Labsheet 4
Lab-4 Image Transforms

Ragja Palakkadavath
(SC18M003)

Sub: Image Processing Lab **Department**: Maths(ML)
Date of Lab sheet: August 30, 2018 **Date of Submission**: September 6, 2018

---

1. Discrete Fourier Transform (use cameraman,horizontal,vertical images)

Display the Fourier Transform of given images. Analyze how the image is getting transformed into frequency domain and display the Fourier kernels also.

Steps:

1. Read a image, im
2. Use the package fftpack from scipy to get FFT
3. If necessary , scale the real part of obtained FFT and plot

Aim

The **Aim** of the above program is to understand how discrete fourier transform modifies an image.

Discussion

The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image.

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)}$$

Figure 1: Discrete Fourier Transform

Algorithm

- Image is read and loaded and converted to an array
- fft is performed via the fft2 function from scipy pack
- The obtained fft is then shifted to make it symmetric around the origin
- The amplitude of the transform obtained from abs() function is then plotted after taking log so as to scale properly

Program Code

```
from PIL import Image
import numpy as np
from scipy import fftpack as fftp
from matplotlib import pyplot as plt

def get_image(image_path,image_name):
    path=image_path+image_name
    print(path)
    img = Image.open(path)

    img.load()
    data = np.asarray( img, dtype="float64" )
    find_dft(data)
    find_dct(data)

def find_dft(matrix):
    image_fft=fftp.fft2(matrix)
    img_shft = np.fft.fftshift(image_fft)
    amp_fft=20*np.log(np.abs(img_shft))
    plt.figure()
    plt.imshow(amp_fft)
```

Result

Discrete Fourier Transform Results

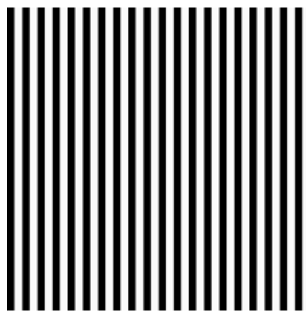

Figure 2: Original Vertical Image

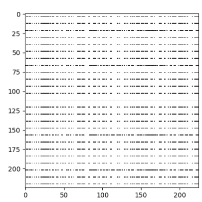

Figure 3: Discrete Fourier Transform of Vertical Image

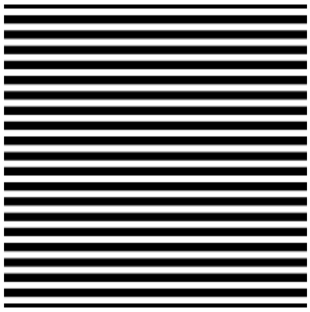

Figure 4: Original Horizontal Image

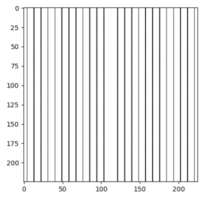

Figure 5: Discrete Fourier Transform of Horizontal Image

Figure 6: Cameraman Image

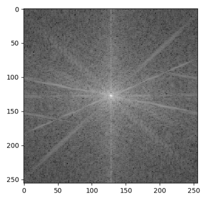

Figure 7: DFT of Cameraman Image

Inference

Transformation of an image to its Frequency Domain was performed via Discrete Fourier Transform. The amplitude/magnitude spectrum of the image was taken and plotted to realize it.

2. Discrete Cosine Transform (use cameraman,horizontal,vertical images)

Obtain the DCT of given images. Analyze them. Plot the DCT kernels also.

Aim

The **Aim** of the above program is to understand how discrete cosine transform modifies an image.

Discussion

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies.
In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers.

Algorithm

- Image is read and loaded and converted to an array
- Discrete Cosine Transform is performed via the dct function from scipy pack
- The obtained transform is then shifted to make it symmetric around the origin
- The amplitude of the transform obtained from abs() function is then plotted after taking log so as to scale properly

Program Code

```
from PIL import Image
import numpy as np
from scipy import fftpack as fftp
from matplotlib import pyplot as plt

def get_image(image_path,image_name):
    path=image_path+image_name
    print(path)
    img = Image.open(path)

    img.load()
    data = np.asarray( img, dtype="float64" )
    find_dft(data)
    find_dct(data)

def find_dct(matrix):
    image_fft=fftp.dct(matrix)
    img_shft = np.fft.fftshift(image_fft)
    amp_fft=20*np.log(np.abs(img_shft))
    plt.figure()
    plt.imshow(amp_fft, cmap='gray')
```

