DEPARTMENT OF COMPUTER & INFORMATION SCIENCE

FACULTY OF SCIENCE

Laboratory Manual

19PCSP306- Image Processing Lab
for
MSc in Computer Science

In-charge
Dr Mari Kamarasan
Assistant Professor

ANNAMALAI UNIVERSITY
ANNAMALAI NAGAR-608002

Developed & Compiled by Dr Mari Kamarasan, Assistant Professor, Dept. of Computer and Information Science.

DO'S AND DON'TS

DO'S

1. Student should get the record of previous experiment checked before starting the new experiment.

2. Read the manual carefully before starting the experiment.

3. Checked the program by the instructor.

4. Get your results checked by the teacher.

5. Computers must be handled carefully.

6. Maintain strict discipline.

7. Keep your mobile phone switched off or in vibration mode.

8. Students should get the experiment allotted for next turn, before leaving the lab.

DON'TS 1.

1. Do not touch or attempt to touch the mains power supply Wire with bare hands.

2. Do not overcrowd the tables.

3. Do not tamper with equipments.

4. Do not leave the without permission from the teacher.

List of Exercises

1. Write a java or python program to convert image into intensity value array.
2. Write a Java or python program convert Color image into Grayscale image.
3. Write a Java or python program convert negative image.
4. Write a Java or python program to perform smoothing operations on an image in spatial domain.
5. Write a Java or python program to perform sharpening operations on an image in spatial domain.
6. Write a Java or python program to transform the image into Fourier transform.
7. Write a Java or python program to performance of gradient operators.
8. Write a Java or python program to perform DCT image compression method.
9. Write a Java or python program to implement image segmentation based on color.
10. Write a Java or python program to implement erosion and dilation operations on image.

Exercise -1

Aim: write a java program to convert the digital image into its corresponding intensity values

Methods used:

1. **AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.
2. **FileOutputStream** is an output stream used for writing data to a file.If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.
3. **DataOutputStream** class allows an application to write primitive Java data types to the output stream in a machine-independent way. ava application generally uses the data output stream to write data that can later be read by a data input stream.
4. **javax.imageio**. ImageIO is a utility class which provides lots of utility method related to images processing in Java. Most common of them is reading form image file and writing images to file in java.

Source Code:

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
class Pixel {
  BufferedImage image;
  int width;
  int height;
```

```
  public Pixel() {
    try {
      File input = new File("lena_128.jpg");
      File file = new File("foo.out");
         FileOutputStream fos = new FileOutputStream(file);
         BufferedOutputStream bos = new BufferedOutputStream(fos);
         DataOutputStream dos = new DataOutputStream(bos);
      image = ImageIO.read(input);
      width = image.getWidth();
      height = image.getHeight();
      int output;
      for(int i=0; i<height; i++) {
      for(int j=0; j<width; j++) {

           Color c = new Color(image.getRGB(j, i));
           output= (c.getRed() + c.getGreen() +  c.getBlue())/3;
                String s=String.valueOf(output+" ");
                dos.writeBytes(s);

