Adaptive Thresholding

Konrad Krzysztof Jarczyk   Sarvin Torkamanzadeh

12 November 2023

1 Summary

Image thresholding is an important technique in processing digital images. It helps to separate the objects in the foreground from the background using a cutoff value called threshold, which is based on particular pixel intensity values. Thresholding methods are divided into two groups. The first group is traditional global thresholding, which is straightforward and uses one threshold for the whole image. The problem with this class of method is that it struggles with images that have uneven illumination or objects with various shades. The second thresholding class is called adaptive thresholding, which instead of finding one threshold value, finds many of them dynamically, to obtain the desired output.

Throughout our project, we show and explain several methods for both thresholding. From global thresholding we explain methods such as Ridler-Calvard and Otsu's, which try to find the best threshold by examining how the brightness values spread out. However, these methods don't always work well when the lighting is patchy or the foreground and background don't stand out from each other enough.

The focus then shifts to adaptive thresholding, which is more flexible. Methods like Chow-Kaneko adjust the threshold in different parts of the image, while Niblack's method takes into account the mean and local variation in a neighborhood of pixel. Sauvola's method improves Niblack by adding flexibility to the threshold within that neighborhood. Experiments are carried out using libraries like OpenCV and scikit-image. The study concludes that adaptive thresholding is very useful for complex images, offering a better way to highlight the important parts of an image.

2 Introduction

Image thresholding stands as one of the fundamental techniques in the realm of digital image processing, playing an important role in simplifying visual data for further analysis. At its core, thresholding involves the segmentation of an image into foreground and background by applying a threshold value to the pixel intensities. This seemingly simple technique lays the groundwork for a broad spectrum of applications, from object detection to image improvement [8].

The essence of traditional global thresholding lies in its simplicity: a single threshold value is applied uniformly across the entire image. However, this simplicity is both its strength and its Achilles' heel. Global thresholding, despite its wide usage, often falters in scenarios where lighting conditions are inconsistent or where the objects of interest exhibit varying intensities.

This is where adaptive thresholding enters the scene. Unlike its global counterpart, adaptive thresholding tailors the threshold value to local image characteristics, offering a dynamic solution to the challenges posed by uneven illumination and complex backgrounds.

In this project, we delve into the theory of both global and adaptive thresholding, aiming to show their strengths, limitations, and their applicability.

Our format through this project is twofold: first, to provide a theoretical understanding of these thresholding techniques and secondly, to compare global and adaptive thresholding through a series of experiments. By implementing and analyzing these methods using Python and some libraries, we aim to demonstrate the scenarios where adaptive thresholding outshines global thresholding. Through this research, we intend to underscore the significance of adaptive thresholding in overcoming the limitations of global thresholding.

3 Theory

The theory section provides a structured overview of image thresholding, it progresses from the foundational grey-level concepts to sophisticated adaptive thresholding strategies and finishes with the latest advancements in the field.

3.1 Introduction to Grey-level Thresholding

Grey-level thresholding is a fundamental method that is used in both the segmentation and transformation of images, primarily to differentiate the image into foreground and background components. This differentiation is based on the similarity in the intensity values of the pixels, which is determined by analyzing the image's histogram alongside a predefined threshold [5].

$$I'(x,y) = \begin{cases} 1 & \text{if } I(x,y) > \tau \\ 0 & \text{otherwise} \end{cases} \quad [5] \tag{1}$$

Here, pixels identified as part of the foreground are labeled as 'ON' (represented by the value '1'), while those in the background are labeled as 'OFF' (indicated by the value '0'). As a result of the thresholding process is a binary image where the gray-scale values are converted into black and white, a process known as binarization [4]. While grey-level thresholding is praised for its straightforward approach, the main challenge lies in the proper selection of the threshold value. This selection is crucial as it heavily relies on the peaks within the histogram, yet spatial details are not taken into account [11].

