CSC420 Assignment 1 Report

🌟 Owner's Information

First Name: Kwan Kiu
Last Name: Choy
Student No: 1005005879
UtorId: choykwan

Theoretical Part

Question 1

1) Prove that $T[x(n)] = h(n) * x(n)$

Step 1: Express $x(n)$ as something made from $\delta(n)$

1) We know that $\delta(x) = \begin{cases} 1 & \text{if } x == 0 \\ 0 & \text{if else} \end{cases}$

$$x(n) = 0 + \ldots + 0 \cdot x(2) + \ldots 1 \cdot x(n) + 0 \cdot x(n+1) + \cdots$$

$$x(n) = \sum_{m=-\infty}^{\infty} x(m)\,\delta(n-m) \quad \text{—— ①}$$

Step 2: Evaluate L.H.S [i.e: $T[x(n)]$]

We know that $x(n) = \sum_{m=-\infty}^{\infty} x(m)\,\delta(n-m)$

$$T[x(n)] = T[x(1)\cdot\delta(n-1) + \ldots x(n)\delta(n-n) + \ldots]$$

$$= x(1)\cdot T[\delta(n-1)] + \ldots x(n) T[\delta(n-n)] + \ldots$$ by linear property of T.

$$= \sum_{m=-\infty}^{\infty} x(m)\, T[\delta(n-m)]$$

$$= \sum_{m=-\infty}^{\infty} x(m)\, h(n-m)$$ by time invarience property

$$= x(n) * h(n)$$

$$= h(n) * x(n)$$ by commutativity

Question 2

2) We want to prove that the coefficients of polynomial multiplication will be equivalent as convolving 2 vectors that represent the polynomials.

Let

$$u(x) = \sum_{i=0}^{M} a_i x^i \quad \text{and} \quad v(x) = \sum_{j=0}^{N} b_j x^j \quad \text{and } M, N \in \mathbb{N} \wedge M \leq N$$

$$u(x) \cdot v(x) = \sum_{i=0}^{M} a_i x^i \sum_{j=0}^{N} b_j x^j$$

$$= \sum_{j=0}^{N} \sum_{i=0}^{M} a_i x^i b_j x^j$$

$$= \sum_{i+j=0}^{M+N} \sum_{i=0} a_i x^i b_{i+j-i} x^{i+j-i}$$

Let $k = i+j$, so we can substitute k into the summation.

$$= \sum_{k=0}^{M+N} \sum_{i=0} a_i x^i b_{k-i} x^{k-i}$$

$$= \sum_{k=0}^{M+N} \sum_{i=0}^{\min(k,N)} a_i b_{k-i} x^k$$

Let $s \in \mathbb{N}$ and $0 \leq s \leq M+N$.

Therefore, the s^{th} term of the polynomial product would be $\sum_{i=0}^{\min(s,N)} a_i b_{s-i} x^s$, and the coefficient will be $\sum_{i=0}^{\min(s,N)} a_i b_{s-i}$.

Assume now u, v be the 2 vectors that represent $u(x)$ and $v(x)$ s.t $u = \begin{bmatrix} a_0 \\ \vdots \\ a_M \end{bmatrix}$ $v = \begin{bmatrix} b_0 \\ \vdots \\ b_N \end{bmatrix}$.

by definition of u and v

$$u * v(s) = \sum_{k=0}^{\min(s,N)} u(k) v(s-k) = \sum_{k=0}^{\min(s,N)} a_k b_{s-k}$$

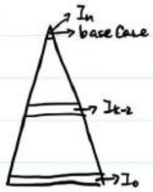
(*) The original formula of convolution according to lecture notes is $u * v(s) = \sum_{i=-\infty}^{\infty} u(i) v(s-i)$.
The reason why we can change the range of the summation from $-\infty$ to ∞ to from 0 to $\min(s,N)$ is because of the assumption of zero padding. Any product $u(i)v(s-i)$ with i outside the 0 to $\min(s,N)$ range will give us a 0

Therefore, we have shown that $\forall s \in \mathbb{N}$, $0 \leq s \leq M+N$, the coefficient of the s^{th} term of the polynomial product is $\sum_{i=0}^{\min(s,N)} a_i b_{s-i}$ which is equivalent to the s^{th} term of the convolution returned vector $\sum_{k=0}^{\min(s,N)} a_k b_{s-k}$.

