COMP 6771 Image Processing: Assignment 2

Student name: Yunqi Xu

Student id: 40130514

December 4, 2022

1. Theoretical Question 1

Based on the question, the mask is:

$$g(x, y) = \frac{1}{4}[f(x, y-1) + f(x, y+1) + f(x-1, y) + f(x+1, y)] \tag{1}$$

Also,

$$f(x - x_0, y - y_0) = F(u, v)e^{-j2\pi(ux_0/M + vy_0/N)} \tag{2}$$

Based on the Eq. 2, the Eq. 1 can be calculated like:

$$\begin{aligned} f(x, y-1) &= f(x-0, y-(1)) \\ &= F(u, v)e^{-j2\pi(u(0)/M + v(1)/N)} \\ &= F(u, v)e^{-j2\pi v/N} \end{aligned} \tag{3}$$

$$\begin{aligned} f(x, y+1) &= f(x-0, y-(-1)) \\ &= F(u, v)e^{-j2\pi(u(0)/M + v(-1)/N)} \\ &= F(u, v)e^{j2\pi v/N} \end{aligned} \tag{4}$$

$$\begin{aligned} f(x-1, y) &= f(x-(1), y-0) \\ &= F(u, v)e^{-j2\pi(u(1)/M + v(0)/N)} \\ &= F(u, v)e^{-j2\pi u/M} \end{aligned} \tag{5}$$

$$\begin{aligned} f(x+1, y) &= f(x-(-1), y-0) \\ &= F(u, v)e^{-j2\pi(u(-1)/M + v(0)/N)} \\ &= F(u, v)e^{j2\pi u/M} \end{aligned} \tag{6}$$

So, based on the Eq. 2 4 5 6,

$$G(u, v) = \frac{1}{4}F(u, v)[e^{-j2\pi v/N} + e^{j2\pi v/N} + e^{-j2\pi u/M} + e^{j2\pi u/M}] \tag{7}$$

$$H(u, v) = \frac{1}{4}[e^{-j2\pi v/N} + e^{j2\pi v/N} + e^{-j2\pi u/M} + e^{j2\pi u/M}] \tag{8}$$

Based on the Euler's Formula, $\cos\theta = \frac{1}{2}(e^{i\theta} + e^{-i\theta})$,

$$\begin{aligned} H(u, v) &= \frac{1}{4}F(u, v)[2\cos(\frac{2\pi v}{N} + 2\cos(\frac{2\pi u}{M}))] \\ &= \frac{1}{2}F(u, v)[\cos(\frac{2\pi v}{N} + \cos(\frac{2\pi u}{M}))] \end{aligned} \tag{9}$$

2. Theoretical Question 2

(a) If an equation is linear, which means that:

$$O(af_1(x,y) + bf_2(x,y)) = aO(f_1(x,y)) + bO(f_2(x,y)) \tag{10}$$

In Eq. 10, the $O()$ is an operator. So in this queation:

$$\begin{aligned} O(af_1(x,y) + bf_2(x,y)) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(af_1(x,y) + bf_2(x,y))\delta(x\cos\theta + y\sin\theta - \rho)dxdy \\ &= a\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_1(x,y)\delta(x\cos\theta + y\sin\theta - \rho)dxdy + \\ & b\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_2(x,y)\delta(x\cos\theta + y\sin\theta - \rho)dxdy \\ &= aO(f_1(x,y)) + bO(f_2(x,y)) \end{aligned} \tag{11}$$

So it is linear operator.

