CSE 107: Lab 03: Image Resizing.

Valentinno Cruz

LAB: Thu 4:30-7:20

Yuxin Tian

October 21st, 2022

Abstract:

The resizing of Images is a technique used for manipulating an images. This allows the image to be downsized(shrunk) or upsized(enlarged). There are several ways of doing this process, with some being more complicated than others. These methods include Nearest Neighbor, Bilinear, and Bicubic interpolation. These methods essentially all calculate each resized pixel with the use of the locations surrounding the given pixel. However, some methods are much quicker, while others are more accurate. For instance, Nearest Neighbor is processed much quicker and with less power in comparison to Bilinear. This is because Bilinear has to process much more information than that of Nearest Neighbor.

Qualitative Results:

Downsampled (NN) (100,175)

Downsampled Bilinear (100, 175)

Upsampled (NN) (500, 625)

Upsampled Bilinear (500, 625)

Quantitative Results:

	N.N. interpolation	Bilinear interpolation
Downsample then upsample	30.639599	24.838016
Upsample then downsample	23.793549	12.761159

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?

   At first glance it may be difficult to distinguish the two images, but upon further inspection it is evident that the Bilinear photo has a slightly higher contrast. This is due to Bilinear using a lot more resources when selecting each pixel value. This in turn creates a more dynamic picture than that of Nearest Neighbor.

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 Nearest Neighbor upscaled image looks a lot more accurate to the original than that of the Bilinear image. I was really surprised to see this because I was almost certain the Bilinear image would be much more accurate. My RMSE value for Bilinear is smaller, so I think maybe RMSE might not be the best way to gauge images that are restored from a smaller size.

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 Bilinear image looks a little more clear than that of the Nearest Neighbor. This does make sense in comparision to the RMSE values.

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?

   A RMSE is 0 when there is a perfect fit in the data. This means that the up then downsampled image that used Nearest Neighbor is as accurate to the original image as possible

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

   Coming up with a way to code the split Bilinear and Nearest Neighbor depending on the string that's called is what took me the longest to implement.

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 3\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

```python
import numpy as np
import math

#.....................................................
#               Root Mean Squared Error
#.....................................................
# This function is used for resizing images
# both nearest neighbor and bilinear methods are used
# depending on which is called.

def myImageResize(inImage_pixels, M, N, interpolation_method):
    # set up empty array.
    ImageOut = np.zeros((M, N))

    # determining the length of original image
    r, c = inImage_pixels.shape[:2]
    # M and N are new row and col of image required after scaling
    cRat = c / N
    rRat = r / M

# ----------------------------------------
#      Nearest neighbor interpolation
# ----------------------------------------
    if interpolation_method == 'nearest':
        for i in range(M):
            for j in range(N):
                y, x = int(j * cRat), int(i *  rRat)
                ImageOut[i][j] = inImage_pixels[x][y]

# ----------------------------------------
#          Bilinear interpolation
# ----------------------------------------
    elif interpolation_method == 'bilinear':
        for i in range(M):
            for j in range(N):
                y, x = j * cRat, i * rRat
                x_floor, y_floor = int(x), int(y)
                x_ceil, y_ceil = min(r - 1, int(math.ceil(x))), min(c - 1, int(math.ceil(y)))

                # Set our pixel location values
                p1 = inImage_pixels[x_floor, y_floor]
                p2 = inImage_pixels[x_floor, y_ceil]
                p3 = inImage_pixels[x_ceil, y_floor]
                p4 = inImage_pixels[x_ceil, y_ceil]

                # the output image is set to the new values
                ImageOut[i, j] = mybilinear(x_floor, y_floor, p1,
                                            x_floor, y_ceil, p2,
                                            x_ceil, y_floor, p3,
                                            x_ceil, y_ceil, p4,
                                            x, y)
    return ImageOut

#.....................................................
#               Root Mean Squared Error
#.....................................................
#  Here we are taking a comparision between two sets of values
#  This is done by looping through the pixels and comparing
#  the actual difference between the estimated and measured vals

def myRMSE(first_im_pixels, second_im_pixels):
    # Take the dimensions from the first pixels.
    M, N = np.shape(first_im_pixels)

    # Find the RMSE by looping through pixels and comparing.
    rmse = 0
    for m in range(M):
        for n in range(N):
            rmse += (first_im_pixels[m,n] - second_im_pixels[m,n])**2

    rmse = np.sqrt(rmse/(M*N))
    return rmse

```

MyImageFucntion.py

```python
#.....................................................
#               Bilinear Method
#.....................................................
# With this function we will compute the values of
# the pixels in the interpolated position using
# the other pixel values

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

    # Set up the min and max of x and y
    Min_x = min(np.floor(x1),np.floor(x2),np.floor(x3),np.floor(x4))
    Max_x = max(np.ceil(x1),np.floor(x2),np.floor(x3),np.floor(x4))
    Min_y = min(np.floor(y1),np.floor(y2),np.floor(y3),np.floor(y4))
    Max_y = max(np.ceil(y1),np.floor(y2),np.floor(y3),np.floor(y4))

    # if min x & max x are equal
    if (Min_x == Max_x):
        return p1
    # if min y & max y are equal
    if (Min_y == Max_y):
        return p1

    # if y min & max value are equivalent
    if (Min_x == Max_x):
        totx = (Max_y - y5) * p1 + (y5 - Min_y) * p4
        return totx

    # if y min & max value are equivalent
    if (Min_y == Max_y):
        min_max_y = (Max_x - x5) * p1 + (x5 - Min_x) * p4
        return min_max_y

    val_1 = (Max_x - x5) * p1 + (x5 - Min_x) * p2
    val_2 = (Max_x - x5) * p3 + (x5 - Min_x) * p4
    Bilinear = (Max_y - y5) * val_1 + (y5 - Min_y) * val_2

    return Bilinear

```

Task 3: Creating a gradient grayscale image. Computing the image average.

Figure 3. The Gradient Image

Questions for task 3:

1. What is the average pixel value in your gradient image?
    a. **127.5**

2. Why did you expect to get this value from the gradient image?
    a. **Because the average between 0 & 255 is 127.5**

3. What was the most difficult part of this task?
    a. **Figuring out how to code the average**

Task3.py

```python
# Import pillow
from PIL import Image, ImageOps
# Import numpy
import numpy as np
from numpy import asarray
import math

# The size of the gradient image.
rows = 100
cols = 256


# Create a numpy matrix of this size.
im_pixels = np.zeros(shape=(rows, cols))


# 256 values between 255 - 0 (back - white)
x = np.linspace(255, 0, 256)

#repeat the vector 100 times
image2_grad = np.tile(x, (100, 1)).T

#flip image 90degrees
a = np.rot90(image2_grad, 1, (1,0))
image_pix = asarray(a) # set as an array

#Generate image from array.
newImage = Image.fromarray(np.uint8(image_pix))

```

```python
#--------------------------
# get average pixel value
#--------------------------
x = np.size(image_pix, 0)
y = np.size(image_pix, 1)



total =0
for i in range(x):
    for j in range(y):
        total+=image_pix[(i,j)]
    b=x*y
mean=(total)/b
print("Mean Value is: ",np.mean(mean))

newImage.show()
newImage.save('image.tif')

```

Task3.py