



National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

EE 433: Digital Image Processing

Name: Amal Saqib

CMS: 282496

Class: BSCS 9C

Lab 4: Primitive Transformations

Date: 4th October 2021

Time: 2.00Pm to 5.00Pm

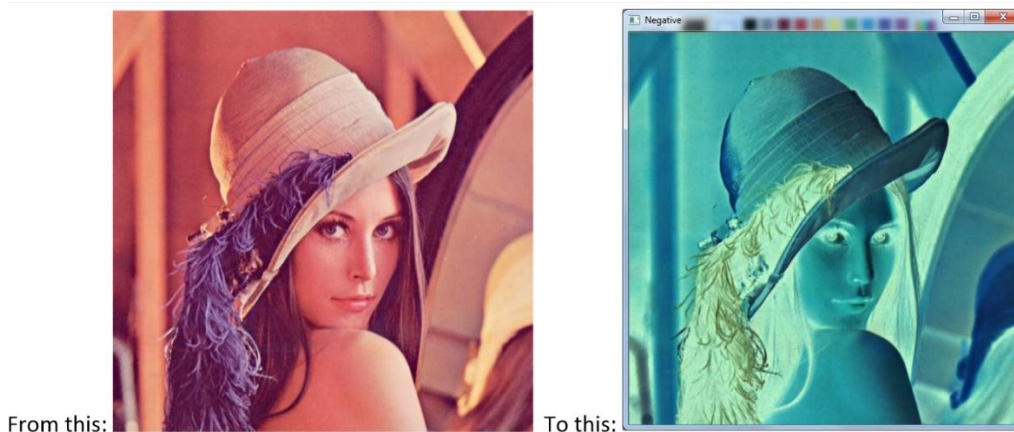
Instructor: Dr. Imran Malik



Primitive Transformations

Task #1: Image Negatives

Implement a function for displaying negative of an input image. Note that the function must handle binary, grayscale, and RGB images. Example of RGB negative:



CODE

```
import os, sys
from PIL import Image

# returns the array that is used to make the histogram
def negatives(img):
    # load the pixels of the image
    pix = img.load()

    # get width and height of the input image
    width, height = img.size

    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # if pixel is greyscale
            if type(pix[x,y]) == int:
                # apply the formula
                S = 255 - pix[x, y]
                # replace pixel
                pix[x, y] = S

            # if pixel is RGB
            else:
                # extract RGB from the pixel
                r, g, b = pix[x, y]
```



```
# apply the formula
S1 = 256 - 1 - r
S2 = 256 - 1 - g
S3 = 256 - 1 - b
# replace the pixel with its negative
pix[x, y] = (S1, S2, S3)
return img
```

```
for infile in sys.argv[1:]:
    try:
        # split the name of the file
        f, e = os.path.splitext(infile)
        # open image
        img = Image.open(infile)

        img = negatives(img)

        # save the new image
        img.save(f + "_negative" + e)

    except IOError:
        print("Error")
```

```
1  import os, sys
2  from PIL import Image
3
4  # returns the array that is used to make the histogram
5  def negatives(img):
6      # load the pixels of the image
7      pix = img.load()
8
9      # get width and height of the input image
10     width, height = img.size
11
12     # iterate through all the pixels
13     for x in range(width):
14         for y in range(height):
15             # if pixel is greyscale
16             if type(pix[x,y]) == int:
17                 # apply the formula
18                 S = 255 - pix[x, y]
19                 # replace pixel
20                 pix[x, y] = S
21
```



```
21
22     # if pixel is RGB
23     else:
24         # extract RGB from the pixel
25         r, g, b = pix[x, y]
26         # apply the formula
27         S1 = 256 - 1 - r
28         S2 = 256 - 1 - g
29         S3 = 256 - 1 - b
30         # replace the pixel with its negative
31         pix[x, y] = (S1, S2, S3)
32     return img
33
34 for infile in sys.argv[1:]:
35     try:
36         # split the name of the file
37         f, e = os.path.splitext(infile)
38         # open image
39         img = Image.open(infile)
40
41         img = negatives(img)
42
43         # save the new image
44         img.save(f + "_negative" + e)
45
46     except IOError:
47         print("Error")
48
```

