AVD641 (Even Sem 2025) Assignment 2

Saurabh Kumar (SC22B146)

Problem 1. *Consider a 4 × 4 grayscale image given by*

$$I = \begin{bmatrix} 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 90 & 100 & 110 & 120 \\ 130 & 140 & 150 & 160 \end{bmatrix}$$

(a) Compute the 2D Discrete Fourier Transform (DFT) of this image.
(b) Explain how the convolution theorem helps in frequency domain filtering.
(c) Suppose we apply an ideal low-pass filter that retains only the DC component. What will be the output image?

Solution. (a) The 2D DFT of an $M \times N$ image $I(x, y)$ is given by:

$$F(f1, f2) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x, y) e^{-i2\pi\left(\frac{xf1}{M} + \frac{yf2}{N}\right)}$$

This is calculated using the following MATLAB code and the result is given:

```
I = [10,20,30,40; 50,60,70,80; 90,100,110,120; 130,140,150,160];
[M,N] = size(I);
F = zeros([M,N]);
for f1=0:M-1
    for f2=0:N-1
    sum = 0;
        for x=0:M-1
            for y=0:N-1
                sum = sum + I(x+1,y+1)*exp(-1i*2*pi*((x*f1/M)+(y*
                    f2/N)));
            end
        end
        F(f1+1,f2+1) = sum;
    end
end
disp(F);
```

$$F = \begin{bmatrix} 1360 + 0i & -80 + 80i & -80 + 0i & -80 - 80i \\ -320 + 320i & 0 + 0i & 0 + 0i & 0 + 0i \\ -320 + 0i & 0 + 0i & 0 + 0i & 0 + 0i \\ -320 - 320i & 0 + 0i & 0 + 0i & 0 + 0i \end{bmatrix}$$

(b) COnvolution theorem says that the convolution in spatial domain is same as the multiplication in frequency domain. This the long process of convolution can be be shortened by

taking frequency transforms of the image and the filter, multiplying them, and then taking their inverse back to spatial domain. In frequency domain, multiplying of image with the filter will result in the filtering of image through the filter. Thus, convolution theorem can be used for frequency domain filtering.

(c) Applying low-pass filter filters out the high-frequency components from the image. The output image will be a DC image in the frequency domain, which corresponds to the average value of the pixel intensities in the spatial domain. It will show a contant pixel of intensity value equal to the average intensity of all the pixels in the original image. □

Problem 2. *Consider the following 3 × 3 image patch:*

$$I = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and the filter (kernel):

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

(a) Compute the convolution of I with K using zero-padding.
(b) Compute the correlation of I with K.
(c) Explain the difference between convolution and correlation.
(d) Show how the kernel K can be decomposed into separable filters.

Solution. (a) Calculating convolution using Python code:

```
import numpy as np

def convolve(I,K):
    I_pad = np.pad(I, ((1,1),(1,1)), 'constant')
    K_flip = np.flip(K) # flipping kernal horizontally and
       vertically
    result = np.zeros((3,3))
    for i in range (0,I.shape[0]):
        for j in range (0,I.shape[1]):
            result[i,j] = (I_pad[i,j]*K_flip[0,0] + I_pad[i,j+1]*
               K_flip[0,1] + I_pad[i,j+2]*K_flip[0,2]) + (I_pad[i
               +1,j]*K_flip[1,0] + I_pad[i+1,j+1]*K_flip[1,1] +
               I_pad[i+1,j+2]*K_flip[1,2]) + (I_pad[i+2,j]*K_flip
               [2,0] + I_pad[i+2,j+1]*K_flip[2,1] + I_pad[i+2,j
               +2]*K_flip[2,2])
    return result

I = np.array([[1,2,3],[4,5,6],[7,8,9]]);
K = np.array([[0,1,0],[1,-4,1],[0,1,0]]);
```

```
print(convolve(I,K))
```

The result is:

$$result = \begin{bmatrix} 2 & 1 & -4 \\ -3 & 0 & -7 \\ -16 & -11 & -22 \end{bmatrix}$$

(b) The python code for correlation is:

```
import numpy as np

def correlate(I,K):
    I_pad = np.pad(I, ((1,1),(1,1)), 'constant')
    result = np.zeros((3,3))
    for i in range (0,I.shape[0]):
        for j in range (0,I.shape[1]):
            result[i,j] = (I_pad[i,j]*K[0,0] + I_pad[i,j+1]*K
               [0,1] + I_pad[i,j+2]*K[0,2]) + (I_pad[i+1,j]*K
               [1,0] + I_pad[i+1,j+1]*K[1,1] + I_pad[i+1,j+2]*K
               [1,2]) + (I_pad[i+2,j]*K[2,0] + I_pad[i+2,j+1]*K
               [2,1] + I_pad[i+2,j+2]*K[2,2])
    return result

I = np.array([[1,2,3],[4,5,6],[7,8,9]]);
K = np.array([[0,1,0],[1,-4,1],[0,1,0]]);

print(correlate(I,K))
```

