COMP 6771 Image Processing: Assignment 4

Student name: Yunqi Xu

Student id: 40130514

December 5, 2022

1 Theoretical Question 1

The detaion has been shown in Fig. 1

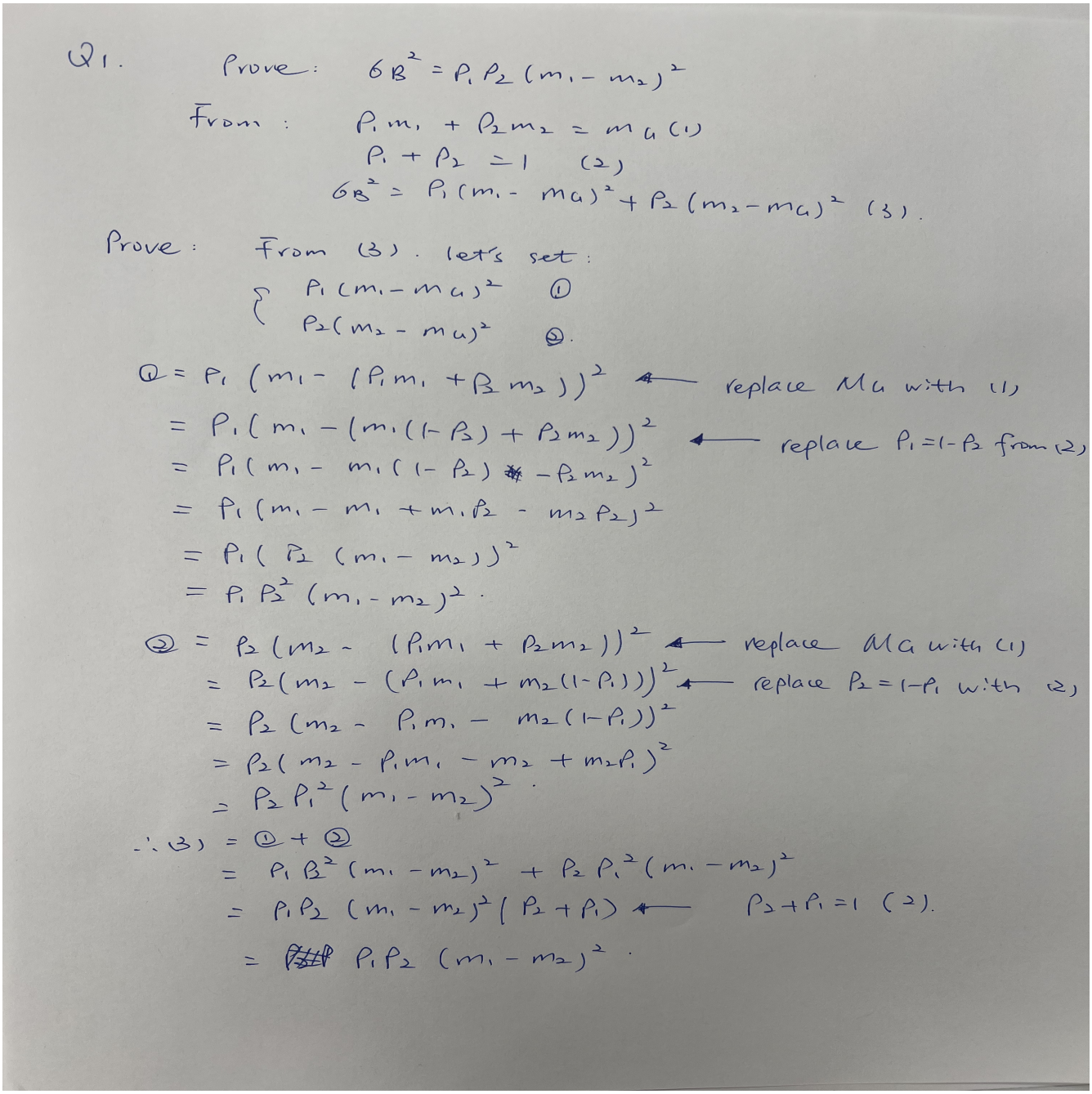


Figure 1: Question1

2 Theoretical Question 2

The main purpose of hough transform is find intersection points in parameter plane. For a point (x_1, y_1), suppose all the lines pass through it and transfor into the parameter space is $b = ax_1 + y_1$ and for another point (x_2, y_2), suppose all the lines pass through this point transfor into the parameter space

is $b = ax_2 + y_2$, the intersection between these two lines are (a_b, b_0), which is the slope and intercept of a line in the Cartesian coordinate system pass through both (x_1, y_1) and (x_2, y_2) if they are on the same line. But for a line $y = kx + b$ if it is a vertical line, the slope k is approach infinity. And it is impossible to represent values of infinity k. Due to this reason, the normal represent is an alternative solution.