RGB -

Running in cmd

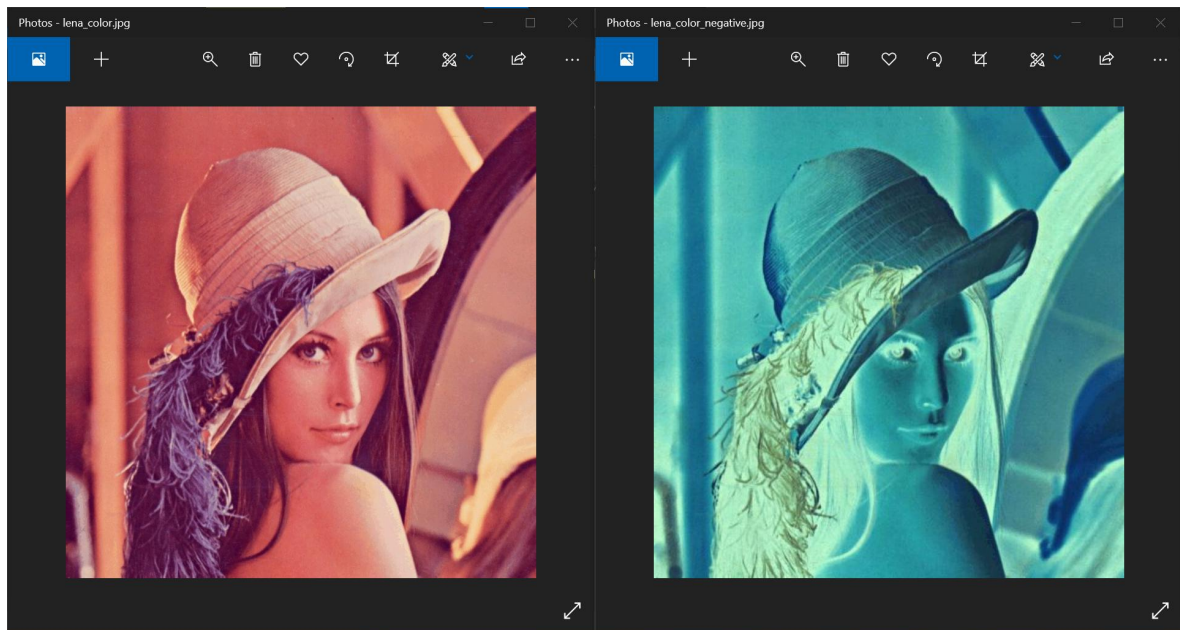
```
C:\Windows\System32\cmd.exe
D:\Folders\SEECs\Semester 5\DIP\Lab\Lab 4>python Task1.py lena_color.jpg
D:\Folders\SEECs\Semester 5\DIP\Lab\Lab 4>
```

OUTPUT



National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science



Greyscale - Using code from previous labs, I saved a greyscale copy of lena-color.jpg.

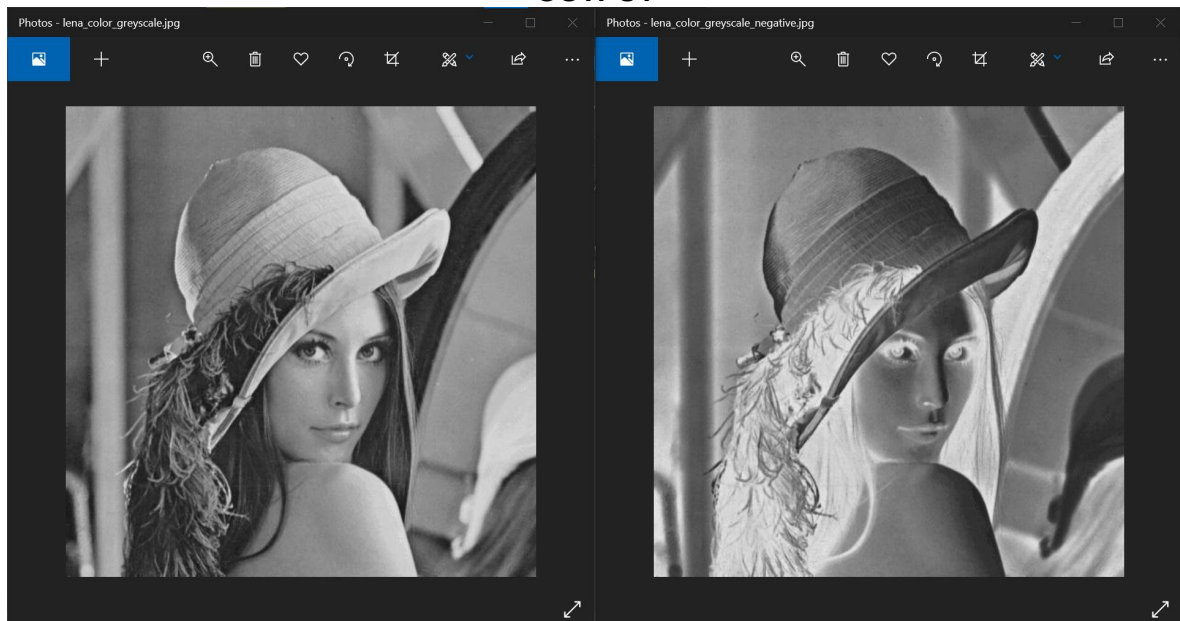
Running in cmd

```
C:\Windows\System32\cmd.exe

D:\Folders\SEECs\Semester 5\DIP\Lab\Lab 4>python Task1.py lena_color_greyscale.jpg

D:\Folders\SEECs\Semester 5\DIP\Lab\Lab 4>_
```