(b) Based on the priciple of Integral by substitution:

$$\begin{aligned} u &= x - x_0 \\ v &= y - y_0 \end{aligned} \tag{12}$$

$$\begin{aligned} x &= u + x_0 \\ y &= v + y_0 \end{aligned} \tag{13}$$

$$\begin{aligned} du &= dx \\ dv &= dy \end{aligned} \tag{14}$$

So,

$$\begin{aligned} f(\rho,\theta) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x-x_0, y-y_0)\delta(x\cos\theta + y\sin\theta - \rho)dxdy \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)\delta[(u+x_0)\cos\theta + (v+y_0)\sin\theta - \rho)dudv \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)\delta(u\cos\theta + x_0\cos\theta + v\sin\theta + y_0\sin\theta - \rho)dudv \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(u,v)\delta(u\cos\theta + v\sin\theta - (\rho - x_0\cos\theta - y_0\sin\theta))dudv \\ &= g(\rho - x_0\cos\theta - y_0\sin\theta, \theta) \end{aligned} \tag{15}$$

3. Programming Question 1

(a) The code is shown blow.

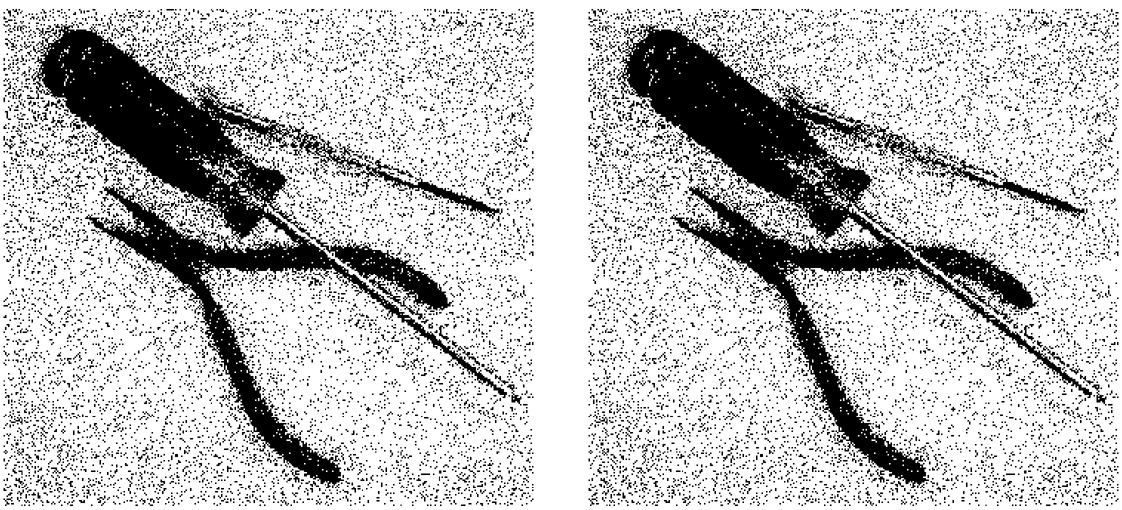
```
import numpy as np
import cv2

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=7))
```

```
    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=(7, 7), 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()
```

(b) The images are shown below:



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

Figure 1: Comparison of re-implement Otsu's algorithm and build-in Otsu's algorithm

The Fig. 1 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.

(c) The Images are shown blow:

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

Figure 2: Comparison of re-implement Otsu's and build-in Otsu's with averaging filter

The fig. 2 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. 1 and Fig. 2, the salt and pepper noises in Fig. 1 are removed after the averaing filter.