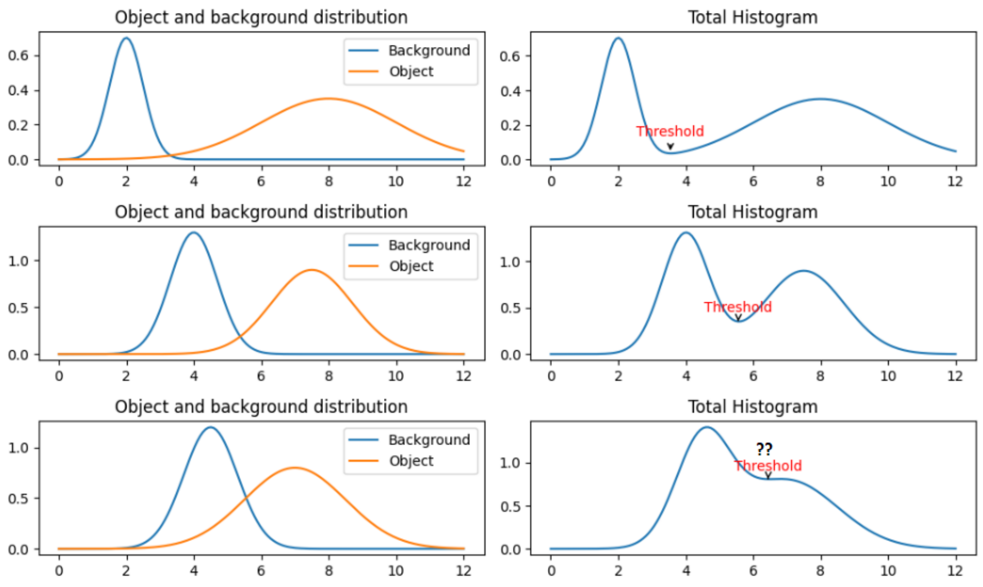

Figure 1: Thresholds founds with histogram

3.2 Global Thresholding

Global thresholding is a method that is used when the object and background pixel distributions are adequately distinct [10]. In this method, the entire image is segmented with a single threshold value. So the value of T will be constant for the whole image. This threshold is chosen by looking at the histogram of the pixel value. Usually, the histogram shows 2 peaks and a valley in between them. The 2 peaks represent the population of pixel values in the background and foreground. Figure 1 illustrates this context. By choosing a value in the minimum of the valley a threshold can be found to differentiate the foreground pixels from the background [5]. Generally, a good threshold can be chosen when the histogram peaks are high, thin, balanced, and separated by deep valleys [3]. Several thresholding methods fall into this classification such as Basic

Histogram-Based Thresholding which is explained above, Iterative-based thresholding like Ridler-Calvard algorithm, and Otsu's.

Ridler-Calvard algorithm This method works optimum when the variance of the the object pixel and background is similar. In this iterative algorithm on the first run that i=0 an initial t0 is chosen. Based on this threshold the mean of the foreground,μf_i, and background, μb_i, is calculated. The new threshold is $\frac{\mu f_i + \mu b_i}{2}$. The next round of iteration, i+1, starts with this new threshold and the algorithm repeats until the threshold values converge [5]. This method is a specific one-dimensional example of K-means clustering. The primary drawback, though, is that a different initial estimate for T can produce a different outcome [10].

Otsu's method This is a more frequently used method and unlike the Ridler-Calvard algorithm, it can work well when foreground and background have different variances on the histogram of pixels population. Otsu's method's goal is to find a threshold that maximizes the variance of the foreground and background between the 2 regions,$\sigma_b^2(t)$, and minimize it within each region, $\sigma_w^2(t)$. Since the total variance of the entire image is a constant variance of $\sigma^2(t)= \sigma_w^2(t)+\sigma_b^2(t)$ it is enough to find the threshold of one of them. For example, we can just find the threshold that maximizes the between classes. We can write the $\sigma_b^2(t)$ as this function:

$$\sigma^2(\tau) = \frac{(m_1(\tau) - \mu m_0(\tau))^2}{m_0(\tau)\,(m_0(H-1) - m_0(\tau))} \quad [5] \tag{2}$$

where τ is the threshold, H is 255 of the maximum pixel value in the image, $m_0(\tau)$ is the zeroth moment, representing the cumulative "weight" of pixel values in an image below or equal to a specified threshold τ. $m_1(\tau)$ is the first moment, signifying the weighted sum of pixel intensities in an image below or equal to a threshold τ.

After that, we test the function(2) with all thresholds from 0 to 255 as input and find the maximum number as output. The threshold that makes the largest output is selected as the best threshold value. If we have the information of the probability density function of the pixel values of background and foreground we can use that detail to optimize the resulted threshold from Otsu's method. Otsu's method's primary flaw is its assumption of a bimodal histogram. This approach doesn't work with two classes of different sizes or with uneven illumination [10]. Another drawback is that The method breaks down when the object and the background have very unequal sizes [3].

3.3 Adaptive Thresholding

In image processing, non-uniform lighting and shadows, often caused by varying light source directions, present significant challenges. These conditions can obscure details and lead to inaccurate interpretations of imagery data. While low-pass filtering [8] and background modeling [6] are viable preprocessing strategies, they may not be sufficient when global thresholding is applied.

Adaptive Thresholding offers a solution by allowing the threshold value to dynamically adjust to the local characteristics of different image regions. This approach acknowledges the spatial variations in illumination and compensates for them accordingly. In literature, terms such as Variable, Adaptive, and Local thresholding often overlap, but they all encapsulate the idea of a spatially variable threshold.