Step1: Set Equation: $\rho = x\cos(\theta) + y\sin(\theta)$
Set Parameters: $threshold$; Point $P_n = (x_n, y_n)$; $\rho \in [\rho_{min}, \rho_{max}]$; $\theta \in [\theta_{min}, \theta_{max}]$

Step2: Split θ generate the $set_\theta = \{\theta_1, \theta_2, ..., \theta_u\}$.

Step3: Split ρ generate the $set_\rho = \{\rho_1, rho_2, ..., \rho_v\}$.

Step4: Set Bin is a 2D-array with (u, v) dimensions, and initialize values in Bin into 0.

Step5: For each point P_i in $P_1, P_2, ..., P_n$ do:
Loop θ_i in set_θ and calculate ρ_j based on the equation defined in Step1.
Based on the (θ_i, ρ_j) find $(position_i, position_j)$ in set_θ and set_ρ.
Then $Bin[position_i][position_j] += 1$

Step6: For all V_m in Bin,if $V_m > threshold$ do:
Find $position_\theta$ and $position_\rho$ in set_θ and set_ρ based on the position in Bin.
Suppose $\theta_i = set_\theta[position_\theta]$, $\rho_j = set_\rho[position_\rho]$ are values from the two positions.
Then those points located on the line $\rho_j = x * \cos(\theta_i) + y * \sin(\theta_i)$ compose an edge.

Step7: Finally find all target edges.

3 Theoretical Question3

The algorithm we use for recognize the bigger dots is Hit-or-Miss Transform. The Hit-or-Miss algorithm is a basic tool for shape detection. Hit-or-Miss use two structuring elements. B_1 for detecting shapes in the foreground, and B_2 for detecting shapes in the background. And following the equation:

$$I \circledast B_{1,2} = (A \ominus B_1) \cap (A^c \ominus B_2)$$

Where, A is the union of objects and A^c is the complement of A.

So, in our case, suppose the radius is r. The steps are described below.

Step1: B_1 is a dot with radius = r, which has the same pattern as the big dot in image.

Step2: Set W is a 1 pixel dilated version of B_1, and $B_2 = W - B_1$.

Step3: Erosion A with B_1 and get the output A_1

Step4: Erosion A^c with B_2 and get the output A_2.

Step5: Find the intersection between A_1 and A_2 and get the output A_3.

Step6: Calculate the number of remaining points in foreground, the number N is the result.