Hence, we have shown that the coefficients of polynomial multiplication will be equivalent as convolving 2 vectors that represent the polynomials.

Question 3

3)



Gaussian pyramid

Given a laplacian pyramid, to reconstruct the original image I_0, we would need the top level of the gaussian pyramid, which is a 1×1 pixel I_n. Let EXPAND() be an arbitrary upsampling function s.t it increase the size of the image by 1 layer of the gaussian pyramid.

We know that the laplacian $L_i = I_i - EXPAND(I_{i+1})$. Therefore, by rearranging the terms, we get:

To reconstruct the image at I_{n-1} level,

$$I_{n-1} = L_{n-1} + EXPAND(I_n)$$

Therefore, to reconstruct the image at k^{th} level,

$$I_k = L_k + EXPAND(I_{k+1})$$

And to reconstruct the original image I_0,

$$I_0 = L_0 + EXPAND(I_1)$$

To make it clear, the minimum information that we need from the gaussian pyramid is I_n which is the image at the top layer of the gaussian pyramid with a 1×1 size. Then, to reconstruct the image at the k^{th} level, all we need to do is to use this closed form expression: $I_k = L_k + EXPAND(I_{k+1})$

Question 4

4) Let (x,y) be a point in the xy plane. Let u, v be the rotated version of (x,y) after rotating the xy plane counterclockwise with an angle θ.

$$\begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

What we want to show $\Delta I = I_{xx} + I_{yy} = I_{ss} + I_{tt}$

$$I_{xx} = \frac{\partial^2 I}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial x}\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial I}{\partial t}\frac{\partial t}{\partial x}\right)$$

$$= \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial s}\cos\theta + \frac{\partial I}{\partial t}\sin\theta\right)$$

$$= \frac{\partial I}{\partial s\partial x}\cos\theta + \frac{\partial I}{\partial x\partial t}\sin\theta$$

$$= \frac{\partial}{\partial s}\frac{\partial I}{\partial x}\cos\theta + \frac{\partial}{\partial t}\frac{\partial I}{\partial x}\sin\theta$$

$$= \frac{\partial}{\partial s}\left(\frac{\partial I}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial I}{\partial t}\frac{\partial t}{\partial x}\right)\cos\theta + \frac{\partial}{\partial t}\left(\frac{\partial I}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial I}{\partial t}\frac{\partial t}{\partial x}\right)\sin\theta$$

$$= \frac{\partial}{\partial s}\left(\frac{\partial I}{\partial s}\cos\theta + \frac{\partial I}{\partial t}\sin\theta\right)\cos\theta + \frac{\partial}{\partial t}\left(\frac{\partial I}{\partial s}\cos\theta + \frac{\partial I}{\partial t}\sin\theta\right)\sin\theta$$

$$= \frac{\partial^2 I}{\partial s^2}\cos^2\theta + \frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta + \frac{\partial^2 I}{\partial s\partial t}\cos\theta\sin\theta + \frac{\partial^2 I}{\partial t^2}\sin^2\theta$$

$$= \frac{\partial^2 I}{\partial s^2}\cos^2\theta + 2\frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta + \frac{\partial^2 I}{\partial t^2}\sin^2\theta$$

$$I_{yy} = \frac{\partial^2 I}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial I}{\partial y}\right)$$

$$= \frac{\partial}{\partial y}\left(\frac{\partial I}{\partial s}\frac{\partial s}{\partial y} + \frac{\partial I}{\partial t}\frac{\partial t}{\partial y}\right)$$

$$= \frac{\partial}{\partial y}\left(\frac{\partial I}{\partial s} -\sin\theta + \frac{\partial I}{\partial t}\cos\theta\right)$$

$$= \frac{\partial^2 I}{\partial y\partial s} -\sin\theta + \frac{\partial^2 I}{\partial y\partial t}\cos\theta$$

$$= \frac{\partial}{\partial s}\frac{\partial I}{\partial y} -\sin\theta + \frac{\partial}{\partial t}\frac{\partial I}{\partial y}\cos\theta$$

$$= \frac{\partial}{\partial s}\left(\frac{\partial I}{\partial s}\frac{\partial s}{\partial y} + \frac{\partial I}{\partial t}\frac{\partial t}{\partial y}\right) -\sin\theta + \frac{\partial}{\partial t}\left(\frac{\partial I}{\partial s}\frac{\partial s}{\partial y} + \frac{\partial I}{\partial t}\frac{\partial t}{\partial y}\right)\cos\theta$$