The adaptive approach is subdivided into two categories: Local Threshold, where the threshold T changes according to the pixel's neighborhood (x,y); and Function-Based Adaptive Threshold, where T is a function dependent on the pixel's coordinates [11]. Both strategies are designed to enhance the segmentation quality in the presence of uneven lighting, as demonstrated in Figure 2 with segmented overlapping sub-images and their corresponding histograms.

Chow-Kaneko Method The Chow-Kaneko method involves dividing the image into smaller overlapping sub-images of size m x m and subsequently selecting an appropriate threshold for each of these sub-images [10] based on the individual blocks's own histogram. After we have divided the image into typically 7x7 blocks we can apply global thresholding methods like Otsu's to get the segmentation that we want[9]. A bimodality test is taken over the threshold of the histogram of each region. Unimodal distributions are ignored because they

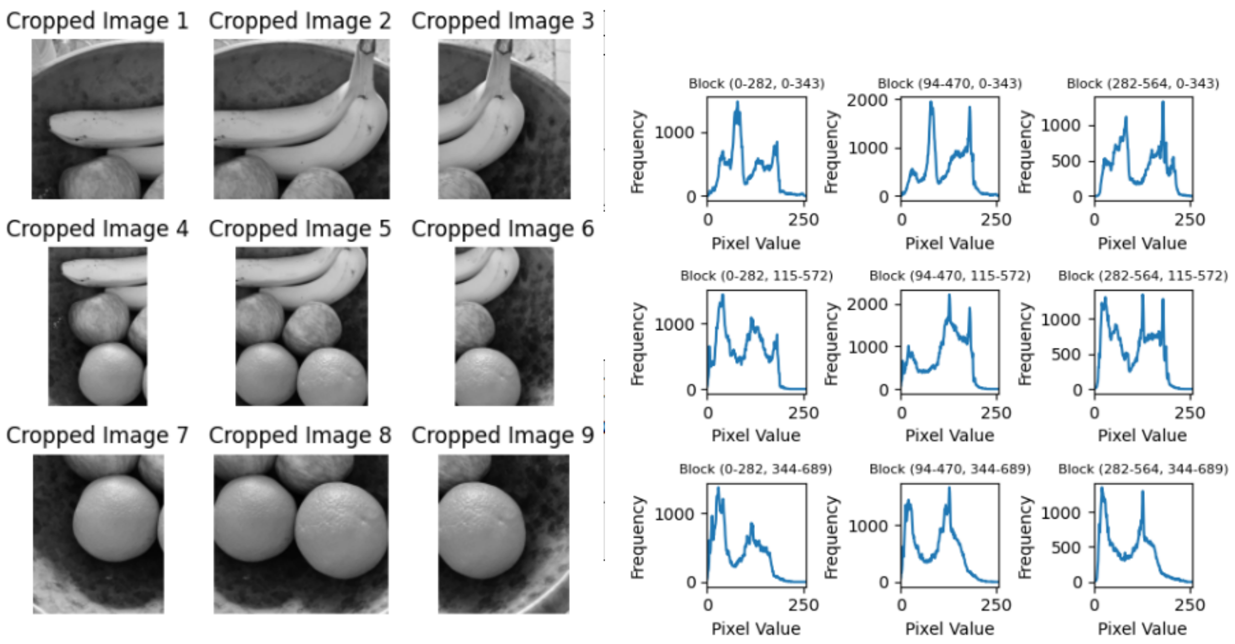

Figure 2: Blocks of overlapped sub-images and their histograms

are assumed to not provide any useful information for modeling the background intensity variation. However, bimodal distributions are well-suited for this task so they are preserved. These are individually fitted to pairs of Gaussian distributions with adjustable height and width, and the threshold values are located. Thresholds are then found for the unimodal distributions by interpolation, and finally, a second stage of interpolation is necessary to find the correct thresholding value at each pixel [6]. This method needs heavy computation, and in real-time applications, it may not be very useful especially if the sub-images are too small in size.

Mean-Based Local Thresholding One way to obtain a more reliable local threshold for each pixel is to analyze the intensities in the neighborhood of that pixel. This can be done by using a suitable function to input the nearby intensity values and output the threshold value. One simple approach is to find the mean of the local intensity distribution [6]. A straightforward approach involves applying a smoothing kernel, such as a Gaussian filter or Box filter, to convolve the image at the x and y coordinates. Subsequently, the threshold is determined as:

$$\tau_{(x,y)} = t \cdot \mu_{(x,y)} \quad [5] \tag{3}$$

where $0 \le t \le 1$ is a scalar value, and $m_{1_{x,y}}$ represents the mean of the local neighborhood. This process results in each pixel being activated (ON) if its value exceeds 100 times t percent of the neighborhood mean. For example, if t is 0.7 and the mean of the neighborhood of a pixel value is 120 then the pixel will be set to OFF if the pixel value is less than or equal to 0.7*120= 84. Other than Box filter and Gaussian filter other operations such as median can be used on the neighborhood [5].