4. 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")
```

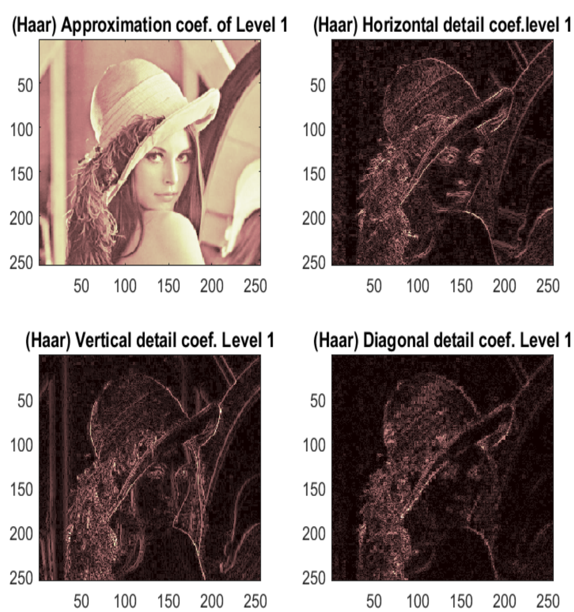


Figure 3: Haar level 1

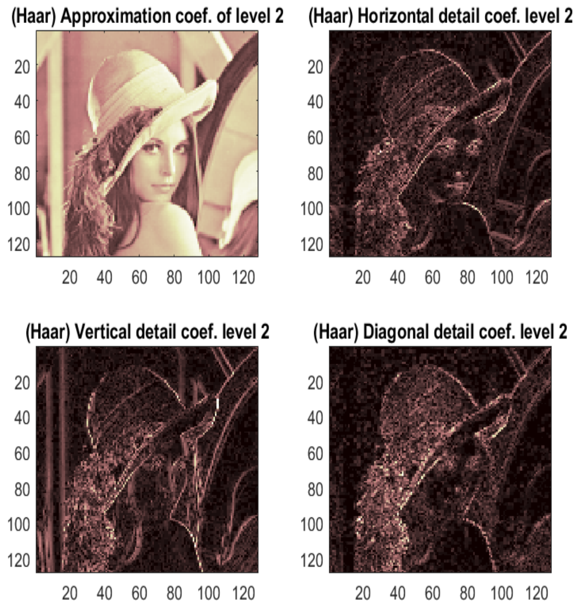


Figure 4: Haar level 2

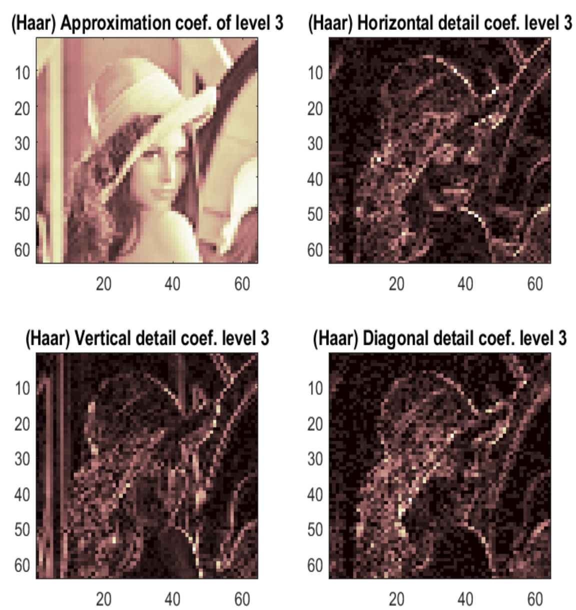


Figure 5: Haar level 3

Fig. 3, 4, 5 shown the result of the output $Wavedec2()$ 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")
```

