CS 5330: Pattern Recognition and Computer Vision

Northeastern University

Lab 7: Thresholding

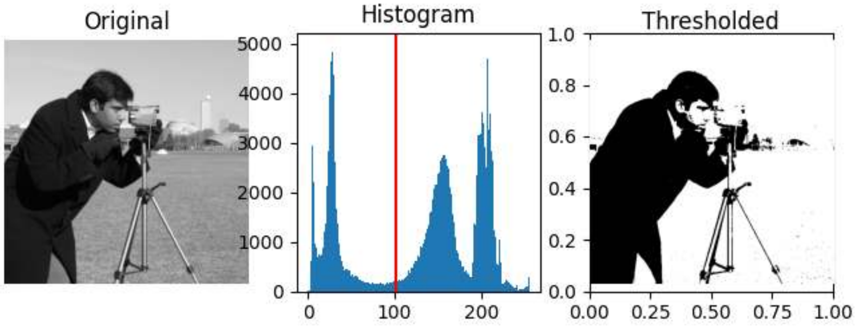
** Contributed by Fall 2024 TAs: Byunghyun Ko, Yihan Wang and Taiwei Cui*

Thresholding

1. Intro and Types of Thresholding
    1. Simple Thresholding
    2. Adaptive Thresholding
    3. Otsu’s Binarization
2. Simple Thresholding Example
3. Adaptive Thresholding Example
4. Otsu’s Binarization Example
5. Summary

Introduction to Thresholding

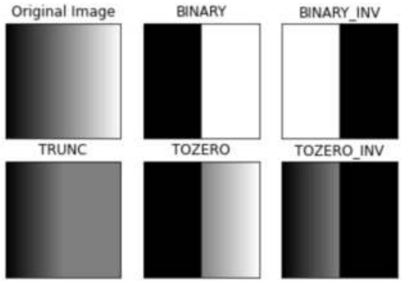
- Thresholding: A simple way to segment images by turning grayscale image into binary images
- There are different kinds of thresholding:
  - Our lab will focus on:
    - Simple Thresholding
    - Adaptive Thresholding
    - Otsu's Binarization



https://scikit-image.org/docs/stable/auto_examples/segmentation/plot_thresholding.html

Simple Thresholding

- Simple Thresholding: a fixed threshold value is used for the entire image. Each pixel value is compared to this threshold value and the pixel is assigned a binary value.
- Pros: easy to implement, works well for uniform lighting
- Cons: not good for images with varying lighting conditions



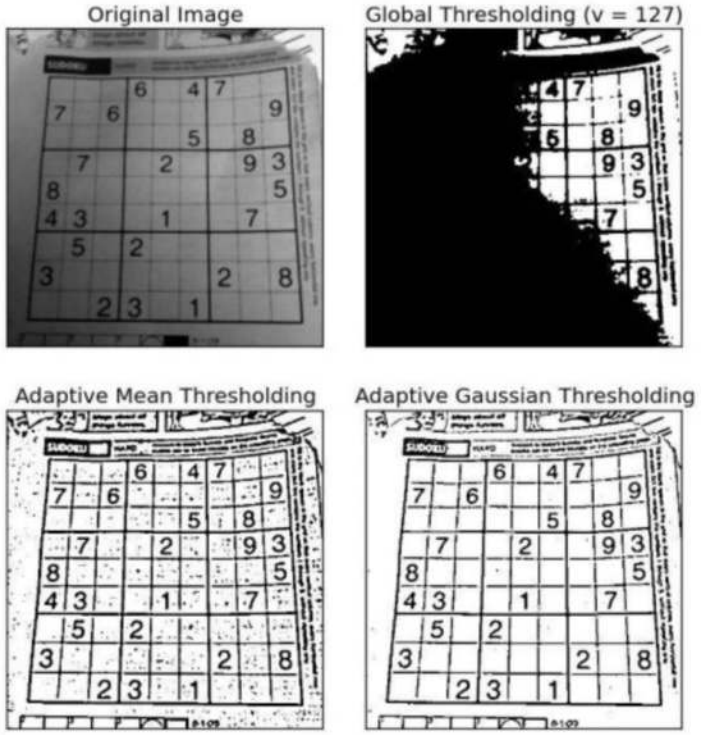
image

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Adaptive Thresholding

- Adaptive Thresholding: a threshold value is calculated for smaller regions of the image rather than the entire image.
- Helpful in cases where lighting conditions vary across an image
- Two main types of adaptive thresholding:
  - Mean Adaptive Thresholding: threshold is set to be the mean of the neighborhood values minus a constant
  - Gaussian Adaptive Thresholding: threshold is set as the weighted sum of the neighborhood values minus a constant
- Pros: handles varying lighting conditions, robust
- Con: slower than simple thresholding

