

CS 4732

MACHINE VISION

PROJECT 1

Image Resolution

#### INSTRUCTOR

Dr. Sanghoon Lee

**Your Name: Zaid Khan**

**KSU ID: 001089961**

**1. ABSTRACT**

In this project, I used python, to downsample 1024x1024 images by half 3 times, going to 512x512 to 256x256 to 128x128. I then upsampled them so it would be easier to see the results and finally put them all onto one 2048x2048 pixel canvas. For the second part of the project, I took an input image, converted it to grayscale, and then lowered the gray levels of the image down from 8 bits to 1 bit. The assignment wanted from 8 to 5 bits, but the difference was too little that I decided to go to 1. I then put all 8 images on a 4x2 image canvas. For both of these, I used pillow to read the image files and convert the new arrays to images and write them to files, and I used numpy to convert the images to arrays to make them easier to work with.

**2. Test RESULTS**

**2.1 Test Results for Image Negative**

Figure 1(a), **................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

Figure 2,**................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

|  |  |
| --- | --- |
| **(a)** | **(b)** |
| **(c)** | **(d)** |

**Figure 1:** (a) Original 1024x1024 grayscale image (rose.jpg), (b-d) Downscale of original images to 512x512 to 256x256 to 128x128.

|  |  |
| --- | --- |
| **(a)** | **(b)** |
| **(c)** | **(d)** |

**Figure 2:** (a) Original 1024x1024 grayscale image (rose.jpg), (b-d) Downscale of original images to 512x512 to 256x256 to 128x128.

**2.2 Test Results for Image Upside-down**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**...............................................................................................................................................**

**3. CODES**

**3.1 Code for downsampling images**

#name: Zaid Khan

#KSUID: 001089961

#Project 1

import numpy as np

# import image

from PIL import Image as pillow

# Down-sample any color or grayscale image from a 1024x1024 pixel-sized image

# (such as rose.jpg) to 512x512, 256x256, 128x128, respectively. Then, up-sample

# the images generated before, back to 1024x1024 pixels. Save each image to show

# the effect.

# Down-sample the image to 512x512, 256x256, and 128x128 pixels

def downSample(img):

    # Initialize a new array to hold the new images

    downSampledImages = []

    # Convert the image to a numpy array

    image = np.array(img)

    for i in range(3):

        #get the size of the image

        sideSize = image.shape[0]

        #get the new size of the image by dividing by 2(1024, 512, 256, 128)

        newSideSize = sideSize // 2

        #check for rgb or grayscale image and make the new image array depending on that

        if len(image.shape) == 3:

            # RGB image

            newImageArray = np.zeros((newSideSize, newSideSize, image.shape[2]), dtype=np.uint8)

        else:

            # Grayscale image

            newImageArray = np.zeros((newSideSize, newSideSize), dtype=np.uint8)

        #loop through the new image array and get the average of the 4 pixels in the original image

        for y in range(newSideSize):

            for x in range(newSideSize):

                #get the average of the 4 pixels in the original image(multiply each value by .1 to avoid overflow)

                newVal = ((image[y\*2,x\*2]\*.1+image[y \* 2, x \* 2+1]\*.1)/2 +(image[y\*2+1,x\*2]\*.1+image[y\*2+1,x\*2+1]\*.1)/2)/2

                #divide by .1 to get the actual value

                newVal = newVal/.1

                #cant use this because it will overflow as max value is 32 for uint8

                # newVal = np.uint8(((image[y\*2,x\*2]+image[y \* 2, x \* 2+1]) +(image[y\*2+1,x\*2]+image[y\*2+1,x\*2+1]))/4)

                #set the new value to the new image array

                newImageArray[y, x] = newVal

        #append the new image to the downsampled images

        downSampledImages.append(pillow.fromarray(newImageArray))

        #make the new image our input image for the next iteration

        image = newImageArray

    #return the downsampled images

    return downSampledImages

# Up-sample the images back to 1024x1024 pixels

def upSampleTo1024(img):

    # Initialize a new array to hold the new images

    upSampledImages = []

    for i in range(3):

        # Convert the image to a numpy

        image = np.array(img[i])

        #check for rgb or grayscale image and make the new image array depending on that

        if len(image.shape) == 3:

            # RGB image

            newImageArray = np.zeros((1024, 1024, image.shape[2]), dtype=np.uint8)

        else:

            # Grayscale image

            newImageArray = np.zeros((1024, 1024), dtype=np.uint8)

        #get the size of the image

        ogSize = image.shape[0]

        #use the size to get the scale factor

        scale = 1024 // ogSize

        #loop through the new image array and set the value of the pixel to the value of the pixel in the original image

        for y in range(1024):

            for x in range(1024):

                newImageArray[y, x] = image[y // scale, x // scale]