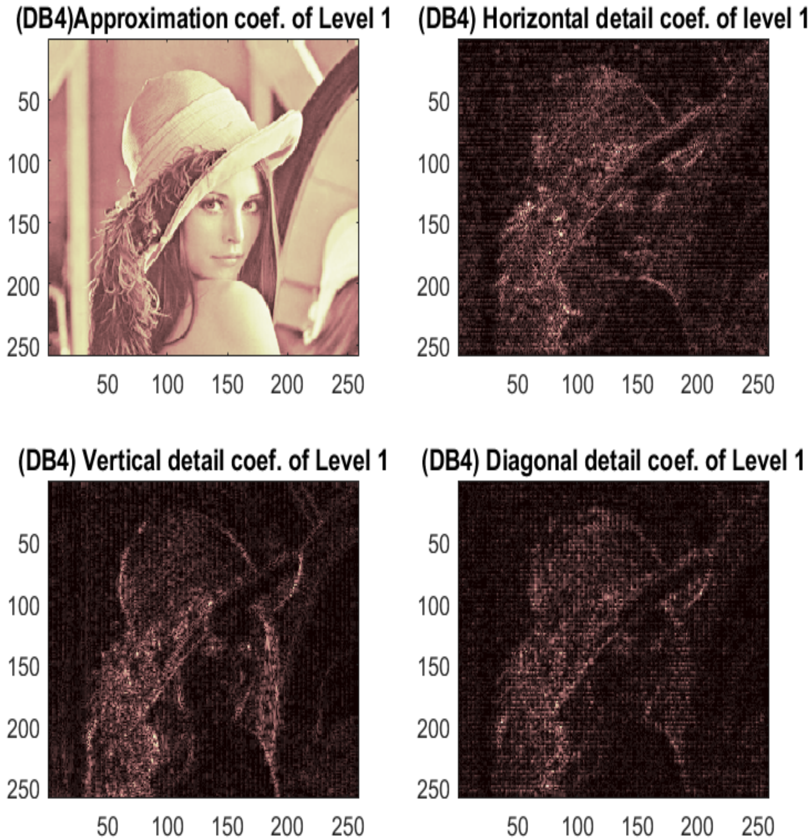


Figure 6: Daubechies-4 level 1

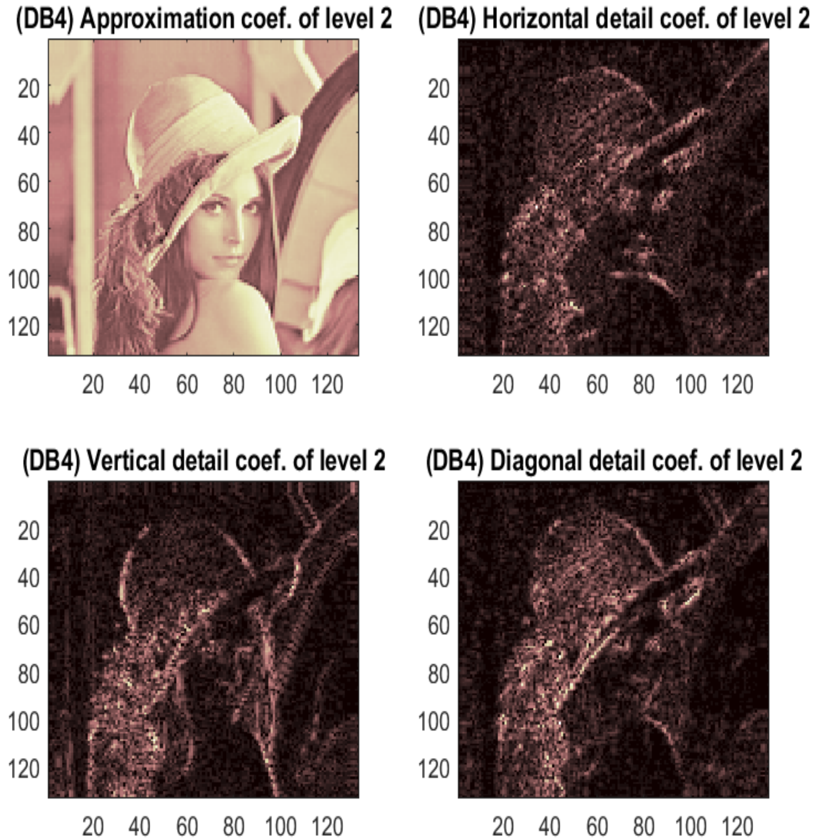


Figure 7: Daubechies-4 level 2

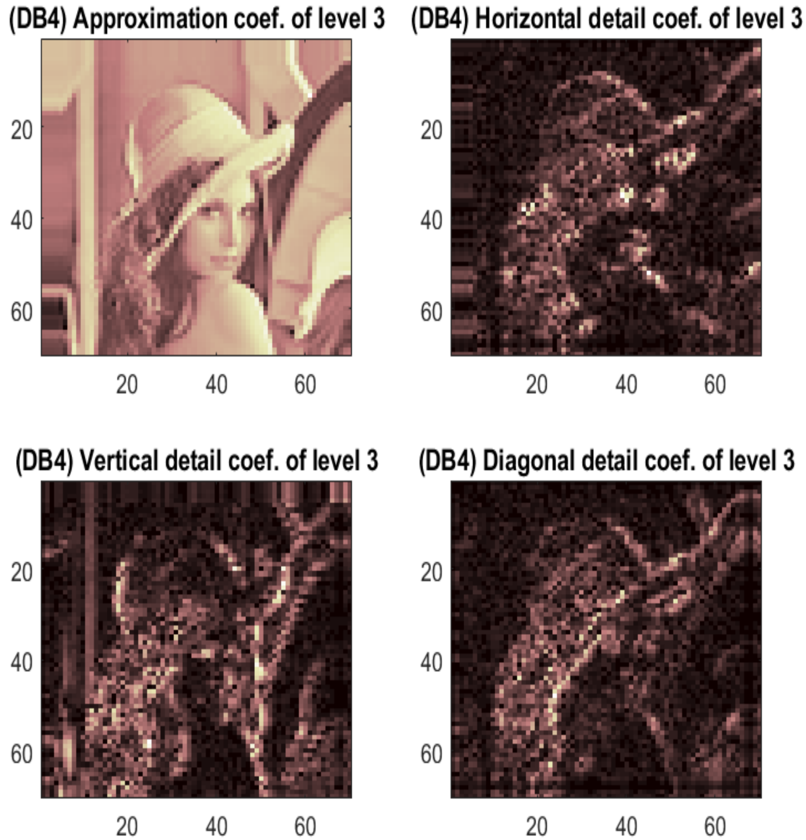


Figure 8: Daubechies-4 level 3

Fig. 6, 7, 8 present the result of the Daubechies4 wavelet.

