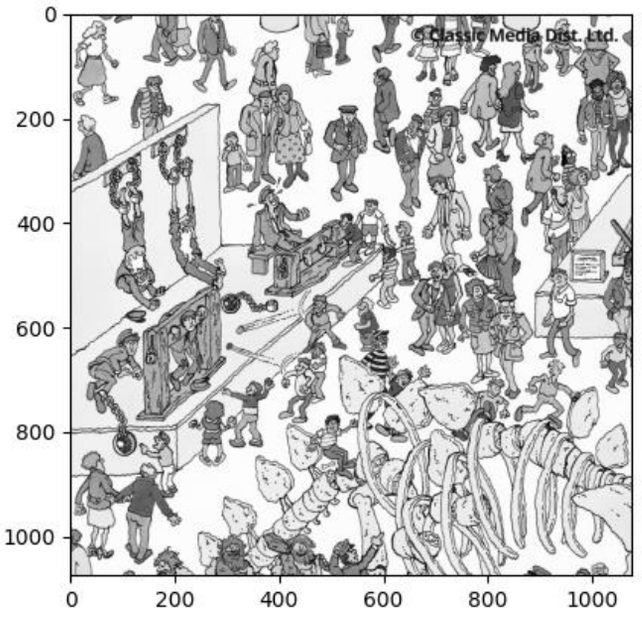
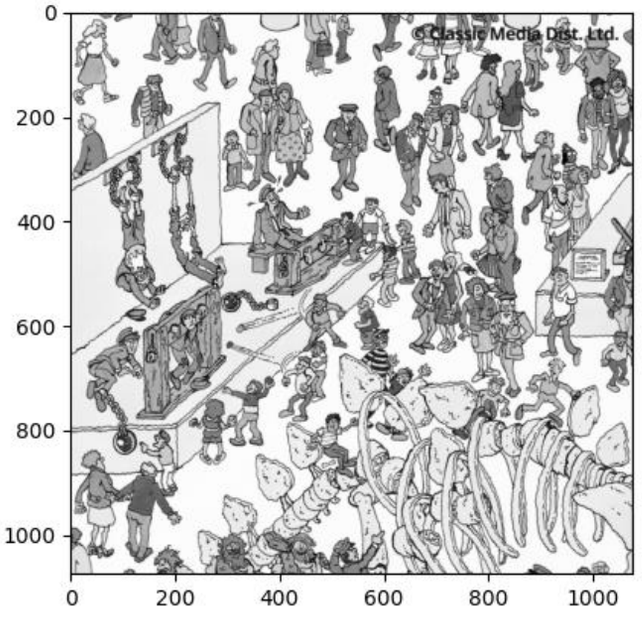
CSC420 Assignment 1 Coding Part

Q1 a)

Original img(gray scale):



After convolution:



Q1 a) Code

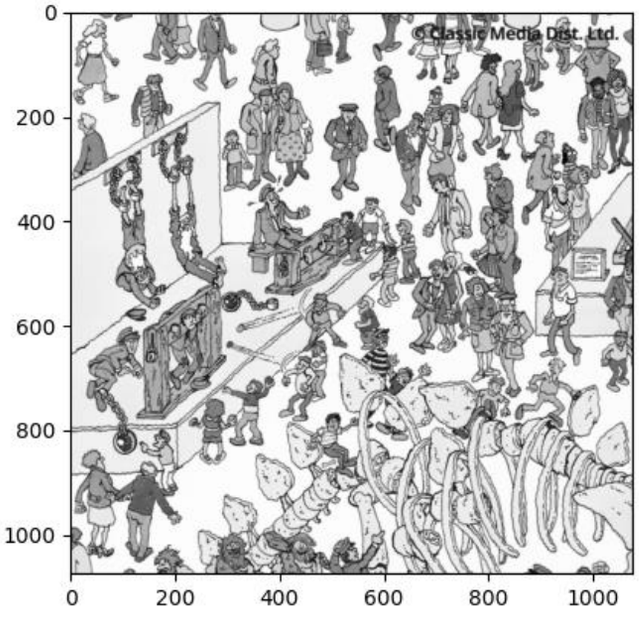
```python
def q1a_conv(img, kernel):
    # conv needs to flip the kernel both directions
    kernel = np.flipud(kernel)
    kernel = np.fliplr(kernel)
    k = int((kernel.shape[0] - 1) / 2)
    org_size = img.shape
    result = np.zeros((org_size[0], org_size[1]))

    # pad the img to make sure the output remains the same size as input
    img = np.pad(img, (k, k), 'constant', constant_values=(0, 0))
    for i in range(1, org_size[0]+1):
        for j in range(1, org_size[1]+1):
            result[i-1][j-1] = np.sum(img[i-k:i+k+1,j-k:j+k+1]*kernel)
    plt.imshow(result, cmap='gray')
    plt.gray()
    plt.show()
```