4 Programming Question 1

(a) The code is shown blow.

```
import numpy as np
import cv2

def imgread(path):
    return cv2.imread(path, 0)

def generateHis(img, L):
    img_height, img_width = img.shape
    value_his = np.zeros(L)
    for y in range(img_height):
        for x in range(img_width):
            intensity = img[y,x]
            value_his[intensity] += 1
    value_prob = value_his * 1.0 / (img_height * img_width)
    return value_his, value_prob

def Ostu(img, value_his, value_prob, L):
    img_height, img_width = img.shape
    final_var = 0
    var_list = []
    for k in range(1, L):
        p1 = np.sum(value_prob[:k])
        p2 = 1.0 - p1
        m1 = 0.0
        m2 = 0.0
        if p1 !=0 and p2 != 0:
            for i in range(k):
                m1 += i * value_prob[i]
            m1 = m1/p1
            for j in range(k, L):
                m2 += j * value_prob[j]
            m2 = m2/p2
            var = p1 * p2 * (m1- m2) **2
            var_list.append(var)

            if var > final_var:
                final_var = var
        else:
            var_list.append(0)
    var_list = np.array(var_list)
    threshold_list = np.where(var_list == np.max(var_list))[0]
    threshold = np.average(threshold_list)
    return threshold

def paddingReflect(img, kernel_size):
    # pre-processing:
    img_height, img_width= img.shape
    padded_img = np.zeros((img_height + kernel_size - 1, img_width +
        kernel_size - 1))
    padding_size = int((kernel_size - 1) / 2)
    padded_img[padding_size: padding_size + img_height, padding_size:
        padding_size + img_width] = img
    top_value = img[:padding_size, :]
    reversed_top_value = np.flip(top_value, axis = 0)
    padded_img[:padding_size, padding_size:padding_size+img_width] =
        reversed_top_value
```

```
        bottom_img_value = img[-padding_size:,:]
        reversed_bottom_value = np.flip(bottom_img_value, axis = 0)
        padded_img[-padding_size:,padding_size:padding_size+img_width] =
            reversed_bottom_value
        left_value = img[:, :padding_size]
        reversed_left_value = np.flip(left_value, axis = 1)
        padded_img[padding_size:padding_size+img_height, :padding_size] =
            reversed_left_value
        right_value = img[:, -padding_size:]
        reversed_right_value = np.flip(right_value, axis = 1)
        padded_img[padding_size:padding_size+img_height, -padding_size:] =
            reversed_right_value
        # 2. generate four corner
        lt_corner = img[:padding_size, :padding_size]
        reversed_lt_corner = np.flip(np.flip(lt_corner, axis=1), axis = 0)
        padded_img[:padding_size, :padding_size] = reversed_lt_corner
        rt_corner = img[:padding_size, -padding_size:]
        reversed_rt_corner = np.flip(np.flip(rt_corner, axis = 1), axis = 0)
        padded_img[:padding_size, -padding_size:] = reversed_rt_corner
        lb_corner = img[-padding_size:, :padding_size]
        reversed_lb_corner = np.flip(np.flip(lb_corner, axis = 1), axis = 0)
        padded_img[-padding_size:, :padding_size] = reversed_lb_corner
        rb_corner = img[-padding_size:, -padding_size:]
        reversed_rb_corner = np.flip(np.flip(rb_corner, axis = 1), axis = 0)
        padded_img[-padding_size:, -padding_size:] = reversed_rb_corner
        return padded_img

    def averagingFilter(input_image, filter_size):
        input_filter = np.zeros((filter_size, filter_size))
        image_height, image_width = input_image.shape[0],input_image.shape[1]
        output_image = np.zeros(input_image.shape)
        padding_size = int((filter_size - 1)/2)
        padded_image = paddingReflect(img=input_image, kernel_size=
            filter_size)

        for y in range(padding_size, image_height + padding_size):
            for x in range(padding_size, image_width + padding_size):
                sub_image = padded_image[y - padding_size: y + (padding_size
                    + 1), x - padding_size: x + (padding_size +1)]
                output_image[y - padding_size, x - padding_size] = np.sum(
                    sub_image)/(filter_size * filter_size)
        return output_image

    img = imgread(path="tools_noisy.png")
    value_his, value_prob = generateHis(img=img, L = 256)
    th = Ostu(img=img, value_his=value_his, value_prob=value_prob, L=256)
    print(th)

    img[img<=th] = 0
    img[img>th] = 255
    cv2.imshow("test", img)
    cv2.imwrite("Figures/otsu_result.png", img)

    py_img = imgread(path="tools_noisy.png")
    retVal_b, b_img = cv2.threshold(py_img, 0, 255, cv2.THRESH_OTSU)
    print(retVal_b)
```

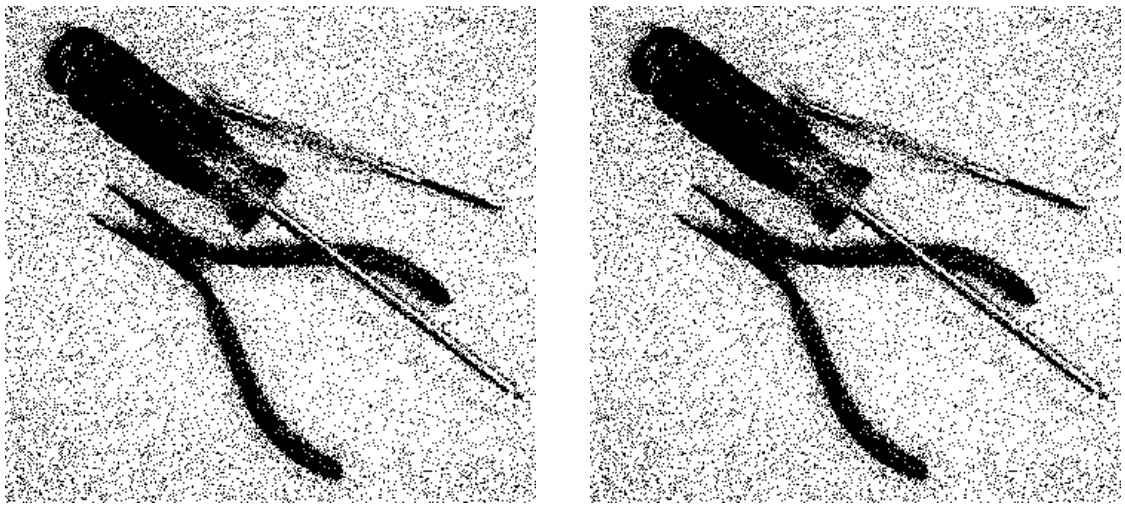
```
    cv2.imshow('test_2', b_img)
    cv2.imwrite("Figures/otsu_py.png", b_img)


    img = imgread(path = "tools_noisy.png")
    blur_img = np.uint8(averagingFilter(input_image=img, filter_size=5))
    blur_value_his, blur_value_prob = generateHis(img=blur_img, L = 256)
    blur_th = Ostu(img=blur_img, value_his=blur_value_his, value_prob=
        blur_value_prob, L=256)
    print(blur_th)
    blur_img[blur_img<=blur_th] = 0
    blur_img[blur_img>blur_th] = 255
    cv2.imshow("test blur", blur_img)
    cv2.imwrite("Figures/otsu_average.png", blur_img)

    py_img = imgread(path="tools_noisy.png")
    py_blur_img = cv2.blur(img, ksize=(5, 5), borderType=cv2.BORDER_REFLECT)
    retVal_b, blur_b_img = cv2.threshold(py_blur_img, 0, 255, cv2.THRESH_OTSU
        )
    print(retVal_b)
    cv2.imshow('test_blur python', blur_b_img)
    cv2.imwrite("Figures/otsu_py_average.png", blur_b_img)


    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

The images are shown below:



(a) Re-implement Otsu's algorithm (b) Build-in python Otsu's algorithm

Figure 2: Comparison of re-implement Otsu's algorithm and build-in Otsu's algorithm

The Fig. 2 presents the result between the re-implement Otsu's algorithm compared with the build-in python otsu's algorithm. And the result indicates that our re-implement can achieve the same output as the build-in python algorithm.

(b) The Images are shown blow:

(a) Re-implement Otsu's (with Averaging filter) (b) Build-in python Otsu's (with Averaging fitler)

Figure 3: Comparison of re-implement Otsu's and build-in Otsu's with averaging filter

The fig. 3 present the result between the re-implement Otsu's algorithm with averaging filter compared with the build-in python otsu's algorithm. And the result indicates taht our re-implemnet can achieve the same output as the build-in python algorithm. Compared with the Fig. 2 and Fig. 3, the salt and pepper noises in Fig. 2 are removed after the averaing filter. Because after smooth, the threshold moved from 149 to 140, in this case, the noise can be removed.

5 Programming Q2

(a) The code is shown below