Niblack's method In order to reduce the noise, in 1985, Niblack reported adding the local standard deviation to the mean to give a more suitable threshold value. He was working on thresholding stars in astronomical images [6].The formula is :

$$t_{1_{(x,y)}} = \mu_{(x,y)} + k \cdot \sigma_{(x,y)} \quad [5] \tag{4}$$

In this formula the μ is the mean of the neighborhood of x and y, the σ is the standard deviation, and the k is a scalar value that is recommended to be 0.2 [5]. The problem with this method is that it is susceptible to noise if there is very little variation in the image and the threshold is just close to the mean.

Sauvola's method Sauvola method just adds weight to the standard deviation by its dynamic range to fix the problem in Niblack's method.

$$\tau(x,y) = \mu(x,y) \cdot \left[1 - k \cdot \left(1 - \frac{\sigma(x,y)}{r}\right)\right] \quad [5] \tag{5}$$

k can be a value between 0.2 and 0.5, while r is assumed to be the maximum variance value equal to 128 for an 8-bit image. In this method, we consider the dynamicity of the neighborhood's variances. If the variance is high in a neighborhood then the value will be close to r which means that the threshold will be close to the Mean since the Mean will be multiplied with 1. On the other hand, if the contrast is very low then the amount of decrease is a non-negative number calculated from the fraction of $\frac{\sigma(x,y)}{r}$ which is between 0 and 1 (since σ is always less than or equal to r). This decrease in threshold value could be at minimum equal to $\mu \cdot (1-k)$. For example, if k is 0.4 then the threshold varies between $\mu(x,y)$ and $0.6\mu(x,y)$

Bernsen's technique Bernsen's Technique is a method for local binarization that leverages the concept of local contrast to compute individual pixel thresholds. It calculates the local threshold value for each pixel (x, y) by analyzing the maximum (Imax) and minimum (Imin) gray levels within a w × w window centered at that pixel. This threshold assignment relies on the evaluation of local contrast conditions and can be described as follows:

If the difference between Imax and Imin exceeds a specified contrast threshold, denoted as L, indicating that the grayscale variation is significant in the local region, then T (x, y) is set to :

$$T(x,y) = \begin{cases} \frac{Imax+Imin}{2} & \text{if } Imax - Imin > L \text{ (if the grayscale image is not uniform)} \\ GT & \text{else (threshold value calculated by global thresholding technique)} \end{cases} \quad [13] \tag{6}$$

However, if the contrast variation within the window is not substantial, implying a relatively uniform area in the grayscale image, T (x, y) adopts a threshold value calculated using a global thresholding technique and referred to as GT. Here, L represents the contrast threshold which is a value that should be tuned by you.

3.4 Advanced Thresholding methods

Recent advances in adaptive thresholding have seen the development of techniques that use machine learning to get optimal thresholding parameters dynamically. Convolutional neural networks (CNNs), for instance, can be trained on a dataset of images to learn the variability in illumination and texture, thus predicting threshold values that adapt to the specific content of each image [17].

Furthermore, researchers have explored the use of genetic algorithms and other optimization strategies to automate the parameter selection process for adaptive thresholding methods [12]. These approaches simulate evolution by iteratively selecting, combining, and mutating parameters to optimize a fitness function that measures the quality of the thresholding.

In practical terms, these advanced techniques have the potential to reduce the manual effort required in parameter tuning and to improve the robustness of thresholding in challenging scenarios. For example, in medical imaging, where accurate segmentation of images can be critical, advanced adaptive thresholding can lead to more precise diagnostics and treatment plans [7].

4 Implementation

This section details the practical application of the theoretical concepts outlined in the Theory section, specifically how various thresholding techniques are realized in a computational environment. It begins with the essential preprocessing steps that prepare images for subsequent algorithms, transitions into the application of global thresholding methods, and then into adaptive thresholding techniques. Nexly, we discuss the process of parameter tuning, followed by experimentation and lastly, the software and libraries utilized throughout the implementation are listed.

4.1 Preprocessing

In the chapter "Segmentation" from the book "Image Proccessing and Analysis" [5], Birchfield highlights the importance of preprocessing techniques before applying thresholding algorithms to images. He suggests that these preprocessing steps can improve the effectiveness of thresholding algorithms.

Smoothing A Gaussian blur is applied to reduce image noise and detail using the `cv2.GaussianBlur` function, which helps in making the histogram peaks more distinct and thus facilitates better threshold selection. Implemented with `cv.GaussianBlur(img, (5, 5), 0)` [14].

