ASSESSED EXERCISE 1

jOEL BAILEY – S5015120

2019

**Otsu’s Thresholding Technique**

Otsu’s thresholding technique is a popular thresholding algorithm used to binarize grayscale images. Otsu’s technique is used to automatically determine the optimum threshold value for a grayscale image, using the image’s histogram. Once the threshold value has been determined the grayscale image can then easily be converted into a binary image. Otsu’s algorithm works by obtaining the intensity histogram of the grayscale image. It will then iterate through every possible threshold value, this is usually a value between 0 and 255. In each iteration the spread of the pixels on each side of the threshold (background and foreground) is calculated. The goal of the algorithm is to find the threshold value where the sum of the background and foreground spreads is at its minimum. Once the optimum threshold has been determined a binary image can then be created. This is done by setting the pixel values that are lower than or equal to the threshold value to 0 (the background) and setting the pixel values that are greater than the threshold value to 255 (the foreground).

Otsu’s algorithm is represented mathematically in Equation (1), where σ2W is the overall variance or the ‘Within-Class Variance’. The threshold with the smallest overall variance is considered the optimum threshold. The weight of both the background and foreground are then represented with Wb and Wƒ respectively. Equations (2) and (3) show how the weights are calculated. The variances of both the background and foreground are illustrated with σ2b and σ2ƒ . Equation (4) shows how these variances are calculated. Within Equation (4) u(t) represents the average weight of either the foreground or background, depending on which one is being calculated. The respective weight is then also represented by W.

σ2W = Wb σ2b + Wƒ σ2ƒ (1)

Wb = (2)

Wƒ = (3)

σ2b = σ2ƒ = (4)

Figure 1 and Figure 2 demonstrate how Otsu’s algorithm works and the affect that it has on the image. Figure 1 is the original greyscale image. Figure 2 represents the binary image that was created using the threshold value determined by Otsu’s algorithm. As can be seen Figure 2 clearly defines the foreground and background elements of the image displayed in Figure 1. It must be noted however as Otsu’s algorithm performs its calculations based on the image’s intensity histogram, the nosier the image is, the harder it will be to determine an optimal threshold value.

Figure 1. Greyscale Image Figure 2. Binary Image

**Implementation of Otsu’s Thresholding Technique**

The above approach to Otsu’s algorithm is useful in theory however, when it comes to implementing it into a computer program it becomes computationally intensive. To overcome this problem, algebraic rules can be applied to Equation (1) in order to achieve Equation (5). Equation (5) sets out to find the ‘Between Class Variance’. Both the ‘Between Class Variance’ and the ‘Within Class Variance’ (overall variance) share a relationship in which the threshold with the largest ‘Between Class Variance’ also contains the smallest ‘Within Class Variance’. This means that the threshold that contains the largest ‘Between Class Variance’ is the optimum threshold. As can be gathered, the σ2B symbol in Equation (5) represents the ‘Between Class Variance’. Just like in Equation (1), Wb and Wƒ represent the weight of the background and foreground. The average background weight and foreground weight are both represented by Ub and Uƒ respectively. It is evident that it will be far quicker to calculate the optimal threshold using Equation (5), thus making it the ideal approach to use when implementing Otsu’s algorithm into a program.

σ2B = Wb Wƒ(Ub - Uƒ)2  (5)

To successfully implement Otsu’s algorithm Equation (5) will need to be transformed into writeable code. Figure 3 shows the pseudocode implementation of Equation (5) to perform Otsu’s Thresholding in a program. Comments are represented with ***Bold italics***and nesting is representing with the use of indentations.

1 ***Intialize variables***2 total = image\_width \* image\_height  
3 threshold, sum, sumBackground, weightBackground, weightForeground, varianceMax,   
 varianceBetween, averageBackground, averageForeground = 0

4 ***Calculate sum value***5 for i = 0; i < 256; i++  
6 sum += i \* histogram[i]

7 ***Iterate through every threshold value***8 for i = 0; i < 256; i++  
9 ***Add correct value to weightbackground. Continue to next iteration if weightBackground = 0.***10 weightBackground += histogram[i]  
11 if weightBackground == 0  
12 continue

13 ***Calculate weightForeground value. Break loop if weightForeground = 0.***14 weightForeground = total – weightBackground  
15 if weightForeground == 0  
16 break

17 ***Calculate sumBackground***18 sumBackground += i \* histogram[i]

19 ***Calculate average values***20 averageBackground = sumBackground / weightBackground  
21 averageForeground = (sum – sumBackground) / weightBackground

22 ***Calculate variance between***23 varianceBetween = (weightBackground \* weightForeground) \*   
 (averageBackground – averageForeground)2

24 ***If varianceBetween is larger than varianceMax set new threshold value and max value***25 if varianceBetween > varianceMax  
26 varianceMax = varianceBetween  
27 threshold = i

28 ***Return threshold value***29 return threshold

Figure 3. Pseudocode implementation of Otsu’s algorithm.

Figure 3 shows a simplified implementation of Otsu’s algorithm. The algorithm has been successfully implemented in C++ within the ‘otsu.cpp’ file. The ‘otsu.cpp’ file contains a class name “Otsu” which contains a number of functions that allow for a bitmap image to be read and transformed into both a grayscale image and a binary image. To transform the image into a binary image a function called “otsuThreshold” is used. The “otsuThreshold” function implements the pseudocode algorithm presented in Figure 3 to find the optimum threshold value for the image. This threshold value is then used by the “createBinary” function to create a binary image from the grayscale image.