CSE 107: Lab 03: Image Resizing.

<your name>
LAB: T 10:30-1:20pm
Yuxin Tian
October 17, 2022

Abstract:

In a few sentences, describe the purpose of this lab. (Do not mention specific Python functions. This description should be at a very high level.)

Qualitative Results:

Include figures with captions of the following four resized versions of `Lab_03_image.tif`:

- Downsampled to size (100, 175) using nearest neighbor interpolation.
- Downsampled to size (100, 175) using bilinear interpolation.
- Upsampled to size (500, 625) using nearest neighbor interpolation.
- 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	22.746414	16.839830
Upsample then downsample	0.000000	5.414557

<Note your values will probably not be the same as mine.>

Questions:

<Your answers to the assignment questions>

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

```
# MyImageFunctions.py

# Import pillow
from PIL import Image, ImageOps

# Import numpy
import numpy as np
from numpy import asarray

# For sqrt(), floor()
import math

def myImageResize( inImage_pixels, M, N, interpolation_method ):

< your implementation>

def myRMSE( first_im_pixels, second_im_pixels ):

< your implementation>

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

< your implementation>
```