$$= \frac{\partial}{\partial s}\left(\frac{\partial I}{\partial s} -\sin\theta + \frac{\partial I}{\partial t}\cos\theta\right) -\sin\theta + \frac{\partial}{\partial t}\left(\frac{\partial I}{\partial s} -\sin\theta + \frac{\partial I}{\partial t}\cos\theta\right)\cos\theta$$

$$= \frac{\partial^2 I}{\partial s^2}\sin^2\theta - \frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta - \frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta + \frac{\partial^2 I}{\partial t^2}\cos^2\theta$$

$$= \frac{\partial^2 I}{\partial s^2}\sin^2\theta - 2\frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta + \frac{\partial^2 I}{\partial t^2}\cos^2\theta$$

$$I_{xx} + I_{yy} = \frac{\partial^2 I}{\partial s^2}\cos^2\theta + \cancel{2\frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta} + \frac{\partial^2 I}{\partial t^2}\sin^2\theta + \frac{\partial^2 I}{\partial s^2}\sin^2\theta \cancel{- 2\frac{\partial I}{\partial s\partial t}\sin\theta\cos\theta} + \frac{\partial^2 I}{\partial t^2}\cos^2\theta$$

$$= \frac{\partial^2 I}{\partial s^2}(\cos^2\theta + \sin^2\theta) + \frac{\partial^2 I}{\partial t^2}(\sin^2\theta + \cos^2\theta)$$

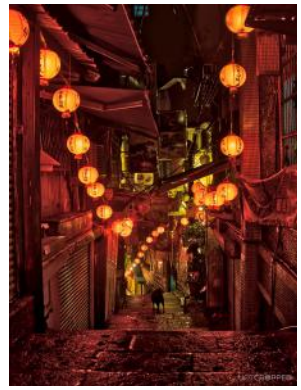
$$= \frac{\partial^2 I}{\partial s^2} + \frac{\partial^2 I}{\partial t^2}$$

$$= I_{ss} + I_{tt}$$

Therefore, we've shown that $\Delta I = I_{xx} + I_{yy} = I_{ss} + I_{tt}$.

Implementation Part

Chosen Images

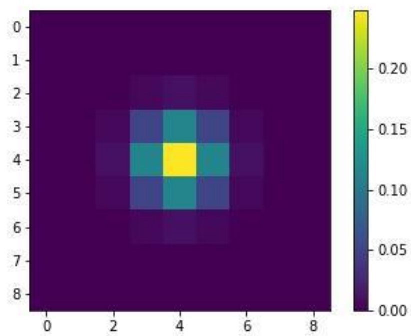
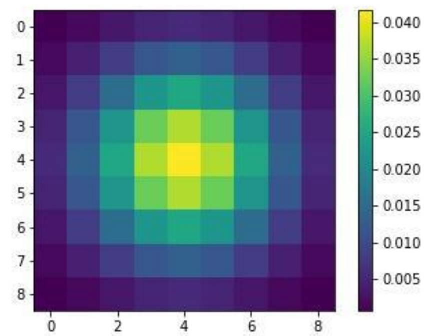
Image 1	**Image 2**	**Image 3**
		

Step 1: Get Gaussian Matrix

Code

```
def step_1_get_gaussian_matrix(ksize: int, scale:float):
    """
    Returns a 2D GaussianMatrix for input ksize and scale.
    :param ksize: The kernel size
    :param scale: sigma in the gaussian distribution
    :return: a 2D gaussian matrix for input ksize and scale.
    """
    # create array with [- (ksize-1)/2, ..., 0,... (ksize-1/2)]
    start, end = -(ksize-1)/ 2, (ksize-1)/2
    row_n_col = np.linspace(start, end, ksize)
    if ksize % 2 == 0:
        row_n_col[int(end): int(end)+2] = [0, 0]
    # compute with formula 1/(sqrt(2*pi)*sigma) * e^(-(x-mu)^2/2sigma^2)
    gaussian_exp = np.exp(-np.square(row_n_col)/(2*np.square(scale)))
    gaussian_mtx = (1/(np.sqrt(2*np.pi) *scale) * gaussian_exp)
    # divide the whole gaussian by np.sum such that it is normalized
    gaussian_mtx /= np.sum(gaussian_mtx)
    # Create the 2D matrix
    gaussian = np.outer(gaussian_mtx, gaussian_mtx)
    return gaussian
```