Q1 b) Code

```python
def q1b_is_separable(kernel):
    u, s, vh = np.linalg.svd(kernel)
    zero = np.zeros(s.shape[0]-1)
    # make sure only one sigular value is non-zero
    if s[0] != 0 and np.allclose(s[1:], zero):
        print("This filter is separable")
        return [True, u[:, 0], s, vh[0, :]]
    return [False]
```

Q1 c)

After convolution:



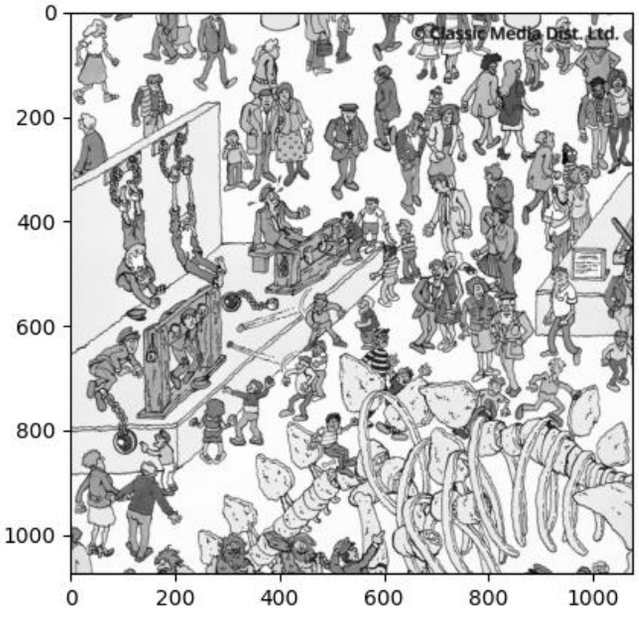
Q1 c) Code

```python
def q1c_conv_separable(img, kernel):
    # for this question I used kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])
    separable = q1b_is_separable(kernel)
    if not separable[0]:
        print(separable[0])
        return False
    u, s, vh = separable[1], separable[2], separable[3]
    u = math.sqrt(s[0])*u  # vertical filter
    vh = math.sqrt(s[0])*vh #horizontal filter
    # conv needs to flip the kernel both directions
    kernel = np.flipud(kernel)
    kernel = np.fliplr(kernel)
    k = int((kernel.shape[0] - 1) / 2)
    org_size = img.shape
    result = np.zeros((org_size[0], org_size[1]))
    img = np.pad(img, (k, k), 'constant', constant_values=(0, 0))
    # apply horizontal filter
    for i in range(1, org_size[0]+1):
        for j in range(1, org_size[1]+1):
            result[i-1][j-1] = np.sum(img[i,j-k:j+k+1]*vh)
    # apply vertical filter
    img = np.pad(result, (k, k), 'constant', constant_values=(0, 0))
    result = np.zeros((org_size[0], org_size[1]))
    for i in range(1, org_size[0]+1):
        for j in range(1, org_size[1]+1):
            result[i-1][j-1] = np.sum(img[i-k:i+k+1,j]*u)
    plt.imshow(result, cmap='gray')
    plt.gray()
    plt.show()
```

not separable filter takes: 7.35 secs | separable filter takes: 11.65 secs

Q1 d) Yes, we can still take advantage of separability

After cross correlation:



