MA4268 Project 2

Jake Khoo A0217099W

April 16, 2023

Helper functions

Convolution

```
def circular_conv(a,b):
    return signal.convolve2d(a,b,mode="full")
```

We create a circular convolution and use the method full. However we do not want the first element so we splice it out in the transformations.

Down-sampling

```
def scale_down(im):
    new_array = []
    for i in range(len(im)):
        row = im[i].tolist()
        new_array += [row[1::2]]
    new_array = new_array[1::2]
    return np.array(new_array)
```

We remove the first column and row as it is unwanted from the convolution.
Let f be the input and $\tilde{f}$ be the output
Then f is a $n \times n$ array where $n \in \{2^a\}, a \in Z^+$
The function maps the $n \times n$ array to an array, $\tilde{f}$ of dimensions $\frac{n}{2} \times \frac{n}{2}$ with the following transformation:

$$\tilde{f}[n_1, n_2] = f[2n_1, 2n_2]$$

$$\forall n_1, n_2 \mod 2 = 0$$

Up-sampling

```
def scale_up(im):
    n = int(im.shape[0]*2)
    new_array = []
    add_zeros = False
    for i in range(int(n/2)):
        new_row = []
        for  j in range(int(n/2)):
            new_row += [im[i,j], 0]

        new_array += [new_row]
        new_array += [[0] * n]
    return np.array(new_array)
```

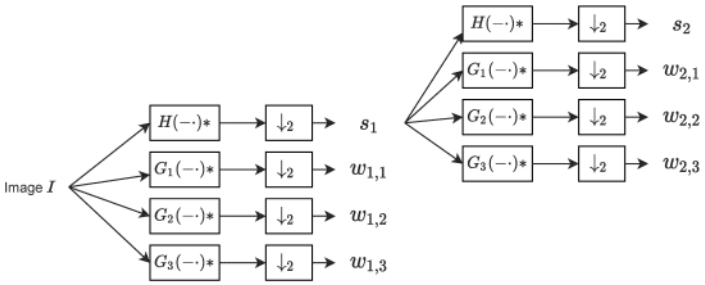
Let f be the input and $\overline{f}$ be the output
Then f is a $n \times n$ array where $n \in \{2^a\}, a \in Z^+$
The function maps the $n \times n$ array to an array, $\overline{f}$ of dimensions $2n \times 2n$ with the following transformation:

$$\overline{f}[2n_1, 2n_2] = f[n_1, n_2]$$

$$\overline{f}[2n_1 + 1, 2n_2] = 0$$
$$\overline{f}[2n_1, 2n_2 + 1] = 0$$
$$\overline{f}[2n_1 + 1, 2n_2 + 1] = 0$$
$$\forall n_1, n_2$$

2D discrete Haar Wavelet Transform

Algorithm from lecture notes:



Since we use the Haar wavelet, our wavelet filters with reversed orders are:

$$H(-\cdot) = 0.5 \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$
$$G_1(-\cdot) = 0.5 \begin{pmatrix} -1 & -1 \\ 1 & 1 \end{pmatrix}$$
$$G_2(-\cdot) = 0.5 \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix}$$
$$G_3(-\cdot) = 0.5 \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

We perform the circular convolution above on the image with each of the reversed filters and apply the down-sampling function to get the first-level coefficients and arrange it as follows:

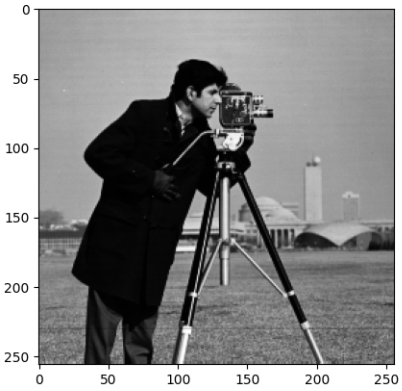
s_1	$w_{1,1}$
$w_{1,2}$	$w_{1,3}$

We repeat the process using s_1 as the new Image to get the next levels and replace s_{l-1} with $\begin{pmatrix} s_l & w_{l,1} \\ w_{l,2} & w_{l,3} \end{pmatrix}$

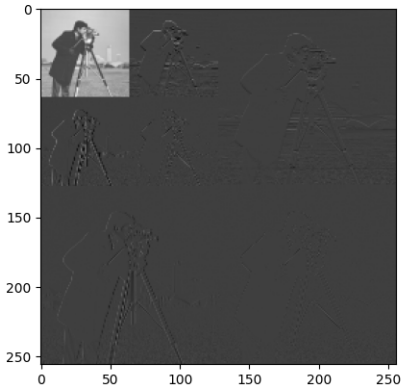
2D discrete Wavelet Transform

```
def dwt2d(im, lvl):
    result = im

    n = im.shape[0]
    H = np.array([[0.5,0.5],[0.5,0.5]])
    G1 = np.array([[-0.5,-0.5],[0.5,0.5]])
    G2 = np.array([[-0.5,0.5],[-0.5,0.5]])
    G3 = np.array([[0.5,-0.5],[-0.5,0.5]])

    new_im = scale_down(circular_conv(im,H))
    tr = scale_down(circular_conv(im,G1))
    bl = scale_down(circular_conv(im,G2))
    br = scale_down(circular_conv(im,G3))

    im = new_im
    result = np.bmat([[new_im, tr], [bl, br]])

    for i in range(lvl-1):

        H = np.array([[0.5,0.5],[0.5,0.5]])
        G1 = np.array([[-0.5,-0.5],[0.5,0.5]])
        G2 = np.array([[-0.5,0.5],[-0.5,0.5]])
        G3 = np.array([[0.5,-0.5],[-0.5,0.5]])

        new_im = scale_down(circular_conv(im,H))
        tr = scale_down(circular_conv(im,G1))
        bl = scale_down(circular_conv(im,G2))
        br = scale_down(circular_conv(im,G3))

        im = new_im
        change = np.bmat([[new_im, tr], [bl, br]])
        result[0:change.shape[0], 0:change.shape[0]] = change

    return result
```

