Part I: Theory

Q1) LTI Systems and Convolution

First I will prove

$$x(n) = \sum_{k=0}^{+\infty} x(k)\delta(n-k)$$

We know

$$\begin{aligned} x(n) &= \sum_{k=0}^{n-1} 0x(k) + 1x(n) + \sum_{k=n+1}^{+\infty} 0x(k) \\ &= \sum_{k=0}^{n-1} 0x(k) + \delta(0)x(n) + \sum_{k=n+1}^{+\infty} 0x(k) \\ &= \sum_{k=0}^{n-1} \delta(n-k)x(k) + \delta(n-n)x(n) + \sum_{k=n+1}^{+\infty} \delta(n-k)x(k) \qquad \text{property of } \delta \\ &= \sum_{k=0}^{+\infty} x(k)\delta(n-k) \end{aligned}$$

Therefore,

$$\begin{aligned} T(x(n)) &= T(\sum_{k=0}^{+\infty} x(k)\delta(n-k)) \\ &= \sum_{k=0}^{+\infty} x(k)T(\delta(n-k)) \qquad \text{by linearity} \\ &= \sum_{k=0}^{+\infty} x(k)h(n-k) \\ &= x(n) * h(n) \end{aligned}$$

Q2) Polynomial Multiplication and Convolution

Let $\mathbf{u}$ be a vector with $n+1$ dimension and $\mathbf{v}$ be a vector with $m+1$ dimension, where $m, n \in \mathbb{N}^+$

$\mathbf{Note}$: index starts from 0.

By definition of convolution, we can know

$$(u * v)(k) = \sum_{i+j=k \wedge i \le n \wedge j \le m} u_i v_j$$

Consider the coefficient of x^k in the polynomial multiplication, where $k \in \mathbb{N}^+ \wedge k \le m+n$

By the property of polynomial multiplication,

$$c(k) = \sum_{i=0}^{min\{k,n\}} u_i v_{k-i} = \sum_{i+j=k \wedge i \le n \wedge j \le m} u_i v_j = (u * v)(k)$$

Therefore, they are equivalent.

Q3) Laplacian Operator

As we know, the rotation matrix is

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

Let

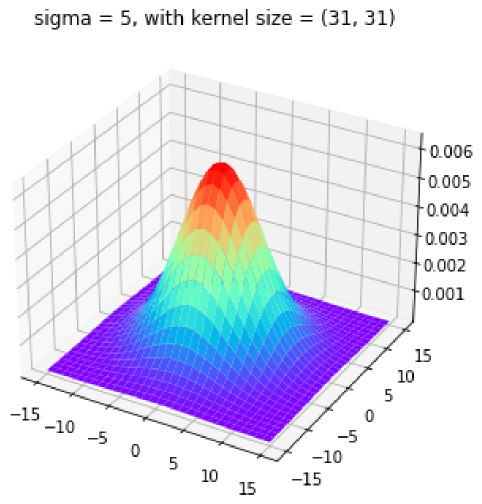
$$x = x'cos\theta - y'sin\theta$$
$$y = x'sin\theta + y'cos\theta$$

