Artificial Intelligence and Computer Vision

Laboratory 4

Report Sheet

Jakub Bujak 249120

Task description:

1. **Perform an image histogram stretching for quantized images:**
    a. Take low and high frequency images, perform quantization with levels of intensity 32, 64 and 128
    b. Perform histogram equalization for low and high frequency images
    c. Describe conclusions regarding histogram equalization for low and high frequency images
2. **Perform:**
    a. Image thresholding (binary and normal) with chosen value
    b. Finding the negative of image

    For Lena Forsen's image. Generate histogram, explain its form.

3. **Perform following steps**:
    a. Create Lena 's DFT
    b. Using DFT result generate inversed DFT image
    c. Compare original Lena and Lena after FT. Note the differences.

Report:

First we import the library packages to PyCharm: Numpy, OpenCV and the MatPlotLib for histogram plotting.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
```

Defining a function to display the images:

```
# Function displaying the image
def print_img(temp, name):
    cv.imshow(name, temp)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

It is a simple function using the OpenCV library using the function *imshow* taking the image from path file

Defining a function to plot the histograms of images:

```
# Function making the histogram of image
def make_histogram(filename, file):
    cv.imshow(filename, file)  # show current image
    plt.hist(file.ravel(), 256, [0, 256])
    plt.title('The histogram of ' + filename, fontdict=None, loc='center', pad=None)  # title for histogram
    plt.show()  # plotting histogram
    cv.waitKey(0)
    cv.destroyAllWindows()
```

A function combining the OpenCV and MatPlotLib libraries. Again we use the imshow function using the previously defined image printing. Then we use the MatPlotLib functions, to plot the histogram, set the ravel, and title the image to be displayed.

Defining a function to equalize histograms:

```
# Function presenting the histogram of image
def equalize_histogram(filename, file):
    img_output = cv.cvtColor(file, cv.COLOR_BGR2LAB)  # black and white colours of the histogram
    eql = cv.equalizeHist(img_output, None)
    cv.imshow('Black&white histogram of ' + filename, img_output)

    make_histogram('Equalized histogram of ' + filename, eql)  # equalized histogram
```

The OpenCV library lets us do that – through it, one can set the color array for equalizing the histogram. After that we use the library function for equalizing the histogram, and displaying it, and finally we can use our previously defined function for making the histogram.

Defining a function to quantize images:

```
# Function responsible for quantization of image
def quantize_image(filename, level=32):
    quant_out = np.zeros((filename.shape[0], filename.shape[1]), dtype='uint8')
    max = np.max(filename)
    min = np.min(filename)

    difference = (max - min) // level
    temp = 0

    for x in range(filename.shape[0]):
        for y in range(filename.shape[1]):
            quant_out[x, y] = filename[x, y]
            while True:
                if quant_out[x, y] <= min + difference or temp >= level:
                    break
                else:
                    quant_out[x, y] = quant_out[x, y] - difference
                    temp += 1
            quant_out[x, y] = temp * difference + min + difference // 2
            temp = 0
    return quant_out
```

Firstly we create, using numpy, a zeros matrix, in which we'll put our images. Through numpy we can also define the max & min levels to later display the difference in steps that the program should take to find next step of quantization levels in *for* loop.

Defining a function to find thresholding:

```
def thresholding(img, type, title):
    ret, thresh = cv.threshold(img, 127, 255, type)
    make_histogram(title, thresh)
```

Simple thresholding - for every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. The first argument is the source image, which should be a **grayscale image**. The second argument is the threshold value which is used to classify the pixel values. The third argument is the maximum value which is assigned to pixel values exceeding the threshold.

