Dhirubhai Ambani Institute of Information and Communication Technology

# Assignment 2

Ganesh Iyer
201311019
Developed using: Python(Using SimpleCV library)

February 4, 2014

## 1 Bitplanes

### 1.1 Bitplanes slicing

In this problem we implemented a function mybitplane() that extracts all 8 bit planes of any input grayscale image I. As we can observe the most significant bits seem to have

Figure 1.1: Original Image

more information pertaining to the image

### 1.2 Watermarking

In this exercise we use the binary image `daiict.bmp` that was provided as a watermark and replace the $i^{th}$ bit plane of the image `lena.jpg` and reconstruct the gray scale image

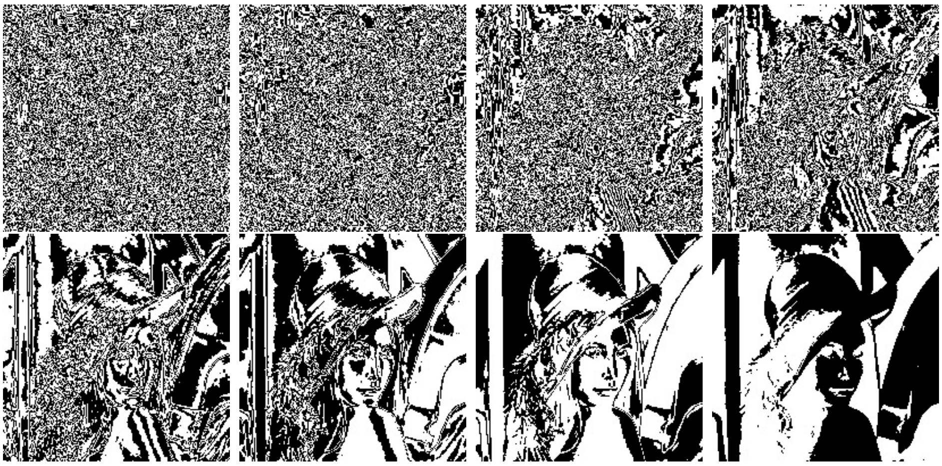

Figure 1.2: Bitplanes, Starting from Least significant bit(TopLeft) to Most significant bit in clockwise direction


Figure 1.3: Watermark

Figure 1.4: WaterMark applied on different layers of the original image. Starting from level 0(top-left) to level 7 in a clock-wise direction

$J_i$ for $1 \leq i \leq 8$.

## 2 Histogram equalization

In this exercise we wrote a function `myhisteq()` that applies histogram equalization on any input grayscale image. We use the transfomration function
$T(r_k) = (L-1)\sum_{j=0}^{k} p_R(r_j)$

Figure 2.1: Clockwise starting from top left; Original Image; Image output of our histogram equalization; Image output of built-in histogram equalization