OUTPUT





Binarized - Using code from previous labs, I saved a binarized copy of lena-color.jpg.

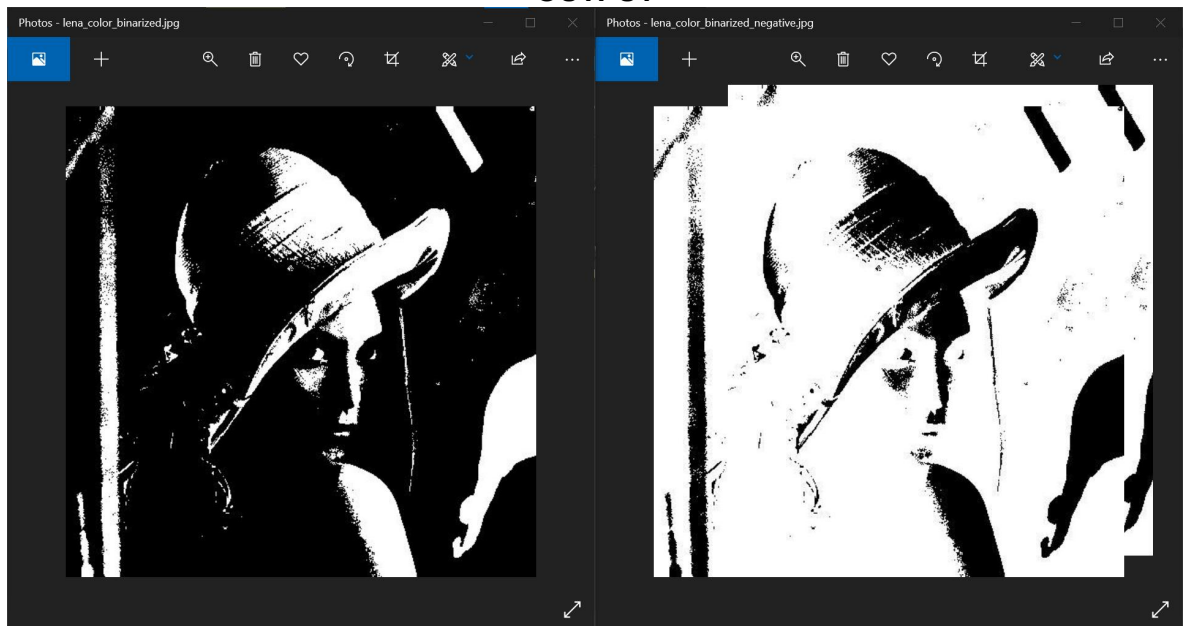
Running in cmd

```
C:\Windows\System32\cmd.exe

D:\Folders\SEECs\Semester 5\DIP\Lab\Lab 4>python Task1.py lena_color_binarized.jpg

D:\Folders\SEECs\Semester 5\DIP\Lab\Lab 4>
```

