Caden Thompson Ct1764 HW1

Q1: Original Image 256x256

Nearest Neighbor 512x512

Bilinear 512x512

Bicubic 512 x512

Negative 256x256

Original for Log, Gamma, Piecewise

Logarithmic c=25

Gamma γ= 1.5

Piecewise Contrast Stretching

The Bicubic interpolation used an a value of -.5 and was the most computationally expensive taking a noticeable amount of time to compute, but it produced the best result out of nearest neighbor and bilinear. A greyscale image was used for Log, Gamma, and Piecewise due to its use in the lecture slides and for better results. For logarithmic a c of anything less than 3 appeared as a black image, so 25 seemed a good balance to where the image could still be seen, so that is the value that was chosen. The gamma equation used in this implementation used a normalized r value by dividing r by 255. For gamma a c=1 and γ= 1.5 were chosen to darken the image slightly. Piecewise was used to perform contrast stretching using the parameters r1 = 70, r2 = 140, s1 = 0, s2 = 255.

Caden Thompson Ct1764 HW1

Nearest Neighbor

```
def NN():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    image1 = cv2.imread(imagepath, cv2.IMREAD_ANYCOLOR)
    print(type(image1[0,0][0]))
    w, h = image1.shape[:2]
    xNew = int(w * 2);
    yNew = int(h * 2);
    xScale = xNew/(w-1);
    yScale = yNew/(h-1);
    #xScale = xNew/(w-1)
    #yScale = yNew/(h-1)
    newImage = np.zeros([xNew, yNew, 3], dtype='uint8')
    print(newImage.shape)
    print(image1[0,0])
    newImage[0,0] = image1[0,0]
    for i in range(xNew-1):
        for j in range(yNew-1):
                newImage[i + 1, j + 1]= image1[1 + int(i / xScale),1 + int(j / yScale)]
    print(newImage[0,0])
    return newImage
```

Imported Libraries

```
import numpy as np
import sys # to acc
import cv2
import math
import time
import os
```

Bilinear

```
def BL(original_img, new_h, new_w):
    old_h, old_w, c = original_img.shape
    resized = np.zeros((new_h, new_w, c))
    w_scale_factor = (old_w ) / (new_w ) if new_h != 0 else 0
    h_scale_factor = (old_h ) / (new_h ) if new_w != 0 else 0
    for i in range(new_h):
        for j in range(new_w):
            x = i * h_scale_factor
            y = j * w_scale_factor
            x_floor = math.floor(x)
            x_ceil = min( old_h - 1, math.ceil(x))
            y_floor = math.floor(y)
            y_ceil = min(old_w - 1, math.ceil(y))
            if (x_ceil == x_floor) and (y_ceil == y_floor):
                q = original_img[int(x), int(y), :]
            elif (x_ceil == x_floor):
                q1 = original_img[int(x), int(y_floor), :]
                q2 = original_img[int(x), int(y_ceil), :]
                q = q1 * (y_ceil - y) + q2 * (y - y_floor)
            elif (y_ceil == y_floor):
                q1 = original_img[int(x_floor), int(y), :]
                q2 = original_img[int(x_ceil), int(y), :]
                q = (q1 * (x_ceil - x)) + (q2    * (x - x_floor)
            else:
                v1 = original_img[x_floor, y_floor, :]
                v2 = original_img[x_ceil, y_floor, :]
                v3 = original_img[x_floor, y_ceil, :]
                v4 = original_img[x_ceil, y_ceil, :]

                q1 = v1 * (x_ceil - x) + v2 * (x - x_floor)
                q2 = v3 * (x_ceil - x) + v4 * (x - x_floor)
                q = q1 * (y_ceil - y) + q2 * (y - y_floor)
            resized[i,j,:] = q
    return resized.astype(np.uint8)
```

Used to save image files

```
os.chdir('C:/Users/caden/Down
filename = 'mandrilcomb.jpg'
#saves file
cv2.imwrite(filename,image4)
```

Bicubic

```
def bicubic(img, ratio, a):
    H,W,C = img.shape
    img = padding(img,H,W,C)
    dH = math.floor(H*ratio)
    dW = math.floor(W*ratio)
    dst = np.zeros((dH, dW, 3), dtype='uint8')
    h = 1/ratio
    print('Start bicubic interpolation')
    print('It will take a little while...')
    inc = 0
    for c in range(C):
        for j in range(dH):
            for i in range(dW):
                x, y = i * h + 2 , j * h + 2
                x1 = 1 + x - math.floor(x)
                x2 = x - math.floor(x)
                x3 = math.floor(x) + 1 - x
                x4 = math.floor(x) + 2 - x
                y1 = 1 + y - math.floor(y)
                y2 = y - math.floor(y)
                y3 = math.floor(y) + 1 - y
                y4 = math.floor(y) + 2 - y
                mat_l = np.matrix([[u(x1,a),u(x2,a),u(x3,a),u(x4,a)]])
                mat_m = np.matrix([[img[int(y-y1),int(x-x1),c],img[int(y-y2),int(x-x1),c],img[int(y+y3),int(x-x1),c],img[int(y+y4),int(x-x1),c]],
                                   [img[int(y-y1),int(x-x2),c],img[int(y-y2),int(x-x2),c],img[int(y+y3),int(x-x2),c],img[int(y+y4),int(x-x2),c]],
                                   [img[int(y-y1),int(x+x3),c],img[int(y-y2),int(x+x3),c],img[int(y+y3),int(x+x3),c],img[int(y+y4),int(x+x3),c]],
                                   [img[int(y-y1),int(x+x4),c],img[int(y-y2),int(x+x4),c],img[int(y+y3),int(x+x4),c],img[int(y+y4),int(x+x4),c]]])
                mat_r = np.matrix([[u(y1,a)],[u(y2,a)],[u(y3,a)],[u(y4,a)]])
                cvar = np.dot(mat_l, mat_m)
                dst[j, i, c] = np.dot(cvar,mat_r)
                inc = inc + 1
                sys.stderr.write('\r\033[K' + get_progressbar_str(inc/(C*dH*dW)))
                sys.stderr.flush()
    sys.stderr.write('\n')
    sys.stderr.flush()
    return dst
```

Bicubic Helper Functions

```
def u(s,a):
    if (abs(s) >=0) & (abs(s) <=1):
        return (a+2)*(abs(s)**3)-(a+3)*(abs(s)**2)+1
    elif (abs(s) > 1) & (abs(s) <= 2):
        return a*(abs(s)**3)-(5*a)*(abs(s)**2)+(8*a)*abs(s)-4*a
    return 0
#https://github.com/rootpine/Bicubic-interpolation
#Paddnig
def padding(img,H,W,C):
    zimg = np.zeros((H+4,W+4,C))
    zimg[2:H+2,2:W+2,:C] = img
    zimg[2:H+2,0:2,:C]=img[:,0:1,:C]
    zimg[H+2:H+4,2:W+2,:]=img[H-1:H,:,:]
    zimg[2:H+2,W+2:W+4,:]=img[:,W-1:W,:]
    zimg[0:2,2:W+2,:C]=img[0:1,:,:C]
    zimg[0:2,0:2,:C]=img[0,0,:C]
    zimg[H+2:H+4,0:2,:C]=img[H-1,0,:C]
    zimg[H+2:H+4,W+2:W+4,:C]=img[H-1,W-1,:C]
    zimg[0:2,W+2:W+4,:C]=img[0,W-1,:C]
    return zimg
#https://github.com/rootpine/Bicubic-interpolation
# https://github.com/yunabe/codelab/blob/master/misc/terminal_progressbar/progress.py
def get_progressbar_str(progress):
    END = 170
    MAX_LEN = 30
    BAR_LEN = int(MAX_LEN * progress)
    return ('Progress:[' + '=' * BAR_LEN +
            ('>' if BAR_LEN < MAX_LEN else '') +
            ' ' * (MAX_LEN - BAR_LEN) +
            '] %.1f%%' % (progress * 100.))
```

Negative, Logarithmic, and Gamma

```
def Negative():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_color_256.tif'
    image1 = cv2.imread(imagepath, cv2.IMREAD_ANYCOLOR)
    w, h = image1.shape[:2]
    newImage = np.zeros([w, h, 3], dtype='uint8')
    for i in range(w):
        for j in range(h):
                values = image1[i,j]
                negvalues = (256 - 1- values[0], 256 - 1- values[0],256 - 1- values[0])
                newImage[i, j]= negvalues
    return newImage
def Log():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_color_256.tif'
    image1 = cv2.imread(imagepath, cv2.IMREAD_ANYCOLOR)
    w, h = image1.shape[:2]
    newImage = np.zeros([w, h, 3], dtype='uint8')
    c = 255/(np.log(1 + np.max(image1)))
    c= 25
    for i in range(w):
        for j in range(h):
                values = image1[i,j]
                negvalues = (c* math.log(1+values[0]), c* math.log(1+values[0]),c* math.log(1+values[0]))
                newImage[i, j]= negvalues
    return newImage

def Gamma():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif'
    image1 = cv2.imread(imagepath, cv2.IMREAD_ANYCOLOR)
    print(image1[0,0])
    w, h = image1.shape[:2]
    newImage = np.zeros([w, h, 1], dtype='uint8')
    c = 1
    gamma = 1.5
    for i in range(w):
        for j in range(h):
                newImage[i, j]= c*(image1[i,j]/255)**gamma * 255
    return newImage
```

Piecewise Contrast Stretching

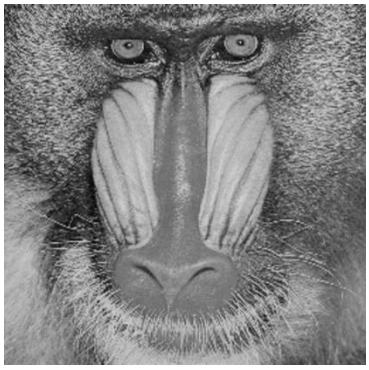
```
#https://www.geeksforgeeks.org/python-intensity-transformation-operations-on-images/
def pixelVal(pix, r1, s1, r2, s2):
        if (0 <= pix and pix <= r1):
            return (s1 / r1)*pix
        elif (r1 < pix and pix <= r2):
            return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
        else:
            return ((255 - s2)/(255 - r2)) * (pix - r2) + s2
#https://www.geeksforgeeks.org/python-intensity-transformation-operations-on-images/
def Piecewise():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif'
    image1 = cv2.imread(imagepath, cv2.IMREAD_ANYCOLOR)
    r1 = 70
    s1 = 0
    r2 = 140
    s2 = 255
    pixelVal_vec = np.vectorize(pixelVal)
    newImage = pixelVal_vec(image1, r1, s1, r2, s2)
    return newImage
```

Main Function

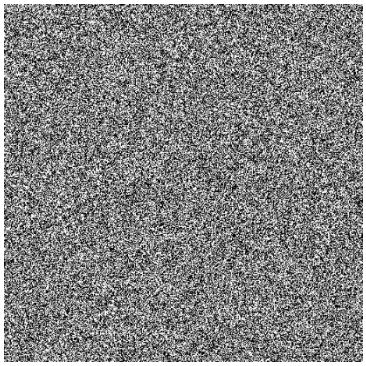
```
def main():
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_color_256.tif'
    image1 = cv2.imread(imagepath, cv2.IMREAD_ANYCOLOR)
    imagepath2 = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif'
    image2 = cv2.imread(imagepath2, cv2.IMREAD_ANYCOLOR)
    imagepathb = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_gray.tif'
    imageb = cv2.imread(imagepathb, cv2.IMREAD_ANYCOLOR)
    print(imageb.shape)
    NNimage = NN()
    BLimage = BL(image1,512,512)
    ratio = 2
    a = -1/2
    dst = bicubic(image1, ratio, a)
    print(dst.shape)
    negimage = Negative()
    logimage = Log()
    gammaimage = Gamma()
    pimage = Piecewise()
```

Caden Thompson Ct1764 HW1

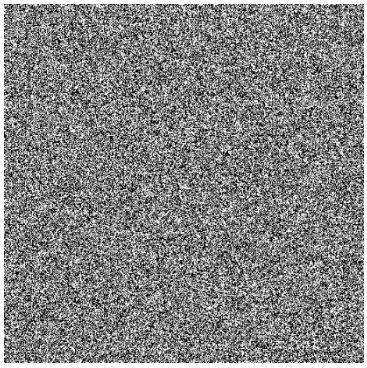
Q2: Original Image 512x512



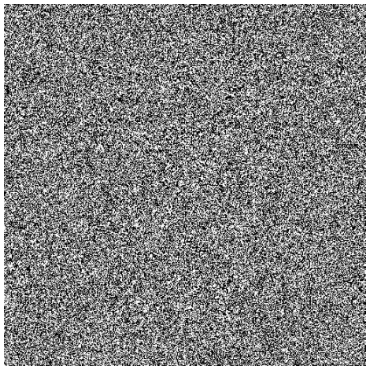
Bit plane 0



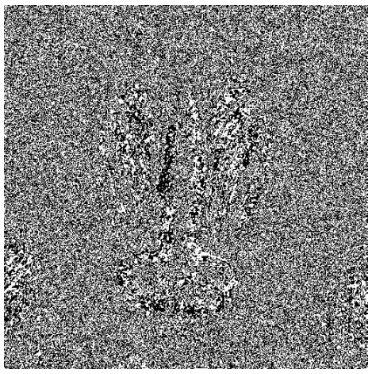
Bit plane 1



Bit plane 2



Bit plane 3



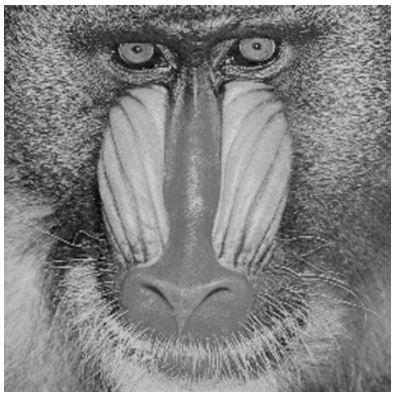
Bit plane 4

Bit plane 5

Bit plane 6

Bit plane 7(MSB)

All Bit planes combined



NumPy was used to convert the image arrays into binary values and into separate image arrays. Each bit plane array was multiplied by its scaling factor (0*1, 1*2… 7*128) to obtain the original values and then all arrays were summed together to get the recovered image. Each bit plane image array was converted into a binary image containing either 0 or 256 rather than 0 or 1 due to constraints on the cv2 saving and viewing function used. This was a workaround I used, but there are probably much better way to do it.

Caden Thompson Ct1764 HW1

Bit plane slicing helper function.

```python
#https://hardikkamboj1.medium.com/intensity-tranformation-bit-plane-slicing-in-python-a48a909121e1
def bitPlaneSlicing(r, bit_plane):
    dec = np.binary_repr(r, width = 8)
    return np.uint8(dec[8-bit_plane])
```

Function that converted data from a binary image containing values of 0 and 1 to an image containing values 0 and 255 to use cv2 file display and save.

```python
def correctvals(image1):
    newImage = np.zeros([512, 512, 1], dtype='uint8')
    for i in range(512):
            for j in range(512):
                if(image1[i,j] == 1):

                    newImage[i, j]= 255
    return newImage
```

Main function that called the helper function and created a dictionary of the bit plane images. Then it combined all of the images to create the combined image. Finally it displayed all of the images. The last few lines were used to save individual files for the report.

```python
#bitplane slicing
#https://hardikkamboj1.medium.com/intensity-tranformation-bit-plane-slicing-in-python-a48a909121e1
bitPlaneSlicingVec = np.vectorize(bitPlaneSlicing)
eight_bitplace = bitPlaneSlicingVec(imageb, bit_plane = 8)
bit_planes_dict = {}
for bit_plane in np.arange(8,0, -1):
    bit_planes_dict['bit_plane_' + str(bit_plane)] = bitPlaneSlicingVec(imageb, bit_plane = bit_plane)
#recombine bitplane https://janithabandara.medium.com/image-compression-using-bit-plane-slicing-opencvsharp-without-pre-defined-functions-608a61d252b7
image4 = bit_planes_dict['bit_plane_1']
+ bit_planes_dict['bit_plane_2'] * 2
+ bit_planes_dict['bit_plane_3'] * 4
+ bit_planes_dict['bit_plane_4'] * 8
+ bit_planes_dict['bit_plane_5'] * 16
+ bit_planes_dict['bit_plane_6'] * 32
+ bit_planes_dict['bit_plane_7'] * 64
+ bit_planes_dict['bit_plane_8'] * 128
#display all bitplanes
cv2.imshow('bit_plane_1', correctvals(bit_planes_dict['bit_plane_1']))
cv2.imshow('bit_plane_2', correctvals(bit_planes_dict['bit_plane_2']))
cv2.imshow('bit_plane_3', correctvals(bit_planes_dict['bit_plane_3']))
cv2.imshow('bit_plane_4', correctvals(bit_planes_dict['bit_plane_4']))
cv2.imshow('bit_plane_5', correctvals(bit_planes_dict['bit_plane_5']))
cv2.imshow('bit_plane_6', correctvals(bit_planes_dict['bit_plane_6']))
cv2.imshow('bit_plane_7', correctvals(bit_planes_dict['bit_plane_7']))
cv2.imshow('bit_plane_8', correctvals(bit_planes_dict['bit_plane_8']))
cv2.imshow('combined', image4)
cv2.waitKey(0)
cv2.destroyAllWindows()
#save file
os.chdir('C:/Users/caden/Downloads/standard_test_images/standard_test_images/')
filename = 'mandrilcomb.jpg'
#saves file
cv2.imwrite(filename,image4)
```

References:

https://www.geeksforgeeks.org/image-processing-without-opencv-python/

https://meghal-darji.medium.com/implementing-bilinear-interpolation-for-image-resizing-357cbb2c2722

https://github.com/rootpine/Bicubic-interpolation

https://github.com/yunabe/codelab/blob/master/misc/terminal_progressbar/progress.py

https://www.geeksforgeeks.org/python-intensity-transformation-operations-on-images/

https://hardikkamboj1.medium.com/intensity-tranformation-bit-plane-slicing-in-python-a48a909121e1

https://janithabandara.medium.com/image-compression-using-bit-plane-slicing-opencvsharp-without-pre-defined-functions-608a61d252b7