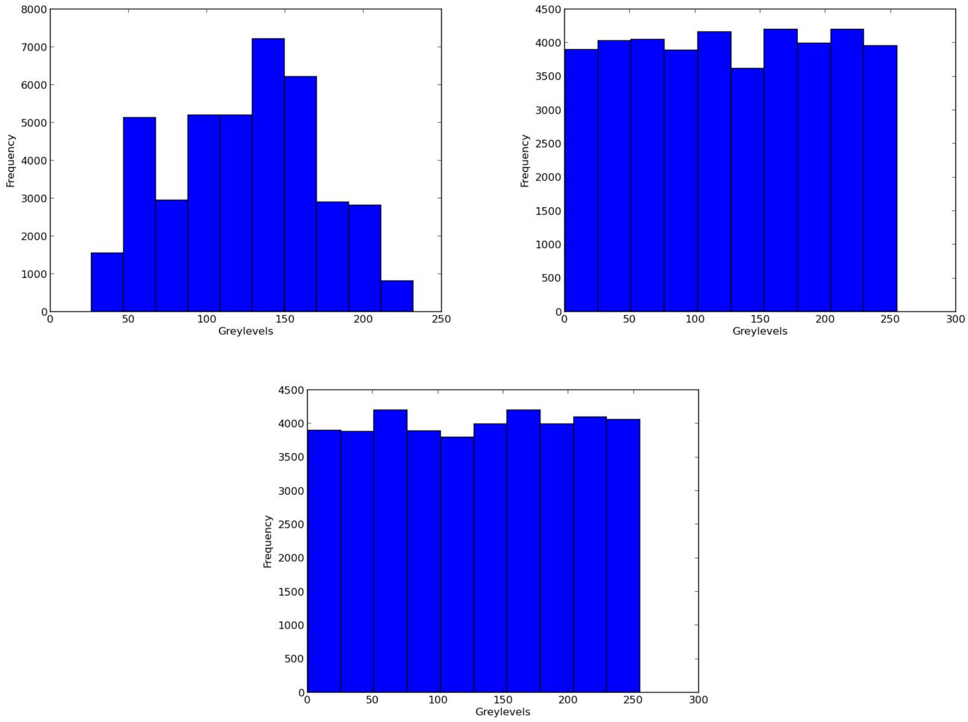


Figure 2.2: Clockwise starting from top left; Original Histogram; Output of our histogram equalization; Output of built-in histogram equalization

# 3 Convolution

We implemeneted a function `my2Dconv()` that takes an Image($I$) and kernel($k$) and output $y = I * k$. We use the following kernel: $k = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$

Figure 3.1: Outputs using kernel(k). Left: Using my2DConv function. Right: Using built-in function

# 4 Deriving popular convolution masks

Given a $3X3$ neighbourhood($F$) of an image

$F = \begin{bmatrix} f(-1,-1) & f(-1,0) & f(-1,1) \\ f(0,-1) & f(0,0) & f(0,1) \\ f(1,-1) & f(1,0) & f(1,1) \end{bmatrix}$

The equation of the actual plane

$g(x, y) = a + bx + cy$. Now we have $F = G + \eta$

## 4.1 Finding $\beta$

For this we use the `lstsq` function built-in Numpy for finding least square solution to a linear matrix equation. This function solves the equation $F = X\beta$ by computing $\beta$ such that it minimizes the Euclidean 2-norm $\|F - X\beta\|^2$.

We get the following values of a,b and c for the first $3X3$ neighbourhood.

$\beta = \begin{bmatrix} 161.0 & -0.5 & -0.5 \end{bmatrix}$

## 4.2 Finding convolution masks

We find the pseudo-inverse of X using the built-in function `pinv`. We get the following matrices for our convolution masks

$M1 = \begin{bmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{bmatrix}$

$M2 = \begin{bmatrix} -0.2 & -0.2 & -0.2 \\ 0 & 0 & 0 \\ 0.2 & 0.2 & 0.2 \end{bmatrix}$

$$M3 = \begin{bmatrix} -0.2 & 0 & 0.2 \\ -0.2 & 0 & 0.2 \\ -0.2 & 0 & 0.2 \end{bmatrix}$$

### 4.3 Finding convolution masks using error weights

We multiply the noise weights with original masks to obtain new masks.

$$M_{w1} = \begin{bmatrix} 0.4 & 0.9 & 0.4 \\ 0.4 & 0.9 & 0.4 \\ 0.4 & 0.9 & 0.4 \end{bmatrix}$$

$$M_{w2} = \begin{bmatrix} -0.67 & -1.33 & -0.67 \\ 0 & 0 & 0 \\ 0.67 & 1.33 & 0.67 \end{bmatrix}$$

$$M_{w3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

### 4.4 Results

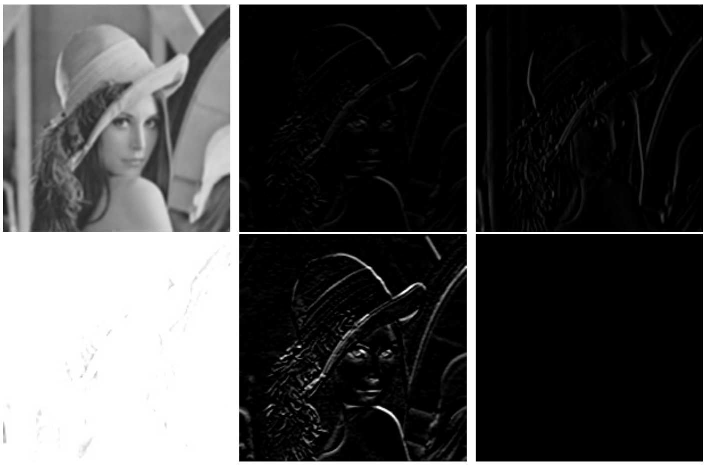

Figure 4.1: Starting from top-left, results of applying the convolution masks dervied($M1, M2, M3, M_{w1}, M_{w2}, M_{w3}$) in clockwise direction.

We observe that these masks work like the Sobel masks for edge detection. Results of $M2, M3$ are similar to Sobel mask in X and Y orientation.