OUTPUT



Task #2: Image Gradients

The horizontal gradient image can be used to detect vertical edges in an image. Implement a function for displaying the horizontal gradient of a grayscale image. The gradient can be approximated by forward differences:

$$I_{\text{gradient}}(x, y) = I(x + 1, y) - I(x, y)$$

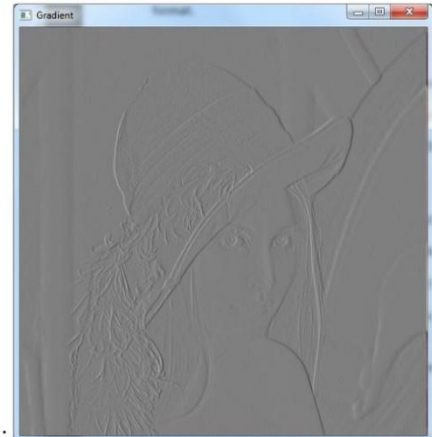
Note that the gradient values can be both positive and negative! So you need to find a way to display the gradient values in the range: 0, 1, 2, ..., 255. The following link can be helpful here:

<https://www.cis.rit.edu/people/faculty/rhody/EdgeDetection.htm>

The resulting image should look something like this:



From this:



To this:

CODE

```
import os
from PIL import Image

# returns the array that is used to make the histogram
def gradient(img):
    # load the pixels of the image
    pix = img.load()

    # get width and height of the input image
    width, height = img.size

    # iterate through all the pixels
    for x in range(width - 1):
        for y in range(height):
            # apply the formula
            pix[x, y] = pix[x + 1, y] - pix[x, y]
            # scale it
            pix[x, y] = int((pix[x, y] + 255) / 510 * 255)

    # change last row of pixels to black
    for y in range(height):
        pix[width - 1, y] = 0

    return img

infile = "lena_color_greyscale.jpg"
try:
    # split the name of the file
    f, e = os.path.splitext(infile)
    # open image
    img = Image.open(infile)

    img = gradient(img)

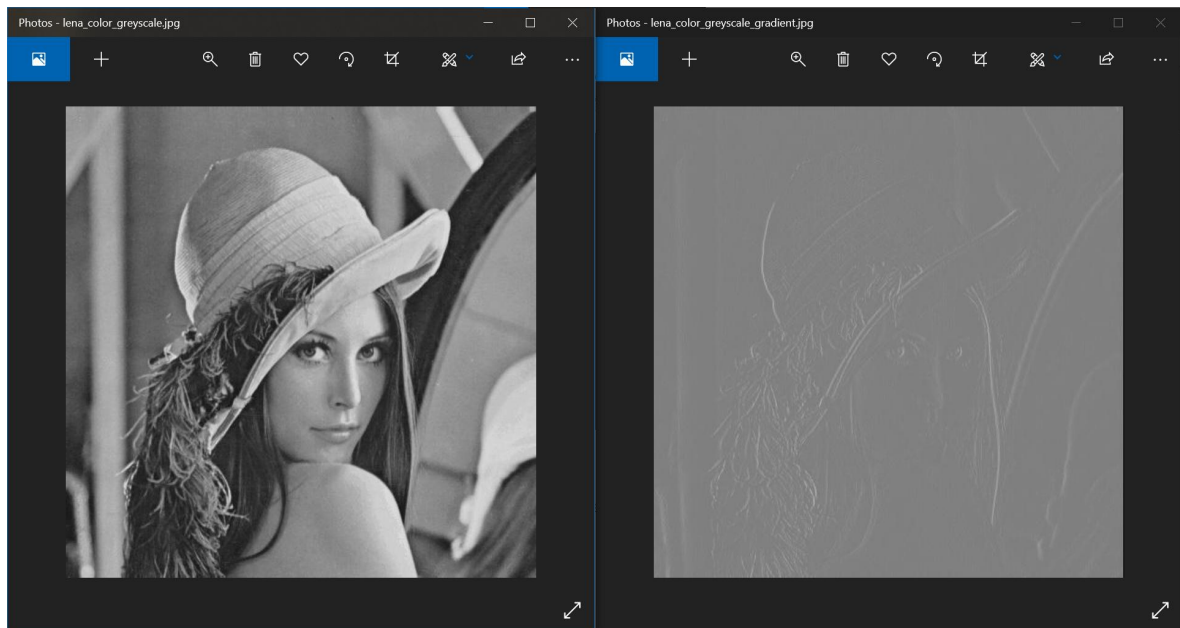
    # save the new image
    img.save(f + "_gradient" + e)
```



```
except IOError:  
    print("Error")
```

```
1  import os  
2  from PIL import Image  
3  
4  # returns the array that is used to make the histogram  
5  def gradient(img):  
6      # load the pixels of the image  
7      pix = img.load()  
8  
9      # get width and height of the input image  
10     width, height = img.size  
11  
12     # iterate through all the pixels  
13     for x in range(width - 1):  
14         for y in range(height):  
15             # apply the formula  
16             pix[x, y] = pix[x + 1, y] - pix[x, y]  
17             # scale it  
18             pix[x, y] = int((pix[x, y] + 255) / 510 * 255)  
19  
20     # change last row of pixels to black  
21     for y in range(height):  
22         pix[width - 1, y] = 0  
23  
24     return img  
25  
26  infile = "lena_color_greyscale.jpg"  
27  try:  
28      # split the name of the file  
29      f, e = os.path.splitext(infile)  
30      # open image  
31      img = Image.open(infile)  
32  
33      img = gradient(img)  
34  
35      # save the new image  
36      img.save(f + "_gradient" + e)  
37  
38  except IOError:  
39      print("Error")  
40
```