Gray-scale Conversion Images are converted to grayscale with `cv2.cvtColor` to simplify the data for thresholding, as color information is not essential for the segmentation process.

4.2 Global Thresholding Techniques

Global thresholding techniques, facilitated by `OpenCV`, apply a uniform threshold value across the entire image.

Basic Binary Thresholding Implemented with `cv2.threshold(img, thresh, maxval, cv2.THRESH_BINARY)`, this method assigns the maximum value to pixels above the threshold and zero otherwise.

Inverse Binary Thresholding Implemented with `cv2.threshold(img, thresh, maxval, cv2.THRESH_BINARY_INV)`, this method inverts the binary outcome, assigning the maximum value to pixels below the threshold.

Truncate Thresholding Using `cv2.threshold(img, thresh, maxval, cv2.THRESH_TRUNC)`, this method truncates all pixel values above the threshold to the threshold value itself.

Threshold to Zero This technique, implemented via `cv2.threshold(img, thresh, maxval, cv2.THRESH_TOZERO)`, sets pixels below the threshold to zero, leaving others unchanged.

Threshold to Zero Inverse An inverse of the above, implemented with `cv2.threshold(img, thresh, maxval, cv2.THRESH_TOZERO_INV)`, sets pixels above the threshold to zero.

Otsu's Binarization Otsu's method is implemented using `cv2.threshold(img, 0, maxval, cv2.THRESH_BINARY + cv2.THRESH_OTSU)` that automatically determines the optimal threshold value based on the minimization of within-class variance.

Ridler-Calvard The Ridler-Calvard algorithm is applied to segment images based on an iterative thresholding approach. Here we describe our method.

1. An initial threshold T_0 is selected as the average intensity of the image pixels.
2. At each iteration i, the image is partitioned into foreground and background based on the current threshold T_i.
3. The average intensity of the pixels classified as foreground ($\mu_{\text{fg}}(T_i)$) and background ($\mu_{\text{bg}}(T_i)$) are computed.
4. The threshold for the next iteration T_{i+1} is updated to $\frac{\mu_{\text{fg}}(T_i)+\mu_{\text{bg}}(T_i)}{2}$.
5. Steps 2-4 are repeated until the change in threshold value between consecutive iterations is below a predefined small threshold ϵ, indicating convergence [8].

The process is encoded using the Python programming language and OpenCV library functions. The initial threshold is chosen by calculating the mean of the grayscale image using the OpenCV function `cv2.mean()`. The pixel classification and mean calculations at each iteration are performed using array operations facilitated by the NumPy library. Iterations continue until the absolute difference between successive threshold values is less than ϵ, set to a value of 0.01 to ensure an accurate convergence without excessive computation. The code for the Ridler-Calvard algorithm is provided in Appendix A.

4.3 Adaptive Thresholding Techniques

Adaptive thresholding techniques are implemented using the OpenCV and scikit-image libraries.

Adaptive Mean Thresholding This method is based on the mean of the neighbourhood pixels. The function call in OpenCV is:

```
cv2.adaptiveThreshold(src, maxValue, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, blockSize, C)
```

Adaptive Gaussian Thresholding This method uses a Gaussian window to weigh the neighbourhood pixels. The corresponding function in OpenCV call is:

```
cv2.adaptiveThreshold(src, maxValue, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, blockSize, C)
```

Niblack's Method Implemented in scikit-image, this method considers the local mean and standard deviation of the pixels. The function call is:

```
skimage.filters.threshold_niblack(image, window_size, k)
```

Sauvola's Method Implemented in scikit-image, it adapts the thresholding based on local contrast, with the standart deviation weighted by its dynamic range:

```
skimage.filters.threshold_sauvola(image, window_size, k, r)
```

4.4 Parameter tuning

Parameter tuning is important for refining the performance of adaptive thresholding methods. The parameters such as window size, block size, constants k and C, and the dynamic range r in Sauvola's method, require optimization. The block size influences the amount of local detail that can be captured, while C adjusts the threshold to accommodate variations in brightness or contrast. The dynamic range r in Sauvola's method is crucial for determining the sensitivity to local standard deviations. Experimentation was essential to discover the settings that produced the best segmentation without adding unnecessary noise.

4.5 Experimentation

A series of experiments with two images with uneven illumination are conducted. The effectiveness of each thresholding method is evaluated based on the clarity of the foreground object or text segmentation. Results from global and adaptive thresholding are compared to assess their abilities to handle variations in illumination.

4.6 Software and Libraries Used

The implementation was carried out with Python programming language, with the following libraries:

OpenCV (cv2) For basis image processing operations [15].

scikit-image For advanced image processing operations not available in OpenCV [1].