        }
                System.out.println();
      }

    } catch (Exception e) {}

 }
  static public void main(String args[]) throws Exception {
    Pixel obj = new Pixel();
  }
}
```

Input image:

Sample Output:

[63 163 162 161 159 157 156 156 154 159 164 171 173 170 159 140 113 105 109 114 116 117
115 114 115 115 117 120 125 128 133 133 133 135 136 136 134 132 132 132 136 140 141 138
137 139 139 136 137 139 141 141 140 137 135 134 137 138 137 134 134 137 139 138 139 137
137 138 138 134 132 133 134 136 136 133 131 131 131 129 115 110 123 144 157 162 158 154
152 151 153 156 157 154 154 154 152 155 155 155 156 155 164 183 201 215 200 129 102 110
117 127 129 127 132 133 126 124 128 127 129 126 130 133 132 135 118 116 164 162 160 158
158 156 155 155 153 159 165 172 175 170 156 137 109 103 108 114 114 114 113 115 113 113
116 121 126 130 131 130 133 134 135 136 135 134 134 133 136 139 140 139 138 139 138 136
138 139 140 140 139 137 135 135 135 137 136 134 135 137 138 137 138 136 136 137 136 133
132 132 133 134 135 133 132 133 130 126 117 111 121 139 152 162 162 159 151 151 153 155
157 154 154 154 152 155 155 155 157 153 159 174 201 214 204 145 117 107 108 126 128 127]

Exercise -2

Aim: write a java program to convert color image into gray-scale image.

Methods used:

1. **AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.
2. **javax.imageio.ImageIO** is a utility class which provides lots of utility method related to images processing in Java. Most common of them is reading form image file and writing images to file in java. You can write any of **.jpg, .png, .bmp** or .gif images to file in Java. Just like writing, reading is also seamless with ImageIO and you can read BufferedImage directly from URL.
3. **setRGB()**. Sets an array of integer pixels in the default RGB color model (TYPE_INT_ARGB) and default sRGB color space, into a portion of the **image** data. Color conversion takes place if the default model does not match the **image** ColorModel.
4. **getRGB**. Returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace. Color conversion takes place if this default model does not match the **image** ColorModel . There are only 8-bits of precision for each color component in the returned data when using this method.

Source code:

```
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
class GrayScale {
  BufferedImage  image;
  int width;
```

```
  int height;
  public GrayScale() {
    try {
      File input = new File("digital_image_processing.jpg");
      image = ImageIO.read(input);
      width = image.getWidth();
      height = image.getHeight();
       for(int i=0; i<height; i++) {
        for(int j=0; j<width; j++) {
         Color c = new Color(image.getRGB(j, i));
           int red = (int)(c.getRed() * 0.299);
           int green = (int)(c.getGreen() * 0.587);
           int blue = (int)(c.getBlue() *0.114);
           Color newColor = new Color(red+green+blue,red+green+blue,red+green+blue);
           image.setRGB(j,i,newColor.getRGB());
         }
       }
      File ouptut = new File("grayscale.jpg");
       ImageIO.write(image, "jpg", ouptut);
     } catch (Exception e) {}
  }
 static public void main(String args[]) throws Exception {
   GrayScale obj = new GrayScale();
  }
}
```

Input image:

Output image:

Exercise -3

Aim: Write a Java program convert image into negative image

Methods used:

1. **javax.imageio.ImageIO** is a utility class which provides lots of utility method related to images processing in Java. Most common of them is reading form image file and writing images to file in java. You can write any of **.jpg, .png, .bmp** or .gif images to file in Java. Just like writing, reading is also seamless with ImageIO and you can read BufferedImage directly from URL.
2. **setRGB()**. Sets an array of integer pixels in the default RGB color model (TYPE_INT_ARGB) and default sRGB color space, into a portion of the **image** data. Color conversion takes place if the default model does not match the **image** ColorModel.
3. **getRGB**. Returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace. Color conversion takes place if this default model does not match the **image** ColorModel . There are only 8-bits of precision for each color component in the returned data when using this method.
4. **image.getWidth():** Determines the width of the image. If the width is not yet known, this method returns `-1` and the specified `ImageObserver` object is notified later.
5. **image.getHeight():** Determines the height of the image. If the height is not yet known, this method returns `-1` and the specified `ImageObserver` object is notified later.
6. **Nagative image transform**: The negative of an image is achieved by replacing the intensity 'i' in the original image by 'i-1', i.e. the darkest pixels will become the brightest and the brightest pixels will become the darkest. Image negative is produced by subtracting each pixel from the maximum intensity value.

   Formula:

   s = T(r) = (L – 1) – r,
   where L - 1 is the max intensity value, s is the output pixel value and r is the input pixel value