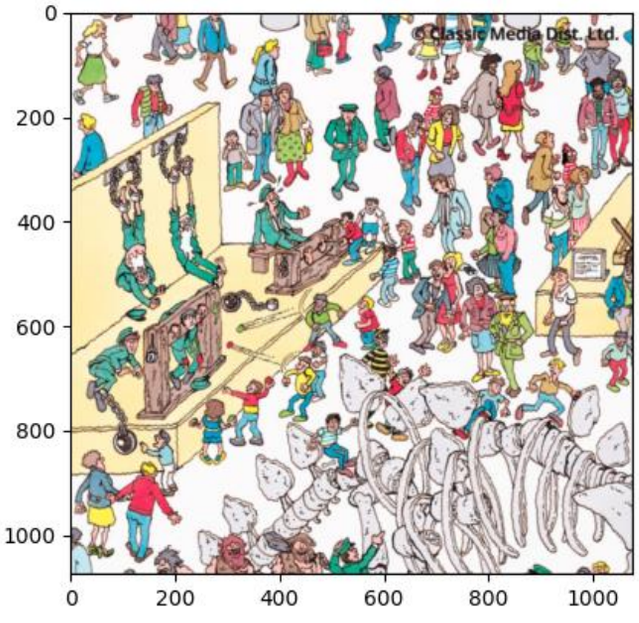
Q1 d) Code

```
def q1d_cross_corr(img, kernel):
    # not separable version of correlation
    # reusing code for convolution for correlation
    # However, convolution filps the kernel in both directions
    # therefore we need to flip it again before passing into the function so it flips it back
    kernel = np.flipud(kernel)
    kernel = np.fliplr(kernel)
    q1a_conv(img, kernel)
    return
```

```
def q1d_cross_corr_separable(img, kernel):
    # separable version of correlation
    # reusing code for convolution for correlation
    # However, convolution filps the kernel in both directions
    # therefore we need to flip it again before passing into the function so it flips it back
    kernel = np.flipud(kernel)
    kernel = np.fliplr(kernel)
    q1c_conv_separable(img, kernel)
    return
```

