

A way to convert Handwritten Core Java Code

Declaration: The project is not yet completed

Things we aim at:

- ❖ Proper binarization of a Core Java Code
- ❖ Convert Handwritten content to text
- ❖ Provide extra facilities such as shortcut for some long words
- ❖ Build a user-friendly application to take advantage of JwriteNscan

As it takes time to build such a project, in this semester we will be doing binarization. Though many assumptions are taken and if the user runs into problem, he/she might consider retaking a photo so that he gets the best binarized form. We are still unable to make sure that the program never crashes but if a photo taken by camera is given we can assure that the program might not crash. The methods we take will be coming up shortly.

#### **Idea behind:**

We had a hard time programming and coming with the exercise book with the code written there and writing the same on keyboard. Can we make something interesting that just allows us to take a photo of the code and the program convert it to the text which we can easily copy into a file?

Wow! That makes our life so much easier.

Minds behind the project:

- i) Sobyasachi Chatterjee (Email Address: [sobyasachichatterjee@gmail.com](mailto:sobyasachichatterjee@gmail.com))
- ii) Sourjya Chatterjee (Email Address: [sourjyachatterjee1998@gmail.com](mailto:sourjyachatterjee1998@gmail.com))

Language used:- Python 3

## **Acknowledgement**

We would really want to thank the teachers of our department especially Ghosh sir and Banerjee Sir whose enthusiasm and contribution made it possible. We would like thank my colleagues who contributed somewhat in the name of the project. We would also like to thank our parents for the financial support they provided.

As discussed Earlier the first step we are doing is to binarize the image given to us. The program runs on console and finds out the binarized image which will help us in the next step.

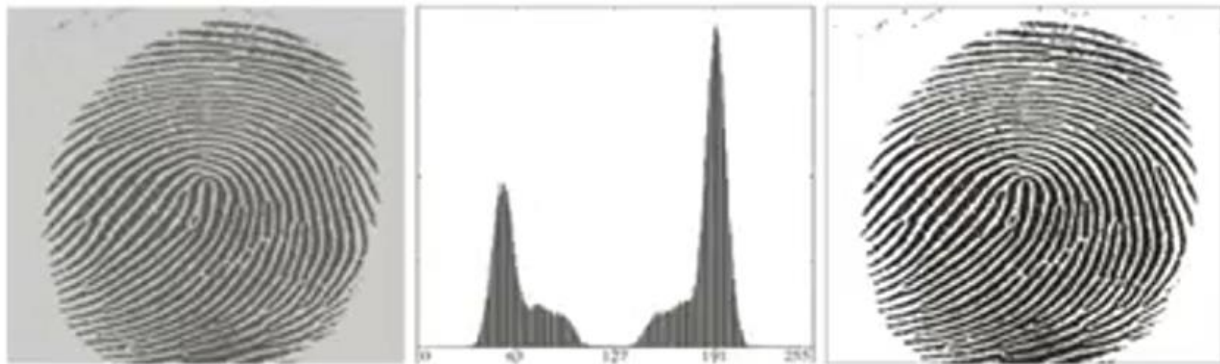
## BINARIZATION

As guessed by many we will be using Otsu's Method in binarizing.

We will briefly discuss Otsu's method and the variation that made it possible

By binarize what we mean is to make a pixel either black and white

Just like this...



In [computer vision](#) and [image processing](#), **Otsu's method**, named after [Nobuyuki Otsu](#) is used to automatically perform clustering-based image [thresholding](#), or, the reduction of a graylevel image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class [variance](#)) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal. Consequently, Otsu's method is roughly a one-dimensional, discrete analog of [Fisher's Discriminant Analysis](#). Otsu's method is also directly related to the [Jenks optimization method](#).