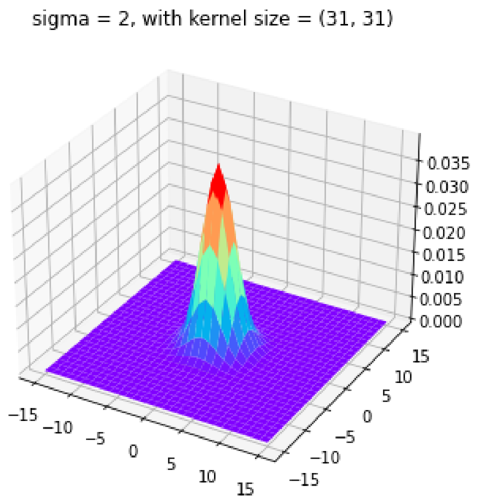
Then

$$\begin{aligned}
\frac{\partial f}{\partial^2 x'} + \frac{\partial f}{\partial^2 y'} &= \frac{\partial}{\partial x'}(\frac{\partial f}{\partial x'}) + \frac{\partial}{\partial y'}(\frac{\partial f}{\partial y'}) \\
&= \frac{\partial}{\partial x'}(\frac{\partial f}{\partial x}\cdot\frac{\partial x}{\partial x'} + \frac{\partial f}{\partial y}\cdot\frac{\partial y}{\partial x'}) + \frac{\partial}{\partial y'}(\frac{\partial f}{\partial x}\cdot\frac{\partial x}{\partial y'} + \frac{\partial f}{\partial y}\cdot\frac{\partial y}{\partial y'}) \\
&= \frac{\partial}{\partial x'}(cos\theta\frac{\partial f}{\partial x} + sin\theta\frac{\partial f}{\partial y}) + \frac{\partial}{\partial y'}(-sin\theta\frac{\partial f}{\partial x} + cos\theta\frac{\partial f}{\partial y}) \\
&= \frac{\partial}{\partial x}(cos\theta\frac{\partial f}{\partial x} + sin\theta\frac{\partial f}{\partial y})\frac{\partial x}{\partial x'} + \frac{\partial}{\partial y}(cos\theta\frac{\partial f}{\partial x} + sin\theta\frac{\partial f}{\partial y})\frac{\partial y}{\partial x'} \\
&+ \frac{\partial}{\partial x}(-sin\theta\frac{\partial f}{\partial x} + cos\theta\frac{\partial f}{\partial y})\frac{\partial x}{\partial y'} + \frac{\partial}{\partial y}(-sin\theta\frac{\partial f}{\partial x} + cos\theta\frac{\partial f}{\partial y})\frac{\partial y}{\partial y'} \\
&= \frac{\partial}{\partial x}(cos\theta\frac{\partial f}{\partial x} + sin\theta\frac{\partial f}{\partial y})cos\theta + \frac{\partial}{\partial y}(cos\theta\frac{\partial f}{\partial x} + sin\theta\frac{\partial f}{\partial y})sin\theta \\
&+ \frac{\partial}{\partial x}(-sin\theta\frac{\partial f}{\partial x} + cos\theta\frac{\partial f}{\partial y})(-sin\theta) + \frac{\partial}{\partial y}(-sin\theta\frac{\partial f}{\partial x} + cos\theta\frac{\partial f}{\partial y})cos\theta \\
&= \frac{\partial f}{\partial^2 x}(cos^2\theta + sin^2\theta) + \frac{\partial f}{\partial^2 y}(cos^2\theta + sin^2\theta) \\
&= \frac{\partial f}{\partial^2 x} + \frac{\partial f}{\partial^2 y}
\end{aligned}$$

