MSAI 495 Introduction to Computer Vision - Assignment 5

Sayantani Bhattacharya

Canny Edge Detector

Algorithms Used:

I. For Gaussian Smoothing:
   A. Load training image and convert it to a grayscale floating-point image.
   B. A Gaussian kernel is a matrix where the values are computed based on the Gaussian distribution. It gives higher weights to pixels near the center and lower weights to those farther away, creating a "blurring" effect.
   C. Use the gausian_smoothening() function to convolve the image with a Gaussian kernel.

II. For Calculating Image Gradient:
   A. Applying Sobel operators to the smoothed image: Sobel operator detects changes in intensity in the horizontal (Gx) and vertical (Gy) directions and gives Ix and Iy.
   B. Solving the gradient magnitude (Mag) using the formula: Mag = sqrt(Ix^2 + Iy^2).
   C. Solving the gradient direction (Theta) using the formula: Theta = arctan2(Iy, Ix).

III. For Selecting high and low thresholds [`find_threshold()`]:
   A. Flatten the gradient magnitude matrix into a 1D array.
   B. Sort the gradient magnitudes in ascending order.
   C. Set the high threshold (`T_high`) as the value at the `percentageOfNonEdge` percentile.
   D. So if the percentage of non-edge pixels is 0.7, then we take the 70th percent-th value of the gradient magnitude pixel values as the high threshold as the Mag value represents much of that pixel is likely to be an edge.
   E. Set the low threshold (`T_low`) as half of the high threshold (`T_high`).

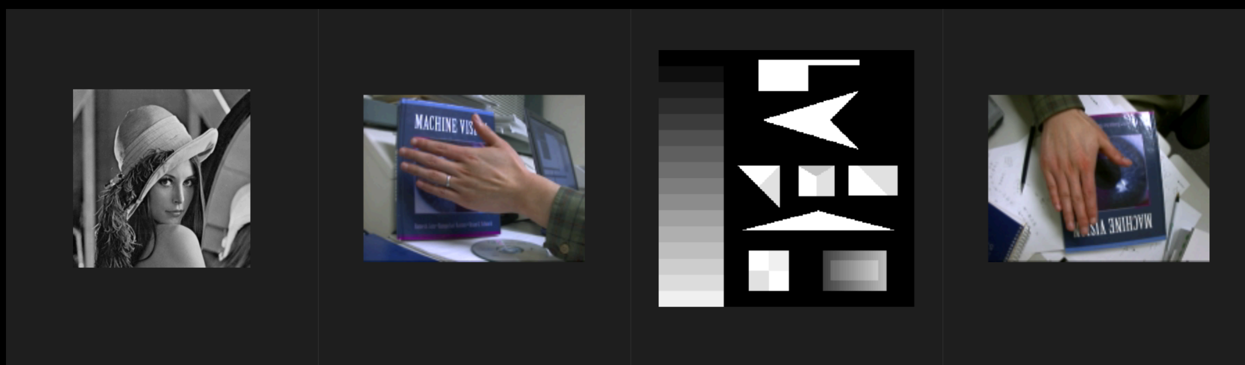
IV. For Suppressing Nonmaxima [`nonmaxima_suppress()`]:
   A. Iterate through each pixel in the gradient magnitude matrix (`Mag`).
   B. For each pixel, check its two neighbors along the gradient direction (`Theta`).
   C. Suppress the pixel if its magnitude is not greater than or equal to both neighbors.
   D. The result is a thinned edge map where only the strongest edge pixels remain.

V. Thresholding and Edge Linking [`edge_linking()`]:
   A. Classify pixels based on the thresholds calculated previously.
   B. Mark pixels with magnitude greater than or equal to `T_high` as strong edges.
   C. Mark pixels with magnitude between `T_low` and `T_high` as weak edges.
   D. Set all other pixels to zero.
   E. For each weak edge pixel, check its 8-connected neighbors.
   F. Promote the weak edge to a strong edge if any neighbor is a strong edge.
   G. Suppress the weak edge if no strong edge is found in its neighbors.
   H. The final edge map contains only strong edges (`255`) and non-edges (`0`).