### Algorithm

1. Compute histogram and probabilities of each intensity level
2. Set up initial  $\omega_i(0)$  and  $\mu_i(0)$
3. Step through all possible thresholds  $t = 1, \dots$  maximum intensity
  1. Update  $\omega_i$  and  $\mu_i$
  2. Compute  $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum  $\sigma_b^2(t)$

Minimize the *weighted within-class variance*:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$q_1(t) = \sum_{i=1}^t P(i) \quad q_2(t) = \sum_{i=t+1}^l P(i)$$

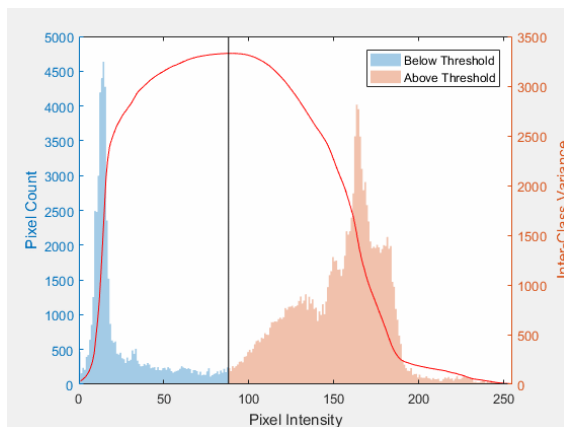
$$\mu_1(t) = \sum_{i=1}^t \frac{iP(i)}{q_1(t)} \quad \mu_2(t) = \sum_{i=t+1}^l \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) = \sum_{i=1}^t [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \sigma_2^2(t) = \sum_{i=t+1}^l [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

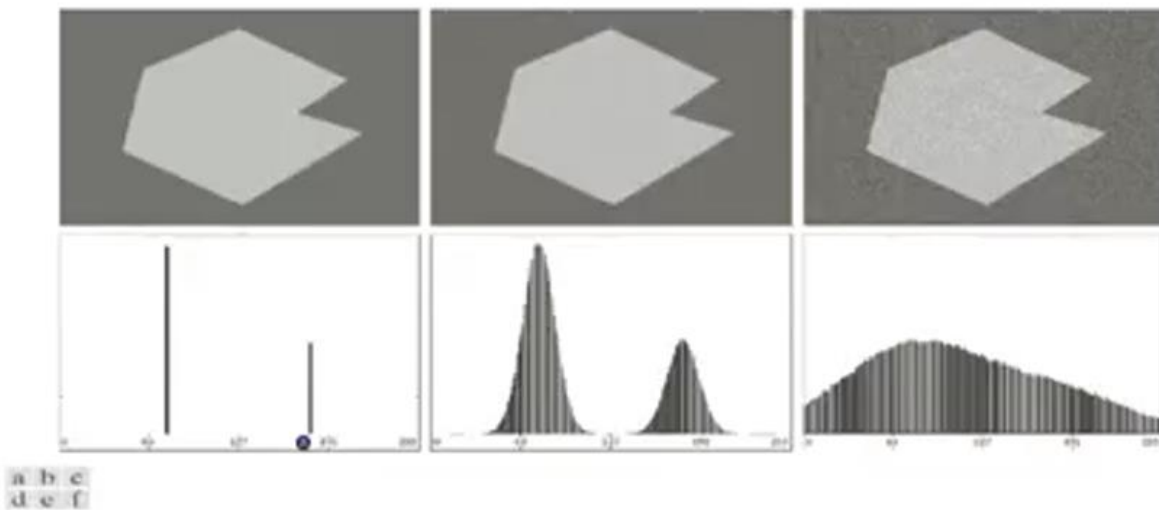
Minimize the *weighted within-class variance*:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

$$\sigma^2 = \underbrace{\sigma_w^2(t)}_{\text{Within-class}} + \underbrace{q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2}_{\text{Between-class}}$$