```
I = imread('lena.tif');

% question a
[c, s] = wavedec2(I, 3, 'haar');
% level 1
[H1, V1, D1] = detcoef2('all', c, s, 1);
A1 = appcoef2(c, s, 'haar', 1);
V1img = wcodemat(V1, 255, 'mat', 1);
H1img = wcodemat(H1, 255, 'mat', 1);
D1img = wcodemat(D1, 255, 'mat', 1);
A1img = wcodemat(A1, 255, 'mat', 1);
figure
subplot(2, 2, 1);imagesc(A1img);colormap pink(255);title("(Haar)
    Approximation coef. of Level 1")
subplot(2, 2, 2);imagesc(H1img);title("(Haar) Horizontal detail coef.
    level 1")
subplot(2, 2, 3);imagesc(V1img);title("(Haar) Vertical detail coef. Level
    1")
subplot(2, 2, 4);imagesc(D1img);title("(Haar) Diagonal detail coef. Level
    1")

% level 2
[H2, V2, D2] = detcoef2('all', c, s, 2);
```

```
A2 = appcoef2(c, s, 'haar', 2);
V2img = wcodemat(V2, 255, 'mat', 1);
H2img = wcodemat(H2, 255, 'mat', 1);
D2img = wcodemat(D2, 255, 'mat', 1);
A2img = wcodemat(A2, 255, 'mat', 1);
figure
subplot(2, 2, 1); imagesc(A2img); colormap pink(255); title("(Haar) Approximation coef. of level 2")
subplot(2, 2, 2); imagesc(H2img); title("(Haar) Horizontal detail coef. level 2")
subplot(2, 2, 3); imagesc(V2img); title("(Haar) Vertical detail coef. level 2")
subplot(2, 2, 4); imagesc(D2img); title("(Haar) Diagonal detail coef. level 2")