Defining a function to perform DFT:

```
def DFT(img):
    img = cv.imread('lena.png', 0)
    img_float32 = np.float32(img)
    make_dft = cv.dft(img_float32, flags=cv.DFT_COMPLEX_OUTPUT)
    shift_dft = np.fft.fftshift(make_dft)  # Shift of the zero frequency component to the centre of spectrum
    spec_mag = 20 * np.log(cv.magnitude(shift_dft[:, :, 0], shift_dft[:, :, 1]) + 1)

    plt.subplot(121), plt.imshow(img, cmap='gray')
    plt.title('Image'), plt.xticks([]), plt.yticks([])

    plt.subplot(122), plt.imshow(spec_mag, cmap='gray')
    plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
    plt.show()
```

The function computes a one-dimensional discrete Fourier Transfrom. This function computes the one-dimensional n-point discrete Fourier Transform (DFT) with the efficient Fast Fourier Transform (FFT) algorithm. FFT refers to a way the DFT can be calculated efficiently, by using symmetries in the calculated terms. The symmetry is highest when n is a power of 2, and the transform is therefore most efficient for these sizes.

Defining a function to perform inverse DFT:

```
def inverse_DFT(img):
    rows, cols = img.shape
    crow, ccol = rows / 2, cols / 2

    img_float32 = np.float32(img)
    make_dft = cv.dft(img_float32, flags=cv.DFT_COMPLEX_OUTPUT)
    shift_dft = np.fft.fftshift(make_dft)  # Shift of the zero frequency component to the centre of spectrum

    tmp = np.zeros((rows, cols, 2), np.uint8)
    tmp[crow - 30:crow + 30, ccol - 30:ccol + 30] = 1

    phshift = shift_dft * tmp
    inv_phshift = np.fft.ifftshift(phshift)
    img_back = cv.idft(inv_phshift)
    img_back = cv.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    plt.subplot(121), plt.imshow(img, cmap='gray')
    plt.title('Image'), plt.xticks([]), plt.yticks([])
    plt.subplot(122), plt.imshow(img_back, cmap='gray')
    plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
    plt.show()
```

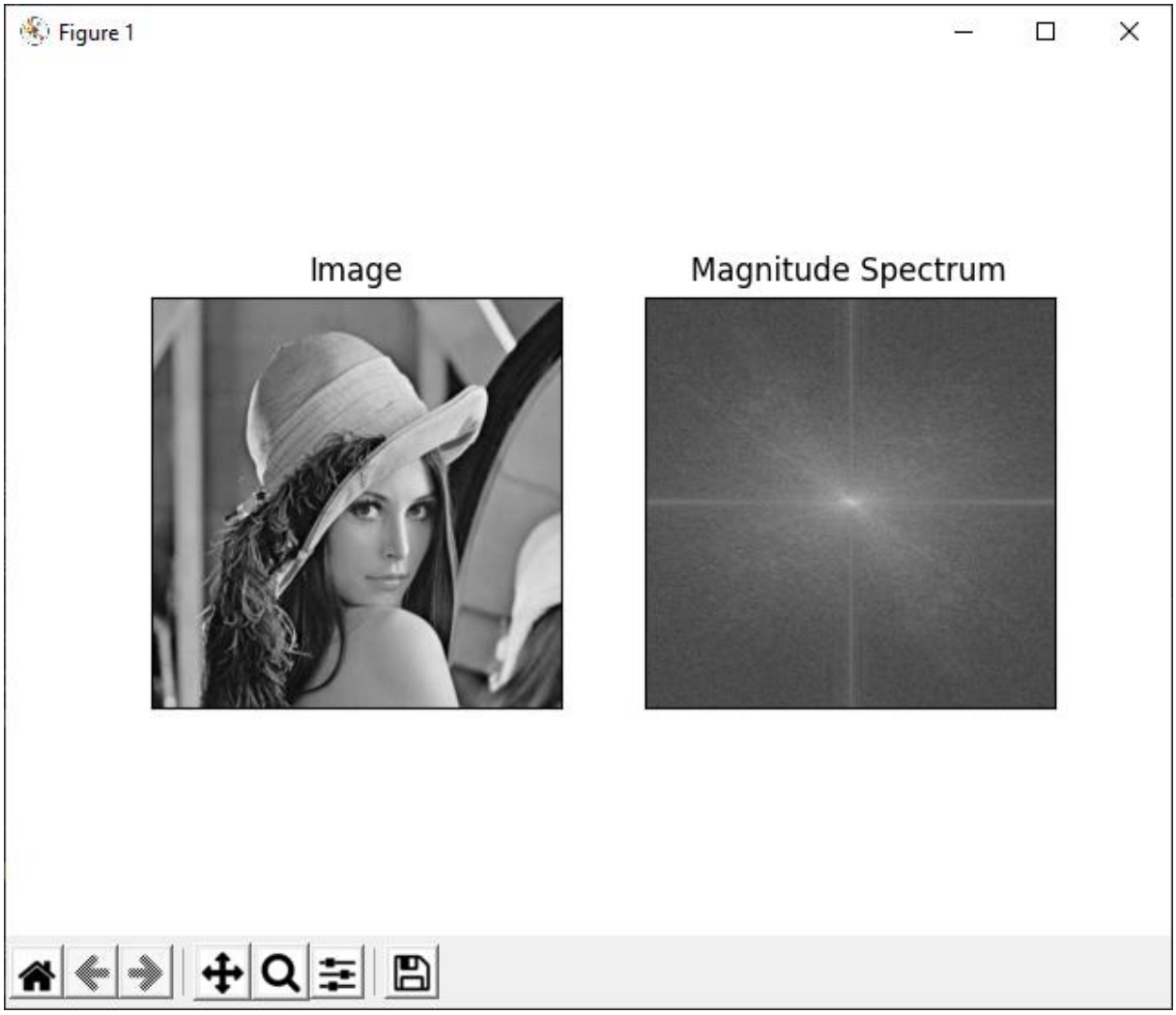
The algorithm behaves similarly to the normal DFT computation, with the addition of rotation of the temporary zeros matrix.