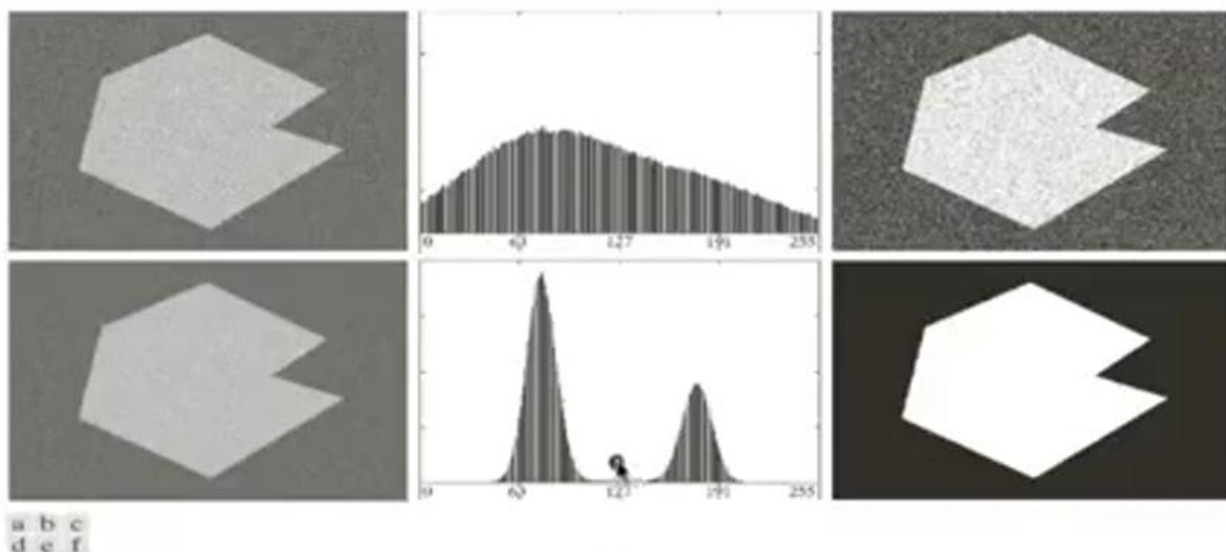
Problems with Otsu's method are:



**FIGURE 10.36** (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d)–(f) Corresponding histograms.

Here the more noisy image tends not to be a bimodal histogram and hence finding a perfect threshold is quite difficult

To remove the noise the image is made smoother by using a 5x5 averaging mask but I have applied it and it does not give fruitful results to our problems as the pixel intensity vary in different area



**FIGURE 10.40** (a) Noisy image from Fig. 10.36 and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a  $5 \times 5$  averaging mask and (e) its histogram. (f) Result of thresholding using

The solution is to divide the problem into multiple blocks and applying otsu on each of the blocks



**FIGURE 10.49** (a) Text image corrupted by spot shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages.

Let's describe our algorithm in which we did qualify the block in accordance with some parameters which we somehow learned after seeing the result.

This is the only part of the algorithm which we will not provide:

- i) Resize the image to 512 x 512
- ii) Test whether it's a photo which is not so strict cause we assume that the difference b/w the highest and lowest pixel intensity should have difference 50
- iii) We convert it to grayscale first if the image is not in grayscale
- iv) Divide the grayscale image into 128 x 64 blocks and apply otsu thresholding. We will qualify the block or else put the pixels to value=255
- v) Show the output to user and get feedback
- vi) If YES then end,  
Else make the block size shorter and check

Input assumptions:

- ✚ We have found writing with a black pen has increased the chance of getting the right answer. Hence we encourage the users to buy a black pen for the algorithm
- ✚ There are not more than 25 lines of code in the image
- ✚ It's is not a 360 degree image

## FEW INPUTS AND OUTPUTS:

### INPUT

Imagine a vast sheet of paper on which straight Lines, Triangles, Squares, Pentagons, Hexagons, and other figures, instead of remaining fixed in their places, move freely about, on or in the surface, but without the power of rising above or sinking below it, very much like shadows - only hard and with luminous edges - and you will then have a pretty correct notion of my country and countrymen. Alas, a few years ago, I should have said "my universe": but now my mind has been opened to higher views of things.