        #append the new image to the upsampled images

        upSampledImages.append(pillow.fromarray(newImageArray))

    #return the upsampled images

    return upSampledImages

#Make a 2x2 image with the original image and the upsampled downsampled images

#not required so I used in built functions but it is a nice way to see the difference

def fullComparisonOfFinalImages(upSampledImages, img):

    # Initialize a new array to hold all the images

    allImages = []

    # Append the original image to the array

    allImages.append(img)

    # Append the upsampled images to the array

    for i in range(3):

        allImages.append(upSampledImages[i])

    # Create a new image with a large enough to contain all images

    newImage = pillow.new("RGB", (1024 \* 2, 1024 \* 2))

    # Paste the images into the new image

    x\_offset = 0

    for i in range(2):

        y\_offset = 0

        for j in range(2):

            newImage.paste(allImages[i + j \* 2], (x\_offset, y\_offset))

            y\_offset += 1024

        x\_offset += 1024

    # Save the new image

    newImage.save("machinevision/Assignment 1/comparisons/rosecomparison.jpg")

# Load the original image

img = pillow.open('machinevision/Assignment 1/images/rose.jpg')

# Down-sample the image to 512x512, 256x256, and 128x128 pixels

downSampledImages = downSample(img)

# Save the down-sampled images

downSampledImages[0].save('machinevision/Assignment 1/images/rose512x512.jpg')

downSampledImages[1].save('machinevision/Assignment 1/images/rose256x256.jpg')

downSampledImages[2].save('machinevision/Assignment 1/images/rose128x128.jpg')

# Up-sample the images back to 1024x1024 pixels

upSampledImages = upSampleTo1024(downSampledImages)

# Save the up-sampled images

upSampledImages[0].save('machinevision/Assignment 1/images/rose512x512to1024x1024.jpg')

upSampledImages[1].save('machinevision/Assignment 1/images/rose256x256to1024x1024.jpg')

upSampledImages[2].save('machinevision/Assignment 1/images/rose128x128to1024x1024.jpg')

# Compare the original image with the up-sampled images

fullComparisonOfFinalImages(upSampledImages, img)

**3.2 Code for quantizing image**

#name: Zaid Khan

#KSUID: 001089961

#Project 1

from PIL import Image

import numpy as np

# Change the gray level of an 8-bit gray level image (that has initially 256 gray

# levels) to 128, 64, and 32 gray level images, respectively. Save each image to

# show the effect.

def quantizer(image\_array):

    # Initialize a new array with the same shape as the original image

    finalizedImages = []

    reduced\_image = np.zeros\_like(image\_array)

    # Loop through for 7-1 bits

    for i in range(7):

        # Calculate the number of levels and the quantization

        bits = 7 - i

        levels = 2 \*\* bits

        quantization\_step = 256 // levels

        print(quantization\_step)

        # integer division of the matrix by the quantization step, then multiplying by the quantization step to brighten the image

        quantized\_image\_np = (image\_array // quantization\_step) \* quantization\_step

        reduced\_image = quantized\_image\_np

        # Convert the array to an image

        reduced\_image = Image.fromarray(reduced\_image)

        finalizedImages.append(reduced\_image)

    return finalizedImages

#Make a 4x2 image with the original image and the upsampled downsampled images

#not required so I used in built functions but it is a nice way to see the difference

def fullComparisonOfFinalImages(reducedImages, img):

    # Initialize a new array to hold all the images

    allImages = []

    # Append the original image to the array

    allImages.append(img)

    # Append the upsampled images to the array

    for i in range(reducedImages.\_\_len\_\_()):

        allImages.append(reducedImages[i])

    # Create a new image large enough to contain all images

    x, y = img.size

    newImage = Image.new("RGB", (x \* 4, y\*2))

    # Paste the images into the new image

    x\_offset = 0

    k=0

    for i in range(2):

        y\_offset = 0

        for j in range(4):

            print(k)

            newImage.paste(allImages[k], (y\_offset, x\_offset))

            print(x\_offset/x, y\_offset/y)

            k += 1

            y\_offset += y

        x\_offset += x

    # Save the new image

    newImage.save("machinevision/Assignment 1/comparisons/rosequantizecomparison.jpg")

# Load the original image

image\_path = "machinevision/Assignment 1/images/rose.jpg"

image = Image.open(image\_path).convert("L")  # Ensure it's grayscale

imageArray = np.array(image)

# Reduce to 7-1 gray levels manually

reducedImages = quantizer(imageArray)

#save the images

for i in range(reducedImages.\_\_len\_\_()):

    reducedImages[i].save(f"machinevision/Assignment 1/images/rosequantizeas{2\*\*(7-i)}.jpg")

    # print(2\*\*(7-i))

fullComparisonOfFinalImages(reducedImages, image)