Original sample image:



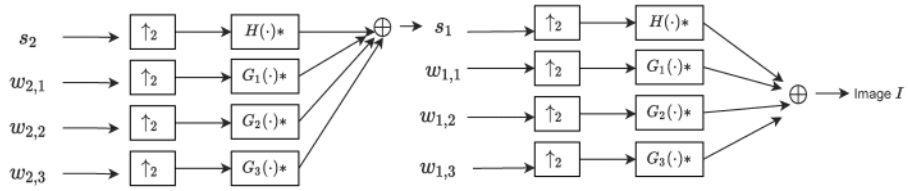
2-Level Haar wavelet coefficients:



The code only allows images of size $2^n \times 2^n$ we can do padding to increase images to 2^n. The image used is 256 by 256, in our test code the same code works for an image of 1024 by 1024.

2D inverse discrete Haar Wavelet Transform

Algorithm from lecture notes:



In the inverse case, we upscale the image first using the function above.
Since we use the Haar wavelet, our wavelet filters are:

$$H(\cdot) = 0.5 \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

$$G_1(\cdot) = 0.5 \begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$$

$$G_2(\cdot) = 0.5 \begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$$

$$G_3(\cdot) = 0.5 \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

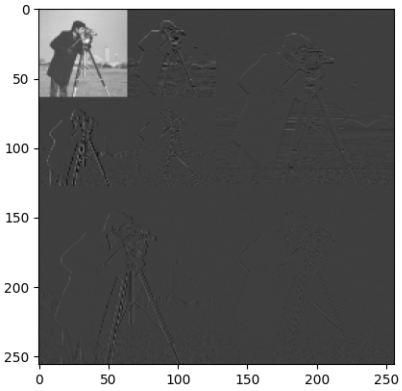
We perform the circular convolution above on the upscaled section with each the corresponding filters and it together to get s_{l-1}

We repeat the process using s_{1-1} and the other 3 quadrants until we get s_0 which is the original image.

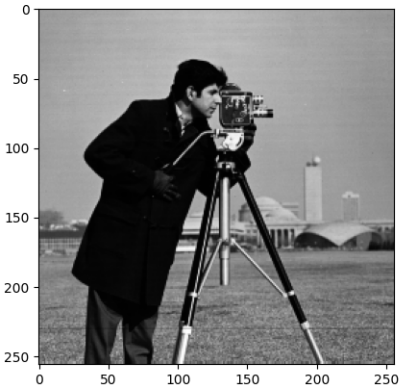
2D inverse discrete Wavelet Transform

```
def idwt2d(im, lvl):
    n = im.shape[0]

    dim = int(n / 2**lvl)
    tl = im[:dim, :dim]
    for i in range(lvl):

        tr = im[:dim, dim:int(dim*2)]
        bl = im[dim:int(dim*2), :dim]
        br = im[dim:int(dim*2), dim:int(dim*2)]

        #Scale up
        tl = scale_up(tl)
        tr = scale_up(tr)
        bl = scale_up(bl)

        br = scale_up(br)

        #Filter
        H = np.array([[0.5,0.5],[0.5,0.5]])
        G1 = np.array([[0.5,0.5],[-0.5,-0.5]])
        G2 = np.array([[0.5,-0.5],[0.5,-0.5]])
        G3 = np.array([[0.5,-0.5],[-0.5,0.5]])

        tl = circular_conv(tl,H)[:-1,:-1]
        tr = circular_conv(tr,G1)[:-1,:-1]
        bl = circular_conv(bl,G2)[:-1,:-1]
        br = circular_conv(br,G3)[:-1,:-1]


        tl = tl + tr + bl + br
        dim *= (2)

    return tl
```

Transformed image:



2-Level inverse Haar wavelet trasnform:



The inverse code is the same as the original sample array. The test code below prints true:

```
print(np.allclose(im, ilvl_2))
```

Conclusion

The main importance of Haar wavelets to image compression using Multi resolution analysis. By using the Haar filter bank, we are able to 'denoise' and compress the image by a factor of 2. Furthermore, if we add back the noise to the compressed image, we are able to get the original image even after reducing and enlarging the dimensions. However, Haar wavelet may not be as good as Daubechies wavelet as the latter has orthogonal filters that can help measure time and frequency.