Defining the main program function:

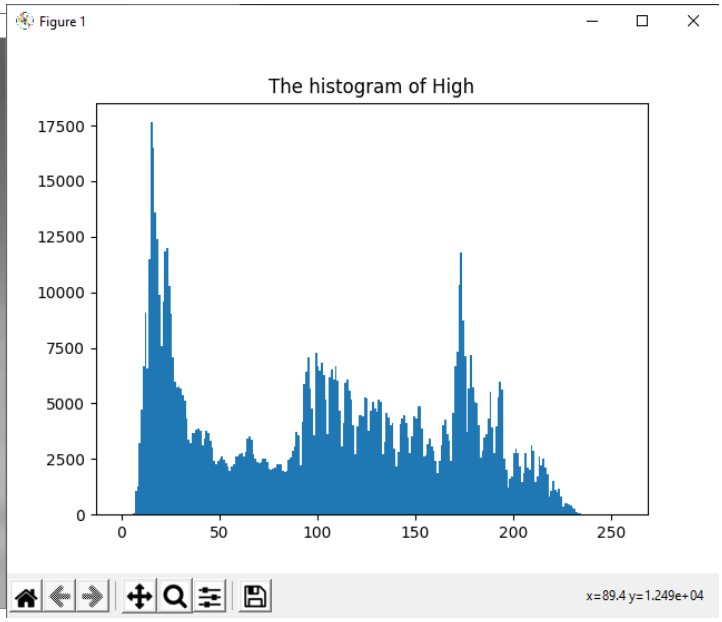
```
def main():
    # Images paths definitions
    path_low = 'img_low.jpg'
    path_high = 'img_high.jpg'
    path_lena = 'lena.png'

    # LOADING UP IMAGES
    img_low = cv.imread(path_low, 0)
    img_high = cv.imread(path_high, 0)
    img_lena = cv.imread(path_lena, 1)
    img_lena_nocol = cv.imread(path_lena, 0)

    DFT(img_lena_nocol) # Run in scientific mode!!!

    make_histogram('High', img_high)
    make_histogram('Low', img_low)

    thresholding(img_lena, cv.THRESH_BINARY, 'Binary')
    thresholding(img_lena, cv.THRESH_BINARY_INV, 'Inverted Binary')


if __name__ == "__main__":
    main()
```

Simple initialization of previously defined functions.