Visualization Results

Kernel Size = 9, Scale = 0.8	Kernel Size = 9, Scale = 2
	

Step 2 Getting Gradient Magnitude

Code

```
def pad_image(image, pad_widths):
    """
    Pad the image with 0s according to dimensions in pad_widths
    :param image: 2D matrix representation of image to be padded
    :param pad_widths: tuple wiht r_pad and c_pad which represents the
number
    of rows to be added at the top and the bottom and the number of columns
to
    be added at both sides respectively.
    :return: padded image 2D matrix representation
    """
    row_image, col_image = image.shape
    r_pad, c_pad = pad_widths
    result = np.zeros((row_image + 2* r_pad, col_image + 2* c_pad))
    for i in range(r_pad, result.shape[0]-r_pad):
        for j in range(c_pad, result.shape[1]-c_pad):
            result[i][j] = image[i-r_pad][j-c_pad]
    return result
```

```python
def convolve(kernel, image):
    """
    Performs 2D convolution between kernel and image.
    :param kernel: a kernel matrix
    :param image: an image np array
    :return: the convolution result 2D matrix
    """
    # h_flip and v_flip the kernel
    kernel_flipflatten = np.flip(np.flip(kernel, 0), 1).flatten()
    # compute sizes of image matrices and kernel matrices, and value of k
    i_row, i_column = image.shape
    k_row, k_col = kernel.shape
    # initiate return result matrix of image dimensions
    result = np.zeros((i_row, i_column))
    # k_r & k_c should be equal since the kernel should be a square matrix
    k_r, k_c = (k_row-1)//2, (k_col-1)//2
    # pad the image with 0s, in order to allow convolution for corner cells
    image_padded = pad_image(image, (k_r,k_c))
    # compute cross correlation between flipped kernel and image
    for i in range(i_row):
        for j in range(i_column):
            # refer to formula in notes: G(i, j) = np.dot(f, t_ij)
            i_neighbouhood = image_padded[i:i+k_row, j:j+k_col].flatten()
            result[i][j] = np.dot(kernel_flipflatten, i_neighbouhood.T)
    return result


def step_2_compute_gradient_magnitude(gray_image, ksize=3, scale=0.5):
    """
    Computes the gradient magnitude given a grayscale image
    :param gray_image: a gray scale image represented by a numpy array
    :param ksize: the kernel size of the gaussian filter. Note that ksize
    should be odd number
    :param scale: the sigma of the gaussian filter
    :return: a gradient magnitude matrix of gray_image
    """
    # getting gaussian filter
    gauss = step_1_get_gaussian_matrix(ksize, scale)
    # getting sobel operators
    sobel_x = np.matrix([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = sobel_x.T
    # convolving gaussian and sobel operators first for easier computation
    gauss_cv_sobel_x = convolve(sobel_x, gauss)
```

```
    gauss_cv_sobel_y = convolve(sobel_y, gauss)

    # convolving with the image to compute the derivatives along x & y
direction
    g_x = convolve(gauss_cv_sobel_x, gray_image)
    g_y = convolve(gauss_cv_sobel_y, gray_image)

    # compute the gradient
    gradient = np.sqrt(np.square(g_x) + np.square(g_y))
    return gradient
```