Result

<u>Discrete Cosine Transform Results</u>

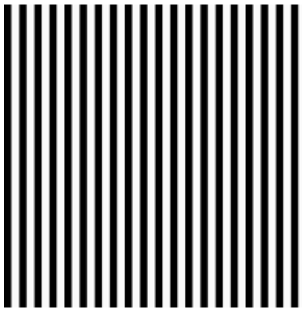

Figure 8: Original Vertical Image

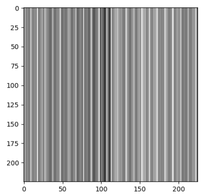

Figure 9: Discrete Cosine Transform of Vertical Image

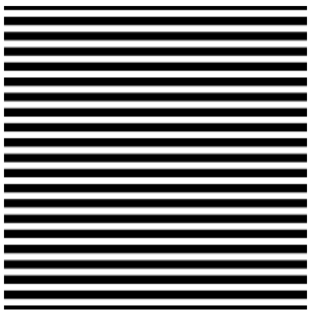

Figure 10: Original Horizontal Image

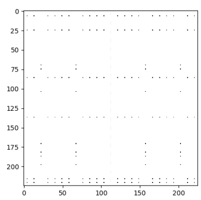

Figure 11: Discrete Cosine Transform of Horizontal Image

Figure 12: Cameraman Image

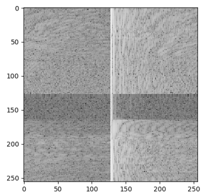

Figure 13: DCT of Cameraman Image

Inference

Transformation of an image to its Frequency Domain was performed via Discrete Cosine Transform. The amplitude/magnitude spectrum of the image was taken and plotted to realize it.

3. Haar Transform.

Show the different stages of the Haar transform of the image with different basis:

$$X = \begin{bmatrix} 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 200 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 255 & 255 & 50 & 255 & 255 & 255 \\ 50 & 50 & 50 & 50 & 255 & 255 & 255 & 255 \end{bmatrix}$$

Aim

The **Aim** of the above program is to understand how Haar Transform modifies the given matrix and view the basis images formed from Haar

Discussion

A Haar function is defined by the recursive equation:

$$\begin{aligned} H_0(t) &\equiv 1 \text{ for } 0 \le t < 1 \\ H_1(t) &\equiv \begin{cases} 1 & \text{if } 0 \le t < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \le t < 1 \end{cases} \\ H_{2^p+n}(t) &\equiv \begin{cases} \sqrt{2}^p & \text{for } \frac{n}{2^p} \le t < \frac{n+0.5}{2^p} \\ -\sqrt{2}^p & \text{for } \frac{n+0.5}{2^p} \le t < \frac{n+1}{2^p} \\ 0 & \text{elsewhere} \end{cases} \end{aligned}$$

where $p = 1, 2, 3, \ldots$ and $n = 0, 1, \ldots, 2^p - 1$.

Figure 14: Recursive Equation to Compute Haar Transform

Once the Haar Matrix is created using the recursive equation, The Haar transform of image g is given by, A = HgH^T

Algorithm

- Python code is written to form the normalized Haar matrix using the recursive equations
- The Haar transform matrix is used to pre and post multiply the image matrix to obtain the Haar Transforms
- The original image was reconstructed after the transform by multiplying with Haar Matrix transposes (as Haar Kernel is orthogonal)

Program Code

```
import numpy as np
from PIL import Image
from matplotlib import pyplot as plt

def get_image(matrix):
    data = np.asarray( matrix, dtype="uint8" )
    #Plot original Image
    plt.figure()
    plt.imshow(data, cmap='gray')
    #Obtain Haar Matrix
```

```
    h=haarMatrix(8)
    ht = h.T
    ght = np.dot(data,ht)
    hght = np.dot(h,ght)
    haarimg = Image.fromarray(hght.astype('uint8'))
    plt.figure()
    plt.imshow(haarimg, cmap='gray')
    #Obtain Reconstructed Image
    a = hght
    ah = np.dot(a,h)
    htah = np.dot(ht,ah)
    plt.figure()
    plt.imshow(htah, cmap='gray')

def haarMatrix(n, normalized=True):
    n = 2**np.ceil(np.log2(n))
    if n > 2:
        h = haarMatrix(n / 2)
    else:
        return np.array([[1, 1], [1, -1]])
    h_n = np.kron(h, [1, 1])
    if normalized:
        h_i = np.sqrt(n/2)*np.kron(np.eye(len(h)), [1, -1])
    else:
        h_i = np.kron(np.eye(len(h)), [1, -1])
    h = np.vstack((h_n, h_i))
    return h
get_image([[255,255,255,255,255,255,255,255],
     [255,255,255,100,100,100,255,255],
     [255,255,100,150,150,150,100,255],
     [255,255,100,150,200,150,100,255],
     [255,255,100,150,150,150,100,255],
     [255,255,255,100,100,100,255,255],
     [255,255,255,255,50,255,255,255],
     [50,50,50,50,255,255,255,255]])
```