Adaptive Thresholding


Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

image

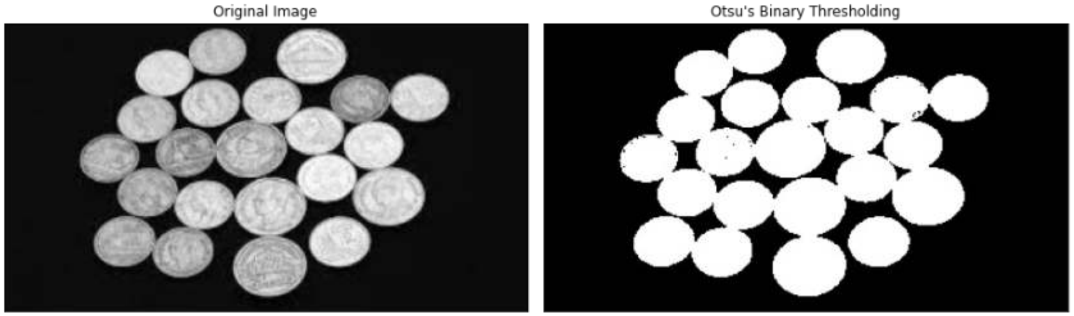
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

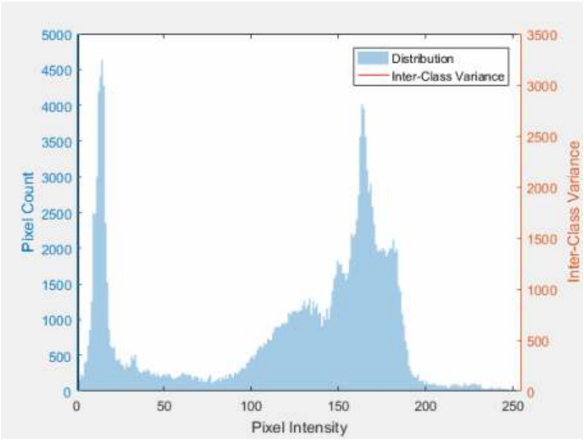
Otsu’s Binarization

- Otsu’s Binarization: automatically calculates the optimal threshold value based on the image’s histogram
- Minimizes the variance with the foreground & background pixels to find the best threshold
- Process:
  - Compute the histogram of the grayscale image
  - Automatically find the optimal threshold value
  - Apply thresholding to binarize the image
- Pros: automatic thresholding, good to use with images with bimodal histogram
- Con: doesn’t work well with more complex histograms or uneven lighting

https://www.projectpro.io/recipes/what-is-otsus-binarization-opencv
https://en.wikipedia.org/wiki/Otsu%27s_method

Otsu’s Binarization





https://www.projectpro.io/recipes/what-is-otsus-binarization-opencv
https://en.wikipedia.org/wiki/Otsu%27s_method

Simple Thresholding Example

Code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image in grayscale
image = cv2.imread('your_image.jpg', cv2.IMREAD_GRAYSCALE)

# Apply simple thresholding
_, thresh_simple = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Show the image
plt.imshow(thresh_simple, cmap='gray')
plt.title('Simple Thresholding')
plt.show()
```

cv2.threshold(src, thresh, maxval, type):

- **src**: The source image (must be a grayscale image).
- **thresh**: The threshold value.
- **maxval**: The value to be given to pixels exceeding the threshold.
- **type**: The type of thresholding to be applied (e.g., cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, etc.).

Adaptive Thresholding Example

Code

```
# Apply adaptive mean thresholding
thresh_adaptive_mean = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_MEAN_C
, cv2.THRESH_BINARY, 11, 2)

# Apply adaptive Gaussian thresholding
thresh_adaptive_gaussian = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C
,  cv2.THRESH_BINARY, 11, 2)

# Show both results
plt.subplot(1, 2, 1), plt.imshow(thresh_adaptive_mean, cmap='gray')
plt.title('Adaptive Mean Thresholding')

plt.subplot(1, 2, 2), plt.imshow(thresh_adaptive_gaussian, cmap='gray')
plt.title('Adaptive Gaussian Thresholding')

plt.show()
```

cv2.adaptiveThreshold(src, maxval, adaptiveMethod, thresholdType, blockSize, C):

- **src**: The source image (grayscale).
- **maxval**: The value to be given to the pixels that exceed the threshold.
- **adaptiveMethod**: The method for calculating the threshold (e.g., cv2.ADAPTIVE_THRESH_MEAN_C or cv2.ADAPTIVE_THRESH_GAUSSIAN_C).
- **thresholdType**: The type of thresholding to apply (e.g., cv2.THRESH_BINARY).
- **blockSize**: Size of the neighborhood area used to calculate the threshold for each pixel (must be an odd number).

Otsu’s Binarization Example

Code

```
# Apply Otsu’s binarization
_, thresh_otsu = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Show the result
plt.imshow(thresh_otsu, cmap='gray')
plt.title('Otsu’s Binarization')
plt.show()
```

cv2.threshold(src, thresh, maxval, type + cv2.THRESH_OTSU):

•**src**: The source image (grayscale).

•**thresh**: Set to 0 because Otsu’s method automatically determines the optimal threshold.

•**maxval**: The maximum value to assign to thresholded pixels.

•**type**: The thresholding method (e.g., cv2.THRESH_BINARY) combined with cv2.THRESH_OTSU for automatic threshold determination.

- In the example above, **cv2.THRESH_BINARY + cv2.THRESH_OTSU** tells OpenCV to first apply binary thresholding, but to determine the threshold value using Otsu's method.

Summary

- **Thresholding** is a great and efficient tool for image segmentation!
- **Simple thresholding** works for well-lit, simple images, but isn’t good for images with varying lighting conditions.
- **Adaptive thresholding** is works well when the lighting varies but is slower.
- **Otsu’s binarization method** is best for images with clear intensity separation, and but may struggle with images with uneven lighting or more complex images.