% level 3
[H3, V3, D3] = detcoef2('all', c, s, 3);
A3 = appcoef2(c, s, 'haar', 3);
V3img = wcodemat(V3, 255, 'mat', 1);
H3img = wcodemat(H3, 255, 'mat', 1);
D3img = wcodemat(D3, 255, 'mat', 1);
A3img = wcodemat(A3, 255, 'mat', 1);
figure
subplot(2, 2, 1); imagesc(A3img); colormap pink(255); title("(Haar) Approximation coef. of level 3")
subplot(2, 2, 2); imagesc(H3img); title("(Haar) Horizontal detail coef. level 3")
subplot(2, 2, 3); imagesc(V3img); title("(Haar) Vertical detail coef. level 3")
subplot(2, 2, 4); imagesc(D3img); title("(Haar) Diagonal detail coef. level 3")
```

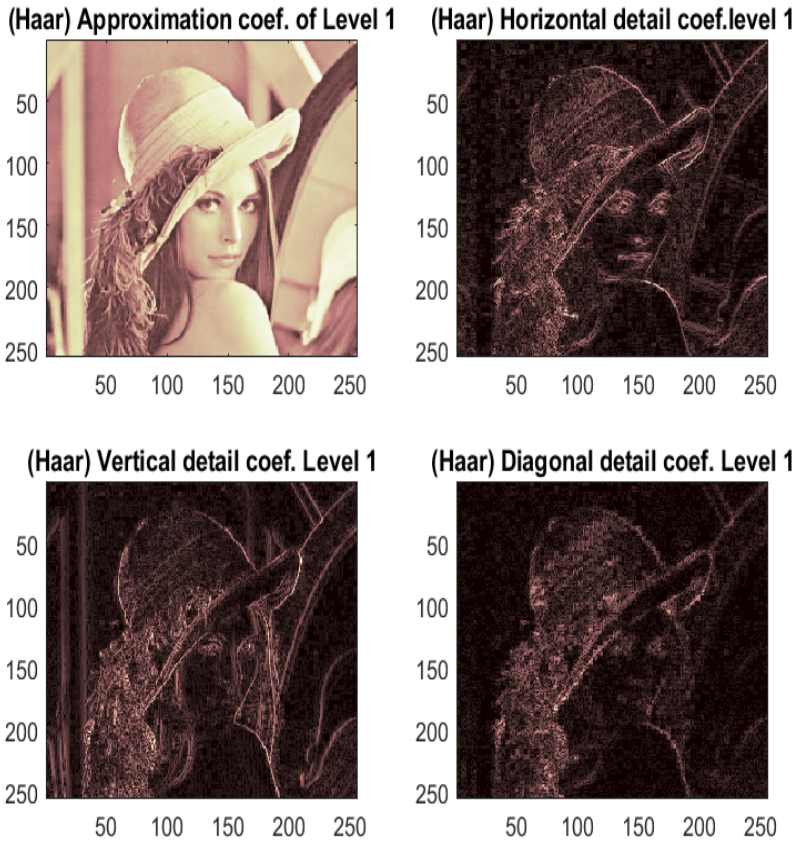


Figure 4: Haar level 1

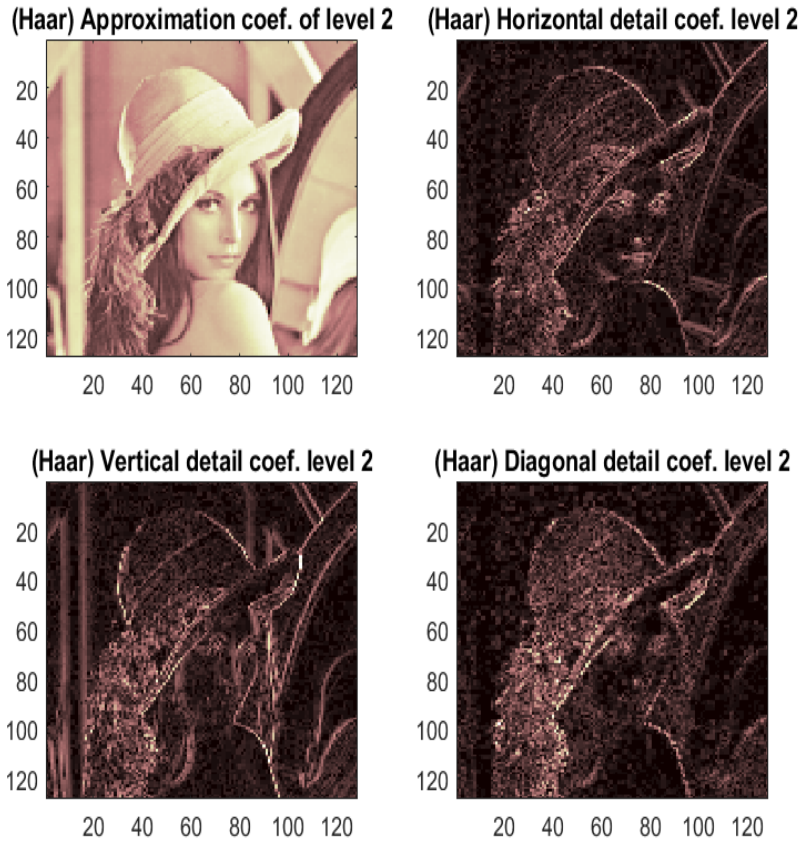


Figure 5: Haar level 2

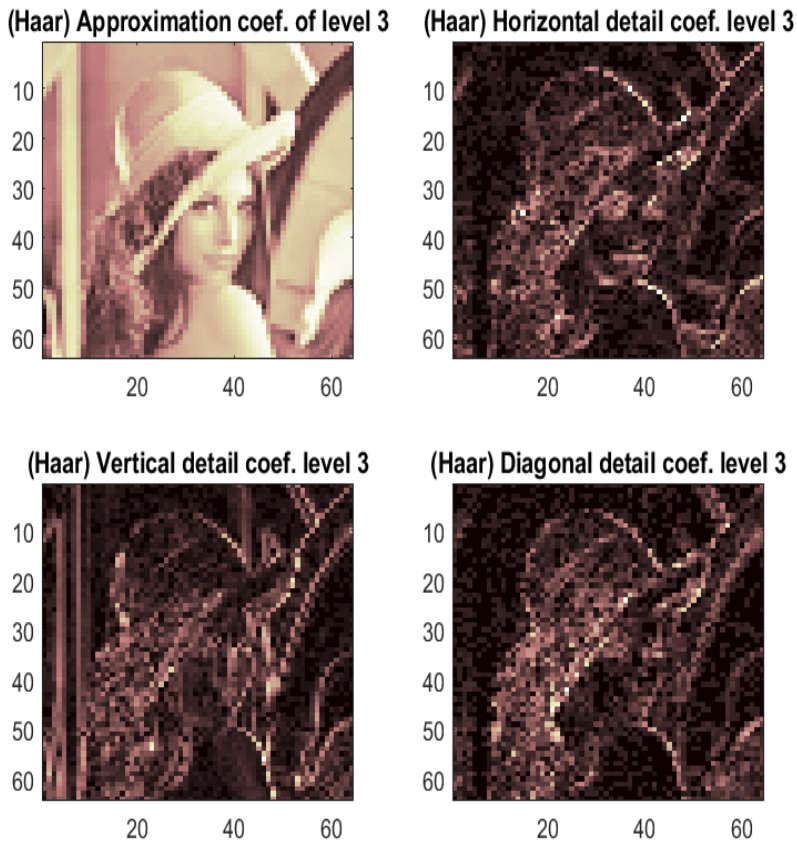


Figure 6: Haar level 3

Fig. 4, 5, 6 shown the result of the output *Wavedec*2() using Haar wavelet.

(b) The code is showing below:

```
I = imread('lena.tif');

% question b
[c, s] = wavedec2(I, 3, 'db4');
% level 1
[H1, V1, D1] = detcoef2('all', c, s, 1);
A1 = appcoef2(c, s, 'db4', 1);
V1img = wcodemat(V1, 255, 'mat', 1);
H1img = wcodemat(H1, 255, 'mat', 1);
D1img = wcodemat(D1, 255, 'mat', 1);
A1img = wcodemat(A1, 255, 'mat', 1);
figure
subplot(2, 2, 1);imagesc(A1img);colormap pink(255);title("(DB4)
    Approximation coef. of Level 1")
subplot(2, 2, 2);imagesc(H1img);title("(DB4) Horizontal detail coef. of
    level 1")
subplot(2, 2, 3);imagesc(V1img);title("(DB4) Vertical detail coef. of
    Level 1")
subplot(2, 2, 4);imagesc(D1img);title("(DB4) Diagonal detail coef. of
    Level 1")

```

```
    % level 2
    [H2, V2, D2] = detcoef2('all', c, s, 2);
    A2 = appcoef2(c, s, 'db4', 2);
    V2img = wcodemat(V2, 255, 'mat', 1);
    H2img = wcodemat(H2, 255, 'mat', 1);
    D2img = wcodemat(D2, 255, 'mat', 1);
    A2img = wcodemat(A2, 255, 'mat', 1);
    figure
    subplot(2, 2, 1); imagesc(A2img); colormap pink(255); title("(DB4) Approximation coef. of level 2")
    subplot(2, 2, 2); imagesc(H2img); title("(DB4) Horizontal detail coef. of level 2")
    subplot(2, 2, 3); imagesc(V2img); title("(DB4) Vertical detail coef. of level 2")
    subplot(2, 2, 4); imagesc(D2img); title("(DB4) Diagonal detail coef. of level 2")