Visualization Results

Image 1

Image 2

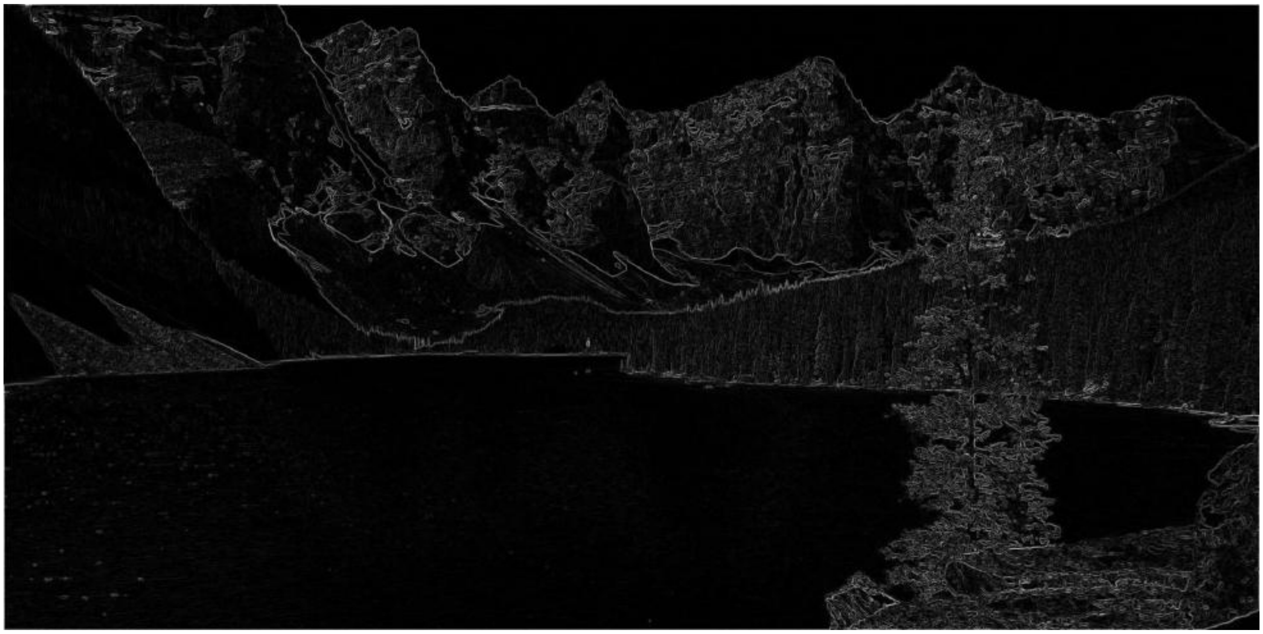


Image 3

Step 3 Applying Threshold Algorithm

Code

```
def step_3_threshold(gradient, epsilon=0.000000001):
    """
    Returns the gradient magnitude matrix after applying threshold such
that
    values in the matrix becomes either 0 or 255.
    :param gradient: a gradient magnitude matrix of an image
    :param epsilon: a float that is very close to 0
    :return: a gradient magnitude matrix with values either equal to 0 or
255.
    """
    # step 1: compute tau_0
    grad_row, grad_col = gradient.shape
    tau_i = np.sum(gradient)/(grad_row * grad_col)
    loop = True
    i = 0
    while loop:
        # step 2: set i = 0, find cell value < or > than tau_0
        lower = gradient[gradient < tau_i]
        higher = gradient[gradient >= tau_i]
        # step 3: compute the mean of the lower and higher groups
        ml, mh = np.mean(lower), np.mean(higher)
        # record previous tau
        prev = tau_i
        # compute tau_i
        tau_i = (ml + mh) / 2
        # check whether we should continue looping
        loop = np.abs(tau_i - prev) > epsilon
        i += 1

    grad_copy = gradient.copy()
    grad_copy = (grad_copy >= tau_i).astype(int) * 255
    return grad_copy
```

Visualization Results

(Using a gaussian filter of kernel size = 3, and sigma = 0.5)

