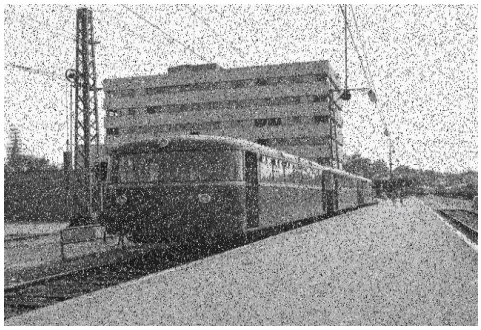
CSE 509 Fall 2022

Homework 5

Part 1: Noise Reduction by Median Filtering [20pts]

In this problem, we study how to use different types of median filtering to reduce salt-and-pepper noise in vintage photographs.



Part A:
Use images hw5_building.jpg and hw5_train.jpg. Apply median filtering with a 3x3 window and a 5x5 window (MATLAB function: medfilt2). Display and submit the resulting images. For each window size, comment on how effectively the noise is reduced while sharp edges and features in the image are preserved.

Part B:
Denoise the images of Part A using a weighted median filter.

Median filter definition: Given a set of input values $f_1, f_2, \ldots, f_N$ and weights $w_1, w_2, \ldots, w_N$, weighted median filtering repeats f_i by w_i times and then computes the median of all the repeated values:

$$g = median\left(w_1 \otimes f_1, w_2 \otimes f_2, \mathrm{L}\ , w_N \otimes f_N\right)$$

$$w \otimes f = \underbrace{f, f, \cdots, f}_{w \text{ times}}$$

Use the following 5x5 weighted median filter. The brackets around 4 indicate the center of the window:

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 2 & 2 & 2 & 1 \\ 1 & 2 & [4] & 2 & 1 \\ 1 & 2 & 2 & 2 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

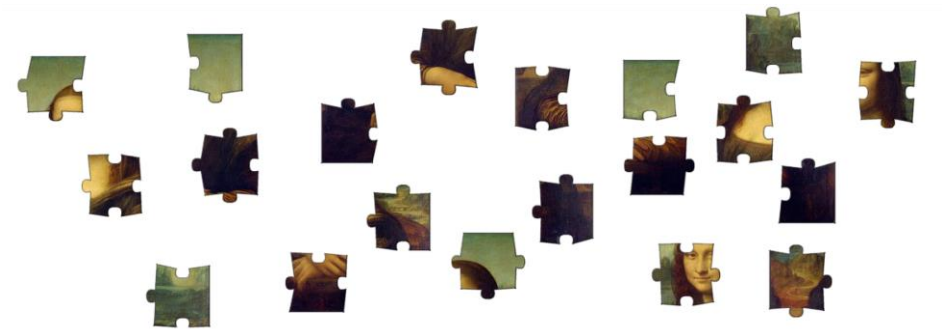
5x5 Weighted Median Filtering

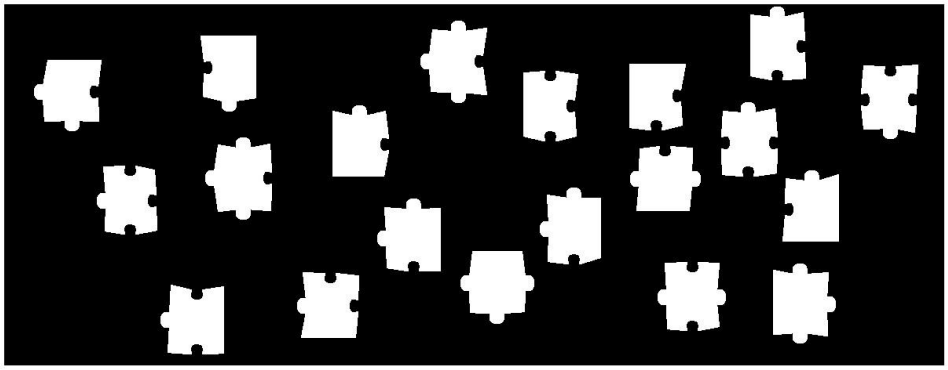
Report which method gives the best visual quality.

