CS-663 Assignment 1 Q1

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

August 30, 2020

1 (30 points) Image Resizing and Rotation.

1.1 (3 points) Image Shrinking.

Input image: *1/data/circles_concentric.png* .

Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes. Shrink the image size by a factor of d along each dimension using image subsampling by sampling / selecting every d-th pixel along the rows and columns.

- Write a function *myShrinkImageByFactorD.m* to implement this.

- Display the original and subsampled images, with the correct aspect ratio, for d = 2 and d = 3 appropriately to clearly show the Moire effects. Display the pixel units along each axis and the colorbar.

```python
1   # IMPORTING MODULES FOR BASIC COMPUTATION
2   import numpy as np
3   import matplotlib.pyplot as plt # for plotting
4   import matplotlib.image as mpimg # for image reading
5   import matplotlib as mpl
6
7   from numpy import zeros,zeros_like,array
8
9   def myShrinkImageByFactorD(input_file,d,cmap="gray"):
10      """
11      d : List of shrinkage factors
12      Shrink the image size by a factor of d along each dimension using image subsampling by
13      sampling / selecting every d -th pixel along the rows and columns.
14      usage : myShrinkImageByFactorD([2,3])
15      input : <input_image_path>,D (list of shrink factors D)
16      output : None
17      Saves the shrinked image data in the ../data folder
18      """
19
20      name = input_file.split(".")[2]
21      input_image = mpimg.imread(input_file,format="png")
22      num_plots = len(d)+1
23
24      width = input_image.shape[0]
25      height = input_image.shape[1]
26      output_images = []
27
28      fig,axes = plt.subplots(1,num_plots, constrained_layout=True)
29
30      # PLOTTING PARAMETERS
31      parameters = {'axes.titlesize': 10}
32      plt.rcParams.update(parameters)
33
34      axes[0].imshow(input_image,cmap=cmap)
35      axes[0].axis("on")
```

```
36        axes[0].set_title("Original Image")

37

38        count = 0

39

40        for i in d:
41            count = count + 1
42            new_width = int(width/i)
43            new_height = int(height/i)
44            output = zeros((new_width,new_height))
45            for W in range(new_width):
46                for H in range(new_height):
47                    output[W][H] = input_image[W*i][H*i]
48            output_images.append(output)

49

50            im = axes[count].imshow(output, cmap=cmap)
51            axes[count].axis("on")
52            axes[count].set_title("Shrink by Factor"+str(i))

53

54        cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
55        plt.savefig(".."+name+"Shrink.png",bbox_inches="tight",pad=-1,cmap=cmap)

56

57        for i in range(len(d)):
58            plt.imsave(".."+name+"ShrinkByFactor"+str(d[i])+".png",output_images[i],cmap=cmap)
```
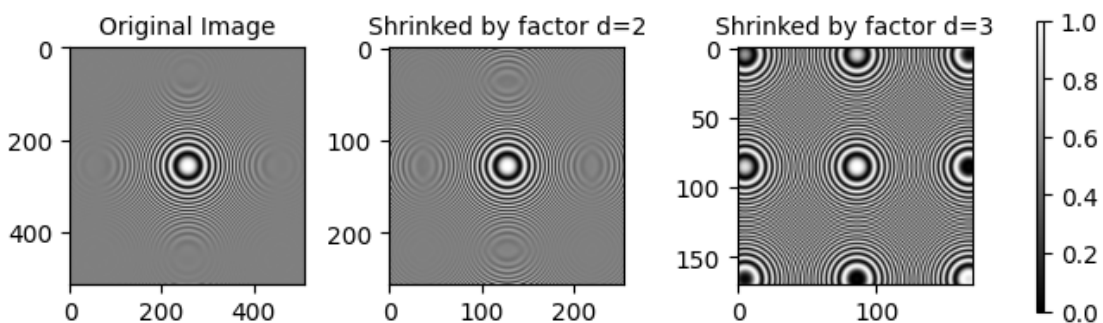

Figure 1: Output of myShrinkageByD.py

Observtion : By shrinking by a factor D, we are effectively changing the sampling frequency of the image, thereby, changing the moire pattern of the image.

1.2 (6 points) Image Enlargement using Bilinear Interpolation

Input image : *1/data/barbaraSmall.png*.

Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of1:1 for the axes. Consider this image as the data. Consider the number of rows as M and the number of columns as *N*.Resize the image to have the number of rows $= 3M2$ and the number of columns $= 2N1$, such that the first and last rows, and the first and last columns, in the original and resized images represent the same data. Use bilinear interpolation for resizing.

- Write a function *myBilinearInterpolation.m* to implement this.

- Display the original and resized images, without changing the aspect ratio of objects present in the image. Display the pixel units along each axis and the colorbar.

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3   import matplotlib.image as mpimg

4

5   from numpy import zeros,zeros_like,array

6

7   from math import floor,ceil

8
```

```
 9    def myBilinearInterpolation(input_file,cmap="gray",region=[]):
10        """
11        input = <input_file_path>,cmap(optional),region(optional)
12        output = None
13        Saves the bilinear Interpolated image to the ../data folder
14        """
15        input_image = mpimg.imread(input_file,format="png")
16        name = input_file.split(".")[2]
17
18        # PLOTTING PARAMETERS
19        parameters = {'axes.titlesize': 10}
20        plt.rcParams.update(parameters)
21
22        if len(region)!=0:
23            input_image = input_image[region[0]:region[1],region[2]:region[3]]
24            name="/data/region"
25
26        rows,columns = input_image.shape
27        new_cols = 2*columns-1
28        new_rows = 3*rows-2
29        row_ratio = ceil(new_rows/rows)
30        col_ratio = ceil(new_cols/columns)
31
32        output = np.zeros((new_rows,new_cols))
33
34        for row in range(new_rows):
35            r = row/row_ratio
36            r1 = floor(r)
37            r2 = ceil(r)
38            for col in range(new_cols):
39                c = col/col_ratio
40                c1 = floor(c)
41                c2 = ceil(c)
42                if(r1<=rows and r2<=rows and c1<=columns and c2<=columns):
43                    bottom_left = input_image[r1][c1]
44                    bottom_right = input_image[r2][c1]