    % level 3
    [H3, V3, D3] = detcoef2('all', c, s, 3);
    A3 = appcoef2(c, s, 'db4', 3);
    V3img = wcodemat(V3, 255, 'mat', 1);
    H3img = wcodemat(H3, 255, 'mat', 1);
    D3img = wcodemat(D3, 255, 'mat', 1);
    A3img = wcodemat(A3, 255, 'mat', 1);
    figure
    subplot(2, 2, 1); imagesc(A3img); colormap pink(255); title("(DB4) Approximation coef. of level 3")
    subplot(2, 2, 2); imagesc(H3img); title("(DB4) Horizontal detail coef. of level 3")
    subplot(2, 2, 3); imagesc(V3img); title("(DB4) Vertical detail coef. of level 3")
    subplot(2, 2, 4); imagesc(D3img); title("(DB4) Diagonal detail coef. of level 3")
```

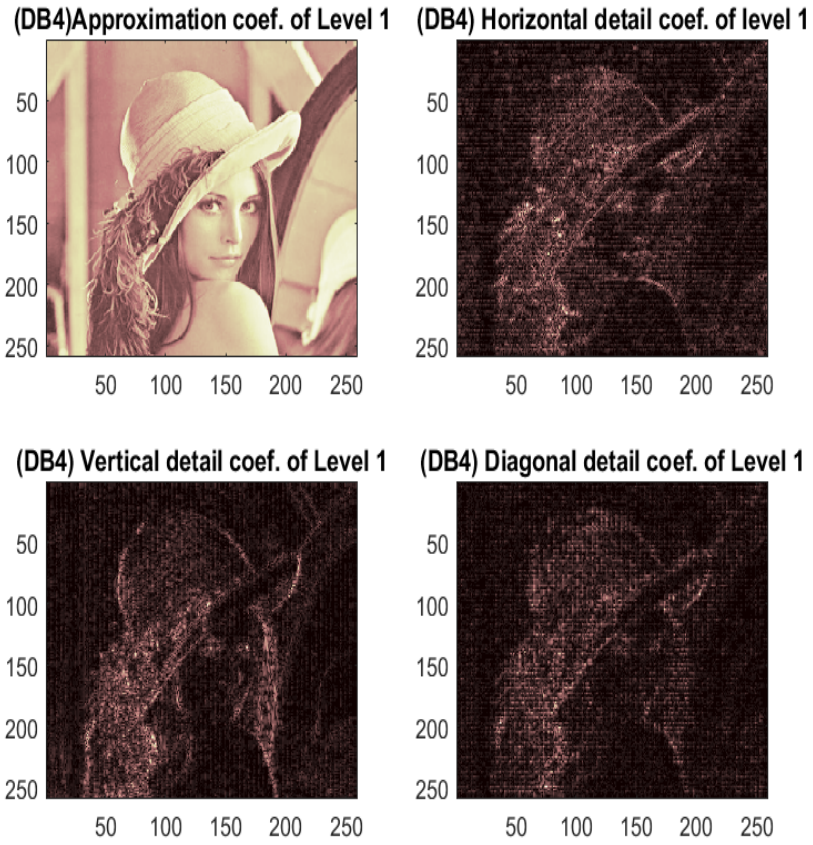


Figure 7: Daubechies-4 level 1

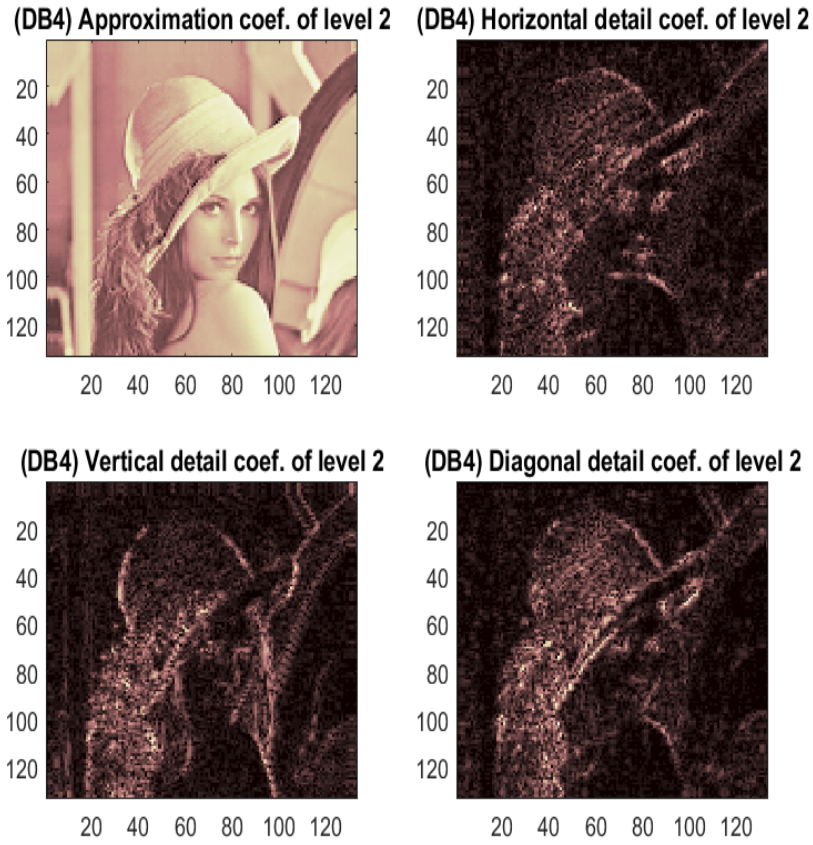


