + W_f(\mu_f - \mu)^2 \quad (\text{where } \mu = W_b \mu_b + W_f \mu_f) \\ &= W_b W_f (\mu_b - \mu_f)^2 \end{aligned}$$

The table below shows the different variances for each threshold value.

Threshold	T=0	T=1	T=2	T=3	T=4	T=5
Within Class Variance	$\sigma_W^2 = 3.1196$	$\sigma_W^2 = 1.5268$	$\sigma_W^2 = 0.5561$	$\sigma_W^2 = \mathbf{0.4909}$	$\sigma_W^2 = 0.9779$	$\sigma_W^2 = 2.2491$
Between Class Variance	$\sigma_B^2 = 0$	$\sigma_B^2 = 1.5928$	$\sigma_B^2 = 2.5635$	$\sigma_B^2 = \mathbf{2.6287}$	$\sigma_B^2 = 2.1417$	$\sigma_B^2 = 0.8705$

### Connected Component Labelling

Connected-component labeling (alternatively connected-component analysis, blob extraction, region labeling, blob discovery, or region extraction) is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. Connected-component labeling is not to be confused with segmentation.

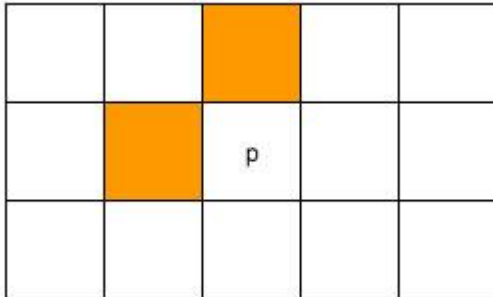
The algorithm designed by **Rosenfeld and Pfaltz in 1966** uses the union-find data structure to solve this problem (read about the union-find data structure), and that too quite efficiently. Its "classical" because it uses a result from the the classical algorithm for connectedness in graph theory.

The algorithm consists of two passes. In the first pass, the algorithm goes through each pixel. It checks the pixel above and to the left. And using these pixel's labels (which have already been assigned), it assigns a label to the current pixel. And in the second pass, it cleans up any mess it might have created, like multiple labels for connected regions.

So that was the overview of how the algorithm works. Now we get into the details.

## The First Pass

In the first pass, every pixel is checked. One by one, starting at the top left corner, and moving linearly to the bottom right corner.



If you're considering the pixel 'p', you'll only check the orange pixels. Thus, at any given time, you only need to have two rows of the image in memory. This helped in memory efficiency in the past, but now we have 2GB RAMs. So its not an issue these days.

We'll go through each step of the first pass one by one.

### Step 1

Here we check if we're interested in a pixel or not. If the pixel is a background pixel (its value is zero, or whatever other criteria you want), we simply ignore it and move on to the next pixel. If not, you go to the next step.

### Step 2 and 3

Here, you're fetching the label of the pixels just above and to the left of 'p'. And you store them (into A and B here). Now, there are a few possible cases here:

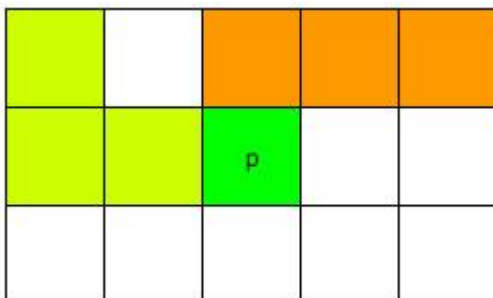
- The pixel above or/and the the left aren't background pixels: In this case, things proceed as usual. You just go to the next step. A or/and B will have actual values (the labels).
- Both pixels are background pixels: In this case, you cannot get labels. So, you create a new label, and store it into A and B.

### Step 4 and 5

You figure out which one is smaller: A or B and then you set that label to pixel 'p'.

### Step 6

Suppose you have a situation where pixel above has a label A and the pixel to the left has a label B. But you know that these two labels are connected (because the current pixel "connects them" ).



Thus, you need to store the information that the labels 'A' and 'B' are actually the same. And you do that using the union-find data structure. You set the label 'A' as the child of 'B'. Using this information, the algorithm will clean up the mess in the second mess.

The only thing you need to remember is that the smaller label get assigned to 'p', and the larger number becomes a child of the smaller number.

#### *Step 7*

Go to the next pixel.

#### **The second pass**

Again, the algorithm goes through each pixel, one by one. It checks the label of the current pixel. If the label is a 'root' in the union-find structure, it goes to the next pixel.

Otherwise, it follows the links to the parent until it reaches the root. Once it reaches the root, it assigns that label to the current pixels.

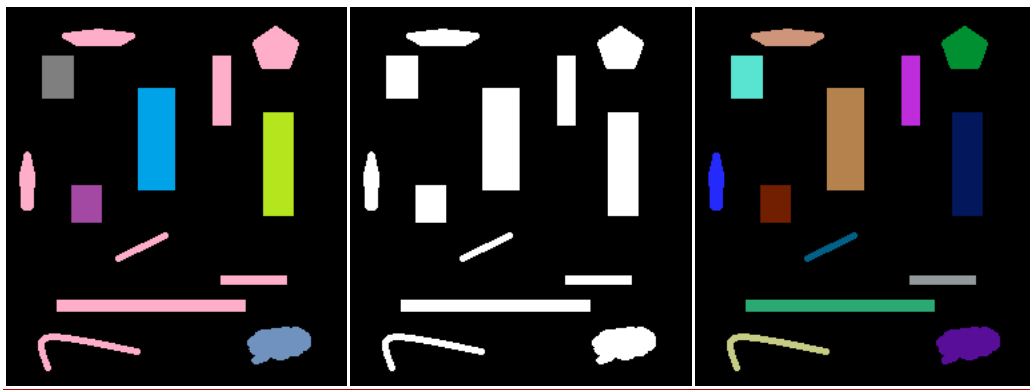
**Results:**



Original image

Otsu threshold

Labelled image



Original image

Otsu threshold

Labelled image

### Discussion and Inference:

A measure of region homogeneity is variance (i.e., regions with high homogeneity will have low variance). Otsu's method selects the threshold by minimizing the within-class variance of the two groups of pixels separated by the thresholding operator. It does not depend on modelling the probability density functions, however, it assumes a bimodal distribution of grey-level values (i.e., if the image approximately fits this constraint, it will do a good job). The method assumes that the histogram of the image is bimodal (i.e., two classes). The method breaks down when the two classes are very unequal (i.e., the classes have very different sizes).

Connected Component Labeling is commonly used to refer to the task of grouping the connected pixels in an image. (Note that there are different ways to define connectedness.) The basic approach is to scan the image and assign labels to each pixel until the labels for the pixels no longer change. This basic approach is slow because the labels propagate one layer in an iteration. There are two common strategies to speed up this process. The most common strategy leads to a two-pass algorithm. This algorithm uses a data structure to record label equivalence information. It scans the image once to assign provisional labels and discover the label equivalence information, and scans the image a second time to assign the final labels. Another very successful strategy is the one-pass algorithms, that find all connected pixel in one shot, for example through recursively visit all the connected neighbors