# CSE 107: Lab 03: Image Resizing

**Angelo Fatali**
**LAB: R 7:30-10:20pm**
**Liang, Haolin**
**October 24, 2022**

**Abstract:**

The point of this lab is to understand the difference between bilinear interpolation and nearest neighbor when resizing an image. To do this we can downsample and upsample using both methods and see the difference. We then can use the root mean square error (RMSE) value to determine which is better mathematically.

**Qualitative Results:**

Figure 1: Downsampled to size (100, 175) using nearest neighbor interpolation.
downsampled_NN.tif

Figure 2: Downsampled to size (100, 175) using nearest bilinear interpolation
downsampled_bilinear.tif

Figure 3: Upsampled to size (500, 625) using nearest neighbor interpolation
upsampled_NN.tif

Figure 4: Upsampled to size (500, 625) using nearest bilinear interpolation
upsampled_bilinear.tif

**Quantitative Results:**

| | Nearest neighbor interpolation | Bilinear interpolation |
|---|---|---|
| **Downsample then upsample** | 22.746414 | 16.882088 |
| **Upsample then downsample** | 0.000000 | 5.449098 |

Table 1: RMSE values between the original image and then the down/upsampled and up/ downsampled versions for both nearest neighbor and bilinear interpolation.

**Questions:**

1. Visually compare the two downsampled images, one using nearest neighbor interpolation and one using bilinear interpolation. How are they different and why based on what you know about the two interpolations?

    - The downsampled image that used nearest neighbor interpolation is much more blocky is pixelated. The bilinear, while still a little pixelated, is much better when compared to the nearest neighbor. This is because nearest neighbor simply gets the pixel value based off the closest distance the other pixels are around it. While bilinear will get the average of the other pixels and get a much better estimate of pixel value.

2. Visually compare the two down then upsampled images, one using nearest neighbor interpolation and one using bilinear interpolation. How are they different? Which one looks better to you? Does this agree with the RMSE values?

    - The down then upsampled image using nearest neighbor resulted in a very pixelated but crisp image. The bilinear result was less pixelated but seemed blurry, almost like it was out of focus. I personally like the bilinear result more since we get more of an idea of what each pixel is, rather than super pixelated. The RMSE value agrees with me as 22 (NN result) is greater than 16 (Bilinear result).

3. Visually compare the two up then downsampled images, one using nearest neighbor interpolation and one using bilinear interpolation. How are they different? Which one looks better to you? Does this agree with the RMSE values?

    - The up then downsampled image using nearest neighbor resulted in an identical image to the original. The bilinear result was pretty good but seemed a little blurry. I personally like the nearest neighbor result more since we got an identical image to the original. The RMSE value agrees with me as 0 (NN result) is less than 5 (Bilinear result).

4. If your image resizing is implemented correctly, you should get an RMSE value of zero between the original image and the up then downsampled one using nearest neighbor interpolation. Why is this the case?

    - Since nearest neighbor, when up scaling, is getting the closest pixel value to the other pixels, we get a very pixelated up scale image. However, when you take that same up scaled image and then down sample it to the original size, we actually don't get any pixel values to be different since the closest pixels are simply the original pixel values.

5. What was the most difficult part of this assignment?

    - I kept having a ton of index out of range errors and had to figure out where I needed to add a -1 and where I didn't. Other than that it was fairly easy.