Figure 8: Daubechies-4 level 2

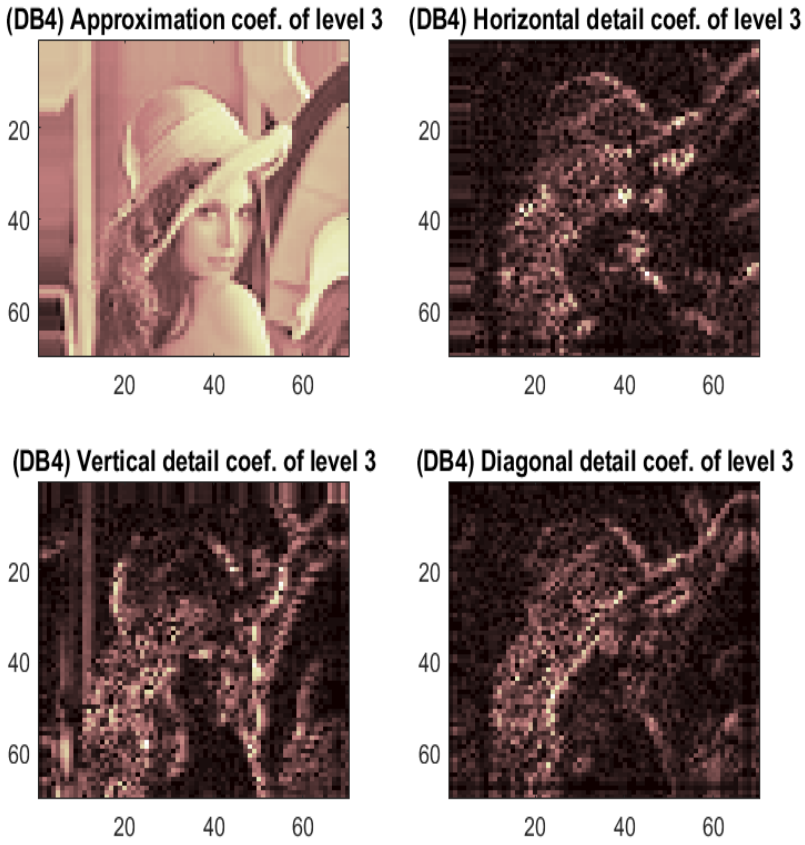


Figure 9: Daubechies-4 level 3

Fig. 7, 8, 9 present the result of the Daubechies4 wavelet.

(c) The Fig. 10 present the comparison between the level 3 approximation image of Haar and the lavel 3 approximation image of Daubechies-4. The result indicates that the Daubechies-4 has smoother result compared with Haar. Because, the Haar wavelet is resembles a step functions and have one vanishing moment. But Daubechies-4 averages over more pixels and it has more vinishing moments for giving compact support. More vinishing moment will make the result smoother than the Haar wavelet.

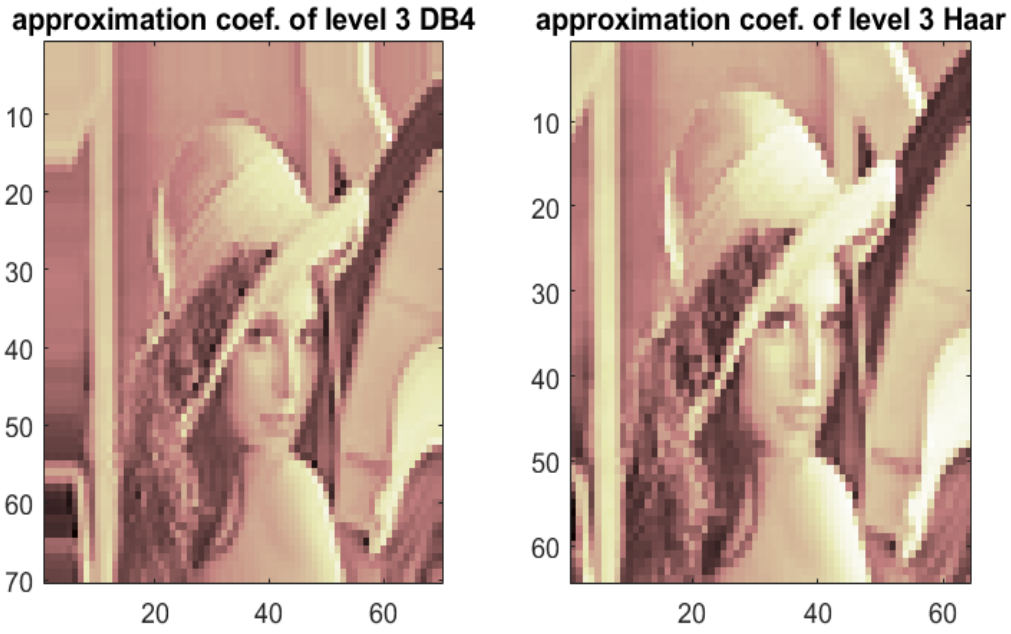


Figure 10: Comparison between Haar and Daubechies-4 in level 3