Image 1

Image 2

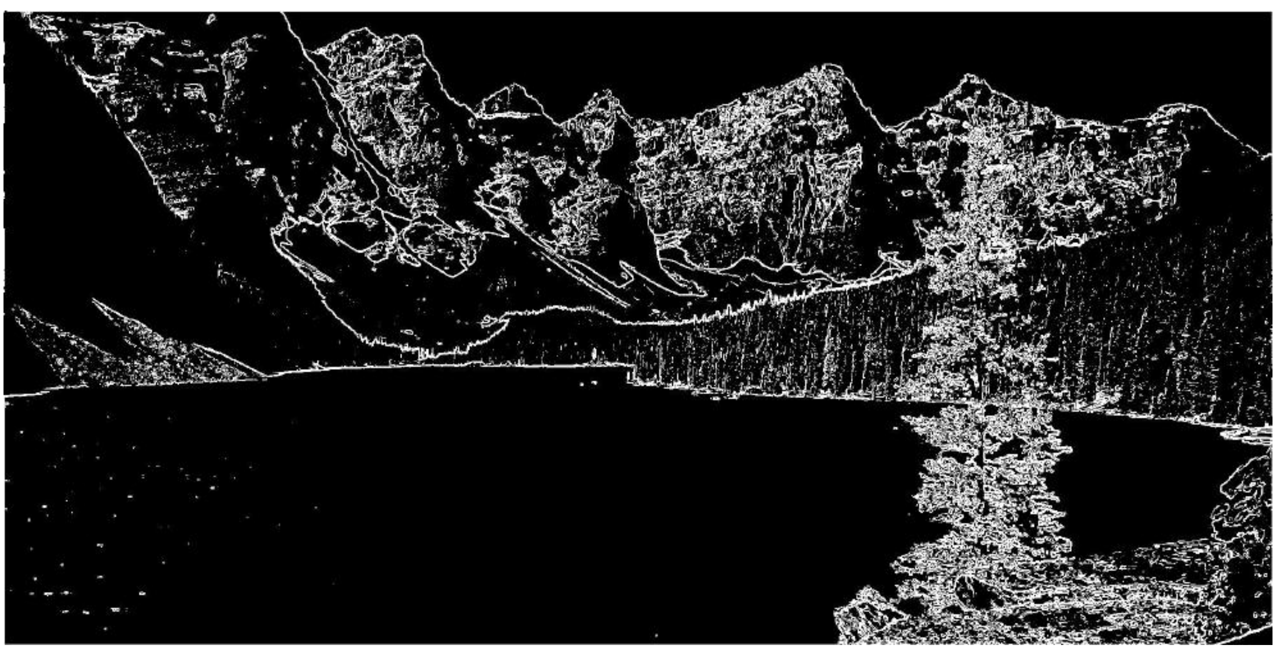
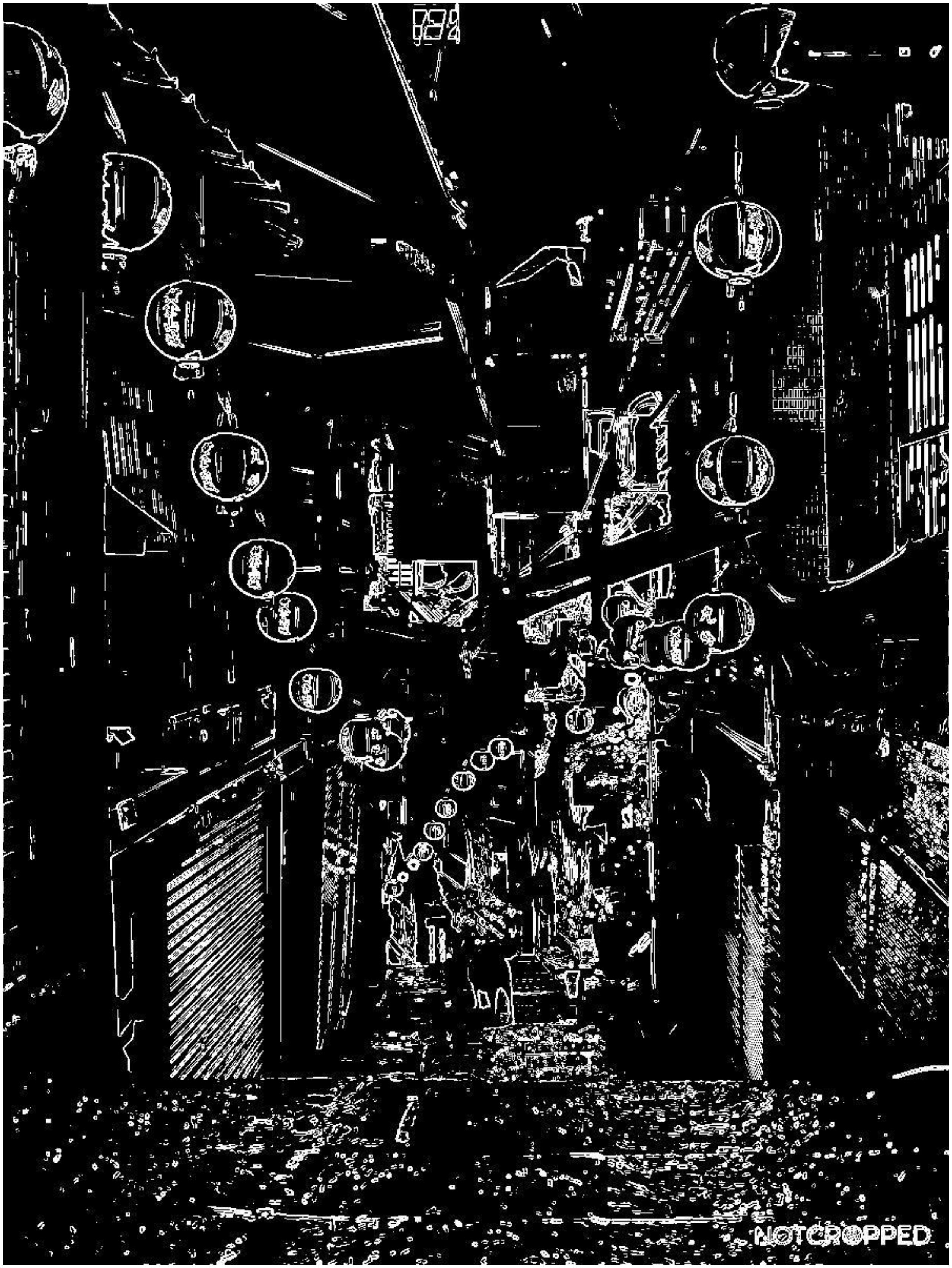


