CSC420: Assignment 1

Xiangyu Kong, kongxi16

September 24, 2019

Problem 1

1. The computational cost for $h * I$ when h is not separable is $O(n^2 m^2)$.

   This because each pixel in I gets computed for m^2 time, and there are n^2 pixels in total.

2. The computational cost for $h * I$ when h is not separable is $O(m^2 2n)$.

Problem 2

Canny Edge Detection Steps:

1. Filter the image with derivative of Gaussian in both horizontal and vertical directions.

   - The purpose of this is to smooth the image and remove the noise.
   - To do this, we apply Gaussian filter to convolve with the image.

2. Find the magnitude and direction for the gradients

   - The purpose of this is to find the possible edges
   - To do this, we apply edge detection filters (for example, Sobel) with different directions and convolve with the image.

3. Non-maximum suppression

   - Get rid of the spurious response from edge detection produced by noise.
   - To do this, we only take local maximum or minimum of the edges.

4. Linking and Thresholding

   - The purpose of this is to connect the unlinked edges.
   - To do this, define 2 thresholds low and high. We use the high threshold's results to start the edge curves and use low threshold's results to connect the unlinked edges.

Problem 3

Laplacian of Gaussian filters are symmetrical. It has the shape of 1. We can see that there is a decrease and then an increase going towards the center. This activates the edges because edges are composed of sudden change of pixel intensities.
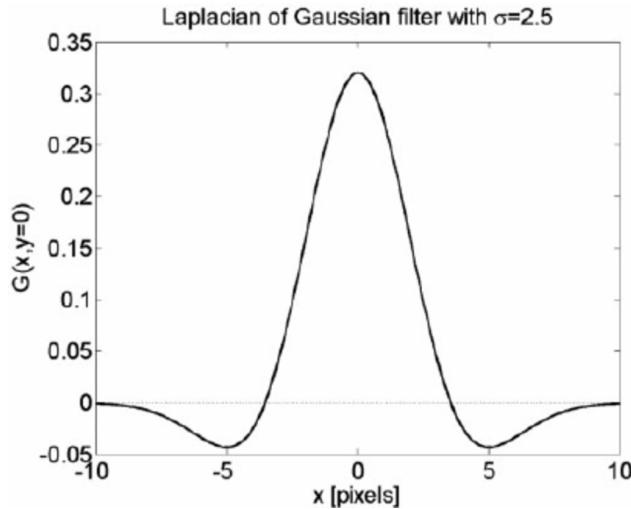

Figure 1: Laplacian of Gaussian with $\sigma = 2.5$

Problem 4

See code implementations in question_4.py.

1. See sample results in **??**. The results are the grayscale image correlation and convolution with the sharpening filter

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figure 2: Result of my_correlation

Figure 3: Original image

Figure 4: Result of my_convolution

2. To achieve portrait mode, we want to blur the background. To do this a blur filter is chosen.

The filter chosen was $\frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$.

Note that we are not using the normal 3×3 blur matrix. This is because the output for $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

is not obvious enough.

See portrait mode and original photo bellow. The face for the dog is the focus rectangle. Notice that the hand and the ears are fuzzed out.
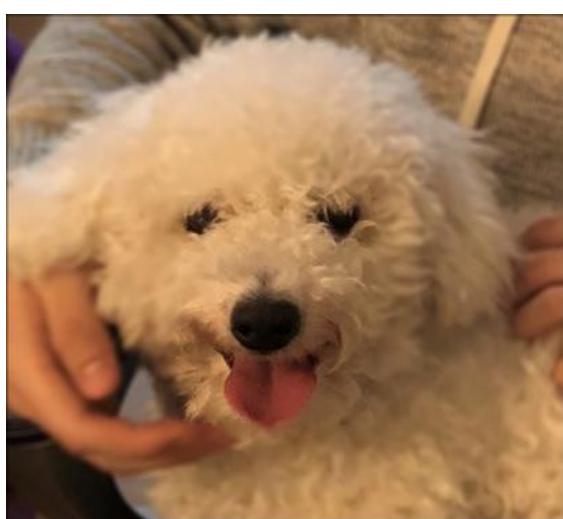
Figure 5: Portrait mode

Figure 6: Original image

Problem 5

See code implementations in question_5.py.

1. Separable Filters allow correlation and convolution operations to be performed at $2K$ operation cost instead of K^2. This is because separable filters are able to be split into $1D$ horizontal and $1D$ vertical filters. The result of convolving the image with the 2 $1D$ filters is the same as convolving the image with the original separable filter.

   This is achieved when when the Singular Value Decomposition has only one non-zero single value.

   $$F = U\Sigma V^T = \Sigma_{i=0}^k \sigma_i u_i v_i^T$$

   where $\sqrt{\sigma_1}\boldsymbol{u}_1$ and $\sqrt{\sigma_1}\boldsymbol{v}_1$ are the vertical and horizontal filters.

2. The Separable filter is $\dfrac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$.

   The output for the separable filter is

   $$[[-0.25]$$
   $$[-0.5\ \ ]$$
   $$[-0.25]]$$

   $$[[-0.25\ \ -0.5\ \ \ -0.25]]$$
   True

   The Inseparable filter is $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$.

   The output for the separable filter is

   $$False$$

Problem 6

See code implementations in question_6.py.

1. The image with noise is as follows

Figure 7: Result of add_rand_correlation

Figure 8: Original image

2. The recovered image is as follows. The chosen filter was the mean filter $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$. This was chosen because averaging the pixels will reduce the effect of the noise and make the noisy pixel resemble the average of its neighbors.

Figure 9: add_rand_correlation

Figure 10: The recovered image

Figure 11: The original image

3. The image with salt and pepper noise is as bellow:

Figure 12: add_salt_and_pepper

Figure 13: The original image

4. Trying to recover salt and pepper noisy image does not yield a very good result: the output image is still noisy. The chosen filter is the median filter. This is achieved with OpenCV's *medianBlur* method. This works because salt and pepper noise is very extreme. The pixel has signal of either 1 or 0 (white or black). This affects the mean vastly, where as it does not really affect the median of the nearby pixels.

Figure 14: mean filter

Figure 15: median filter

Figure 16: The original image

5.