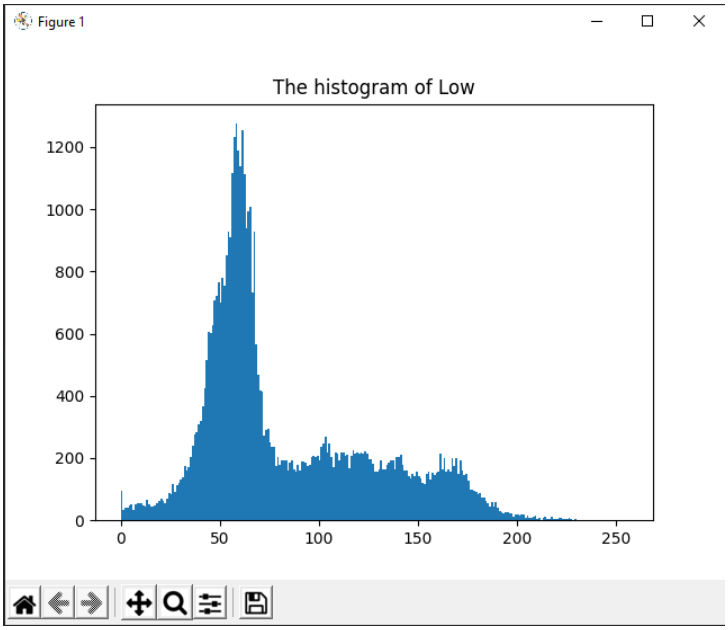
Results of used functions:



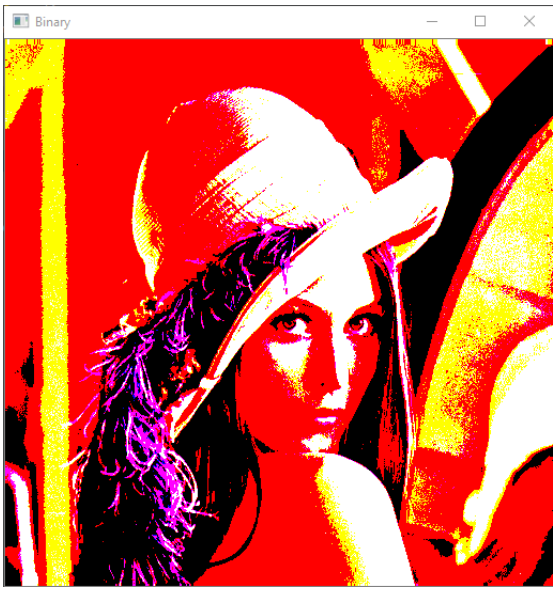
In the first figure we can see the magnitude spectrum of grayscale image of Lena Forsen. The spectrum itself kind of resembles an asterix.

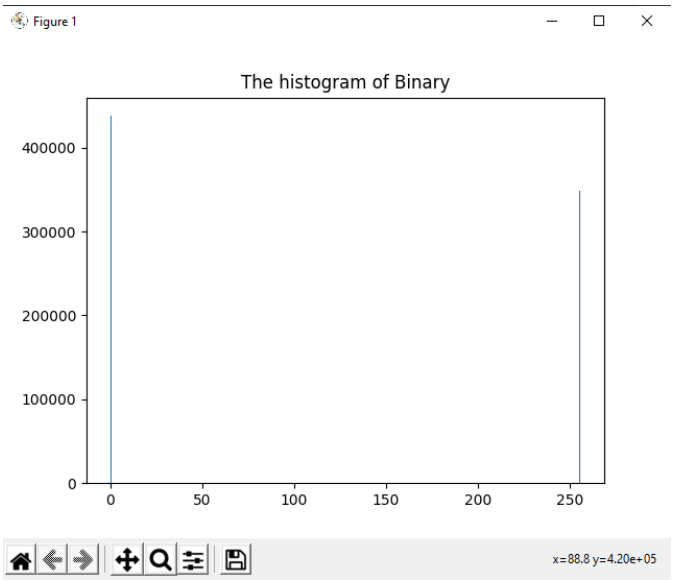


Here we can see the grayscale image of high frequency and its histogram. The histogram's ravel seems to be shifted more to the left, with the peak value, from the centre.