NumPy For handling high-level mathematical functions and arrays [2].

Matplotlib For illustrating the results [14].

5 Experiments and Results

5.1 Experimentation setup

For the experimental evaluation, two types of images with uneven illumination were selected: one featuring objects and the other containing text. The images were first smoothed using Gaussian Blur.

5.2 Global Thresholding

After applying global thresholding methods, we can see that they did not yield satisfactory results for the text image, as depicted in Appendix B on Figure 3. Although some global methods like Truncate and Otsu's thresholding showed slightly better results on the image presenting objects. (Appendix B, Figure 4) Generally, they were still not optimal, especially with image containing textual content.

5.3 Adaptive methods and parameter tuning

Subsequent experiments were conducted using adaptive thresholding methods. These methods produced better segmentation on both types of images. In methods Adaptive Mean and Adaptive Gaussian thresholding adjusting the constant C improved clarity the text image, by increasing the constant C contrast of image is enhanced. Conversely, by lowering this value image gets smoother and edges less pronounced. We found out that the best results are 5-10 for image with text and 2-3 for image with objects of constant C. Results of altered images are shown in Appendix C on Figure 5 and 6.

We further tested adaptive methods such as Niblack and Sauvola. For each method, we set the constant k to 0.2 and the window size to 13. During the experiments, we noticed that using an excessively large window in Sauvola's method led to background noise and overly thick and noisy edges. On the other hand, reducing the window size causes troubles in detecting the edges and objects. Sauvola's method produced clearer results in images with textual content by effectively removing background noise. However, Niblack's method caused background noise in both images. Note that we did not tune the r parameter in Sauvola's method as it did not significantly impact the results, except when the r value was set very low, which led to a higher number of black pixels within objects, indicative of over-segmentation. Similarly, we tried to adjust constant k. We observed that increasing k led to increasing contrast, lowering threshold in darker areas, which caused background elements be classified as foreground. By decreasing k threshold raises in darker areas, causing foreground elements be classified as background. The outcomes of these methods, including the noted effects of parameter adjustments, are illustrated in the last two pictures of Appendix C, shown in Figure 5 and 6.

5.4 Results and observations

The experimentation led to several observations:

- Global thresholding techniques were generally inadequate for images with uneven illumination, especially for text.
- Adaptive Gaussian and mean thresholding methods outperformed global methods, with particular improvements by adjusting the constant C and the filter size.
- The Sauvola method provided good results for text images, given an appropriate window size and k value.
- Niblack's method was prone to background noise, suggesting its limited applicability in scenarios with uneven illumination.

These results underscore the importance of parameter tuning and the selection of suitable thresholding methods based on the characteristics of the images and the desired outcome.

6 Conclusion

The outcomes of our experiments emphasize that the choice of right thresholding method should be chosen, based on the specific needs of the application. For tasks that require precision in variable lighting environments, adaptive thresholding methods are irreplaceable. Their ability to adjust to local characteristics allows for accurate segmentation. However, using algorithms for adaptive thresholding might be computationally expensive, so in scenarios when we need real-time processing, such as live streaming, the use of global thresholding might be a better option.

References

[1] Niblack and sauvola thresholding. `https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_niblack_sauvola.html`. Accessed: 2023-11-06.

[2] Numpy documentation. `https://numpy.org/doc/stable/`. Accessed: 2023-11-06.

[3] Thresholding. PDF document. Sections 3.2.1, 3.2.2 from Jain et al.; Chapter 7 from Petrou et al.

[4] Stan Birchfield. *Graylevel Transformations*, pages 83–86. Cengage Learning, 2018.

[5] Stan Birchfield. *Segmentation*. Cengage Learning, 2018.

[6] E. R. Davies. *Thresholding Techniques*, page 4. Elsevier, 2012.

[7] L. M. Davis and R. Thompson. The impact of advanced thresholding techniques in medical imaging. *International Journal of Medical Imaging*, 2023.

[8] Kjersti Engan. Lecture notes on image processing and computer vision (2023). Lecture Notes.

[9] GeeksforGeeks. Thresholding-based image segmentation, January 16 2023. GeeksforGeeks.

[10] Singaraju Jyothi and K.Bhargavi. A survey on threshold based segmentation technique in image processing. *26. K. Bhargavi, S. Jyothi*, 3, 11 2014.

[11] Dilpreet Kaur and Yadwinder Kaur. Various image segmentation techniques: A review. 2014.

[12] K. J. Lee and H. Y. Kim. Automated parameter tuning for image thresholding using genetic algorithms. *Computational Intelligence and Optimization*, 2022.

[13] S. N and V. S. Image segmentation by using thresholding techniques for medical images. *Computer Science & Engineering: An International Journal*, 6(1):1–13, 2016.