The result is:

$$result = \begin{bmatrix} 2 & 1 & -4 \\ -3 & 0 & -7 \\ -16 & -11 & -22 \end{bmatrix}$$

(c) Convolution gives the measure of effect of one image over the other, or filtering one with the other in frequency domain. Whereas, correlation gives the similarity between two images. Mathematically, in correlation, we slide the kernel over the each pixel of image and compute sum of products of the corresponding pixels, which gives the output for the central pixel. In case of convolution, the difference is that the kernel is flipped vertically and horizontally before this operation.

(d) For K to be decomposable, it must be written as the outer product of two vectors:

$$K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} . \begin{bmatrix} d & e & f \end{bmatrix}$$

This gives,

$$ad = 0, ae = 1, af = 0$$

or,

$$a \neq 0, d = 0, but af = 0$$

That means no solution. Hence the K cannot be decomposed into separable filters. □

Problem 3. *Consider the following 5 × 5 grayscale image:*

$$I = \begin{bmatrix} 10 & 20 & 30 & 40 & 50 \\ 60 & 70 & 80 & 90 & 100 \\ 110 & 120 & 130 & 140 & 150 \\ 160 & 170 & 180 & 190 & 200 \\ 210 & 220 & 230 & 240 & 250 \end{bmatrix}$$

(a) Compute the gradient magnitudes using the Sobel operator.
(b) Identify the edge directions.
(c) Explain how non-maximum suppression in the Canny edge detector refines edges.

Solution. (a) The python code for computing the gradient magnitudes using Sobel operator is:

```
import numpy as np

# convolution / correlation function
def convolve(I,K):
    I_pad = np.pad(I, ((1,1),(1,1)), 'constant')
    result = np.zeros(I.shape)
    for i in range (0,I.shape[0]):
        for j in range (0,I.shape[1]):
            result[i,j] = np.sum(I_pad[i:i+3, j:j+3] * K)
    return result

# Image
I = np.array([[10, 20, 30, 40, 50],
              [60, 70, 80, 90, 100],
              [110, 120, 130, 140, 150],
              [160, 170, 180, 190, 200],
              [210, 220, 230, 240, 250]])

# Sober operator -- Kernals
Gx = np.array([[-1, 0, 1],
               [-2, 0, 2],
               [-1, 0, 1]])

Gy = np.array([[-1, -2, -1],
               [0, 0, 0],
               [1, 2, 1]])
```

```
# convolution results
Ix = convolve(I, Gx)
Iy = convolve(I, Gy)

# gradient magnitude
G = np.round(np.sqrt(Ix**2 + Iy**2),2)
print(G)
```

The result is:

$$G = \begin{bmatrix} 219.54 & 286.36 & 325.58 & 364.97 & 336.15 \\ 410.37 & 407.92 & 407.92 & 407.92 & 468.61 \\ 566.04 & 407.92 & 407.92 & 407.92 & 635.3 \\ 743.24 & 407.92 & 407.92 & 407.92 & 817.07 \\ 782.43 & 682.64 & 722.5 & 762.36 & 892.75 \end{bmatrix}$$

(b) Using python:

```
theta = np.degrees(np.arctan2(Iy, Ix))
print(theta)
```

This shows the edge direction horizontally.

(c) In the Canny edge detector, the gradient magnitude is computed for the image, identifying areas with high intensity changes (possible edges). However, these potential edges may be thick or noisy. Non-Maximum Suppression is used to refine these edges by eliminating pixels that are not part of the actual edge, ensuring that only the strongest pixels (local maxima) remain.
Firstly, the gradient magnitude and edge direction are computed for the image. The, for each pixel, the gradient magnitudes is compared with its two neighbours in the direction of edge direction. If both the gradient magnitude is greater than both the neighbours, it is preserved as edge and if it is lesser than either of its neighbours, it is suppressed, i.e., not considered as edge.

□

Problem 4. *A 1D signal is given as: S = [3, 7, 15, 20, 23, 18, 12, 5, 2]*

A Gaussian kernel with $\sigma = 1$ *is given as:*

$$G = \frac{1}{16}[1, 4, 6, 4, 1]$$

(a) Apply Gaussian filtering to smooth the signal using discrete convolution.
(b) Explain the effect of increasing σ *on the output.*
(c) Why does Gaussian filtering help in reducing noise?