Part 2: Solving a Jigsaw Puzzle [40pts]

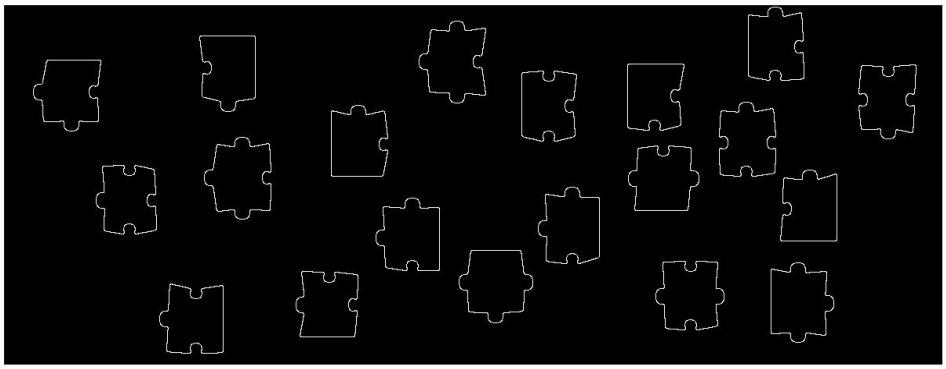
Use the input image hw4_puzzle_pieces.jpg, which shows the pieces of a jigsaw puzzle. Since we know that the completed puzzle should show a picture of the famous Mona Lisa painting, as a reference when solving the puzzle, use hw4_puzzle_reference.jpg

Step 1:
We binarize the pieces image, with the result shown below.





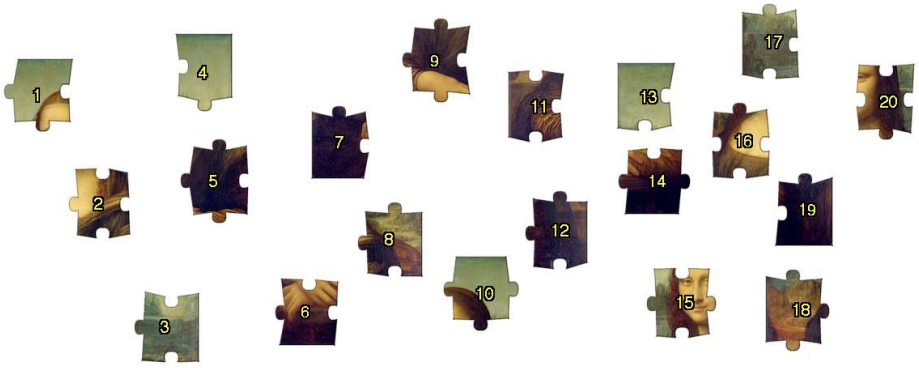
Step 2:
We extract an edge map from the binary image (result shown below), which we will use later to draw the boundaries of the jigsaw pieces on top of the reference image.



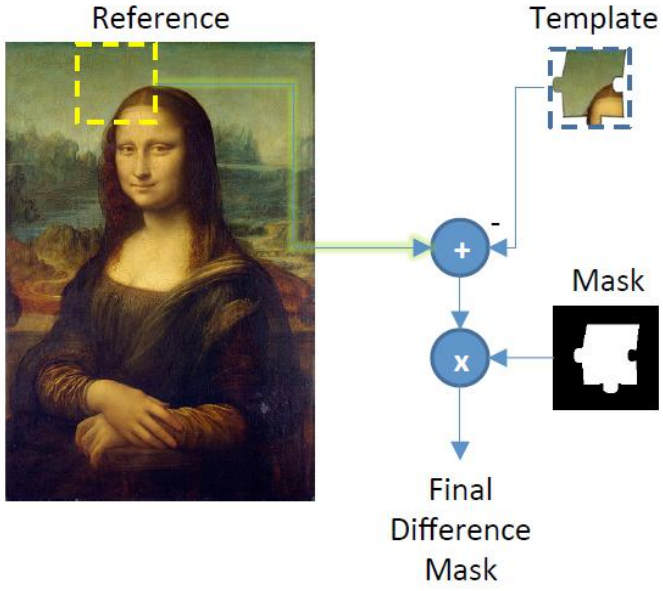
Step 3:
When the jigsaw pieces are fully assembled, the assembled image may have a different size than the reference image. To account for this mismatch, we resize the reference image

(maintaining the same aspect ratio) to have approximately the same number of pixels as the total number of white pixels in the binary pieces image.

Step 4:
We perform region labeling on the binary pieces image.