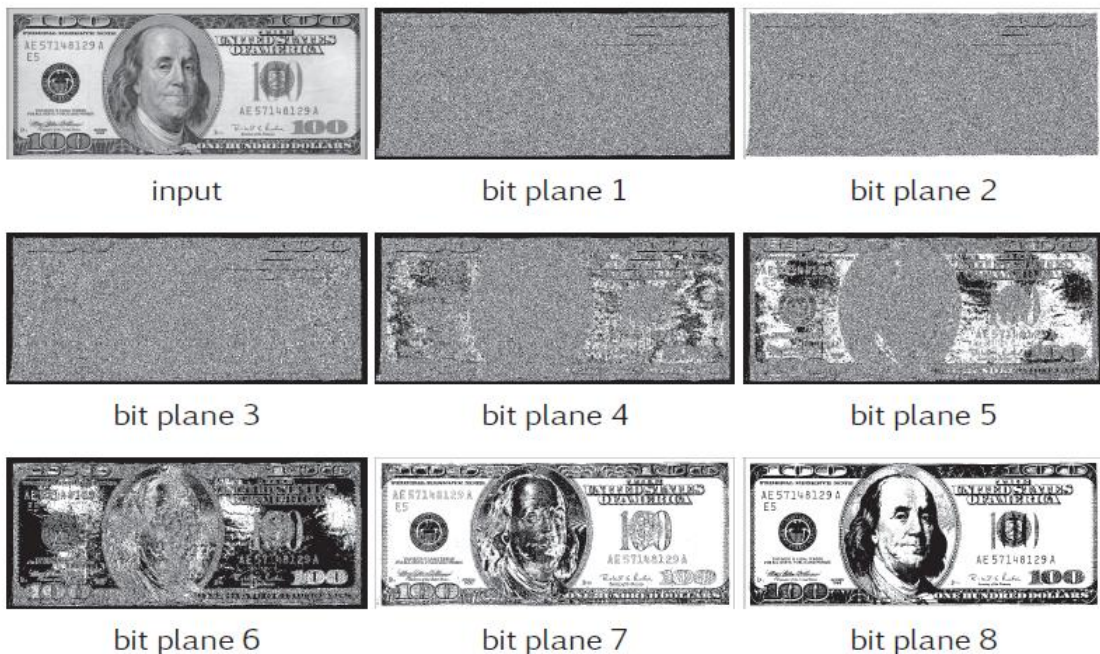
OUTPUT



Task #3: Bit Plane Slicing

Perform bit slicing of an 8 bit greyscale image as discussed in the lecture. Start from the least significant bit and move towards the most significant bit. You will get eight binary images of the input image as demonstrated below.

Bit-plane slicing. demo



CODE

```
import os  
from PIL import Image
```



```
# convert decimal value to 8-bit binary
def toBinary(pix):
    pix = bin(pix)
    pix = pix[2:]
    while len(pix) < 8:
        pix = '0' + pix

    return pix

# returns the bit plane slice
def slicing(img, i):
    # load the pixels of the image
    pix = img.load()

    # get width and height of the input image
    width, height = img.size

    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # find the binary equivalent of the pixel's intensity
            binary = toBinary(pix[x, y])

            # if bit is 1, make the pixel white (intensity 255)
            if int(binary[7 - i]) == 1:
                pix[x, y] = 255
            # else make the pixel black
            else:
                pix[x, y] = 0

    # return the image for that bit plane
    return img

infile = "bitplane.jpg"
try:
    # split the name of the file
    f, e = os.path.splitext(infile)

    # 8 bit slices
    for i in range(8):
        # open image
        img = Image.open(infile).convert('L')

        # call bit plane slicing for that bit
        img = slicing(img, i)

        # save the images
        img.save(f + "_" + str(i + 1) + ".jpg")

except IOError:
    print("Error")
```

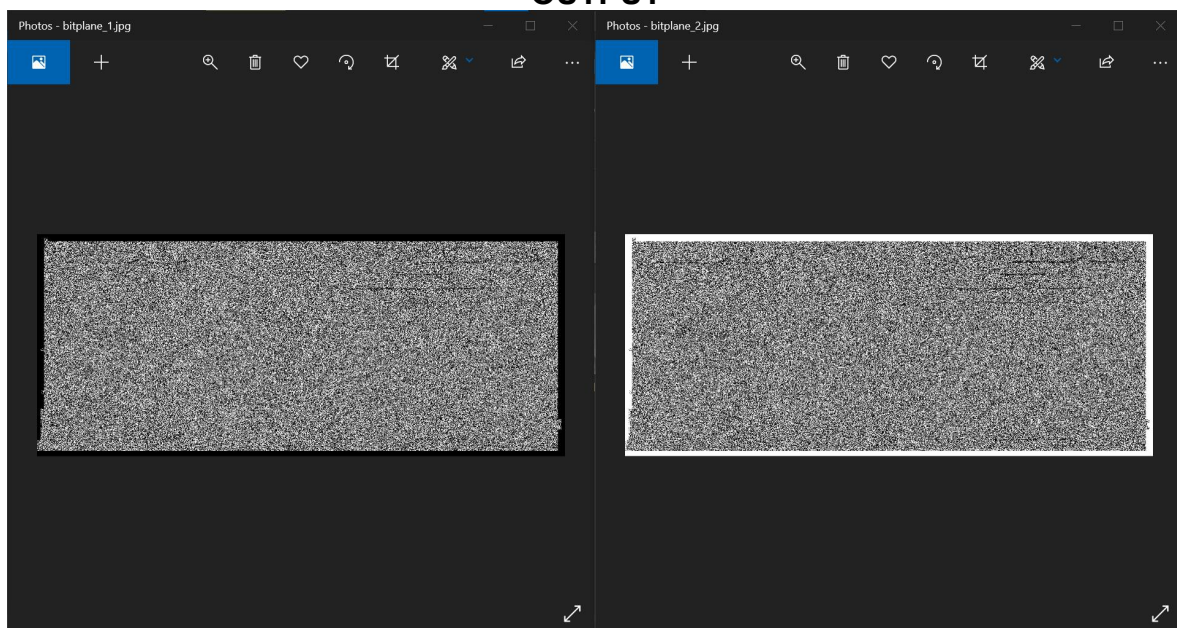


```
1  import os
2  from PIL import Image
3
4  # convert decimal value to 8-bit binary
5  def toBinary(pix):
6      pix = bin(pix)
7      pix = pix[2:]
8      while len(pix) < 8:
9          pix = '0' + pix
10
11     return pix
12
13 # returns the bit plane slice
14 def slicing(img, i):
15     # load the pixels of the image
16     pix = img.load()
17
18     # get width and height of the input image
19     width, height = img.size
20
21     # iterate through all the pixels
22     for x in range(width):
23         for y in range(height):
24             # find the binary equivalent of the pixel's intensity
25             binary = toBinary(pix[x, y])
26
```




```
27         # if bit is 1, make the pixel white (intensity 255)
28         if int(binary[7 - i]) == 1:
29             pix[x, y] = 255
30         # else make the pixel black
31         else:
32             pix[x, y] = 0
33
34     # return the image for that bit plane
35     return img
36
37     infile = "bitplane.jpg"
38     try:
39         # split the name of the file
40         f, e = os.path.splitext(infile)
41
42         # 8 bit slices
43         for i in range(8):
44             # open image
45             img = Image.open(infile).convert('L')
46
47             # call bit plane slicing for that bit
48             img = slicing(img, i)
49
50             # save the images
51             img.save(f + "_" + str(i + 1) + ".jpg")
52
53     except IOError:
54         print("Error")
```

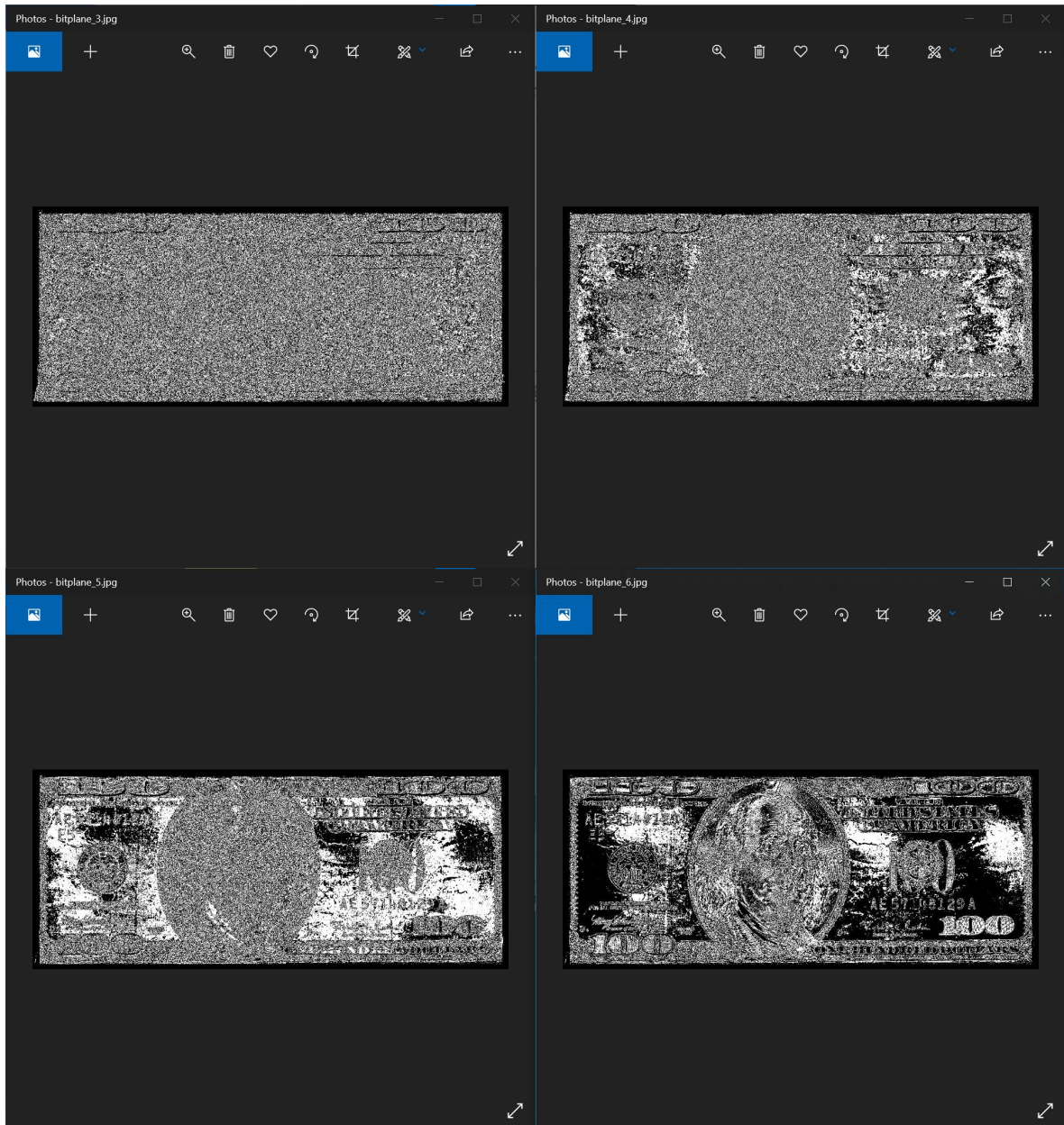
OUTPUT





National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science





National University of Sciences and Technology (NUST)

School of Electrical Engineering and Computer Science