Solution. (a) The python code for discrete convolution is:

```
import numpy as np
```

```
# convolution function
def convolve(I,K):
    I_pad = np.pad(I, (2,2), 'constant')
    result = np.zeros(I.shape)
    for i in range (0,I.shape[0]):
      result[i] = np.sum(I_pad[i:i+5] * K)
    return result

S = np.array([3, 7, 15, 20, 23, 18, 12, 5, 2])
G = np.array([1, 4, 6, 4, 1])/16

print(convolve(S,G))
```

The result is:

$$result = \begin{bmatrix} 3.8125 & 8.375 & 14. & 18.5625 & 19.8125 & 17.0625 & 11.8125 & 6.5 & 2.75 \end{bmatrix}$$

(b) On increasing σ, the kernel size becomes larger, thus it filters over a larger number of pixels for each iteration. This makes the output image blurry.

(c) Gaussian filer is a low-pass filter, which filters out high frequency components which includes noise. This retains the low frequency components which are important part of the image, thus smoothening the image.

□

Problem 5. *Consider a 2D Gaussian function:*

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

where $\sigma = 1$.
(a) Compute the Laplacian of this Gaussian function.
(b) Sketch the zero-crossings of the Laplacian of Gaussian function and explain how they help in blob detection.
(c) Describe how Laplacian of Gaussian compares to Difference of Gaussians in practical implementations.

Solution. (a)

$$G(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

For $\sigma = 1$,

$$G(x,y) = \frac{1}{2\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

First partial derivatives are:

$$\frac{\partial G(x,y)}{\partial x} = \frac{-x}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

$$\frac{\partial G(x,y)}{\partial y} = \frac{-y}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

Second partial derivatives are:

$$\frac{\partial^2 G(x,y)}{\partial x^2} = \frac{-1+x^2}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

$$\frac{\partial^2 G(x,y)}{\partial y^2} = \frac{-1+y^2}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

Laplacian: sum of second partial derivatives

$$\nabla^2 G(x,y) = \frac{-1+x^2}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right) + \frac{-1+y^2}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

This gives,

$$\nabla^2 G(x,y) = \frac{-2+x^2+y^2}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

Thus, the Laplacian of the Gaussian function is:

$$\nabla^2 G(x,y) = \frac{x^2+y^2-2}{\pi} \exp\left(-\frac{x^2+y^2}{2}\right)$$

(b) The zero-crossings happen where the Laplacian of the Gaussian function become zero. The python code for plotting:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import cm
from matplotlib.ticker import LinearLocator

fig, ax = plt.subplots(subplot_kw={"projection": "3d"})

X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
Z = (X**2+Y**2-2)*np.exp((-X**2-Y**2)/2)/3.14

# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter('{x:.02f}')