Q2

After applying gaussian filter

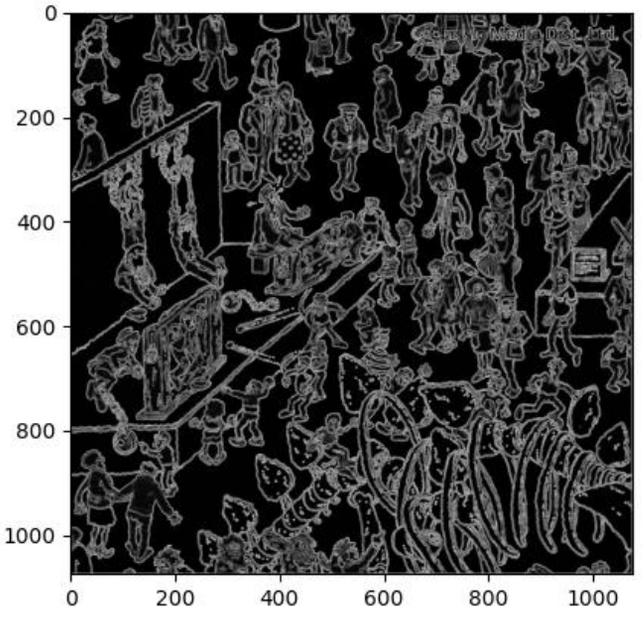


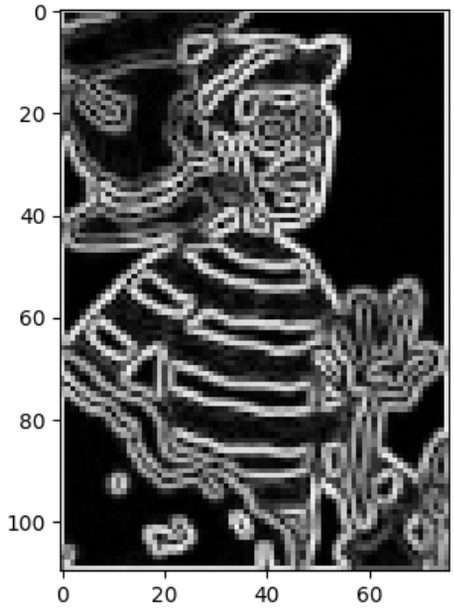
Q2 Code

```
def get_gaussian_filter(kernel_size,sigma):
    x_values = np.linspace(-1* (kernel_size//2), kernel_size//2, kernel_size)
    gaussian = np.zeros(x_values.shape[0])
    for i in range(x_values.shape[0]):
        gaussian[i] = 1 / (sigma * math.sqrt(2*math.pi)) * np.exp(-1*pow(x_values[i],2)/(2*pow(sigma,2)))
    #2d gaussian is the outer product of the 1D gaussian
    kernel = np.outer(gaussian.T, gaussian)
    # normalize
    kernel = kernel / np.sum(kernel)
    return kernel


def q2(size, sigma):
    img = io.imread("./waldo.png")
    gaussian_filter = get_gaussian_filter(size,sigma)
    for i in range(3):
        img[:,:,i] = signal.convolve2d(img[:,:,i], gaussian_filter, mode='same')
    plt.imshow(img)
    plt.show()
```

Q3 a)

Magnitude of gradients





Q3 a) code

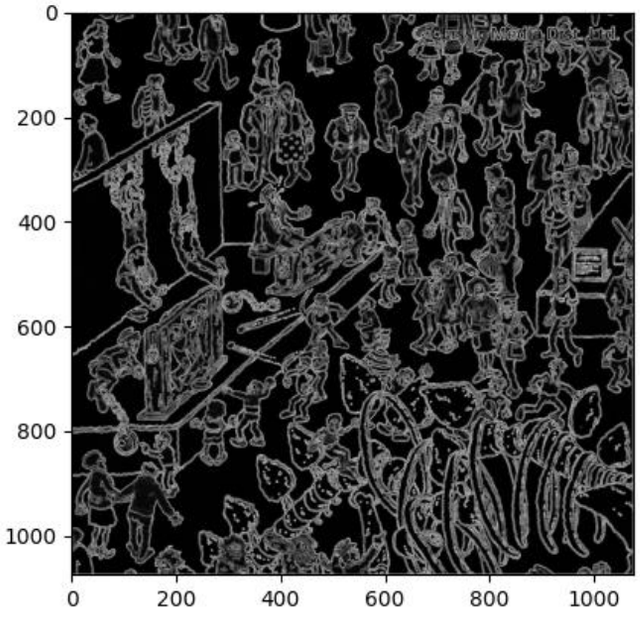
```
def q3a_magnitude_of_gradient(image):
    # using sobel fiter from the slide for egde detection
    Mx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    My = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    gradient_x = signal.convolve2d(image, Mx, mode='same')
    gradient_y = signal.convolve2d(image, My, mode='same')
    magnitude = np.sqrt(np.square(gradient_x) + np.square(gradient_y))
    plt.imshow(magnitude, cmap='gray')
    plt.gray()
    plt.show()
```

Q3 b) Code

```
def q3b_localization(image, target):
    corr = signal.correlate(image, target)
    corr = corr / (np.linalg.norm(image) * np.linalg.norm(target))
    upper_left = np.where(corr == corr.max())
    for location in zip(upper_left[0], upper_left[1]):
        x, y = location
    upper_left_index = [x, y]
    upper_right_index = [x, y-target.shape[1]-1]
    lower_left_index = [x-target.shape[0]-1, y]
    lower_right_index = [x-target.shape[0]+1, y-target.shape[1]-1]
    plt.imshow(image)
    plt.plot([upper_left_index, lower_left_index, lower_right_index, upper_right_index, upper_left_index], 'r')
    plt.show()
```

