Fan Vincent Ka Chun 1002563380

Question 1

(a) Since we are trying to apply a convolution operation on image I ($h * I$), and the filter h is not separable, then the computational cost per pixel would equal to m^2 which is the size of h. Therefore, for I the total cost would be $m^2 * n^2$.
(b) If h is separable, then we can perform a horizontal convolution followed by a vertical convolution. Thus, the computational cost per pixel would equal to $2m$. So, the total cost for image I would be $2m * n^2$.

Question 2

The first step of Canny edge detection is to apply a Gaussian filter to smooth the image and remove the noise by applying derivative of Gaussian in both horizontal and vertical directions.

Then, we can find the magnitude and orientation of the gradients. We can find the magnitude by taking the squared root of the sum of squared partial derivatives, which is as follow: $||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$. We can then calculate the gradient direction by taking the arctangent of the derivatives as follow: $\theta = \tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$.

Thirdly, we can perform non-maximum suppression to thin out the edges. This is done by checking if a pixel is local maximum along the positive and negative gradient direction. If the magnitude of the gradient of this pixel is the largest compared to the neighboring pixels, then it's magnitude value will be preserved. Otherwise, it will be suppressed, i.e. lowering to 0.

Lastly, we will need to further filter out edge pixels that has a relatively low gradient magnitude. We will need to pick a high and low threshold, then if the pixel's gradient magnitude is lower than the low threshold, it will be filtered out. If the pixel's gradient magnitude is in between the thresholds, it will be defined as weak edges. If the pixel's gradient magnitude is larger than the high threshold, then it will be defined as strong edges. Next, we would draw our final edge image by hysteresis thresholding. We will start our edge curves with the strong edges and continue with the weak edges that are connected to the strong edges. The remaining edges that are not connected to any strong edges are usually not extracted from true edges. Therefore, by suppressing these weak edges, we will achieve a more accurate result.

Question 3

Laplacian of Gaussian is a 2D measure of the second derivative of an image intensity. When we apply the Laplacian of Gaussian to an image, it will first smooth the image and then highlight places with high intensity change, then we can find the zero-crossings which are points of zero where there is a local max and min surrounding it. Since it highlights areas with high difference in intensity, this suggests that area to be edges. Therefore, we can use Laplacian of Gaussian to detect edges because it can identify areas of the image where it has rapid change in intensity.

Question 4

(a) Here are the results by using $MyCorrelation$ under valid mode with a sharpening filter:

$$h = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$


The original photo is on the left, while the processed photo is on the right

(b) Here are the results by using $MyConvolution$ under valid mode with the same sharpening filter as above. Note that the results are very similar since convolution is correlation with an h that is flipped horizontally and vertically.


The original photo is on the left, while the processed photo is on the right

(c) I choose a 5 by 5 Gaussian approximation filter with the following matrix:

$$\frac{1}{256}\begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix}$$

I choose to use this because Gaussian blur offers a more gradual smoothing compared to a 5 by 5 box filter since Gaussian takes the nearest neighbouring pixels to have higher influence. Thus, a more smoothing transition than other linear filters.

I first used Adobe Photoshop to segment out the person and background. Then, I apply the filter on the background and then stitched it back together as final output.

In my implementation, I added a depth input which is meant to reflect how much depth the user wants, i.e. the higher depth is the more blurred it is. This is because depth controls how many times the function will apply the filter onto the background. Note the right is processed image with depth=5, while the original is on the left in grayscale.


Question 5

(a) A separable filter is a 2D matrix filter that can be broken down into two 1D filter, one is a row matrix and the other is a column matrix. We can take the 1D row and column matrix and apply it horizontally and vertically onto the image matrix's row and column. If the results are the same as applying the original filter, then we can say this filter is separable.

(b) I picked the following matrices to demonstrate my $isSeparableFilter$:

$$h1 = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}, h2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Here are my results:

```
In [126]: runfile('C:/Users/vince/Documents/CSC420/A1/A1_Q5.py', wdir='C:/Users/vince/Documents/CSC420/A1')
horizontal 1D filter is: [ 1.31607401  0.         -1.31607401]
vertical 1D filter is: [0.75983569 1.51967137 0.75983569]
True
horizontal 1D filter is: [0.33333333 0.33333333 0.33333333]
vertical 1D filter is: [0.33333333 0.33333333 0.33333333]
True
```

The first 3 lines correspond to h1, and the next 3 lines correspond to h2.

Here are the approximated matrices:

$$h1_{vertical} = \begin{pmatrix} 0.76 \\ 1.52 \\ 0.76 \end{pmatrix}, \quad h1_{horizontal} = (1.32 \quad 0 \quad -1.32)$$

$$h2_{vertical} = \begin{pmatrix} 0.33 \\ 0.33 \\ 0.33 \end{pmatrix}, \quad h2_{horizontal} = (0.33 \quad 0.33 \quad 0.33)$$

Question 6

(a) Below is gray.jpg after processed by $AddRandNoise$ with a magnitude of 0.05

(b) I picked $GaussianBlur$ from cv2 module with 3 by 3 kernel size and $\sigma = 0$ to remove the noises. I chose a 3x3 and $\sigma = 0$ because I do not want to lose too much information since we only choose to affect 5% of the value. Therefore, a small filter will be enough.
Since low pass filters are effective for removing noise, and our noise was generated by a uniform distribution. Gaussian blur will be a better approximation than other linear filters because Gaussian distribution can give the neighboring values higher influence. While median blurring will take into all neighboring pixels, thus including all uniformly distributed noise. Therefore Gaussian, is the most effective linear filter in this case.

We can see that the sky is much less noisy than before and smoothed out, while most of the details, i.e. building structure and cars, are preserved and smoothed out.

(c) Below is gray.jpg after processed by $AddSaltAndPepperNoise$ with a density of 0.05.

(d) Upon trying the same filter from (b), I realized that it didn't work as expected in this case. So I picked median filtering to remove salt and pepper noise because by calculating the median of the pixels within the kernel size, it can help to reduce the extrema's influence (the salt and pepper noise) and replace the extrema's value with the median of its surrounding pixels. I choose a kernel size of 3 because I know that the density is relatively low, so a 3x3 has a low chance of including noises while preserving the most details.

(e) The cause of artifacts is due to salt-and-pepper pixel clusters when median filtering is performed, since there are 2 more channels, the density of the noise also higher since we are performing $AddSaltAndPepperNoise$ on each channel. Therefore, if there are lots of neighboring pixels that are noises. Then when computing the median, it would account for that and affect the surrounding area. Thus, creating the artifacts.

So, I propose that after applying the median filters, apply a layer of box filter in attempt to smooth out and blend in the artifacts. This works to help remove artifacts since we can take the average surrounding the artifacts and actual pixels, we can get a result with blended artifacts. Here are the results:

On the left, we have the color.jpg after adding some salt-and-pepper noise with 0.05 density. While on the right we have the image that's processed by median filter and then a layer of box filter. Note that since information was lost when salt-and-pepper noise was added to the image, therefore we cannot retrieve 100% of the original pixel value. We can see that there are still little artifacts left, which we may be able to remove if we apply more filters onto the image. However, the image would decrease drastically. So, I decided that the image after 2 filters looks the best.