[14] OpenAI. Chatgpt, 2023. Large language model.

[15] OpenCV. Image Thresholding — OpenCV 4.x, n.d. [Online; accessed 4-November-2023].

[16] TW Ridler and S Calvard. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man, and Cybernetics*, 8(8):630–632, 1978.

[17] J. A. Smith and E. B. Doe. Adaptive image thresholding using convolutional neural networks. *Journal of Image Processing and Machine Vision*, 2021.

A Code for the Ridler-Calvard Algorithm

Listing 1: Ridler-Calvard Thresholding Algorithm based on the method described in [16].

```python
import numpy as np
import cv2

def ridler_calvard_thresholding(image):

    # Initial guess for the threshold
    T = np.mean(image)

    # Iterate until the threshold stops changing
    delta_T = 1
    while delta_T > 0:
        # Create two groups
        G1 = image[image >= T]
        G2 = image[image < T]

        # Calculate means of both groups
        mean_G1 = np.mean(G1)
        mean_G2 = np.mean(G2)

        # Update the threshold
        new_T = (mean_G1 + mean_G2) / 2
        delta_T = np.abs(new_T - T)
        T = new_T

    # Apply the final threshold to create binary image
    _, thresholded_image = cv2.threshold(image, T, 255, cv2.THRESH_BINARY)

    return thresholded_image
```