plt.show()
```

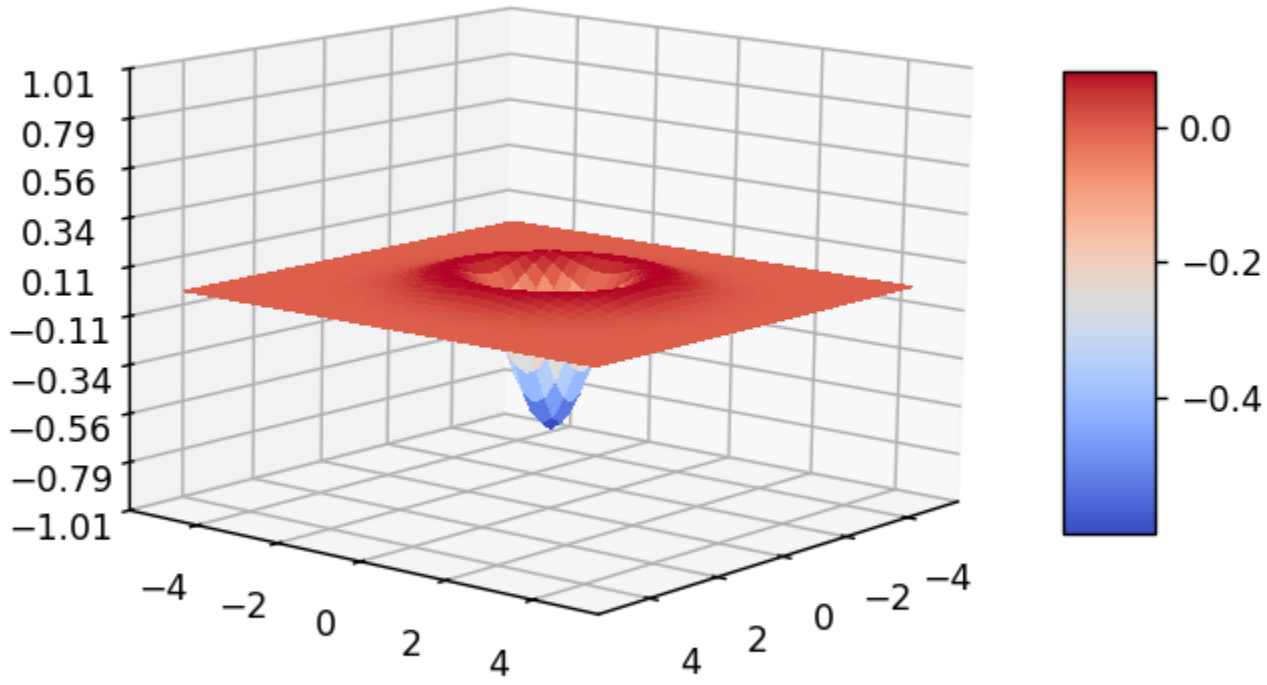

Figure 1: Laplacian of the Gaussian and its zero crossings

Zero-crossings helps in detecting regions in an image that exhibit changes in intensity. These zero-crossings correspond to locations where the image intensity changes from being concave to convex, which often marks the boundaries of blobs in an image.

(c) In practical implementations, both Laplacian of Gaussian and Difference of Gaussians are used for edge detection, blob detection, and scale-space representation. These two methods are mathematically related, but they differ in terms of computation efficiency, accuracy, and ease of implementation.
The Laplacian of Gaussian involves computing the second derivative of the Gaussian function, which requires convolving the image with a kernel that has the Laplacian of the Gaussian shape. This is computationally expensive because it involves more intensive operations and convolution steps compared to DoG. The DoG is computationally more efficient since it only requires two Gaussian convolutions with different standard deviations. However, LoG gives the more accurate result, as it can more precisely identify areas with rapid intensity changes.
□

Problem 6. *Additional Reading: Read and learn about Gabor filter.*
A Gabor filter is defined as:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right)$$

where
$x' = x\cos\theta + y\sin\theta$
$y' = -x\sin\theta + y\cos\theta$
(a) Compute the response of the image patch in Problem 3 to a Gabor filter with parameters $\lambda = 4, \theta = 0, \psi = 0, \sigma = 1, and \gamma = 0.5$.
(b) Explain how Gabor filters are useful for edge detection and texture analysis.

Solution. (a)

```
import numpy as np

# convolution function
def convolve(I,K):
    I_pad = np.pad(I, ((2,2),(2,2)), 'constant')
    result = np.zeros(I.shape)
    for i in range (0,I.shape[0]):
        for j in range (0,I.shape[1]):
            result[i,j] = np.sum(I_pad[i:i+5, j:j+5] * K)
    return result

# Image
I = np.array([[10, 20, 30, 40, 50],
              [60, 70, 80, 90, 100],
              [110, 120, 130, 140, 150],
```

```
                [160, 170, 180, 190, 200],
                [210, 220, 230, 240, 250]])

# Gabor filter
lam = 4; theta = 0; psi = 0; sigma = 1; gamma = 0.5
x, y = np.meshgrid(np.linspace(-2, 2, 5), np.linspace(-2, 2, 5))
x_prime = x*np.cos(theta) + y*np.sin(theta)
y_prime = -x*np.sin(theta) + y*np.cos(theta)
G = np.round(np.exp(-0.5*(x_prime**2 + gamma**2*y_prime**2)/sigma
    **2)*np.cos(2*np.pi*x_prime/lam + psi),2)

print(convolve(I,G))
```

The result is:

$$result = \begin{bmatrix} 105.7 & 127.2 & 131.3 & 183.8 & 205.3 \\ 218.4 & 247.5 & 241. & 324.1 & 353.2 \\ 367.6 & 402. & 377. & 492.4 & 526.8 \\ 403.4 & 432.5 & 396. & 509.1 & 538.2 \\ 353.7 & 375.2 & 339.3 & 431.8 & 453.3 \end{bmatrix}$$

(b) The Gabor filter is a type of linear filter that combines both frequency and spatial domain information. It is derived from Gaussian functions modulated by sinusoidal waves, making it sensitive to specific frequencies and orientations.
For edge detection, edges in an image are transitions between regions with different intensities. Gabor filters are sensitive to changes in the image, especially transitions from light to dark or vice versa. Gabor filters are also highly effective in extracting and analyzing the textures due to their combination of spatial and frequency localization. □