OUTPUT

output

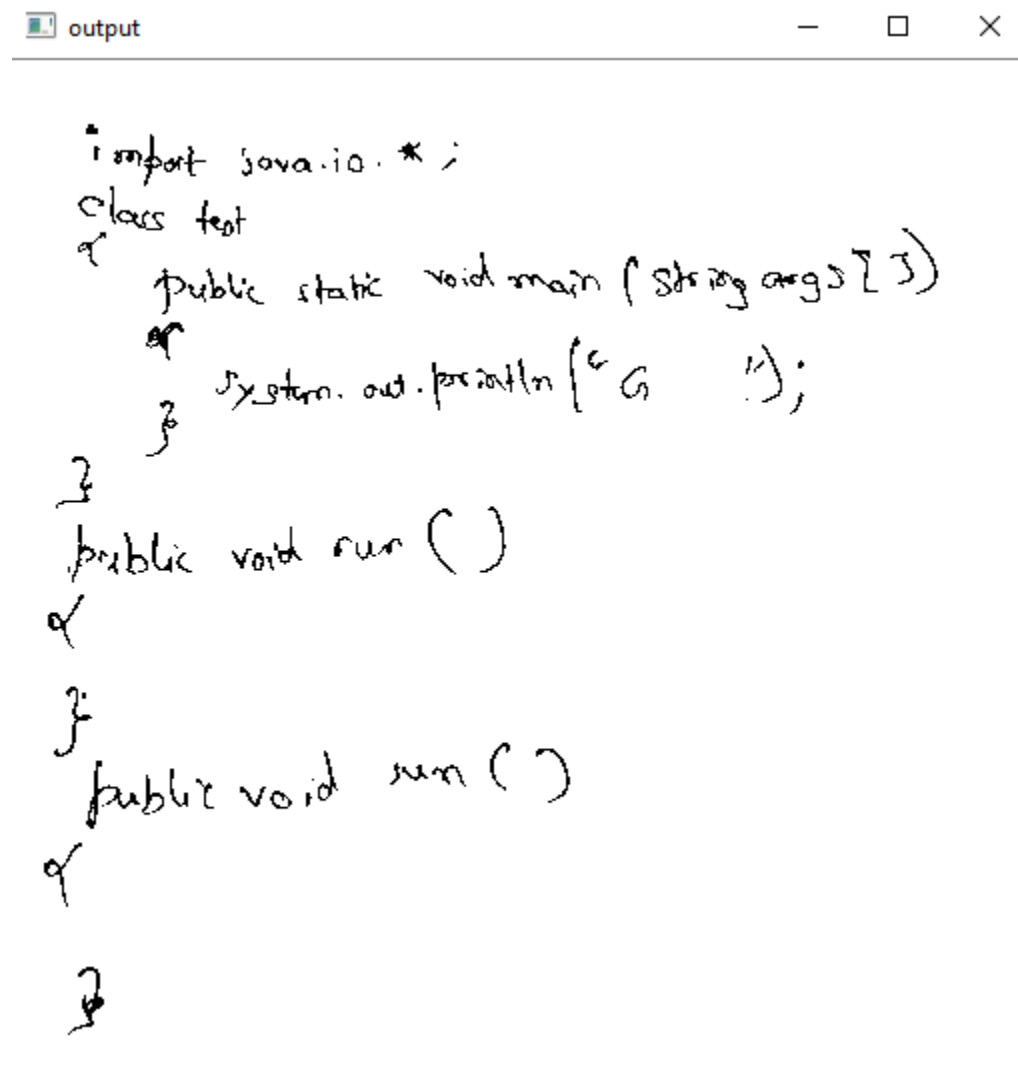
Imagine a vast sheet of paper on which straight lines, Triangles, Squares, Pentagons, Hexagons, and other figures, instead of remaining fixed in their places, move freely about, on or in the surface, but without the power of rising above or sinking below it, very much like shadows - only hard and with luminous edges - and you will then have a pretty correct notion of my country and countrymen. Alas, a few years ago, I should have said "my universe": but now my mind has been opened to higher views of