**test_myresize.py**

```
# Import pillow
from PIL import Image, ImageOps

# Import numpy
import numpy as np
from numpy import asarray

# Read the image from file.
orig_im = Image.open('Lab_03_image.tif')

# Show the original image.
orig_im.show()

# Create numpy matrix to access the pixel values.
# NOTE THAT WE WE ARE CREATING A FLOAT32 ARRAY SINCE WE WILL BE DOING
# FLOATING POINT OPERATIONS IN THIS LAB.
orig_im_pixels = asarray(orig_im, dtype=np.float32)

# Import myImageResize from MyImageFunctions
from MyImageFunctions import myImageResize

#########################################################################
# Experiment 1: Downsample then upsample using nearest neighbor
interpolation.
#########################################################################

# Create a downsampled numpy matrix using nearest neighbor interpolation.
downsampled_im_NN_pixels = myImageResize(orig_im_pixels, 100, 175, 'nearest')

# Create an image from numpy matrix downsampled_im_NN_pixels.
downsampled_im_NN =
Image.fromarray(np.uint8(downsampled_im_NN_pixels.round()))

# Show the image.
downsampled_im_NN.show()

# Save the image.
downsampled_im_NN.save('downsampled_NN.tif')

# Upsample the numpy matrix to the original size using nearest neighbor
interpolation.
down_up_sampled_im_NN_pixels = myImageResize(downsampled_im_NN_pixels, 400,
400, 'nearest')

# Create an image from numpy matrix down_up_sampled_im_NN_pixels.
down_up_sampled_im_NN =
Image.fromarray(np.uint8(down_up_sampled_im_NN_pixels.round()))

# Show the image.
down_up_sampled_im_NN.show()

# Import myRMSE from MyImageFunctions
from MyImageFunctions import myRMSE

# Compute RMSE between original numpy matrix and down then upsampled nearest
neighbor version.
```

```
down_up_NN_RMSE = myRMSE( orig_im_pixels, down_up_sampled_im_NN_pixels)

print('\nDownsample/upsample with myimresize using nearest neighbor
interpolation = %f' % down_up_NN_RMSE)

##########################################################################
# Experiment 2: Downsample then upsample using bilinear interpolation.
##########################################################################

# Create a downsampled numpy matrix using bilinear interpolation.
downsampled_im_bilinear_pixels = myImageResize(orig_im_pixels, 100, 175,
'bilinear')

# Create an image from numpy matrix downsampled_im_bilinear_pixels.
downsampled_im_bilinear =
Image.fromarray(np.uint8(downsampled_im_bilinear_pixels.round()))

# Show the image.
downsampled_im_bilinear.show()

# Save the image.
downsampled_im_bilinear.save('downsampled_bilinear.tif')

# Upsample the numpy matrix to the original size using bilinear
interpolation.
down_up_sampled_im_bilinear_pixels =
myImageResize(downsampled_im_bilinear_pixels, 400, 400, 'bilinear')

# Create an image from numpy matrix down_up_sampled_im_bilinear_pixels.
down_up_sampled_im_bilinear =
Image.fromarray(np.uint8(down_up_sampled_im_bilinear_pixels.round()))

# Show the image.
down_up_sampled_im_bilinear.show()

# Compute RMSE between original numpy matrix and down then upsampled bilinear
version.
down_up_bilinear_RMSE = myRMSE( orig_im_pixels,
down_up_sampled_im_bilinear_pixels)

print('Downsample/upsample with myimresize using bilinear interpolation = %f'
% down_up_bilinear_RMSE)

##########################################################################
# Experiment 3: Upsample then downsample using nearest neighbor
interpolation.
##########################################################################

# Create an upsampled numpy matrix using nearest neighbor interpolation.
upsampled_im_NN_pixels = myImageResize(orig_im_pixels, 500, 625, 'nearest')

# Create an image from numpy matrix upsampled_im_NN_pixels.
upsampled_im_NN = Image.fromarray(np.uint8(upsampled_im_NN_pixels.round()))

# Show the image.
upsampled_im_NN.show()

# Save the image.
```

```
upsampled_im_NN.save('upsampled_NN.tif')

# Downsample the numpy matrix to the original size using nearest neighbor
interpolation.
up_down_sampled_im_NN_pixels = myImageResize(upsampled_im_NN_pixels, 400,
400, 'nearest')

# Create an image from numpy matrix up_down_sampled_im_NN_pixels.
up_down_sampled_im_NN =
Image.fromarray(np.uint8(up_down_sampled_im_NN_pixels.round()))

# Show the image.
up_down_sampled_im_NN.show()

# Compute RMSE between original numpy matrix and down then upsampled nearest
neighbor version.
up_down_NN_RMSE = myRMSE( orig_im_pixels, up_down_sampled_im_NN_pixels)

print('\nUpsample/downsample with myimresize using nearest neighbor
interpolation = %f' % up_down_NN_RMSE)

##########################################################################
# Experiment 3: Upsample then downsample using bilinear interpolation.
##########################################################################

# Create an upsampled numpy matrix using bilinear interpolation.
upsampled_im_bilinear_pixels = myImageResize(orig_im_pixels, 500, 625,
'bilinear')

# Create an image from numpy matrix upsampled_im_bilinear_pixels.
upsampled_im_bilinear =
Image.fromarray(np.uint8(upsampled_im_bilinear_pixels.round()))

# Show the image.
upsampled_im_bilinear.show()

# Save the image.
upsampled_im_bilinear.save('upsampled_bilinear.tif')

# Downsample the numpy matrix to the original size using bilinear
interpolation.
up_down_sampled_im_bilinear_pixels =
myImageResize(upsampled_im_bilinear_pixels, 400, 400, 'bilinear')

# Create an image from numpy matrix up_down_sampled_im_bilinear_pixels.
up_down_sampled_im_bilinear =
Image.fromarray(np.uint8(up_down_sampled_im_bilinear_pixels.round()))

# Show the image.
up_down_sampled_im_bilinear.show()

# Compute RMSE between original numpy matrix and up then downsampled bilinear
version.
up_down_bilinear_RMSE = myRMSE( orig_im_pixels,
up_down_sampled_im_bilinear_pixels)

print('Upsample/downsample with myimresize using bilinear interpolation = %f'
% up_down_bilinear_RMSE)
```