45                    top_left = input_image[r1][c2]
46                    top_right = input_image[r2][c2]
47                    output[row][col] = bottom_right*(r\%1)*(1-(c\%1)) + bottom_left*(1-r\%1)*(1-c\%1) +
48                                        top_right*(r\%1)*(c\%1) + top_left*(1-r\%1)*(c\%1)
49
50        fig,axes = plt.subplots(1,2, constrained_layout=True, gridspec_kw={'width_ratios':[1,2]})
51        axes[0].imshow(input_image,cmap)
52        axes[0].axis("on")
53        axes[0].set_title("Original Image")
54        im = axes[1].imshow(output,cmap)
55
56        axes[1].axis("on")
57        axes[1].set_title("Bilinear Interpolated Image")
58
59        cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
60
61        # SAVING THE IMAGE WITH INTERPOLATED AND ORIGINAL
62        plt.savefig(".."+name+"BilinearInterpolation.png",cmap=cmap,bbox_inches="tight",pad=-1)
63
64        # SAVING THE INTERPOLATED IMAGE
65        plt.imsave(".."+name+"Bilinear.png",output,cmap=cmap)
```
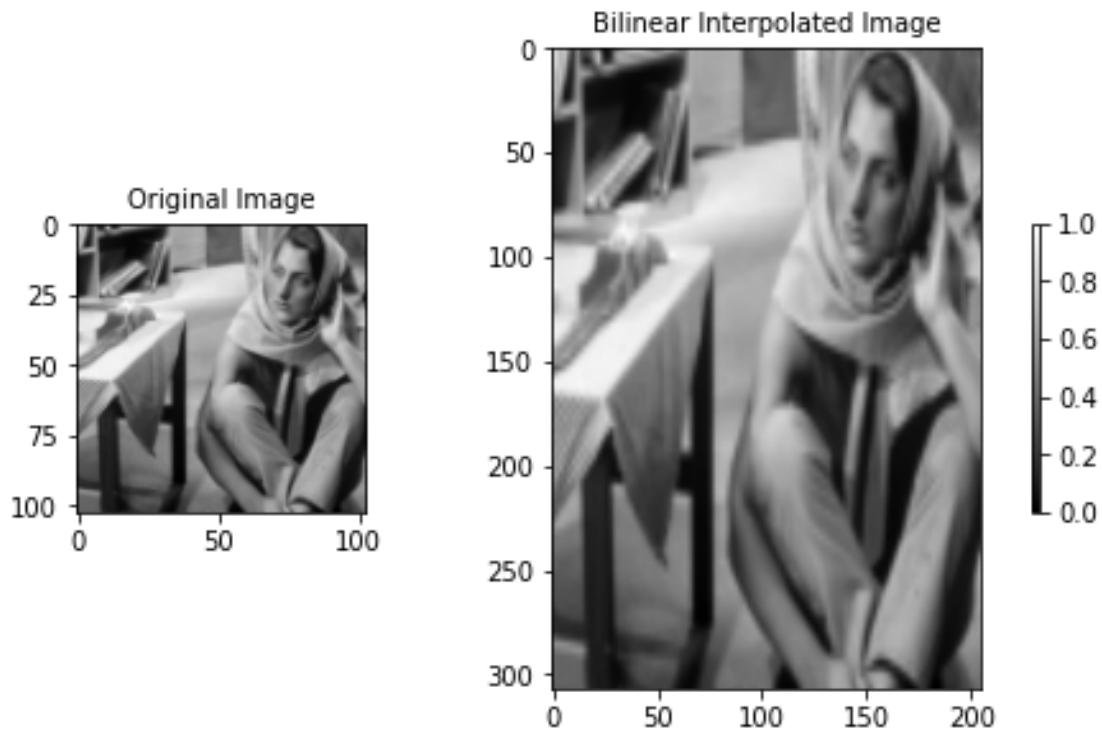
Figure 2: Output of myBilinearInterpolation.py

1.3 (6 points) Image Enlargement using Nearest-Neighbor Interpolation

Redo the previous problem using nearest-neighbor interpolation.

- Write a function *myNearestNeighborInterpolation.m* to implement this.

- Display the original and resized images.

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from numpy import zeros,zeros_like,array
from math import floor,ceil


def myNearestNeighbourInterpolation(input_file,cmap="gray",region=[]):
    """
    input : <input_file_path>,cmap(optional),region (optional)
    output : None
    Saves the Nearest Neighbour interpolated images to the ../data folder.
    """
    name = input_file.split(".")[2]
    input_image = mpimg.imread(input_file,format="png")

    if len(region)!=0:
        input_image = input_image[region[0]:region[1],region[2]:region[3]]
        name="/data/region"

    # PLOTTING PARAMETERS
    parameters = {'axes.titlesize': 10}
    plt.rcParams.update(parameters)

    rows,columns = input_image.shape

    new_cols = 2*columns-1
    new_rows = 3*rows-2

    row_ratio = ceil(new_rows/rows)
    col_ratio = ceil(new_cols/columns)