VI. Comparisons of different parameters
VII. Comparisons with different edge detectors.

Results:

1. Input Test Images:

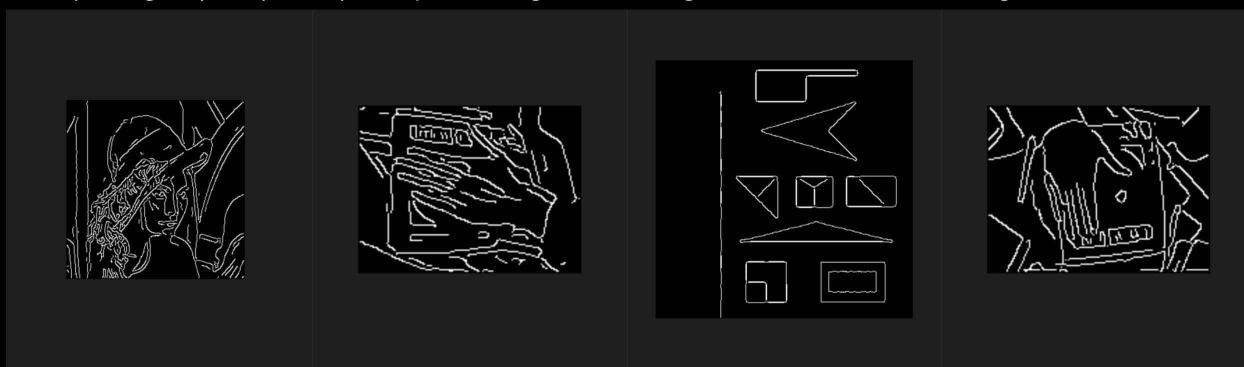


2. With Gausian filtering:

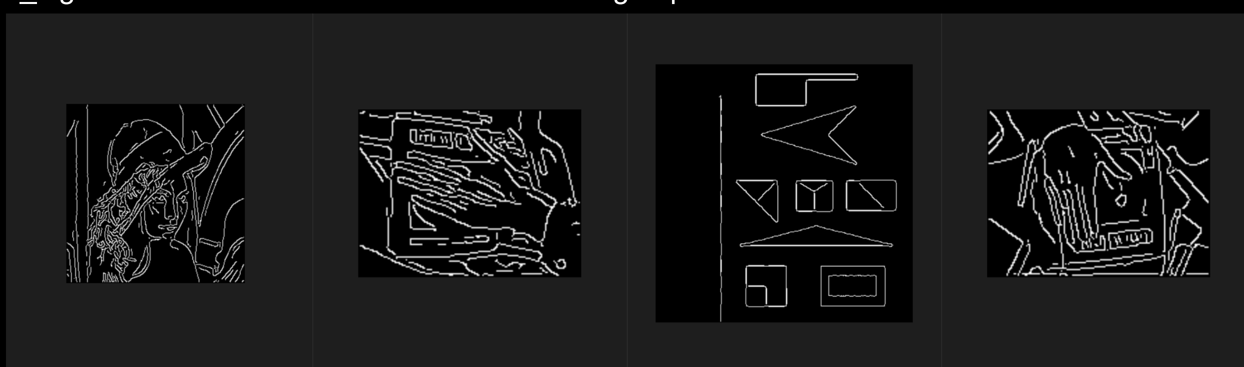
3. With (N, Sigma) as (5, 1.4) and percentageOfNonEdge = 0.7

   Based on the observation on test.bmp it seems that Over-smoothing: The output seems to have lost nearly all edge information, suggesting the edges were smoothed away too aggressively before gradient computation.And Thresholding was too aggressive: It's likely the T_high threshold is too high due to a high percentageOfNonEdge(0.7), causing valid weak edges to be ignored. **And the main issue is** that the gradient of most pixel is not too high as neighbouring colours are pretty similar, so I added a min-max thresholding which gave much better results.