**MyImageFunctions.py**

```
import numpy as np
import math

def myImageResize(inImage_pixels, M, N, interpolation_method ):
# This function will resize the image based on the pixels and the
interpolation method for grayscaled images.
# Syntax:
#   up_down_sampled_im_bilinear_pixels =
myImageResize(upsampled_im_bilinear_pixels, 400, 400, 'bilinear')
#
# Input:
#    inImage_pixels = numpy matrix of grayscale image
#    M = number of rows
#    N = number of columns
#    interpolation_method = 'nearest' or 'bilinear' method
#
# Output:
#    out = Resized numpy matrix image
#
# History:
#   A. Fatali   10/23/22     Created
#
# Based off the TA's sample file
    Minput, Ninput = inImage_pixels.shape
    # create a new output matrix
    out = np.zeros(shape=(M, N))

    for m in range(M+1):  # int
        for n in range(N+1):   # int
            # loop  all the pixels from the out

            # 1. we estimate the index of input image -> estimate row and col
            m_inter = (((m-0.5)/M) * Minput) + 0.5   # float number,
            n_inter = (((n-0.5)/N) * Ninput) + 0.5

            if interpolation_method == 'nearest':
                # use nn
                # round()
                m_inter = round(m_inter)
                n_inter = round(n_inter)
                # print("m_inter: ", m_inter)
                # print("n_inter: ", n_inter)
                out[m-1, n-1] = inImage_pixels[m_inter-1, n_inter-1]

            elif interpolation_method == 'bilinear':
                # use the bilinear

                # find the 4 points
                # use the if else from the hw
                # m_inter -> m1, m2  which are the nearest row index
                if(m_inter == int(m_inter)):
                    m1 = m_inter-1
                    m2 = m_inter-1
                else:
                    if m_inter < 1:
                        m1, m2 = 1, 2
```

```
                    elif m_inter > Minput-1:
                        m1, m2 = Minput-2, Minput-1
                    else:
                        m1 = math.floor(m_inter-1)
                        m2 = math.ceil(m_inter-1)

                # n_inter -> n1, n2  which are the nearest col index
                if(n_inter == int(n_inter)):
                    n1 = n_inter-1
                    n2 = n_inter-1
                else:
                    if n_inter < 1:
                        n1, n2 = 1, 2
                    elif n_inter > Ninput-1:
                        n1, n2 = Ninput-2, Ninput-1
                    else:
                        n1 = math.floor(n_inter-1)
                        n2 = math.ceil(n_inter-1)

                p1 = inImage_pixels[m1, n1]
                p2 = inImage_pixels[m1, n2]
                p3 = inImage_pixels[m2, n1]
                p4 = inImage_pixels[m2, n2]
                p5 = 0

                p5 =
mybilinear(m1,n1,p1,m1,n2,p2,m2,n1,p3,m2,n2,p4,m_inter-1,n_inter-1,p5)
                out[m-1, n-1] = p5


    return out

def mybilinear(x1,y1,p1,x2,y2,p2,x3,y3,p3,x4,y4,p4,x5,y5,p5):
# This function preforms bilinear interpolation
# Syntax:
#   p5 = mybilinear(x1,y1,p1,x2,y2,p2,x3,y3,p3,x4,y4,p4,x5,y5,p5)
#
# Input:
#   x = the location in x
#   y = the location in y
#   p = the pixel value
#
# Output:
#   p5 = the pixel value
#
# History:
#   A. Fatali   10/22/22     Created
#
# Based off the TA's sample file
    # use eq from hw
    # apply eq on p1 and p3  -> t1
    # apply eq on p2 and p4  -> t2
    # ....         t1 and t2  -> p5
    p5_prime = (p3-p1)*((x5-x1)/(x3-x1))+p1
    p5_doubleprime = (p4-p2)*((x5-x2)/(x4-x2))+p2
    p5 = (p5_doubleprime-p5_prime)*((y5-y1)/(y2-y1))+p5_prime
    return p5
```

```
def myRMSE(first_im_pixels, second_im_pixels):
# This function is the root mean squared error (RMSE). It compares matrix of
two images.
# Syntax:
#   up_down_bilinear_RMSE = myRMSE(first_im_pixels, sec_im_pixels)
#
# Input:
#   first_im_pixels = the original image matrix
#   sec_im_pixels = the second image matrix
#
# Output:
#   sqrt(total) = The root mean squared error
#
# History:
#   A. Fatali   10/22/22     Created
#
# Based off the TA's sample file
    total = 0
    M,N = first_im_pixels.shape
    for m in range(0,M):
        for n in range(0,N):
            total += (first_im_pixels[m,n] - second_im_pixels[m,n])**2

    # now you have sum
    total = total / (M * N)

    return math.sqrt(total)
```