```

```
34        output = zeros((new_rows,new_cols))
35
36        for row in range(new_rows):
37            r = row/row_ratio
38            r1 = (floor(r) if (r\%1)<0.5 else ceil(r))
39            for col in range(new_cols):
40                c = col/col_ratio
41                c1 = (floor(c) if (c\%1)<0.5 else ceil(c))
42                output[row][col] = input_image[r1][c1]
43
44        fig,axes = plt.subplots(1,2, constrained_layout=True, gridspec_kw={'width_ratios':[1,2]})
45        axes[0].imshow(input_image,cmap)
46        axes[0].axis("on")
47        axes[0].set_title("Original Image")
48        im = axes[1].imshow(output,cmap)
49        axes[1].axis("on")
50        axes[1].set_title("Nearest Neighbor Interpolated")
51
52        cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
53        plt.savefig(".."+name+"NearestNeighbor.png",cmap=cmap,bbox_inches="tight",pad=-1)
54
55        plt.imsave(".."+name+"NN.png",output,cmap=cmap)
```
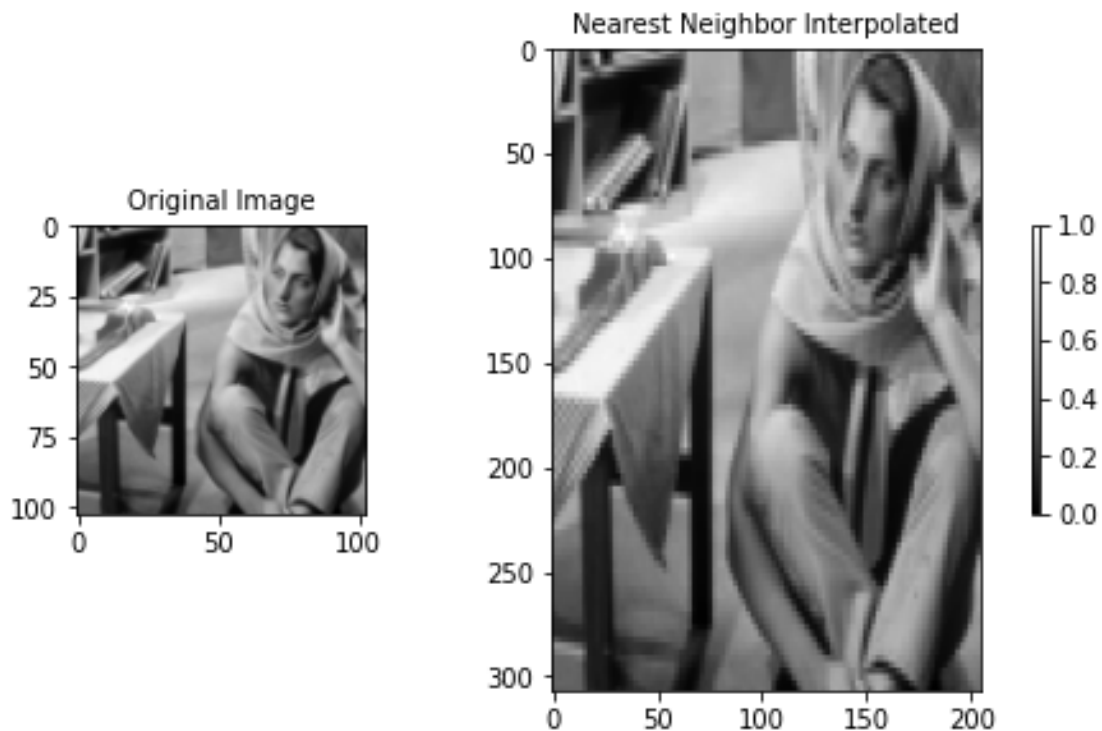

Figure 3: Output of myNearestNeighbourInterpolation.py

1.4 (6 points) Image Enlargement using Bicubic Interpolation.

Redo the previous problem using bicubic interpolation.

- Write a function *myBicubicInterpolation.m* to implement this.

- Display the original and resized images.

```
1    import numpy as np
2    import matplotlib.pyplot as plt
3    import matplotlib.image as mpimg
4
5    from numpy import zeros,zeros_like,array,c_
6    from math import floor,ceil
7
8
```

```python
def coeffUpdate(input_image,x,y):
    """
    This function calculates the 16 coefficients necessary for calculating the
    bicubic interpolation equations.
    input : input_image , present co-ordinates (x=row,y=col)
    output : the 16 coefficients a00 to a33
    """
    p = input_image.copy()
    r,c = input_image.shape
    q = zeros((r,4))
    s = zeros((4,c+4))
    p = np.concatenate((p,q), axis=1)
    p = np.concatenate((p,s), axis=0)

    for i in range(4):
        p[r+i][:] = input_image[r-1][c-1]
        p[:][c+i] = input_image[r-1][c-1]

    a00 = p[x+1][y+1]
    a01 = -.5*p[x+1][y+0] + .5*p[x+1][y+2]
    a02 = p[x+1][y+0] - 2.5*p[x+1][y+1] + 2*p[x+1][y+2] - .5*p[x+1][y+3]
    a03 = -.5*p[x+1][y+0] + 1.5*p[x+1][y+1] - 1.5*p[x+1][y+2] + .5*p[x+1][y+3]
    a10 = -.5*p[x+0][y+1] + .5*p[x+2][y+1]
    a11 = .25*p[x+0][y+0] - .25*p[x+0][y+2] - .25*p[x+2][y+0] + .25*p[x+2][y+2]
    a12 = -.5*p[x+0][y+0] + 1.25*p[x+0][y+1] - p[x+0][y+2] + .25*p[x+0][y+3] + .5*p[x+2][y+0] -
            1.25*p[x+2][y+1] + p[x+2][y+2] - .25*p[x+2][y+3]
    a13 = .25*p[x+0][y+0] - .75*p[x+0][y+1] + .75*p[x+0][y+2] - .25*p[x+0][y+3] - .25*p[x+2][y+0] +
            .75*p[x+2][y+1] - .75*p[x+2][y+2] + .25*p[x+2][y+3]
    a20 = p[x+0][y+1] - 2.5*p[x+1][y+1] + 2*p[x+2][y+1] - .5*p[x+3][y+1]
    a21 = -.5*p[x+0][y+0] + .5*p[x+0][y+2] + 1.25*p[x+1][y+0] - 1.25*p[x+1][y+2] - p[x+2][y+0] +
            p[x+2][y+2] + .25*p[x+3][y+0] - .25*p[x+3][y+2]
    a22 = p[x+0][y+0] - 2.5*p[x+0][y+1] + 2*p[x+0][y+2] - .5*p[x+0][y+3] - 2.5*p[x+1][y+0] +
            6.25*p[x+1][y+1] - 5*p[x+1][y+2] + 1.25*p[x+1][y+3] + 2*p[x+2][y+0] - 5*p[x+2][y+1] +
            4*p[x+2][y+2] - p[x+2][y+3] - .5*p[x+3][y+0] + 1.25*p[x+3][y+1] - p[x+3][y+2] +
            .25*p[x+3][y+3]
    a23 = -.5*p[x+0][y+0] + 1.5*p[x+0][y+1] - 1.5*p[x+0][y+2] + .5*p[x+0][y+3] + 1.25*p[x+1][y+0] -
            3.75*p[x+1][y+1] + 3.75*p[x+1][y+2] - 1.25*p[x+1][y+3] - p[x+2][y+0] + 3*p[x+2][y+1] -
            3*p[x+2][y+2] + p[x+2][y+3] + .25*p[x+3][y+0] - .75*p[x+3][y+1] + .75*p[x+3][y+2] -
            .25*p[x+3][y+3]
    a30 = -.5*p[x+0][y+1] + 1.5*p[x+1][y+1] - 1.5*p[x+2][y+1] + .5*p[x+3][y+1]
    a31 = .25*p[x+0][y+0] - .25*p[x+0][y+2] - .75*p[x+1][0] + .75*p[x+1][2] + .75*p[x+2][0] -
            .75*p[x+2][2] - .25*p[x+3][0] + .25*p[x+3][2]
    a32 = -.5*p[x+0][y+0] + 1.25*p[x+0][y+1] - p[x+0][y+2] + .25*p[x+0][y+3] + 1.5*p[x+1][y+0] -
            3.75*p[x+1][y+1] + 3*p[x+1][y+2] - .75*p[x+1][y+3] - 1.5*p[x+2][y+0] + 3.75*p[x+2][y+1] -
            3*p[x+2][y+2] + .75*p[x+2][y+3] + .5*p[x+3][y+0] - 1.25*p[x+3][y+1] + p[x+3][y+2] -
            .25*p[x+3][y+3]
    a33 = .25*p[x+0][y+0] - .75*p[x+0][y+1] + .75*p[x+0][y+2] - .25*p[x+0][y+3] - .75*p[x+1][y+0] +
            2.25*p[x+1][y+1] - 2.25*p[x+1][y+2] + .75*p[x+1][y+3] + .75*p[x+2][y+0] - 2.25*p[x+2][y+1] +
            2.25*p[x+2][y+2] - .75*p[x+2][y+3] - .25*p[x+3][y+0] + .75*p[x+3][y+1] - .75*p[x+3][y+2] +
            .25*p[x+3][y+3]

    return ([a00,a01,a02,a03,a10,a11,a12,a13,a20,a21,a22,a23,a30,a31,a32,a33])

def myBicubicInterpolation(input_file,row_ratio=3,col_ratio=2,cmap="gray",region=[]):
    """
    inputs : <input_image_path>,row-ratio, col-ratio,cmap(optional),region(optional)
    output = None
    Saves the bi-cubic Interpolated image to the ../data folder
    """
    input_image = mpimg.imread(input_file,format="png")
    name = input_file.split(".")[2]
    if len(region)!=0:
        input_image = input_image[region[0]:region[1],region[2]:region[3]]
        name="/data/region"
```

```
74        rows,columns = input_image.shape
75        new_cols = col_ratio*columns-1
76        new_rows = row_ratio*rows-2
77        output = zeros((new_rows,new_cols))
78
79        # PLOTTING PARAMETERS
80        parameters = {'axes.titlesize': 10}
81        plt.rcParams.update(parameters)
82
83        for row in range(new_rows):
84            r = (row/row_ratio)
85            x = r\%1
86            r1 = floor(r)
87            for col in range(new_cols):
88                c = col/col_ratio
89                y = c\%1
90                c1 = floor(c)
91                if (r1>=0 and c1>=0):
92                    a00,a01,a02,a03,a10,a11,a12,a13,a20,a21,a22,a23,a30,a31,a32,a33 =
93                                        coeffUpdate(input_image,r1,c1)
94                    output[row][col] = (a00 + a01 * y + a02 * y**2 + a03 * y**3) + (a10 + a11 * y + a12 *
95                                    y**2 + a13 * y**3) * x + (a20 + a21 * y + a22 * y**2 + a23 * y**3) *
96                                    x**2 + (a30 + a31 * y + a32 * y**2 + a33 * y**3) * x**3
97
98        fig,axes = plt.subplots(1,2, constrained_layout=True, gridspec_kw={'width_ratios':[1,2]})
99        axes[0].imshow(input_image,cmap)
100       axes[0].axis("on")
101       axes[0].set_title("Original Image")
102       im = axes[1].imshow(output,cmap)
103       axes[1].axis("on")
104       axes[1].set_title("Bicubic Interpolated")
105
106       cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
107       plt.savefig(".."+name+"BicubicInterpolated.png",cmap=cmap,bbox_inches="tight",pad=-1)
108
109       plt.imsave(".."+name+"Bicubic.png",output,cmap=cmap)
```
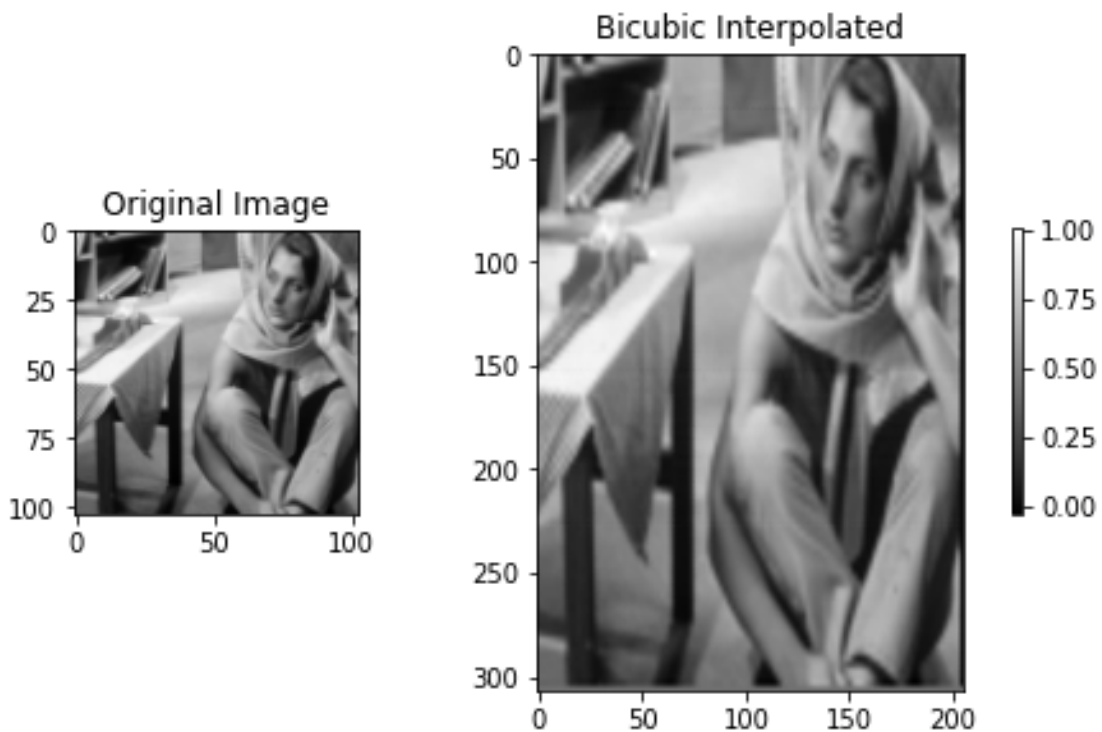

Figure 4: Output of myBicubicInterpolation.py

7

1.5 (3 points) Image Enlargement using Different Interpolation Methods.

- Display a small chosen region in the images resized using the 3 different interpolation methods, i.e., bilinear, nearest-neighbor, and bicubic. Visualize using the "jet" colormap.Compare the results. Describe what you see and justify your observations based on the underlying interpolation theory.

```python
from myNearestNeighbourInterpolation import *
from myBilinearInterpolation import *
from myBicubicInterpolation import *

region = [50,80,50,80]
input_file = "../data/barbaraSmall.png"

myNearestNeighbourInterpolation(input_file,region=region,cmap="jet")
myBilinearInterpolation(input_file,region=region,cmap="jet")
myBicubicInterpolation(input_file,region=region,cmap="jet")
```
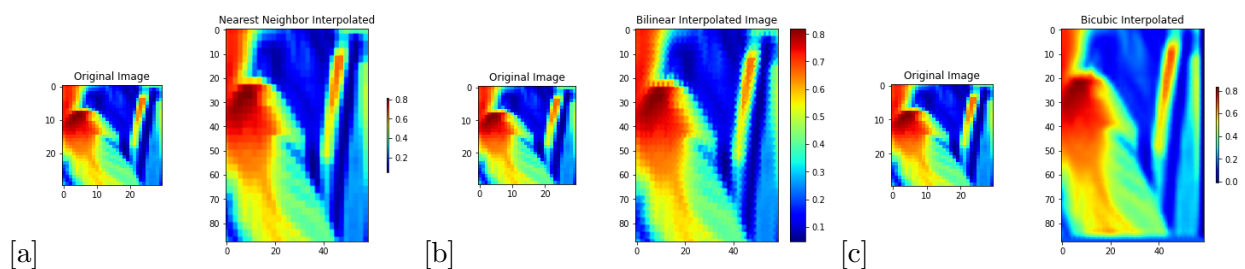

[a] [b] [c]

Figure 5: (a) Nearest Neighbour Interpolation (b) Bi-linear Interpolation (c) Bi-cubic Interpolation

Observation : On going from Nearest Neighbour to Bi-cubic Interpolation, we see that the jaggedness of the image reduces and smoothness increases. The reason for the observation being that in Nearest-Neighbour only the nearest pixel intensity value is used, while in Bi-linear, the intensity value of each pixel is determined on weighted sum of the corner pixels in which the current pixel is located. In Bi-cubic we extends the weighted sum to the 2nd derivatives of the pixel intensities.

1.6 (6 points) Image Rotation using Bilinear Interpolation.Input

Input image:1/data/barbaraSmall.png.

Rotate the entire image (all objects in the image) clockwise by 30 degrees about the central point in the image.

- Write a function *myImageRotation.m* to implement the rotation. Reuse the bi-linear interpolation function that you coded before.

- Display the original and rotated images

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from numpy import zeros,zeros_like,array,c_
from math import cos,sin,pi,floor,ceil

def myImageRotation(input_file,angle,cmap="gray"):
    input_image = mpimg.imread(input_file,format="png")
    name = input_file.split(".")[2]

    theta = angle * pi/180
    translation_matrix = array([[cos(theta),-sin(theta)],[sin(theta),cos(theta)]])

    rows = input_image.shape[0]
    columns = input_image.shape[1]
    new_image = np.ones_like(input_image)
```

```
18        x_mid = (rows -1 )/2
19        y_mid = (columns -1 )/2
20
21        for row in range(rows):
22            for col in range(columns):
23                # rotation matrix transform
24                [row_prime,col_prime] = np.matmul(translation_matrix,array([row-x_mid,col-y_mid]).T)
25                r=x_mid+row_prime
26                r1 = floor(r)
27                r2 = ceil(r)
28                c = y_mid+col_prime
29                c1 = floor(c)
30                c2 = ceil(c)
31
32                if(r1<rows and r2<rows and c1<columns and c2<columns and r1>=0 and r2>=0 and c1>=0 and c2>=0):
33                    bottom_left = input_image[r1][c1]
34                    bottom_right = input_image[r2][c1]
35                    top_left = input_image[r1][c2]
36                    top_right = input_image[r2][c2]
37                    new_image[row][col] = bottom_right*(r\%1)*(1-(c\%1)) +
38                                           bottom_left*(1-r\%1)*(1-c\%1)+top_right*(r\%1)*(c\%1) +
39                                           top_left*(1-r\%1)*(c\%1)
40
41        fig,axes = plt.subplots(1,2, constrained_layout=True)
42        axes[0].imshow(input_image,cmap="gray")
43        axes[0].axis("on")
44        axes[0].set_title("Original Image")
45
46        im = axes[1].imshow(new_image,cmap="gray")
47        axes[1].axis("on")
48        axes[1].set_title(r"Rotated Image by $30^{o}$")
49
50        cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
51        plt.savefig(".."+name+"RotateBy"+str(angle)+".png",cmap=cmap,bbox_inches="tight",pad=-1)
52        plt.imsave(".."+name+"Rotate.png",new_image,cmap=cmap)
```
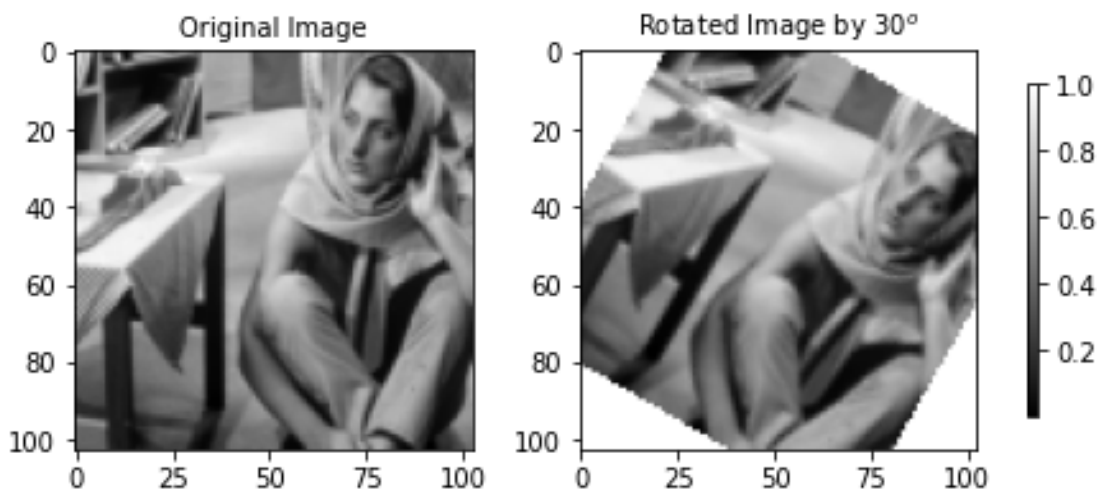

Figure 6: Output of myImageRotation.py

The main script _myMainScript.py_

```
1   from myShrinkImageByFactorD.py import *
2   from myNearestNeighbourInterpolation import *
3   from myBilinearInterpolation import *
4   from myBicubicInterpolation import *
5   from myImageRotation import *
6
7   from time import time
```

9

```python
 8
 9     circle_file = "../data/circles_concentric.png"
10     D = [2,3]
11     barbara_file = "../data/barbaraSmall.png"
12     angle=30
13
14     super_start = time()
15
16     start = time()
17     myShrinkImageByFactorD(circle_file,D,cmap="gray")
18     end = time()
19     print("Time taken to run myShrinkImageByFactorD.py :",end-start,"secs")
20     start = time()
21     myBilinearInterpolation(barbara_file,cmap="gray")
22     end = time()
23     print("Time taken to run myBilinearInterpolation.py :",end-start,"secs")
24     start = time()
25     myNearestNeighbourInterpolation(barbara_file,cmap="gray")
26     end = time()
27     print("Time taken to run myNearestNeighbourInterpolation.py :",end-start,"secs")
28     start = time()
29     myBicubicInterpolation(input_file,cmap="gray")
30     end = time()
31     print("Time taken to run myBicubicInterpolation.py :",end-start,"secs")
32
33     # region test
34     start = time()
35     region = [50,80,50,80]
36     myBilinearInterpolation(barbara_file,region=region,cmap="gray")
37     myNearestNeighbourInterpolation(barbara_file,region=region,cmap="gray")
38     myBicubicInterpolation(barbara_file,region=region,cmap="gray")
39     end = time()
40     print("Time taken to run region tests :",end-start,"secs")
41
42     start = time()
43     myImageRotation(barbara_file,angle,cmap="gray")
44     end = time()
45     print("Time taken to run myImageRotation.py :",end-start,"secs")
46
47
48     super_end = time()
49     print("Time taken to run all the scripts :",super_end-super_start,"secs")
```