Part II: Application

Q4) Edge Detection

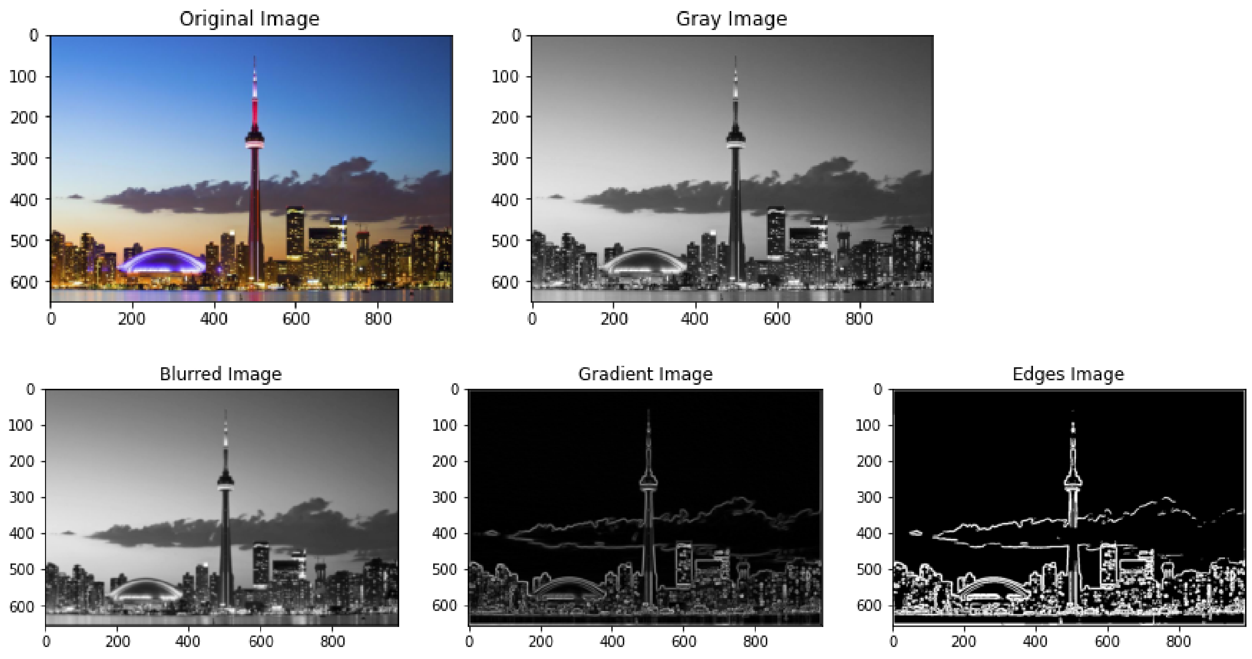




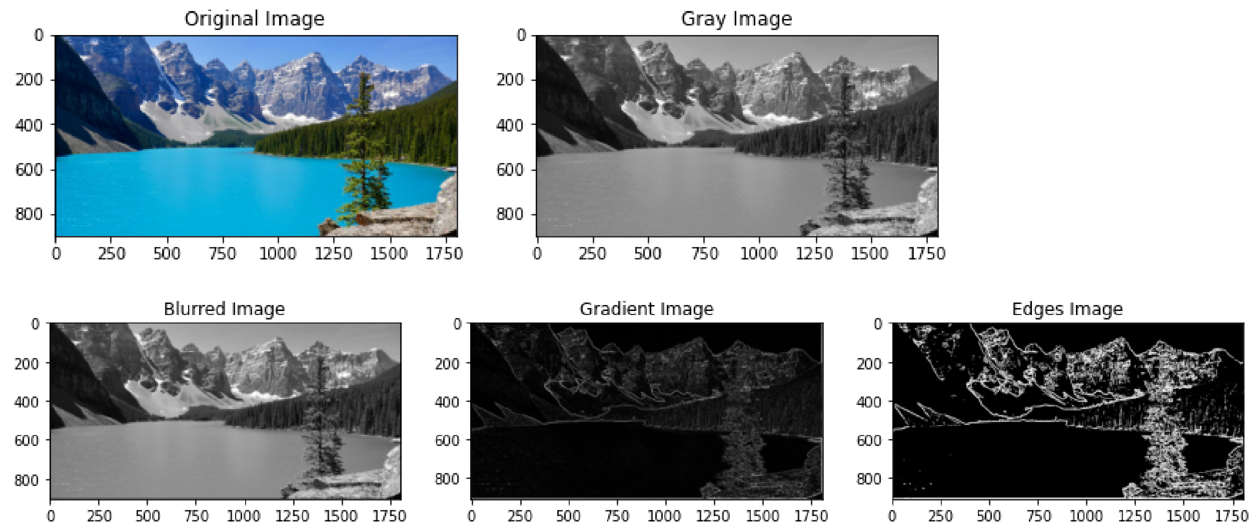
Two plots above are the Gaussian kernel visualization. We can see the kernel size is 31×31 with $\sigma = 5$ and $\sigma = 2$ respectively. We can clearly observe the feature that the kernel with larger standard deviation is much wider than the other.

As for my edge detection part, my results for each picture include:

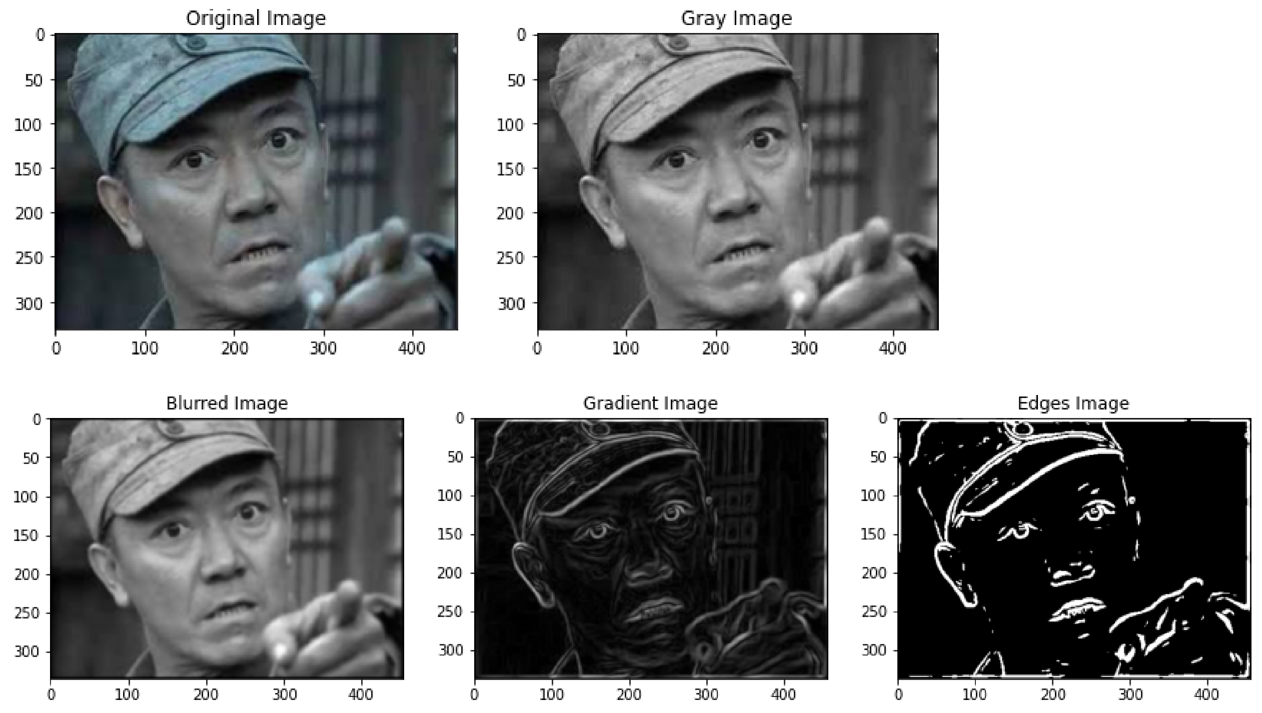
- Original Image
- Gray Image (From RGB Image to Gray Image)
- Blurred Image (with Gaussian kernel convolution)
- Gradient Image (Image get after gradient calculation)
- Edges Image (The final result image with edge detection)