Q4

After Canny



Q4 Code

```
def q4_canny(image):
    magnitude, gradient_dir = gradient_for_q4(image)
    padded = np.pad(magnitude, (1, 1), 'constant', constant_values=(0, 0))

    # non maximum suppression
    for i in range(1, gradient_dir.shape[0]+1):
        for j in range(1, gradient_dir.shape[1]+1):
            if closer_to(gradient_dir[i-1][j-1]) == 0:
                if padded[i][j] < padded[i][j+1] or padded[i][j] < padded[i][j-1]:
                    image[i-1][j-1] = 0
            elif closer_to(gradient_dir[i-1][j-1]) == 45:
                if padded[i][j] < padded[i+1][j-1] or padded[i][j] < padded[i-1][j+1]:
                    image[i-1][j-1] = 0
            elif closer_to(gradient_dir[i-1][j-1]) == 90:
                if padded[i][j] < padded[i+1][j] or padded[i][j] < padded[i-1][j]:
                    image[i-1][j-1] = 0
            elif closer_to(gradient_dir[i-1][j-1]) == 135:
                if padded[i][j] < padded[i-1][j-1] or padded[i][j] < padded[i+1][j+1]:
                    image[i-1][j-1] = 0
    plt.imshow(magnitude, cmap='gray')
    plt.gray()
    plt.show()
```

```python
def gradient_for_q4(image):
    # using sobel fiter from the slide for egde detection
    Mx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    My = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    gradient_x = signal.convolve2d(image, Mx, mode='same')
    gradient_y = signal.convolve2d(image, My, mode='same')
    magnitude = np.sqrt(np.square(gradient_x) + np.square(gradient_y))
    gradient_direction = np.arctan2(gradient_y, gradient_x) # output value is between [-pi, pi]
    # convert radius to degrees and take care of -ve values
    gradient_dir_degree = np.round(gradient_direction*180 / math.pi)
    gradient_dir_degree = np.where(gradient_dir_degree < 0, gradient_dir_degree+180, gradient_dir_degree)
    return magnitude, gradient_dir_degree

def closer_to(direction):
    if direction < 45/2 or 135+45/2 <= direction <= 180:
        return 0
    elif 90-45/2 >= direction >= 45/2:
        return 45
    elif 135-45/2 >= direction >= 90-45/2:
        return 90
    elif 180 - 45/2 >= direction >= 135-45/2:
        return 135
    return 0
```

CSC420 Assignment1
Wu, Yu-chieh 1005473202

Written Part:

Q1. Image size = $H \times W$ → $H \cdot W$ pixels
filter size = $K \times K$ → K^2 pixels

a) * Multiplication:

- everytime we apply the filter, it will cause K^2 multiplication because we need to multiply each pixel in the filter with the corresponding pixel in the image.
- we need to apply the filter for each pixel in the image which is HW times

→ overall HWK^2 multiplications

* Addition:

- everytime we apply the filter, it will cause (K^2-1) addition because we need to sum up the K^2 numbers after the multipication. And summing K^2 number requires (K^2-1) addition
- we need to do this everytime we apply the filter, which is HW times

→ Overall $HW(K^2-1)$ addition. (or approx: HWK^2)

b) • Since we have a separable filter, we will be applying 2 1D filter with size Kx1 / 1xK → K pixels

* Multiplication:

- Everytime we apply a 1D filter, it will require K multiplication because our filter has K pixels
- For every pixel in the image, we need to apply 2 filters in total → 2K multiplications for each image pixel
- we need to apply the filter for each pixel in the image which is HW times
- overall HW2K multiplications

* Addition:

- Everytime we apply a 1D filter, we need to sum up K numbers, which requires (K-1) addition
- For every pixel in the image, we need to apply 2 filters in total → 2(K-1) addition for each image pixel
- we need to apply the filter for each pixel in the image which is HW times
- overall HW2(K-1) addition (or approx: 2HWK)

Q2.