```
/ Ex.3 Write a Java program convert negative image */
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
class Negative{
  BufferedImage  image;
  int width;
  int height;
    public Negative() {
      try {
       File input = new File("digital_image_processing.jpg");
       image = ImageIO.read(input);
       width = image.getWidth();
       height = image.getHeight();
       for(int i=0; i<height; i++) {
       for(int j=0; j<width; j++) {
       Color c = new Color(image.getRGB(j, i));
            int red = (int)(c.getRed() * 0.299);
            int green = (int)(c.getGreen() * 0.587);
            int blue = (int)(c.getBlue() *0.114);
               int gray = (red+green+blue)/3;
               gray = (255 - gray -1);
               //System.out.println("Gray"+gray);
```

```
            //Color newColor = new Color(red+green+blue,red+green+blue,red+green+blue);
            Color newColor = new Color(gray,gray,gray);
            image.setRGB(j,i,newColor.getRGB());
                    System.out.println("Break");
          }
        }
        File ouptut = new File("negativeimage.jpg");
        ImageIO.write(image, "jpg", ouptut);
      } catch (Exception e) {}
  }
    static public void main(String args[]) throws Exception {
    Negative obj = new Negative();
  }
}
```

Input image:

Output image:

Exercise -4

Aim: **Write a Java program to enhance the image in spatial domain using power law transformation**

Methods used:

1. **javax.imageio.ImageIO** is a utility class which provides lots of utility method related to images processing in Java. Most common of them is reading form image file and writing images to file in java. You can write any of **.jpg, .png, .bmp** or .gif images to file in Java. Just like writing, reading is also seamless with ImageIO and you can read BufferedImage directly from URL.
2. **setRGB()**. Sets an array of integer pixels in the default RGB color model (TYPE_INT_ARGB) and default sRGB color space, into a portion of the **image** data. Color conversion takes place if the default model does not match the **image** ColorModel.
3. **getRGB**. Returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace. Color conversion takes place if this default model does not match the **image** ColorModel . There are only 8-bits of precision for each color component in the returned data when using this method.
4. **image.getWidth():** Determines the width of the image. If the width is not yet known, this method returns `-1` and the specified `ImageObserver` object is notified later.
5. **image.getHeight():** Determines the height of the image. If the height is not yet known, this method returns `-1` and the specified `ImageObserver` object is notified later.
6. **Power – Law transformations**:

   s=cr^γ

   This symbol γ is called gamma, due to which this transformation is also known as gamma transformation. Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their own gamma correction, that's why they display their image at different intensity.

```
import java.awt.*;
```

```
import java.awt.image.BufferedImage;
import java.io.*;
import java.lang.*;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
class Powerlaw{
  BufferedImage  image;
  int width;
  int height;
  public Powerlaw() {
     try {
      File input = new File("lena.jpg");
      image = ImageIO.read(input);
      width = image.getWidth();
      height = image.getHeight();
         double gamma;
      gamma=1.25;
      for(int i=0; i<height; i++) {
      for(int j=0; j<width; j++) {
          Color c = new Color(image.getRGB(j, i));
           int red = (int)(c.getRed() * 0.299);
           int green = (int)(c.getGreen() * 0.587);
           int blue = (int)(c.getBlue() *0.114);
              int gray = (red+green+blue)/3;
              int gray1 = (int) Math.pow(gray,gamma);
             // System.out.println("Gray"+gray1);
          Color newColor = new Color(gray1,gray1,gray1);
          image.setRGB(j,i,newColor.getRGB());
        }
      }
      File ouptut = new File("powerlaw1.jpg");
```

```
        ImageIO.write(image, "jpg", ouptut);