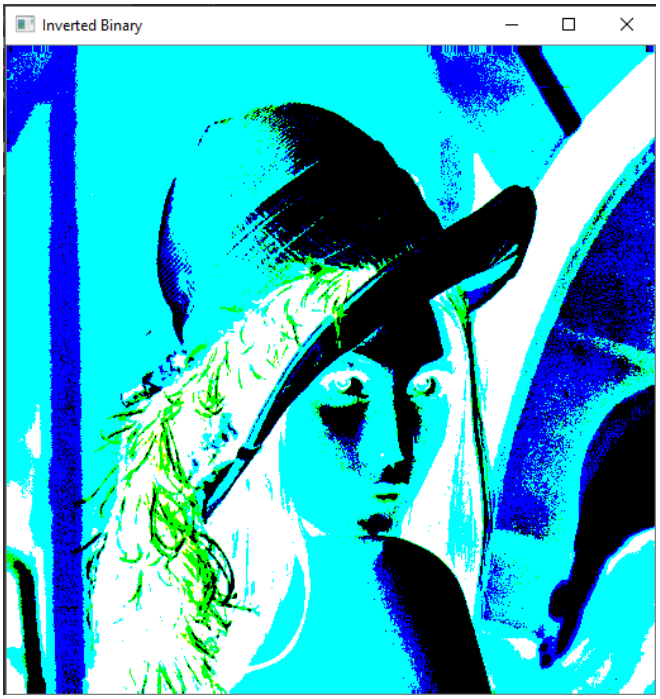


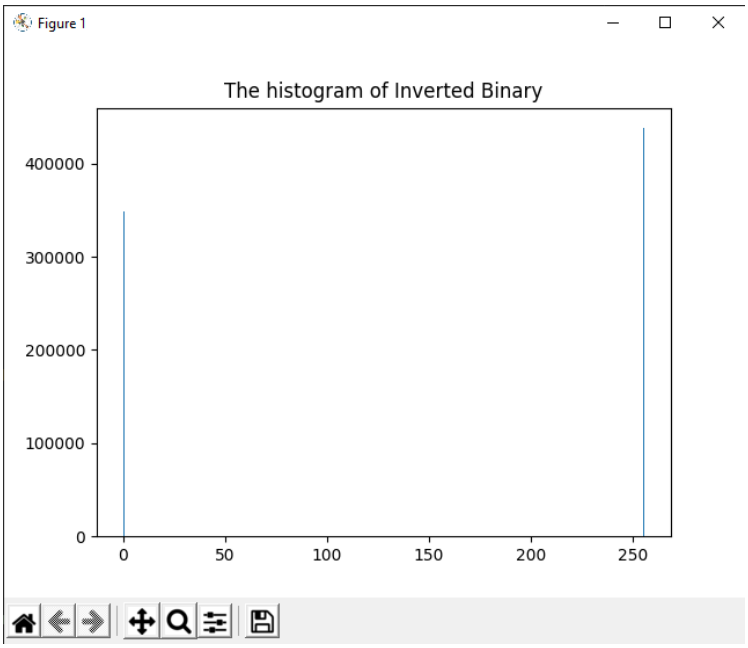
For the low frequency image, the ravel is also shifted to the left, but with much lower values of the Y axis.





Here is the presentation of thresholding – when we put it on the original image through the DFT, the images goes into the reds of RGB array.





The inverted binary thresholding results in the RGB going into blues of the array, and giving the “negative” effect on the image.

When comparing the histograms, we can see that the reds peak on the left of the centre of the histogram, while the negative take the right of the X axis values.