Result

<u>Results of Haar Transform</u>

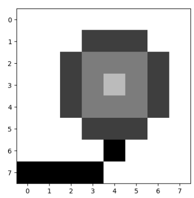

Figure 15: Original Image

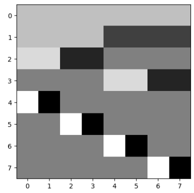

Figure 16: Haar Matrix Kernel

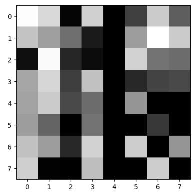

Figure 17: Haar Transformed Image

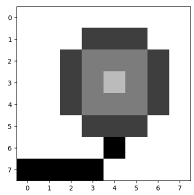

Figure 18: Image Reconstructed after Inverse Haar Transform

Inference

The Haar matrix was generated from the recursive equations. The transform matrix was then pre and post multiplied with the image/matrix to obtain the transformed image/matrix. Haar transform was realised in an image and the image was reconstructed by inverting the Haar Procedure

4. Walsh/Hadamard Transform

Show the different stages of the Walsh/hadamard transform of the image with different basis :

$$X = \begin{bmatrix} 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 200 & 150 & 100 & 255 \\ 255 & 255 & 100 & 150 & 150 & 150 & 100 & 255 \\ 255 & 255 & 255 & 100 & 100 & 100 & 255 & 255 \\ 255 & 255 & 255 & 255 & 50 & 255 & 255 & 255 \\ 50 & 50 & 50 & 50 & 255 & 255 & 255 & 255 \end{bmatrix}$$

Aim

The **Aim** of the above program is to understand how Walsh/Hadamard Transform modifies the given matrix and view the basis images formed from the transform

Discussion

The Walsh function is defined by the recursive equation:

$$W_{2j+q}(t) \equiv (-1)^{\lfloor \frac{j}{2} \rfloor + q}\{W_j(2t) + (-1)^{j+q} W_j(2t-1)\}$$

where $\lfloor \frac{j}{2} \rfloor$ means the largest integer which is smaller or equal to $\frac{j}{2}$, $q = 0$ or 1, $j = 0, 1, 2, \ldots$
and:

$$W_0(t) \equiv \begin{cases} 1 & \text{for } 0 \leq t < 1 \\ 0 & \text{elsewhere} \end{cases}$$

Figure 19: Recursive Equation to Compute Walsh Transform

Hadamard Transform is a special case of Walsh Transform where the entries of the orthogonal transform matrices are +1 or -1

Once the Hadamard Matrix is created using the recursive equation, The Hadamard transform of image g is given by, A = HgH^T

Algorithm

- Hadamard matrix can be defined using the required size of the image. It is obtained from linalg function in python
- The Hadamard transform matrix is used to pre and post multiply the image matrix to obtain the Haar Transforms
- The original image was reconstructed after the transform by multiplying with Hadamard Matrix transposes (as Hadmard Kernel is orthogonal)

Program Code

```
import numpy as np
from scipy import linalg
from PIL import Image
from matplotlib import pyplot as plt


def find_Hadamard(matrix):
    data = np.asarray( matrix, dtype="uint8" )
    #Plot original Image
    plt.figure()
    plt.imshow(data, cmap='gray')

    h = linalg.hadamard(8)
    #Plot Hadamard Matrix
    plt.figure()
    plt.imshow(h, cmap='gray')

    ht = h.T
    ght = np.dot(data,ht)
    hght = np.dot(h,ght)

    plt.figure()
    plt.imshow(hght, cmap='gray')

    a = hght
    ah = np.dot(a,h)
    htah = np.dot(ht,ah)

    plt.figure()
    plt.imshow(htah, cmap='gray')
```

```
find_Hadamard([[255,255,255,255,255,255,255,255],
        [255,255,255,100,100,100,255,255],
        [255,255,100,150,150,150,100,255],
        [255,255,100,150,200,150,100,255],
        [255,255,100,150,150,150,100,255],
        [255,255,255,100,100,100,255,255],
        [255,255,255,255,50,255,255,255],
        [50,50,50,50,255,255,255,255]])
```

Result

Results of Hadamard Transform

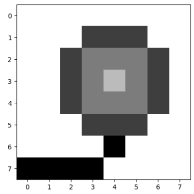

Figure 20: Original Image

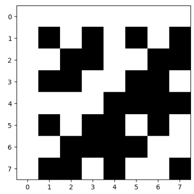

Figure 21: Hadamard Matrix Kernel

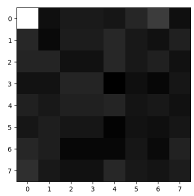

Figure 22: Hadamard Transformed Image

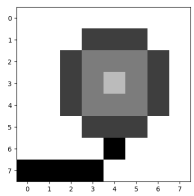

Figure 23: Image Reconstructed after Inverse Hadamard Transform

Inference

The Hadamard matrix was generated from the recursive equations. The transform matrix was then pre and post multiplied with the image/matrix to obtain the transformed image/matrix. Hadamard transform was realised in an image. Hadamard transform is a special case of Waalsh Transform where the entries of the transform matrices are +1 or -1