      } catch (Exception e) {}
   }
static public void main(String args[]) throws Exception {
      Powerlaw obj = new Powerlaw();
   }
}
```

Input image:

Output image:

Exercise -5

Aim: Write a python program to implement image sharpening technique in spatial domain

Methods used:

1. **OpenCV-Python** is a library of Python bindings designed to solve computer vision problems.
   cv2.imread() method loads an image from the specified file. If the image cannot be read (because of missing file, improper permissions, unsupported or invalid format) then this method returns an empty matrix.
   Syntax: cv2.imread(path, flag) where **path:** A string representing the path of the image to be read. **flag:** It specifies the way in which image should be read. It's default value is **cv2.IMREAD_COLOR.**

2. **Numpy()** array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.
   We can initialize numpy arrays from nested Python lists, and access elements using square brackets:
   import numpy as np
   a = np.array([1, 2, 3]) # Create a rank 1 array
3. **The Filter2D()** operation convolves an image with the kernel. You can perform this operation on an image using the **Filter2D()** method of the **imgproc** class. Following is the syntax of this method −
   filter2D(src, dst, ddepth, kernel)
   This method accepts the following parameters −
   src − A **Mat** object representing the source (input image) for this operation.
   dst − A **Mat** object representing the destination (output image) for this operation.
   ddepth − A variable of the type integer representing the depth of the output image.
   kernel − A **Mat** object representing the convolution kernel.
4. cv2.imshow() method is used to display an image in a window. The window automatically fits to the image size.
   Syntax: cv2.imshow(window_name, image)
   window_name: A string representing the name of the window in which image to be displayed.
   image: It is the image that is to be displayed.

Source code:

```
import cv2 as cv
import numpy as np
image = cv.imread('D:\TVA\SOFTWARE\INPUT\lena.bmp')
kernel = np.array(      [[-1, -1, -1],
                         [-1,  9, -1],
                         [-1,-1,-1]])
```

```
sharpened = cv.filter2D(image, -1, kernel) # applying the sharpening kernel to the input image & displaying it.
cv.imshow('original Image', image)
cv.imshow('Image Sharpening', sharpened)
cv.waitKey(0)
cv.destroyAllWindows()
```

Input image:

Output image:

Exercise -6

Aim: Write a python program to implement the two dimensional Fourier Transformation(FT)

Methods used:

1. Matplotlib is a plotting library for the **Python** programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like wxPython,

2. numpy.fft.fft2(*a, s=None, axes=(-2, -1), norm=None*)[source]

Compute the 2-dimensional discrete Fourier Transform

This function computes the *n*-dimensional discrete Fourier Transform over any axes in an *M*-dimensional array by means of the Fast Fourier Transform (FFT). By default, the transform is computed over the last two axes of the input array, i.e., a 2-dimensional FFT.

Parameters **a**array_like

Input array, can be complex **s**sequence of ints, optional

Shape (length of each transformed axis) of the output (s[0] refers to axis 0, s[1] to axis 1, etc.).

This corresponds to n for fft(x, n). Along each axis, if the given shape is smaller than that of the input, the input is cropped. If it is larger, the input is padded with zeros. if *s* is not given, the shape of the input along the axes specified by *axes* is used.

axessequence of ints, optional

Axes over which to compute the FFT. If not given, the last two axes are used. A repeated index in *axes* means the transform over that axis is performed multiple times. A one-element sequence means that a one-dimensional FFT is performed.

norm{None, "ortho"}, optional

Returns

Out complex ndarray

The truncated or zero-padded input, transformed along the axes indicated by *axes*, or the last two axes if *axes* is not given.

Formula 2D FT

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M} \sum_{y=0}^{N} f(x,y)\, e^{-j\,2\pi (u\,x\,/\,M + v\,y\,/\,N)}$$

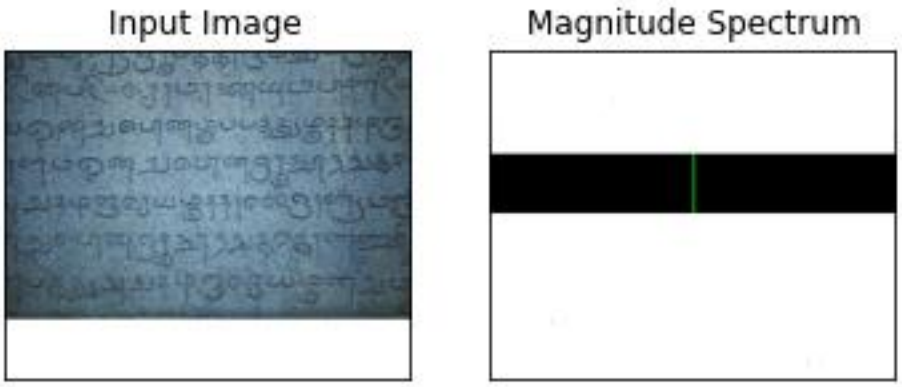
$$f(x,y) = \sum_{u=0}^{M} \sum_{v=0}^{N} F(u,v)\, e^{j\,2\pi\,(u\,x\,/\,M + v\,y\,/\,N)}$$

Source code:

```
import cv2
import numpy as np
```

```
from matplotlib import pyplot as plt
img = cv2.imread('D:\TVA\SOFTWARE\INPUT\epigraphy.png')
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

input/output:



Exercise -7

Aim: Write a python program to performance of gradient operators for edge detections

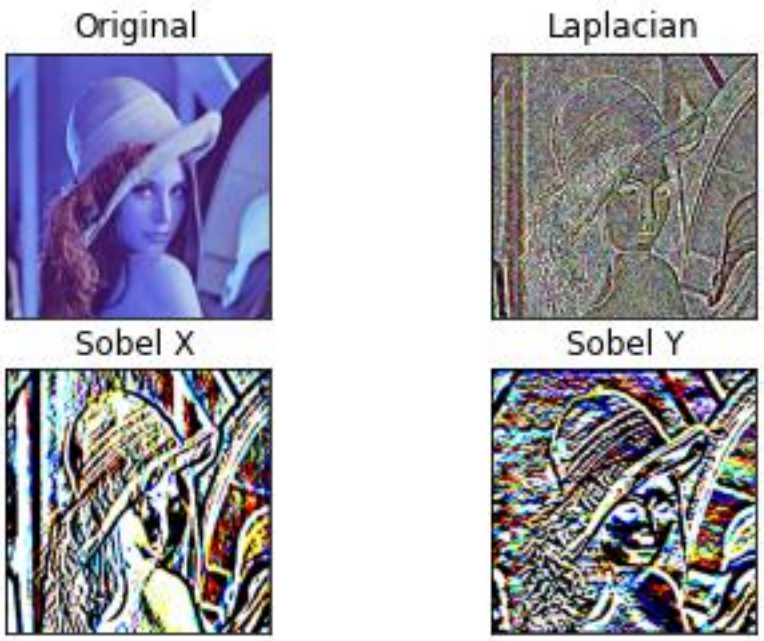
Methods used:

1. Laplacian(src_gray, dst, ddepth, kernel_size, scale, delta, BORDER_DEFAULT);
   src_gray: The input image.
   dst: Destination (output) image
   ddepth: Depth of the destination image. Since our input is *CV_8U* we define *ddepth* = *CV_16S* to avoid overflow
   kernel_size: The kernel size of the Sobel operator to be applied internally. We use 3 in this example.
   scale, *delta* and *BORDER_DEFAULT*: We leave them as default values.
2. `Sobel(src, dst, ddepth, dx, dy)`
   src − An object of the class **Mat** representing the source (input) image.
   dst − An object of the class **Mat** representing the destination (output) image.
   ddepth − An integer variable representing the depth of the image (-1)
   dx − An integer variable representing the x-derivative. (0 or 1)
   dy − An integer variable representing the y-derivative. (0 or 1)

Source code:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('D:\TVA\SOFTWARE\INPUT\lena.bmp')
laplacian = cv.Laplacian(img,cv.CV_64F)
sobelx = cv.Sobel(img,cv.CV_64F,1,0,ksize=5)
sobely = cv.Sobel(img,cv.CV_64F,0,1,ksize=5)
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])
plt.show()
```

input/output:


Original
Laplacian
Sobel X
Sobel Y

Exercise -8

Aim: Write a Java program to implement DCT image compression method.

Methods used:

Discrete Cosine Transform is used in lossy image compression because it has very strong energy compaction, i.e., its large amount of information is stored in very low frequency component of a signal and rest other frequency having very small data which can be stored by using very less number of bits (usually, at most 2 or 3 bit).
To perform DCT Transformation on an image, first we have to fetch image file information (pixel value in term of integer having range 0 – 255) which we divides in block of 8 X 8 matrix and then we apply discrete cosine transform on that block of data.

After applying discrete cosine transform, we will see that its more than 90% data will be in lower frequency component. For simplicity, we took a matrix of size 8 X 8 having all value as 255 (considering image to be completely white) and we are going to perform 2-D discrete cosine transform on that to observe the output.

DCT formula:

$$DCT(i, j) = \frac{1}{\sqrt{2N}} C(i)\, C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x, y) \cos\left[\frac{(2x + 1)i\pi}{2N}\right] \cos\left[\frac{(2y + 1)j\pi}{2N}\right]$$

$$C(x) = \frac{1}{\sqrt{2}} \text{ if x is 0, else 1 if x > 0}$$

Source code:

```
mport java.util.*;

class GFG
{
   public static int n = 8,m = 8;
   public static double pi = 3.142857;

   // Function to find discrete cosine transform and print it
   static strictfp void dctTransform(int matrix[][])
   {
      int i, j, k, l;

      // dct will store the discrete cosine transform
      double[][] dct = new double[m][n];

      double ci, cj, dct1, sum;
```

```
        for (i = 0; i < m; i++)
        {
            for (j = 0; j < n; j++)
            {
                // ci and cj depends on frequency as well as
                // number of row and columns of specified matrix
                if (i == 0)
                    ci = 1 / Math.sqrt(m);
                else
                    ci = Math.sqrt(2) / Math.sqrt(m);

                if (j == 0)
                    cj = 1 / Math.sqrt(n);
                else
                    cj = Math.sqrt(2) / Math.sqrt(n);

                // sum will temporarily store the sum of
                // cosine signals
                sum = 0;
                for (k = 0; k < m; k++)
                {
                    for (l = 0; l < n; l++)
                    {
                        dct1 = matrix[k][l] *
                               Math.cos((2 * k + 1) * i * pi / (2 * m)) *
                               Math.cos((2 * l + 1) * j * pi / (2 * n));
                        sum = sum + dct1;
                    }
                }
                dct[i][j] = ci * cj * sum;
            }
        }

        for (i = 0; i < m; i++)
        {
            for (j = 0; j < n; j++)
                System.out.printf("%f\t", dct[i][j]);
            System.out.println();
        }
    }
public static void main (String[] args)
    {
        int matrix[][] = { { 255, 255, 255, 255, 255, 255, 255, 255 },
                           { 255, 255, 255, 255, 255, 255, 255, 255 },
                           { 255, 255, 255, 255, 255, 255, 255, 255 },
```

```
                { 255, 255, 255, 255, 255, 255, 255, 255 },
                { 255, 255, 255, 255, 255, 255, 255, 255 },
                { 255, 255, 255, 255, 255, 255, 255, 255 },
                { 255, 255, 255, 255, 255, 255, 255, 255 },
                { 255, 255, 255, 255, 255, 255, 255, 255 } };
        dctTransform(matrix);
    }
}
```

Output:

```
[2039.999878   -1.168211   1.190998   -1.230618   1.289227   -1.370580   1.480267   -1.626942
-1.167731      0.000664   -0.000694   0.000698    -0.000748   0.000774    -0.000837   0.000920
1.191004       -0.000694  0.000710    -0.000710   0.000751    -0.000801   0.000864    -0.000950
-1.230645      0.000687   -0.000721   0.000744    -0.000771   0.000837    -0.000891   0.000975
1.289146       -0.000751  0.000740    -0.000767   0.000824    -0.000864   0.000946    -0.001026
-1.370624      0.000744   -0.000820   0.000834    -0.000858   0.000898    -0.000998   0.001093
1.480278       -0.000856  0.000870    -0.000895   0.000944    -0.001000   0.001080    -0.001177
-1.626932      0.000933   -0.000940   0.000975    -0.001024   0.001089    -0          -0.001024   ]
```

Exercise -9

Aim: Write a python program to implement the image segmentation based on color.

Methods used:

K-means clustering:

cv2.kmeans(Z,K, criteria,10,cv2.KMEANS_RANDOM_CENTERS)

1. **samples** : It should be of **np.float32** data type, and each feature should be put in a single column.
2. **nclusters(K)** : Number of clusters required at end
3. **criteria** : It is the iteration termination criteria. When this criteria is satisfied, algorithm iteration stops. Actually, it should be a tuple of 3 parameters. They are `( type, max_iter, epsilon )`:
    a. type of termination criteria. It has 3 flags as below:
        - **cv.TERM_CRITERIA_EPS** - stop the algorithm iteration if specified accuracy, *epsilon*, is reached.
        - **cv.TERM_CRITERIA_MAX_ITER** - stop the algorithm after the specified number of iterations, *max_iter*.
        - **cv.TERM_CRITERIA_EPS +** **cv.TERM_CRITERIA_MAX_ITER** - stop the iteration when any of the above condition is met.
    b. max_iter - An integer specifying maximum number of iterations.
    c. epsilon - Required accuracy
4. **attempts** : Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness.
5. **flags** : This flag is used to specify how initial centers are taken. Normally two flags are used for this : **cv.KMEANS_PP_CENTERS** and **cv.KMEANS_RANDOM_CENTERS**.

cv2.threshold(): First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

cv2.THRESH_BINARY
cv2.THRESH_BINARY_INV
cv2.THRESH_TRUNC
cv2.THRESH_TOZERO
cv2.THRESH_TOZERO_INV

Source code:

```
import numpy as np
```

```
import cv2
img = cv2.imread('D:\TVA\SOFTWARE\INPUT\watermelon.jpg')
Z = img.reshape((-1,3))
# convert to np.float32
Z = np.float32(Z)
# define criteria, number of clusters(K) and apply kmeans()
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
K = 4
# ret,label,center=cv2.kmeans(Z,K, criteria,10,cv2.KMEANS_RANDOM_CENTERS)
_,label,center = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)
# Now convert back into uint8, and make original image
center = np.uint8(center)
res = center[label.flatten()]
res2 = res.reshape((img.shape))
gray = cv2.cvtColor(res2,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
#segmentation
gray = cv2.cvtColor(res2,cv2.COLOR_BGR2GRAY)
ret, threshseg = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
cv2.imwrite('D:\TVA\SOFTWARE\INPUT\img_CV2_95.jpg',threshseg)
cv2.imwrite('D:\TVA\SOFTWARE\INPUT\img_CV2_94.jpg',res2)
cv2.imshow('threshseg',threshseg)
cv2.imshow('thresh',thresh)
cv2.imshow('res2',res2)
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

sample input/output

Exercise-10

Aim: Write a python program to implement the erosion and dilation image morphological operations.

Methods used:

Morphological operations are a set of operations that process images based on shapes. They apply a structuring element to an input image and generate an output image. The most basic morphological operations are two: **Erosion and Dilation**

Basics of Erosion:

- Erodes away the boundaries of foreground object
- Used to diminish the features of an image.

Working of erosion:

1. A kernel(a matrix of odd size(3,5,7) is convolved with the image.
2. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).
3. Thus all the pixels near boundary will be discarded depending upon the size of kernel.
4. So the thickness or size of the foreground object decreases or simply white region decreases in the image.

Basics of dilation:

- Increases the object area
- Used to accentuate features

Working of dilation:

- A kernel(a matrix of odd size(3,5,7) is convolved with the image
- A pixel element in the original image is '1' if atleast one pixel under the kernel is '1'.
- It increases the white region in the image or size of foreground object increases

Source code:

```
# Python program to demonstrate erosion and
# dilation of images.
import cv2
import numpy as np

# Reading the input image
img = cv2.imread('input.png', 0)

# Taking a matrix of size 5 as the kernel
kernel = np.ones((5,5), np.uint8)

# The first parameter is the original image,
# kernel is the matrix with which image is
# convolved and third parameter is the number
```

```
# of iterations, which will determine how much
# you want to erode/dilate a given image.
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
```

Input image:

Output: Dilation

Output: Erosion