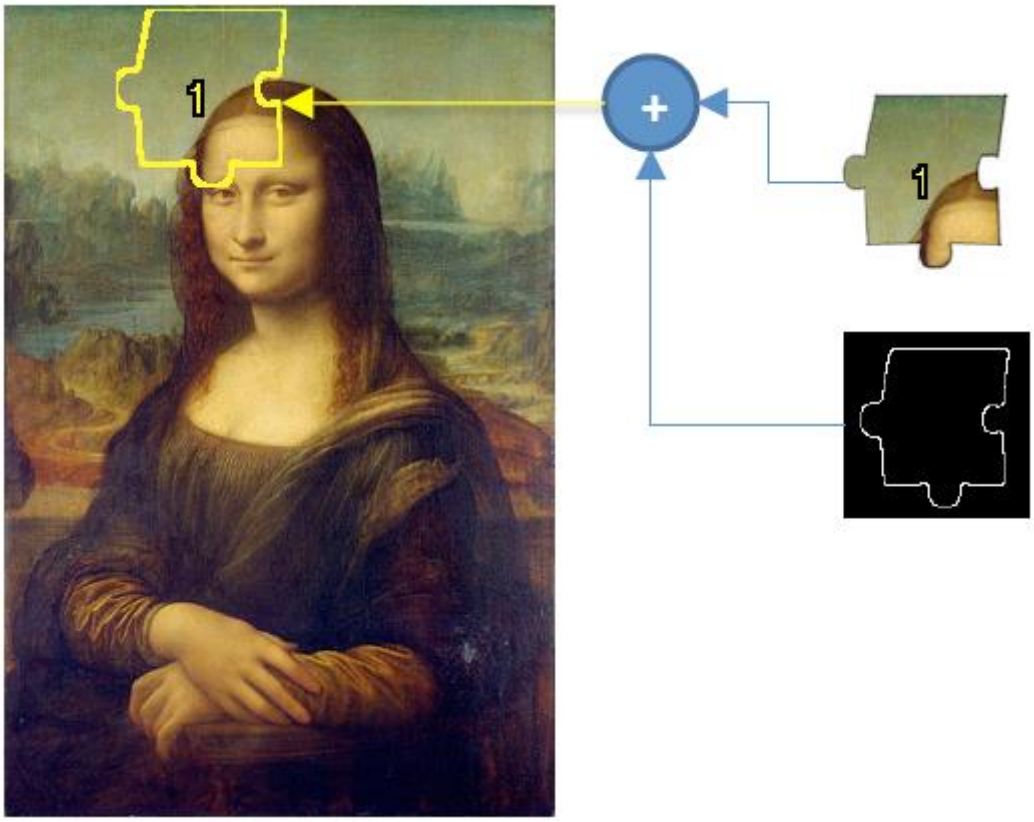
Step 5:
For each region in the binary pieces image, we compare the corresponding region in the original color pieces image to different regions in the reference image. Note that the regions being compared are non-rectangular in shape, because we utilize the regional shapes obtained from the segmentation in Step 1. We find the region in the reference image that best matches (has the smallest mean squared difference relative to) each piece region.



Step 6:
For the matching region in the reference image, we

(i) label that region with the same number that identifies the jigsaw piece

(ii) overlay the boundary of the jigsaw piece using the edge map from Step 2



Part 3: Motion Estimation and Motion Compensation [40pts]

Goal: Write a program to do motion estimation and motion compensation and test it with the two pairs of frames: fb_cur.bmp and fb_ref.bmp; mobile_cur.bmp and mobile_ref.bmp.

1. Perform *full search* motion estimation (ME) [1]with the given frame pairs. Take *_ref.bmp (.raw) as reference image and *_cur.bmp (.raw) as the image you are encoding now. The search range is +16 ~ -16 pixels. The block size is 16×16.
2. Perform motion compensation with the motion vector you get from step 1. Remember to compensate it directly.
3. Calculate the PSNR between the original image (*_cur.bmp (.raw)) and the compensated image.
4. Which PSNR is higher? Does it look better visually? How do you explain that result?

[1] Motion estimation full search code is available on the web if you do not want to write it. E.g., https://www.mathworks.com/matlabcentral/fileexchange/8761-block-matching-algorithms-for-motion-estimation

$$PSNR = 10\log_{10}\frac{255^2}{MSE}$$

$$MSE = \frac{\sum_{n=1}^{FrameSize}(I_n - P_n)^2}{FrameSize}$$

I_n is the n^{th} pixel in the original image, and P_n is the n^{th} pixel in the compensated image.

Requirements for upload:

1. Required files: Compress all the following data into one .zip file, named as "HW5_ASUID.zip".
   i. Readme file: Instructions on how to run your program, include non-image results (e.g. PSNR).
   ii. Source code
   iii. Output images for the 3 parts, plus screen shots if you think are needed. **There is NO NEED to include the input images**.