Image 3



Overview

Image 1 -- *Gaussian blur with size = 3, scale = 0.5*

Step 0: Grayscale Image

Step 1: Blurred Image

Step 2: Gradient Magnitude

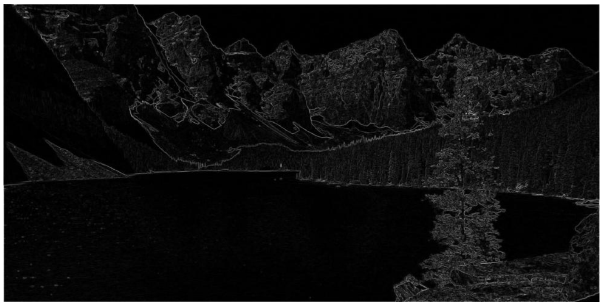
Step 3: Threshold

Image 2 -- *Gaussian blur with size = 3, scale = 0.5*

Step 0: Grayscale Image

Step 1: Blurred Image

Step 2: Gradient Magnitude



Step 3: Threshold

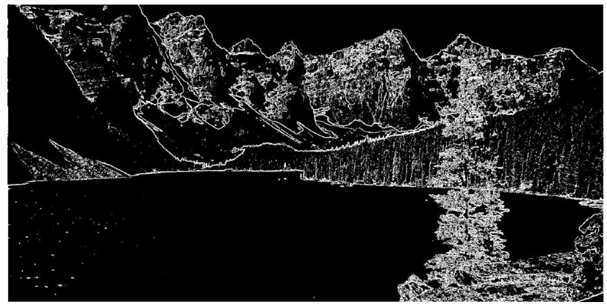


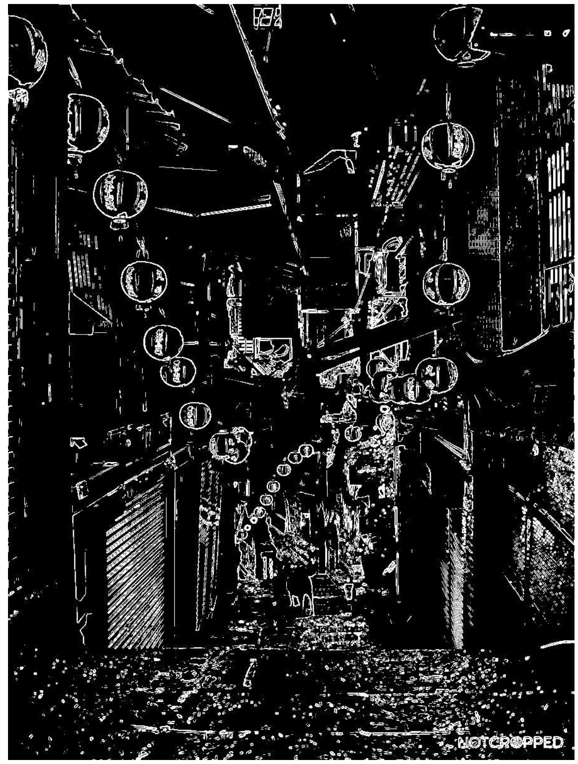
Image 3 -- *Gaussian blur with size = 3, scale = 0.5*