a) If G(x,y) is isotropic:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \cdot e^{\frac{-x^2+y^2}{2\sigma^2}}$$

$$\frac{\partial G(x,y)}{\partial y} = \frac{-y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}} = \underbrace{\left(\frac{1}{\sqrt{2\pi}\,\sigma^2} e^{-\frac{x^2}{2\sigma^2}}\right)}_{f_1(x)} \underbrace{\left(\frac{-y}{\sqrt{2\pi}\,\sigma^2} e^{-\frac{y^2}{2\sigma^2}}\right)}_{f_2(y)}$$

since $\frac{\partial G(x,y)}{\partial y}$ can be factored into a function of x times a function of y $[f_1(x) \cdot f_2(y)]$ it is separable.

If G(x,y) is anisotropic:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} \cdot e^{\frac{-x^2+y^2}{2\sigma_x\sigma_y}}$$

$$\frac{\partial G(x,y)}{\partial y} = \frac{-y}{2\pi\sigma_x^2\sigma_y^2} \cdot e^{\frac{-x^2+y^2}{2\sigma_x\sigma_y}} = \underbrace{\left(\frac{1}{\sqrt{2\pi}\,\sigma_x^2} e^{-\frac{x^2}{2\sigma_x\sigma_y}}\right)}_{f_1(x)} \underbrace{\left(\frac{-y}{\sqrt{2\pi}\,\sigma_y^2} \cdot e^{-\frac{y^2}{2\sigma_x\sigma_y}}\right)}_{f_2(y)}$$

since $\frac{\partial G(x,y)}{\partial y}$ can be factored into a function of x times a function of y $[f_1(x) \cdot f_2(y)]$ it is separable.

---

b) Laplacian of Gaussian is not separable

Laplacian of Gaussian $= \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$, where $G(x,y)$ is the Gaussian function

$$LoG = -\frac{1}{\pi\sigma^4} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \cdot \left[1 - \frac{x^2+y^2}{2\sigma^2}\right]$$

As we can see, LoG cannot be separated into a function of x times a

function of y, therefore it is not separable.

LoG formula Reference:

https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

---

c) Let F be a 3×3 matrix and can be separated to $\begin{bmatrix} a \\ b \\ c \end{bmatrix}$ and $[x\ y\ z]$, therefore F can be expressed as $\begin{bmatrix} a \\ b \\ c \end{bmatrix} * [x\ y\ z] = \begin{bmatrix} ax & ay & az \\ bx & by & bz \\ cx & cy & cz \end{bmatrix}$.

from here, we can see that a 3×3 filter is only separable when both the below conditions are satisfied:

① rows are multiple of each other

② columns are multiple of each other.

Q3, cross-correlation is not commutative

Recall: cross-correlation $G = F \otimes I(i) = \sum_{-\infty}^{\infty} F(u) I(i+u)$

If cross-correlation is commutative, then $F \otimes I = I \otimes F$

$* \quad I \otimes F(i) = \sum_{-\infty}^{\infty} I(u) F(i+u)$

Also: $I \otimes F(-i) = \sum_{-\infty}^{\infty} I(u) F(-i+u) = \sum_{-\infty}^{\infty} I(u) F(u-i)$

Now, take $F \otimes I(i) = \sum_{-\infty}^{\infty} F(u) I(i+u)$ and perform u-substitution

→ let $u = u - i$: $F \otimes I(i) = \sum_{-\infty}^{\infty} F(u-i) I(i+(u-i))$

$= \sum_{-\infty}^{\infty} F(u-i) I(u)$

$= I \otimes F(-i) \neq I \otimes F(i)$

We showed that $F \otimes I \neq I \otimes F$, therefore cross-correlation is not commutative

Q4. According to the slide, applying first gaussian filter with σ_1 and then another with σ_2 is the same as applying one with $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$

→ Here, $\sigma_1 = 1$, $\sigma_2 = 2$

→ $\sigma = \sqrt{1^2 + 2^2}$

→ $\sigma = \sqrt{5}$