Image Processing Application – Spring 2021

Lecturer: Professor Lee Guee Sang

Student: Le Hoai Duy – 218493

**Homework2**

1. Take a gray image and:
2. Generate different bit-depth image as Figure 2.21 of the slide:

**Code:**

from PIL import Image

grayImage = Image.open("./rose.jpg")

# Generate in [20,16,8,6,4,2,1] bit depth

image20bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=20)

image16bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=16)

image8bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=8)

image6bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=6)

image4bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=4)

image2bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=2)

image1bit = grayImage.convert("P", palette=Image.ADAPTIVE, colors=1)

# Show and save result

import numpy as np

import matplotlib.pyplot as plt

fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(12, 12), gridspec\_kw = {'wspace':0, 'hspace':0.1})

np.vectorize(lambda ax:ax.axis('off'))(axes)

ax0, ax1, ax2, ax3= axes.flat

ax0.imshow(grayImage, cmap=plt.cm.gray)

ax0.set\_title('24-bit', fontsize=18)

ax1.imshow(image20bit, cmap=plt.cm.gray)

ax1.set\_title('20-bit', fontsize=18)

ax2.imshow(image16bit, cmap=plt.cm.gray)

ax2.set\_title('16-bit', fontsize=18)

ax3.imshow(image8bit, cmap=plt.cm.gray)

ax3.set\_title('8-bit', fontsize=18)

# Save result.

fig.savefig("D:/result\_HW2\_1.png",bbox\_inches='tight')

fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(12, 12), gridspec\_kw = {'wspace':0, 'hspace':0.1})

np.vectorize(lambda ax:ax.axis('off'))(axes)

ax4, ax5, ax6, ax7 = axes.flat

ax4.imshow(image6bit, cmap=plt.cm.gray)

ax4.set\_title('6-bit', fontsize=18)

ax5.imshow(image4bit, cmap=plt.cm.gray)

ax5.set\_title('4-bit', fontsize=18)

ax6.imshow(image2bit, cmap=plt.cm.gray)

ax6.set\_title('2-bit', fontsize=18)

ax7.imshow(image1bit, cmap=plt.cm.gray)

ax7.set\_title('1-bit', fontsize=18)

# Save result.

fig.savefig("D:/result\_HW2\_2.png",bbox\_inches='tight')

**Result:**

A picture containing plant, flower, bouquet, decorated

Description automatically generated

A picture containing text

Description automatically generated

**Analysis of the results**: when keeping the spatial resolution constant and changing the gray-level resolution, the larger the number of bits, the sharper the image quality is. This is difficult to identify with the naked eye if the number of bits is reduced but still large (e.g. 20-bit or 16-bit) but will be evident when the number of bits is getting smaller (e.g. 6-bit to 1-bit).

1. Generate different resolution as Figure 2.20 of the slide:

**Code:**

grayImage.save("rose1.jpg", quality=50)

grayImage.save("rose2.jpg", quality=25)

grayImage.save("rose3.jpg", quality=12)

grayImage.save("rose4.jpg", quality=6)

grayImage.save("rose5.jpg", quality=3)

**Result**

A picture containing plant, flower, bouquet

Description automatically generated

**Analysis of the result**: when keeping the gray-level resolution constant and changing the spatial resolution, the higher the sampling frequency, the sharper the image quality is. These images show the dimensional proportions between various sampling densities, but keep their size are the same to make it easy to see the effects resulting from a reduction in the number of samples.

1. Restore the images in (2) by interpolation:

Code:

import PIL

grayImage = Image.open("./rose.jpg")

width, height = grayImage.size

#Change spatical resolution of the original image

grayImage1 = grayImage.resize((int(width/2), int(height/2)))

grayImage2 = grayImage.resize((int(width/4), int(height/4)))

grayImage3 = grayImage.resize((int(width/8), int(height/8)))

grayImage4 = grayImage.resize((int(width/16), int(height/16)))

grayImage5 = grayImage.resize((int(width/32), int(height/32)))

#Restore by nearest neighbor interpolation

grayImage3\_nearest = grayImage3.resize((width, height),PIL.Image.NEAREST)

grayImage4\_nearest = grayImage4.resize((width, height),PIL.Image.NEAREST)

grayImage5\_nearest = grayImage5.resize((width, height),PIL.Image.NEAREST)

#Restore by bilinear interpolation

grayImage3\_bilinear = grayImage3.resize((width, height),PIL.Image.BILINEAR )

grayImage4\_bilinear = grayImage4.resize((width, height),PIL.Image.BILINEAR )

grayImage5\_bilinear = grayImage5.resize((width, height),PIL.Image.BILINEAR )

#Show image and save result

fig, axes = plt.subplots(ncols=2, nrows=3, figsize=(12, 18), gridspec\_kw = {'wspace':0, 'hspace':0.01})

np.vectorize(lambda ax:ax.axis('off'))(axes)

ax0, ax1, ax2, ax3, ax4, ax5 = axes.flat

ax0.imshow(grayImage3\_nearest, cmap=plt.cm.gray)

ax0.set\_title('Nearest neighor interpolation', fontsize=18)

ax2.imshow(grayImage4\_nearest, cmap=plt.cm.gray)

ax4.imshow(grayImage5\_nearest, cmap=plt.cm.gray)

ax1.imshow(grayImage3\_bilinear, cmap=plt.cm.gray)

ax1.set\_title('Bilinear Interpolation', fontsize=18)

ax3.imshow(grayImage4\_bilinear, cmap=plt.cm.gray)

ax5.imshow(grayImage5\_bilinear, cmap=plt.cm.gray)

fig.savefig("D:/result\_HW2\_4.png",bbox\_inches='tight')

**Result:**

A picture containing flower, plant, clothes

Description automatically generated

**Analysis of the results:** The results are quite similar to Figure 2.25. Interpolation technique can be used to restore images with a resolution close to the original image. On the contrary, the images had a much smaller resolution than the original, interpolation technique gave very poor results. Bilinear is better than nearest neighbor interpolation.

1. Take a color image:
2. For a patch of image, print out the real pixel representations as in the slide.

**Code:**

from PIL import Image

import numpy as np

colorImage = Image.open("./london.jpg")

area = (colorImage.size[0]/2, colorImage.size[1]/2,colorImage.size[0]/2+ 7, colorImage.size[1]/2+7)

cropImage = colorImage.crop(area)

cropImage.save("./crop\_london.jpg")

a = np.array(cropImage)

print(a[:,:,0]) # r channel

print(a[:,:,1]) # g channel

print(a[:,:,2]) # b channel

**Result:**