B Results of global thresholding methods

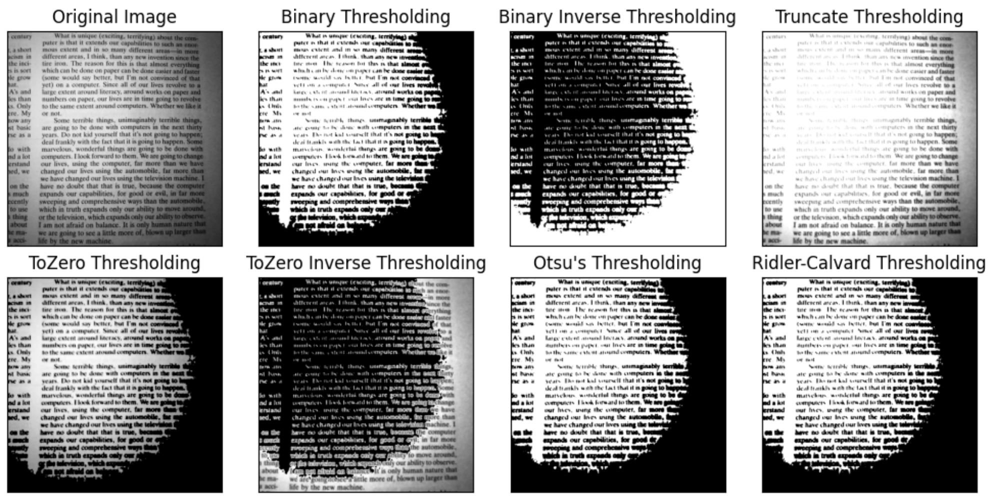

Figure 3: Global methods on image of text with uneven illumination

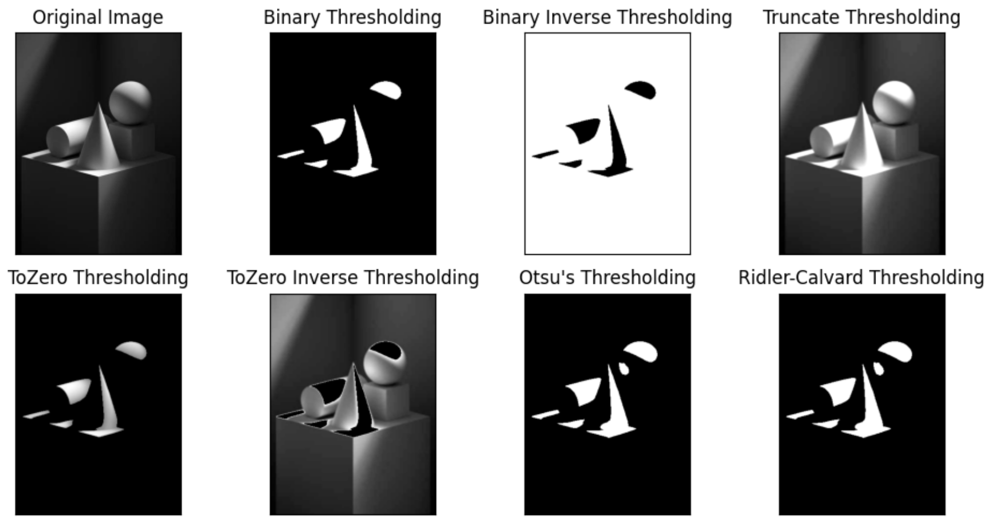

Figure 4: Global methods on image of objects with uneven illumination

C Results of adaptive thresholding methods

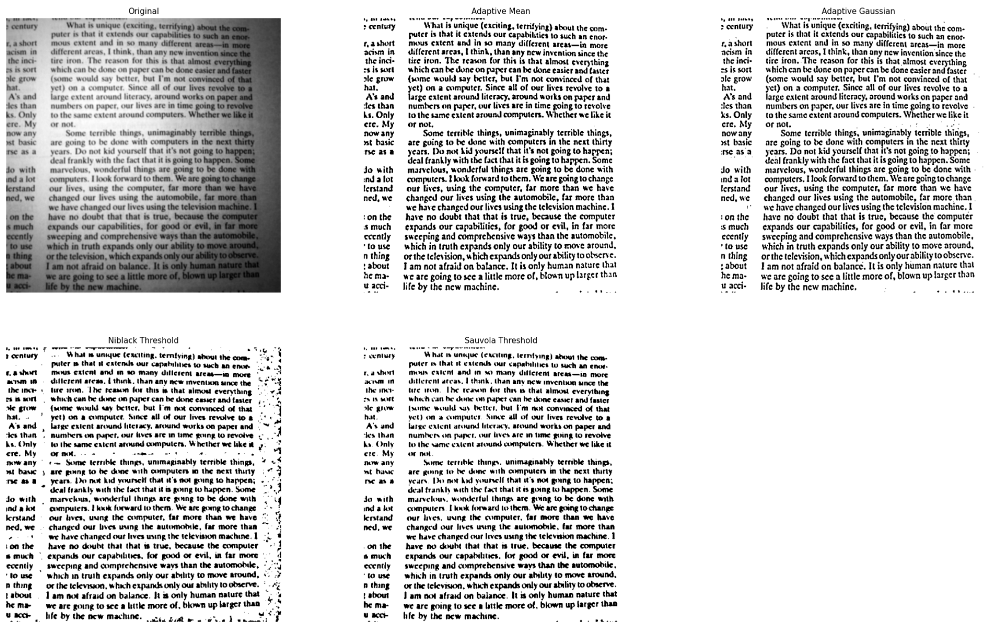

Figure 5: Adaptive thresholds methods on the image of text with uneven illumination

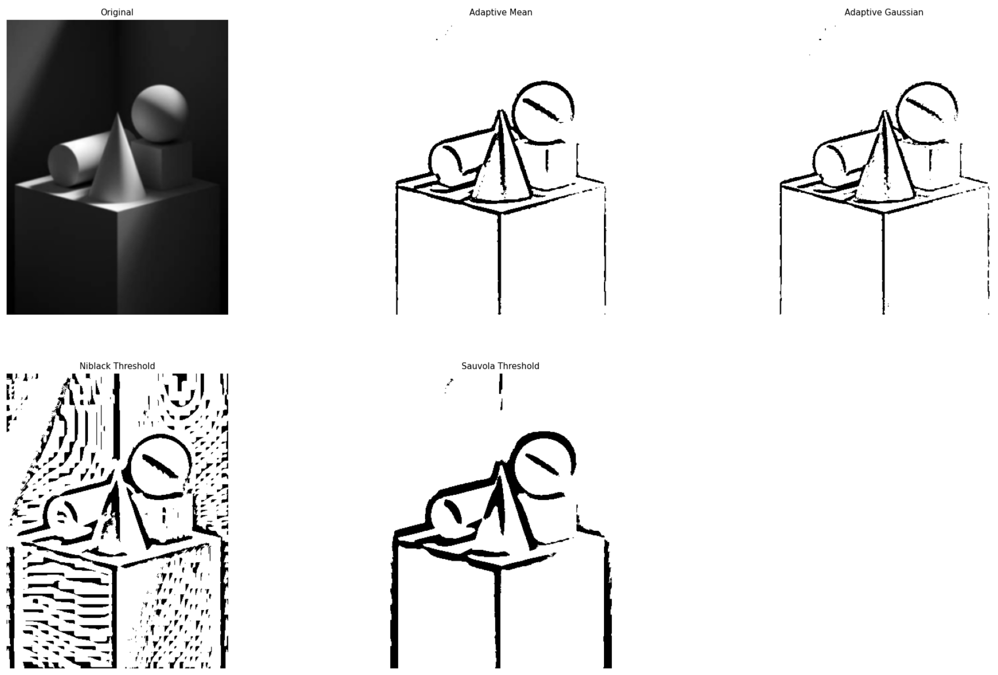

Figure 6: Adaptive thresholds methods on the image of objects with uneven illumination