CSE 107: Lab 03: Image Resizing.

Martin Urueta
LAB: Thursday 7:30pm-10:20pm
Liang, Haolin
October 24, 2022

Abstract:
In this lab, we implement a function that resizes a grayscale image. It will do 2 different functions: nearest neighbor or bilinear interpolation to generate the resized image. Resizing an image to a smaller size is called downsampling. Resizing an image to a larger size is called upsampling. During upsize and downsize our picture, we would compare our values using RMSE values to the original image.

Qualitative Results:

Figure 1: Original Image

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

Figure 3: Downsampled to size (100, 175) using bilinear interpolation.

Figure 4: Upsampled to size (500, 625) using nearest neighbor interpolation

Figure 5: Upsampled to size (500, 625) using bilinear interpolation.

Quantitative Results:

A table showing the RMSE values between the original image and the down/upsampled and up/downsampled versions for both nearest neighbor and bilinear interpolation. For example, a table like
this:

	Nearest neighbor interpolation	Bilinear interpolation
Downsample then upsample	22.807706	16.882088
Upsample then downsample	0.000000	5.449098

Questions:

1. The downsample looks more blocky using nearest neighor interpolation. THe bilinear is still a little pixelated, looks better than nearest neighbor. This is because nearest neighbor just gets the pixel value based off the closest distance the other pixels are around it. Bilinear will get the average of the other pixels and gets a better estimate of the pixel value.
2. The down then up sample using nearest neighbor interpolation is very pixelated but image looks a little better. The bilinear result was less pixelated but looks blurry. I like how Bilinear looks less pixelated than nearest neighbor interpolation. The RMSE values agrees with me as 22 for (NN) is greater than 16 (Bilinear)
3. The up then downsample image using nearest neighbor interpolation showed a identical image as the original. The billinear looks great as well but it is a bit blurry. I like nearest neighbor since it looks more identical to the original image. The RMSE value agrees with me as 0 (NN) is less than 5 (Bilinear)
4. When nearest neighbor when up scaling it gets the closest value and becomes very pixelated. But, when we up scale then down scale to the original size we dont get any pixel value difference since the closest pixels are the original pixel value.
5. The most difficult part of the assignment was figuring out how to start it since having to figure out how what variable are inputed and how we need to do it a certain way.

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)
```

MyImageFunctions.py

```
import numpy as np
from numpy import asarray
# For sqrt(), floor()
import math

def myImageResize( inImage_pixels, M, N, interpolation_method ):
# This function resize the image based on the pixels and interpolation
method for grayscaled images. The methods are nearest and bilinear
#
# Syntax:
#   up_down_sampled_im_bilinear_pixels =
myImageResize(upsampled_im_bilinear_pixels, 400, 400, 'bilinear')

#
# Input:
#   inImage_pixels = the image with pixel values that range from 0-255
#   M = The Row of the image to resize
#   N = The Col of the image to resize
#   interpolation_method = the method to do the resize
#
# Output:
#   up_down_sampled_im_bilinear_pixels = the resized image that was created
with a method
#
# History:
#   M. Urueta   10/22/22     Created
    data = np.zeros(shape = (M, N))
    row, col = inImage_pixels.shape
```

```
    for m in range (1,M+1):
        for n in range (1, N+1):
            x = ((row/M)*(m - 0.5)) + 0.5
            y = ((col/N)*(n - 0.5)) + 0.5
            if interpolation_method == 'nearest' :
                rx = round(x)
                ry = round(y)
                data[m-1,n-1] = inImage_pixels[rx-1, ry-1]
            elif interpolation_method == 'bilinear':
                if (x == int(x)):
                    m1 = x - 1
                    m2 = x - 1
                else:
                    if x < 1:
                        m1, m2 = 1, 2
                    elif x > row - 1:
                        m1, m2  = row - 2, row - 1
                    else:
                        m1 = math.floor(x - 1)
                        m2 = math.ceil(x - 1)
                # y -> n1, n2  which are the nearest col index
                if(y == int(y)):
                    n1 = y - 1
                    n2 = y - 1
                else:
                    if y < 1:
                        n1, n2 = 1, 2
                    elif y > col - 1:
                        n1, n2 = col - 2, col - 1
                    else:
                        n1 = math.floor(y - 1)
                        n2 = math.ceil(y - 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,x - 1,y
- 1,p5)
```

```
                data[m - 1, n - 1] = p5
    return data

def mybilinear(x1,y1,p1,x2,y2,p2,x3,y3,p3,x4,y4,p4,x5,y5,p5):
# This function preforms the bilinear interpolation to get the pixel value
for the interpolated pixel
#
# 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 intensity value
#
#
# Output:
#   p5 = the intensity value
#
# History:
#   M. Urueta   10/22/22     Created
    p5_prime = (p3 - p1) * ((x5 - x1) / (x3 -x1)) + p1
    p5_doubleprime = (p4 - p2) * ((x5 - x2) / (y2 -y1)) + p2
    p5 = (p5_doubleprime - p5_prime) * ((y5 - y1) / (y2 - y1)) + p5_prime
    return p5

def myRMSE(first_im_pixels, sec_im_pixels):
# This function is the root mean squared error. compare the matrix of two
image
#
# Syntax:
#   up_down_bilinear_RMSE = myRMSE( first_im_pixels, sec_im_pixels)
#
# Input:
#   first_im_pixels = the original image matrix
#   sec_im_pixels = the reconstruct image matrix
#
#
# Output:
#   up_down_bilinear_RMSE = The root mean squared error
#
```

```
# History:
#   M. Urueta   10/22/22     Created
    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] - sec_im_pixels[m,n]) ** 2
    total = total / (M * N)

    return math.sqrt(total)
```