For this image, we can observe that the overall quality of this algorithm is great, we can get the outline of buildings, edges of clouds, even if the small cloud on the left part of the image. However, there still are some flaws exist. In Edges Image, the edge of cloud is not continuous, especially on the right part of the image, and because of cloud, the CN Tower lose its middle part.

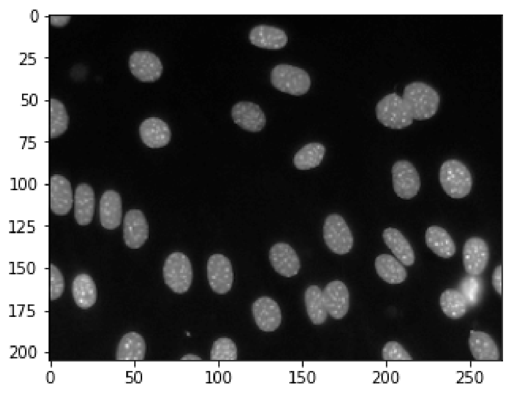


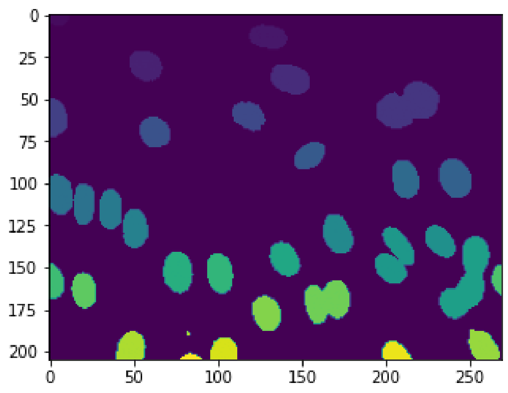
In this image, the algorithm extract the edges for the beautiful lake and the mountain. But this algorithm focus on details too much. There are lots of edges on the mountain, which make the mountain white in the final edges which are we don't want to get. Mountains are uneven, but we don't want to get those too detailed things on the image. In addition, many pines on the mountain, if we look closely, we can observe the vertical edges on the mountain.



This is Yunlong Li, a character from a famous Chinese TV series called "Drawing Sword". This algorithm successfully discard the edges in the background of the image and extract most of edges on this character. For example, it detect his hat, eyes, nose, ears, mouth and hand. However, the outline edge is not that continuous. The outline of his neck (on the left part of the image) and his cheek and chin are discontinuous. This might be caused by the occlusion of his hand or the color similarity between his skin and the background.

Q5) & Q6)





After apply the algorithm to this image, I found there are 32 cells in this image while there should be 37 cells in this image counted by myself. In the image above, I use different color to indicate different cell. 32 is near to 37 and this algorithm behaves well in counting cells. Nevertheless, there are some points can be improved. Compared to the original image, the cell at the top left did not be counted and there is a tiny point is counted as a cell at coordinate around (190, 80). In addition, if there are two cells are in the stage of cell division or they have overlapping, they will be counted as one cell.

There are two main points to improve the algorithm. The first point is improve the threshold algorithm. The values in this image is not "clean". Before I use Connected-component labeling algorithm, I need to make sure the value of each pixel has the binary value, that is either 0 for background or 255 for foreground. In order to do so, I use the auto-threshold algorithm in question 4. It behaves well but not enough. There might be some better algorithm to binary the image and discard the things we don't want e.g. at position (190, 80).

The second point is that we need an algorithm to detect whether a connected area is one cell or several cells. There are three times counting two cells as one cell and once counting three connected cells as a cell. This might cause the detected number of cells is less than the real number of cells. If there are lots of cells overlapping or in the stage of cell division, this might have a larger bias.