Input:

```
import java.io.*;  
class test  
{  
    public static void main (String args[])  
    {  
        System.out.println("G N");  
    }  
}  
public void run ()  
{  
    public void run ()  
}
```

Output:

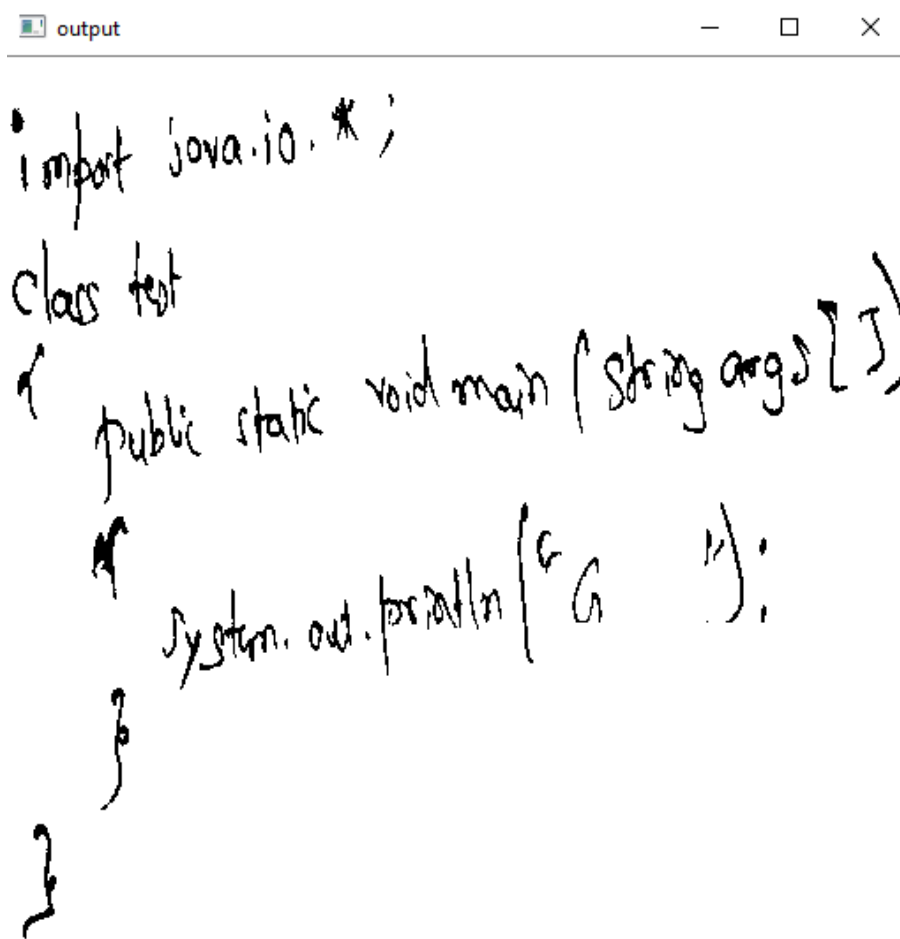


```
import java.io.*;
class test
{
    public static void main (String args[])
    {
        System.out.println("G");
    }
    public void run ()
    {
        public void run ()
    }
}
```

INPUT:

```
import java.io.*;  
class test  
{  
    public static void main (String args[])  
    {  
        System.out.println("G");  
    }  
}
```

OUTPUT



The screenshot shows a window titled "output" with standard Windows window controls (minimize, maximize, close). Inside the window, the same Java code as in the input block is displayed, followed by its output: "G".

```
import java.io.*;  
class test  
{  
    public static void main (String args[])  
    {  
        System.out.println("G");  
    }  
}
```

G