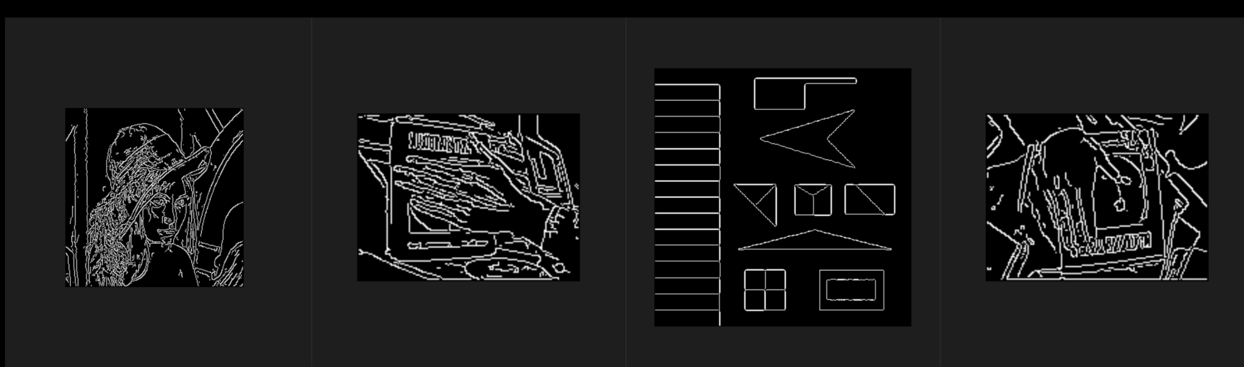
4. With (N, Sigma) as (5, 1.4) and percentageOfNonEdge = 0.7 and thresholding.



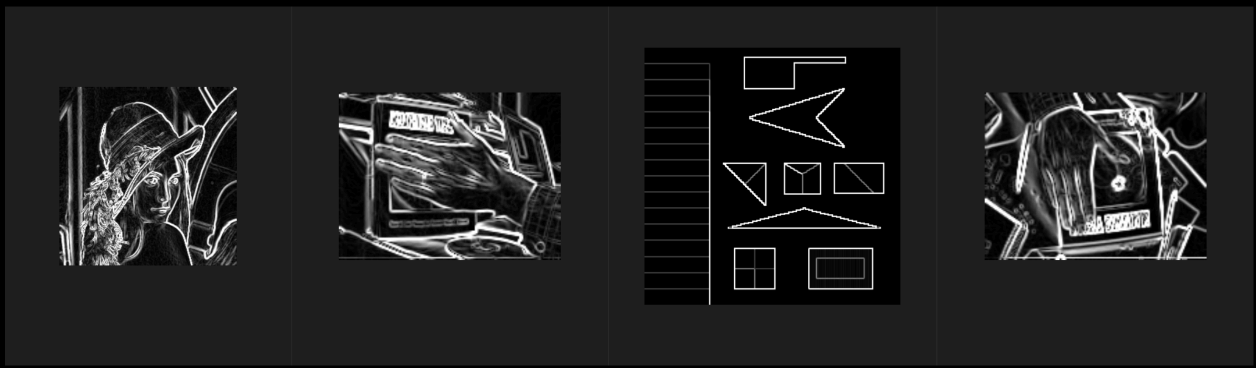
5. With (N, Sigma) as (5, 1.4) and Lowering percentageOfNonEdge to 0.2. This will lower T_high and as we can observe retains more edge options.



6. With (N, Sigma) as (3, 0.4) and percentageOfNonEdge = 0.7. This gives a much noisier output.



7. Comparison with CV’s Sobel function:



8. Comparison with CV’s Canny function: