CS-663 Assignment 1

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

August 29, 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.

```
# IMPORTING MODULES FOR BASIC COMPUTATION
import numpy as np
import matplotlib.pyplot as plt # for plotting
import matplotlib.image as mpimg # for image reading
import matplotlib as mpl

from numpy import zeros,zeros_like,array
from mpl_toolkits.axes_grid1 import ImageGrid  #extra plotting features

def myShrinkImageByFactorD(d,cmap="gray"):
    """
    d : List of shrinkage factors
    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.
    usage : myShrinkImageByFactorD([2,3])
    input : <input_image_path>
    output : None
    Saves the shrinked image data in the ../data folder
    """
    input_file = "../data/circles_concentric.png"
    input_image = mpimg.imread(input_file,format="png")
    num_plots = len(d)+1

    width = input_image.shape[0]
    height = input_image.shape[1]

    fig,axes = plt.subplots(1,num_plots, constrained_layout=True,
                gridspec_kw={'width_ratios':[4,2,1]})

    #vmin = 0
    #vmax = 255

    axes[0].imshow(input_image,cmap=cmap)
    axes[0].axis("on")

```

```
    count = 0

    for i in d:
        count = count + 1
        new_width = int(width/i)
        new_height = int(height/i)
        output = zeros((new_width,new_height))
        for W in range(new_width):
            for H in range(new_height):
                output[W][H] = input_image[W*i][H*i]

        im = axes[count].imshow(output, cmap=cmap)
        axes[count].axis("on")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),ticks=[0,1],shrink=0.45)
    cbar.ax.set_yticklabels([0,255])
    plt.show()

myShrinkImageByFactorD([2,3])
```

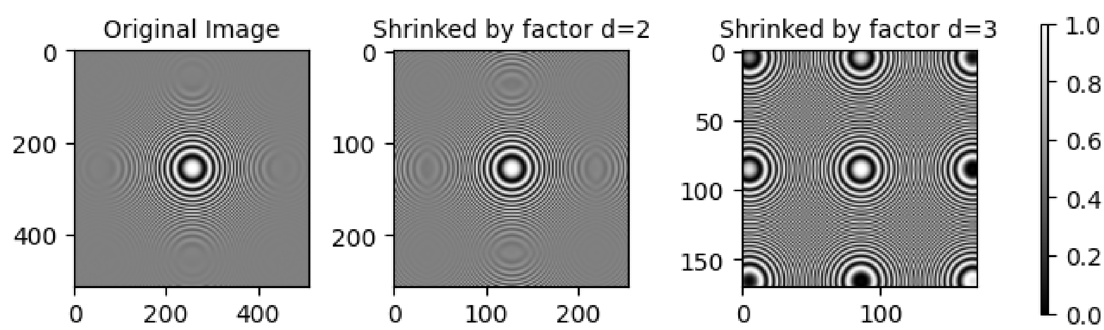

Figure 1: Output of myShrinkageByD.py

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.

```
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 myBilinearInterpolation(input_file,cmap="gray",region=[]):
    """
    input = <input_file_path>,cmap(optional),region(optional)
    output = None
    Saves the bilinear Interpolated image to the ../data folder
```

```
    """
    input_image = mpimg.imread(input_file,format="png")
    name = input_file.split(".")[2]

    # 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)

    output = np.zeros((new_rows,new_cols))

    for row in range(new_rows):
        r = row/row_ratio
        r1 = floor(r)
        r2 = ceil(r)
        for col in range(new_cols):
            c = col/col_ratio
            c1 = floor(c)
            c2 = ceil(c)
            if(r1<=rows and r2<=rows and c1<=columns and c2<=columns):
                bottom_left = input_image[r1][c1]
                bottom_right = input_image[r2][c1]
                top_left = input_image[r1][c2]
                top_right = input_image[r2][c2]
                output[row][col] = bottom_right*(r\%1)*(1-(c\%1)) + bottom_left*(1-r\%1)*(1-c\%1) +
                                   top_right*(r\%1)*(c\%1) + top_left*(1-r\%1)*(c\%1)

    fig,axes = plt.subplots(1,2, constrained_layout=True, gridspec_kw={'width_ratios':[1,2]})
    axes[0].imshow(input_image,cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image")
    im = axes[1].imshow(output,cmap)

    axes[1].axis("on")
    axes[1].set_title("Bilinear Interpolated Image")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)

    # SAVING THE IMAGE WITH INTERPOLATED AND ORIGINAL
    plt.savefig(".."+name+"BilinearInterpolation.png",cmap=cmap,bbox_inches="tight",pad=-1)

    # SAVING THE INTERPOLATED IMAGE
    plt.imsave(".."+name+"Bilinear.png",output,cmap=cmap)
```

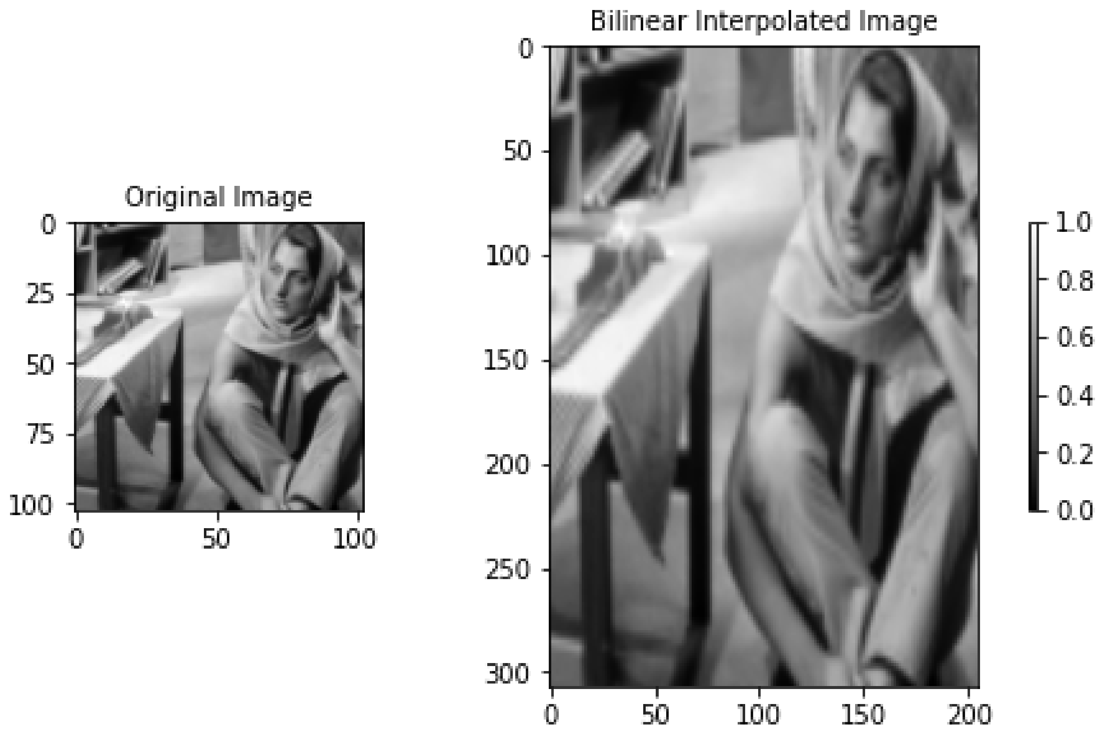

Figure 2: Output of myBilinearInterpolation.py

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.

```
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")

    # 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)

    output = zeros((new_rows,new_cols))

    for row in range(new_rows):
        r = row/row_ratio
```

```
        r1 = (floor(r) if (r\%1)<0.5 else ceil(r))
        for col in range(new_cols):
            c = col/col_ratio
            c1 = (floor(c) if (c\%1)<0.5 else ceil(c))
            output[row][col] = input_image[r1][c1]

    fig,axes = plt.subplots(1,2, constrained_layout=True, gridspec_kw={'width_ratios':[1,2]})
    axes[0].imshow(input_image,cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image")
    im = axes[1].imshow(output,cmap)
    axes[1].axis("on")
    axes[1].set_title("Nearest Neighbor Interpolated")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
    plt.savefig("../data/barbaraSmallNearestNeighbor.png",cmap=cmap,bbox_inches="tight",pad=-1)

    plt.imsave("../data/barbaraSmallNN.png",output,cmap=cmap)
```

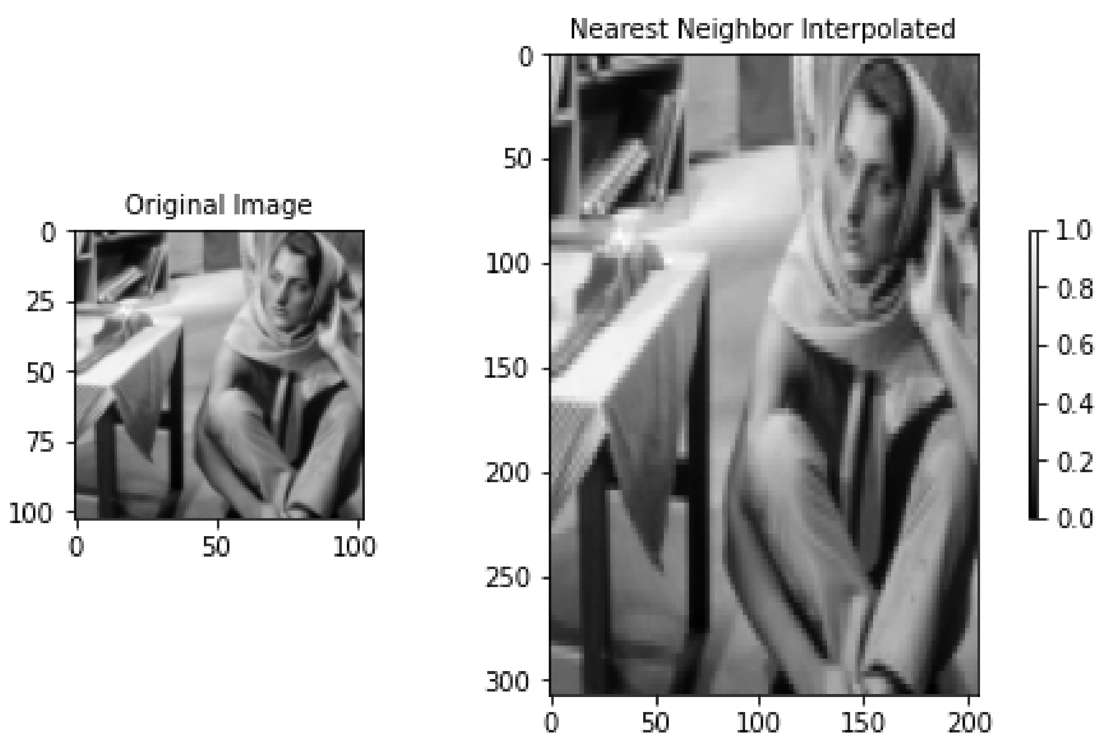

Figure 3: Output of myNearestNeighbourInterpolation.py

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.

```
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 floor,ceil


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)

    """
    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"

    rows,columns = input_image.shape
    new_cols = col_ratio*columns-1
    new_rows = row_ratio*rows-2
    output = zeros((new_rows,new_cols))

```

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

    for row in range(new_rows):
        r = (row/row_ratio)
        x = r%1
        r1 = floor(r)
        for col in range(new_cols):
            c = col/col_ratio
            y = c%1
            c1 = floor(c)
            if (r1>=0 and c1>=0):
                a00,a01,a02,a03,a10,a11,a12,a13,a20,a21,a22,a23,a30,a31,a32,a33 =
                                                coeffUpdate(input_image,r1,c1)
                output[row][col] = (a00 + a01 * y + a02 * y**2 + a03 * y**3) + (a10 + a11 * y + a12 *
                                    y**2 + a13 * y**3) * x + (a20 + a21 * y + a22 * y**2 + a23 * y**3) *
                                    x**2 + (a30 + a31 * y + a32 * y**2 + a33 * y**3) * x**3

    fig,axes = plt.subplots(1,2, constrained_layout=True, gridspec_kw={'width_ratios':[1,2]})
    axes[0].imshow(input_image,cmap)
    axes[0].axis("on")
    axes[0].set_title("Original Image")
    im = axes[1].imshow(output,cmap)
    axes[1].axis("on")
    axes[1].set_title("Bicubic Interpolated")

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

    plt.imsave(".."+name+"Bicubic.png",output,cmap=cmap)
```

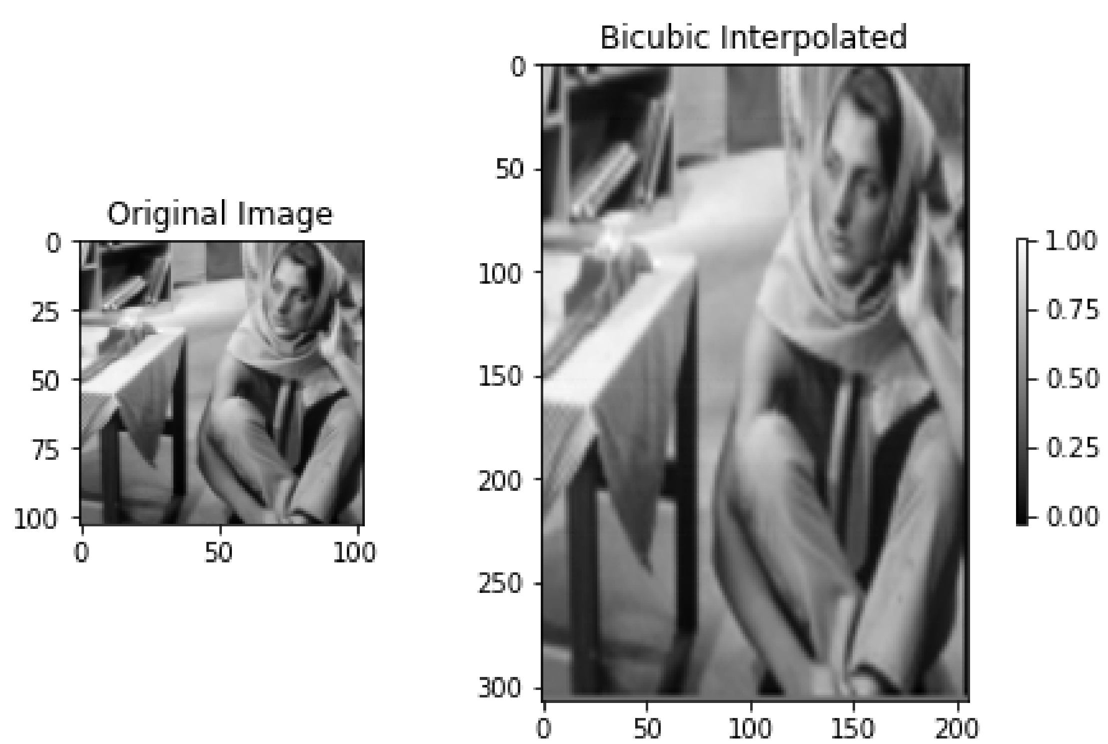

Figure 4: Output of myBicubicInterpolation.py

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

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

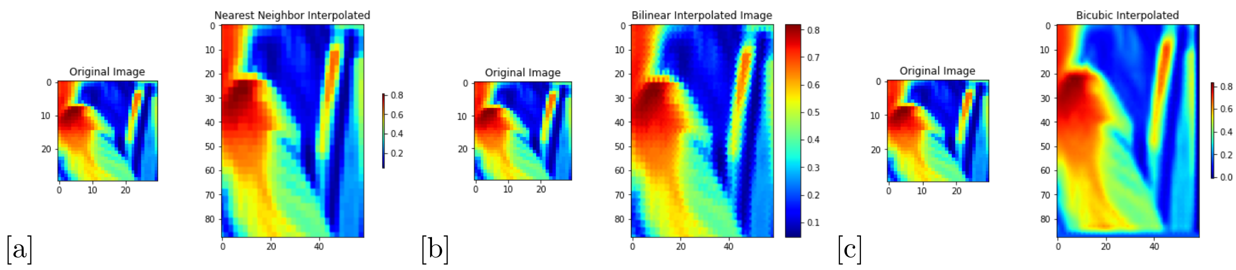

Figure 5: (a) Nearest Neighbour Interpolation (b) Bi-linear Interpolation (c) Bi-cubic Interpolation

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 2^{nd} 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 functionmyImageRotation.mto implement the rotation. Reuse the bilinear 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)
    x_mid = (rows -1 )/2
    y_mid = (columns -1 )/2

    for row in range(rows):
        for col in range(columns):

            [row_prime,col_prime] = np.matmul(translation_matrix,array([row-x_mid,col-y_mid]).T)
            r=x_mid+row_prime
            r1 = floor(r)
            r2 = ceil(r)
            c = y_mid+col_prime
```

```
            c1 = floor(c)
            c2 = ceil(c)

            if(r1<rows and r2<rows and c1<columns and c2<columns and r1>=0 and r2>=0 and c1>=0 and c2>=0):
                bottom_left = input_image[r1][c1]
                bottom_right = input_image[r2][c1]
                top_left = input_image[r1][c2]
                top_right = input_image[r2][c2]
                new_image[row][col] = bottom_right*(r\%1)*(1-(c\%1)) +
                                    bottom_left*(1-r\%1)*(1-c\%1)+top_right*(r\%1)*(c\%1) +
                                    top_left*(1-r\%1)*(c\%1)

    fig,axes = plt.subplots(1,2, constrained_layout=True)
    axes[0].imshow(input_image,cmap="gray")
    axes[0].axis("on")
    axes[0].set_title("Original Image")

    im = axes[1].imshow(new_image,cmap="gray")
    axes[1].axis("on")
    axes[1].set_title(r"Rotated Image by $30^{o}$")

    cbar = fig.colorbar(im,ax=axes.ravel().tolist(),shrink=0.45)
    #cbar.ax.set_yticklabels()
    plt.savefig("../data/barbaraSmallRotateBy30.png",cmap=cmap,bbox_inches="tight",pad=-1)
    plt.imsave("../data/barbaraSmallRotate.png",new_image,cmap=cmap)
```

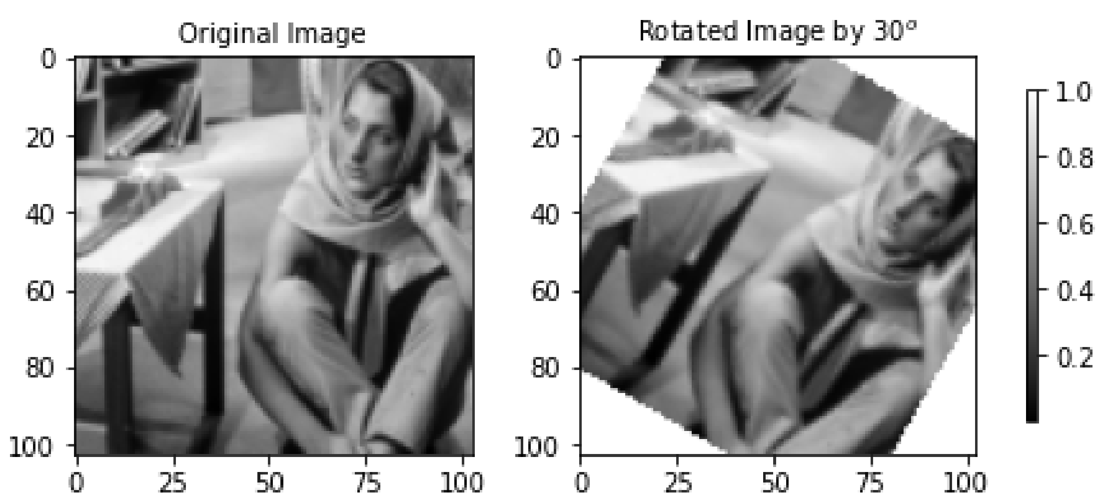

Figure 6: Output of myImageRotation.py