CSE 107 Lab 03: Image Resizing

Cyrus Arellano
Lab: Th 7:30-10:20pm
Haolin Liang
October 24, 2022

Abstract:

The purpose of this lab is to demonstrate what nearest neighbor and bilinear interpolation can do to an image when upsampling and downsampling an image. This lab will also help compare between the two and display images after upsampling and downsampling an image. RMSE values will also be recorded to help understand both interpolations and see errors between images.

Quantitative Results:

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

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

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

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

Quantitative Results:

	Nearest neighbor interpolation	**Bilinear interpolation**
Downsample then upsample	22.746414	16.835790
Upsample then downsample	0.000000	5.405959

Questions:

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?

When zooming in, it seems like bilinear interpolation looks more clearer, smoother and better quality than the nearest neighbor interpolation. This is because bilinear interpolation considers the weighted average to get its interpolated value. Nearest neighbor considers 1 pixel while bilinear considers 4 pixels, making it so much smoother.

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?

When comparing the two down then upsampled images, bilinear interpolation is much more clearer and less chunky than nearest neighbor, so bilinear looks better to me. This does agree with the RMSE values because bilinear RMSE value is 16.835790 while nearest neighbor is 22.746414. The lower the value, the better the smoother image and performance.

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?

When comparing the two up then downsampled image, the nearest neighbor looks a tiny bit clearer than the bilinear. They almost look exactly the same but it seems like nearest neighbor looks much clearer. This does agree with the RMSE values because nearest neighbor has an RMSE value of 0 while bilinear has an RMSE value of 5.405959. Because of these values, this does agree with the images because nearest neighbor looks a little bit more clearer.

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?

This is the case because we are considering one pixel and scaling that up. That one pixel value can be scaled up to a certain number so that we have that pixel value in the same cell it would normally be in but scaled up. Because we can reverse this easily, this can be scaled down using the same pixel values and same cells, causing the RMSE to be the value of zero.

What was the most difficult part of this assignment?

The most difficult part of this lab was trying to figure out the bilinear interpolation function. This is because I got confused with the indexing and some of the equations that were needed to calculate mybilinear.

Code:

MyImageFunctions.py

```
from PIL import Image, ImageOps

# Import numpy
import numpy as np
from numpy import asarray
import math





def myRMSE(I1, I2):


    M, N = np.shape(I1)
    rmse = 0

    #Helps calculate for RMSE
    for m in range(M):
        for n in range(N):
            rmse += (I1[m,n] - I2[m,n])**2
    rmse = np.sqrt(rmse/(M*N))

    return rmse




def myImageResize( inputPixels, M, N, interpolationName ):
```

```
    #Gets shape of original
    ro, co = inputPixels.shape

    #New shape of array
    resized = np.empty((M, N), float)


    #Helps estimate index of input image
    for row in range(M+1):
        for col in range(N+1):

            mt = (((row-0.5)/M) * ro) + 0.5
            nt = (((col-0.5)/N) * co) + 0.5



            if(interpolationName == "nearest"):
                        #Helps calculate for nearest neighbor. Uses round
                        mt = round(mt)
                        nt = round(nt)
                        resized[row-1][col-1] = inputPixels[mt-1][nt-1]

            elif(interpolationName == "bilinear"):


                        #Gets 4 points m1,m2,n1,n2. This is the bilinear
part of it.
                        if(mt == int(mt)):
                            m1 = mt-1
                            m2 = mt-1
                        else:
                            if(mt < 1 ):
                                m1 = 1-1
                                m2 = 2-1
                            elif(mt>ro-1):
                                m1 = ro-2
                                m2 = ro-1
                            else:
```

```
                                m1 = math.floor(mt-1)
                                m2 = math.ceil(mt-1)

                        if(nt == int(nt)):
                            n1 = nt-1
                            n2 = nt-1
                        else:
                            if(nt < 1 ):
                                n1 = 1-1
                                n2 = 2-1
                            elif(nt>co-1):
                                n1 = co-2
                                n2 = co-1
                            else:
                                n1 = math.floor(nt-1)
                                n2 = math.ceil(nt-1)

                        #Gets pixels from original image
                        p1 = inputPixels[m1][n1]
                        p2 = inputPixels[m1][n2]
                        p3 = inputPixels[m2][n1]
                        p4 = inputPixels[m2][n2]
                        x5 = mt -1
                        y5 = nt -1

                       #Goes into mybilinear function to help calculate
using all points
                        pq =
mybilinear(m1,n1,p1,m1,n2,p2,m2,n1,p3,m2,n2,p4,x5,y5)
                        resized[row-1][col-1] = pq
            #Returns new array
    return resized




def mybilinear(x1,y1,p1,x2,y2,p2,x3,y3,p3,x4,y4,p4,x5,y5):

    #Equation gotten from HW to help calculate for bilinear interpolation
```

```
    pri1 = (p3-p1)*((x5-x1)/(x3-x1)) + p1
    pri2 = (p4-p2)*((x5-x2)/(x4-x2)) + p2
    p5 = (pri2-pri1) * ((y5-y1)/(y2-y1)) + pri1

    return p5
```

test_myimresize.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)
```