(c) The figure present the comparison between the level 3 approximation image of Haar and the lavel 3 approximation image of Daubechies4. The result indicates that the Daubechies4 has a more smooth result compared with the result of Haar.

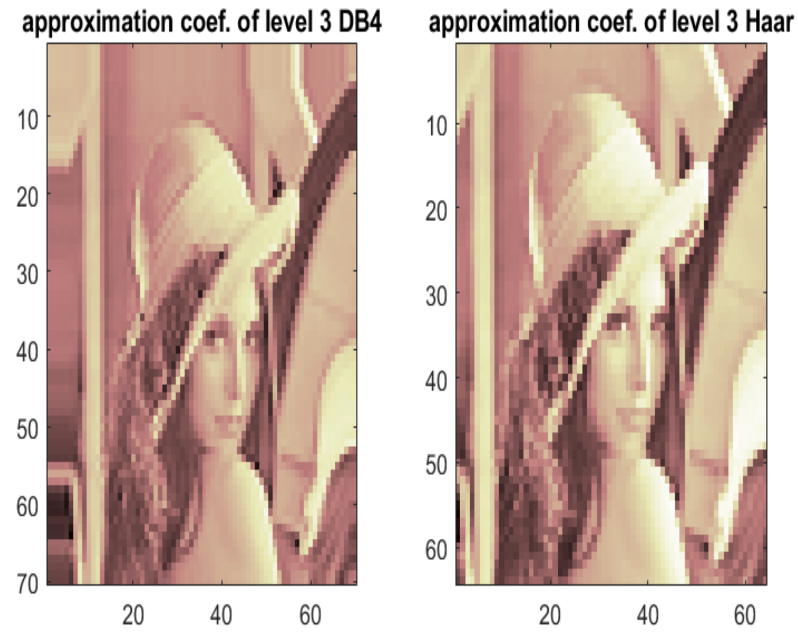


Figure 9: Comparison between Haar and Daubechies-4 in level 3