Step 0: Grayscale Image

Step 1: Blurred Image

Step 2: Gradient Magnitude

Step 3: Threshold



Brief Description of the algorithm

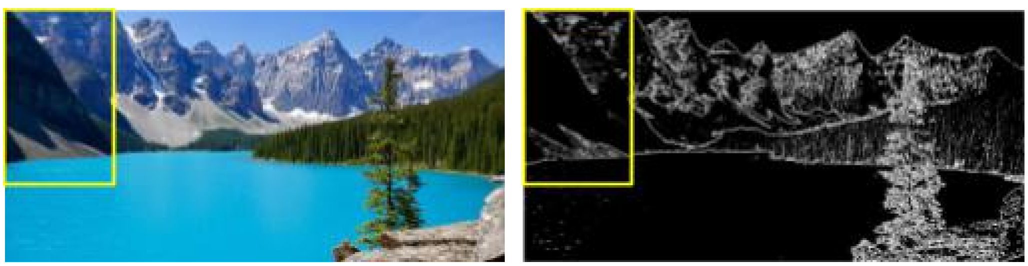
The algorithm first applies the Gaussian filter to remove the noise in the image. Then we applied the Sobel filter on the image to get the partial derivatives along with both the horizontal and vertical directions. After that, we compute the gradient magnitude matrix from the two partial derivatives to represent the change of intensity of that pixel. Finally, we apply a threshold algorithm to the gradient magnitude matrix to divide the cells in the matrix into two groups. One group with lower magnitudes, and another one with higher magnitude. Applied this threshold algorithm until the difference between the mean gradient magnitude across the two groups is equal to a small enough number (0.000000001). After that, turn all the higher magnitude pixels to 255 and lower magnitude pixels into 0 so that edges will appear as white in the final image.

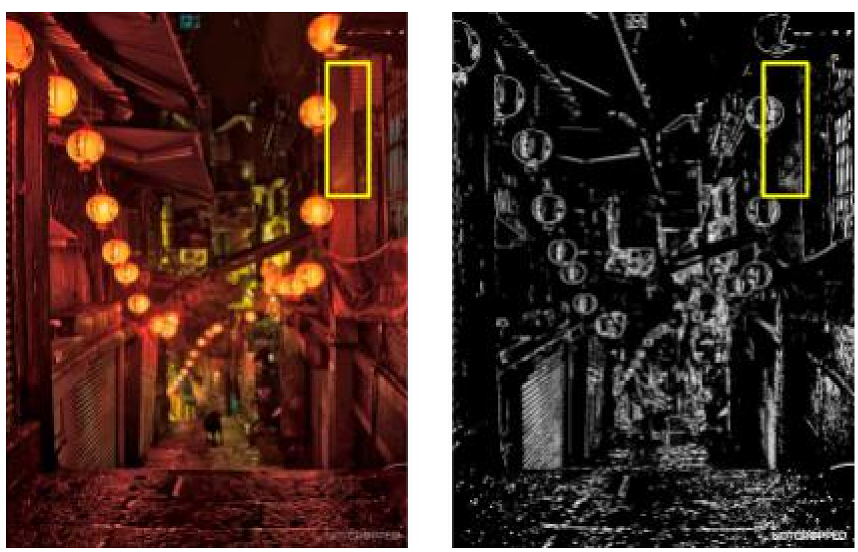
Strengths

- The algorithm is extremely good at detecting edges when the edge is formed by a high contrast color between the foreground and the background. This can be clearly illustrated by the clear edges detected around the lanterns in image 3, in which the lights (the warm colors in the lanterns) formed a high contrast with the dark background.
- The algorithm is not sensitive to gradual color change. This is something that we want. As shown in image 1, both the sky and the lake show a gradual change of colors in the original image. Our algorithm here does not add edges between each “color zones”, which is something that we want, as they are not “sky pieces” or “lake pieces”, they should be viewed as one giant area despite the color difference.

Limitations

- Our algorithm detect edges less well in darker areas as shown in image 2 and image 3. Take image 2 for example, even though there are patterns in the leftmost mountain, no edges are really detected in the final output image. Similarly, in image 3, the grid patterns on the buildings can only be seen in areas that are illuminated by the lantern light but not the darker areas.





- Our algorithm also performs not too well when foreground and background share similar colors. This can be illustrated in image 1, where some edges of the CN tower are not clearly shown as it has a similar color to the color of the cloud behind the tower.