108062213 顏浩昀
HW1

Question 1. Color Quantization and Dithering

In this question, I'm given a 24-bits color jpeg image, and be requested to do color quantization with median-cut algorithm and error diffusion dithering.

1. Preprocess

Initially, I use PIL library to read the image and transform to numpy(short for np) array.

```
originalImg = Image.open('./img/Lenna.jpeg')
rgbArray = np.array(originalImg)
```

2. Median-cut color quantization

These are the step I implement this algorithm:

A. Reshape the input image into a 2D array of pixels.
B. Initialize a list of cubes with the entire array of pixels.
C. While the number of cubes is less than 2 to the power of the number of bits, divide each cube into two smaller cubes and add them to the list of cubes.

```
while len(cubes)<2**bits:
  newCubes = []
  for cube in cubes:
    leftCube, rightCube = util.SplitCube(cube)
    newCubes.append(leftCube)
    newCubes.append(rightCube)
  cubes = newCubes
```

```
def SplitCube(cube):
  ranges = np.max(cube, axis=0) - np.min(cube, axis=0)
  dim = np.argmax(ranges)
  sortedcube = cube[cube[:, dim].argsort()]
  medianidx = len(cube) // 2
  leftcube = sortedcube[:medianidx]
  rightcube = sortedcube[medianidx:]
  return leftcube, rightcube
```

D. Compute the mean color of each cube.

```
colors = [util.MedianColor(cube) for cube in cubes]
```

```
def MedianColor(cube):
  return np.array(np.mean(cube, axis=0).astype(int))
```

E. Replace each pixel in the original image with its closest median color.

```python
for y in range(img.shape[0]):
  for x in range(img.shape[1]):
    pixel = img[y, x]
    newRgbArray[y, x] = util.ClosestColor(pixel, colors)
```

```python
def ClosestColor(color, color_list):
  idx = np.argmin(distance.cdist([color], color_list, metric='euclidean'))
  return color_list[idx]
```

F. Compute the color quantization MSE error by comparing the original and quantized images.

```python
def MSE(oriImg, newImg):
  return np.mean((oriImg - newImg) ** 2)
```

The MSE in case 3-bit is roughly about 64.8677, and the MSE in the case 6- bit is about 22.9243

```
MSE for 3 bits:  64.86774288308497
MSE for 6 bits:  22.924317684131818
```

(median-cut 3-bits color)

(median-cut 6-bits color)

The output image is save as png in /out directory.

108062213 顏浩昀
HW1

3. Error diffusion dithering

These are the step I implement the dithering method:

A. Create a copy of the original image and initialize an empty output image.

B. For each pixel in the original image, find its closest color in the Look-Up-Table(Colors in previous question).

C. Compute the quantization error by subtracting the original color from the quantized color.

D. Distribute the quantization error to neighboring pixels using pre-defined coefficients.

E. Repeat steps B, C and D for all the pixels in the image.

```
for y in range(h):
    for x in range(w):
      pixel = oriImg[y, x]
      newPixel = util.ClosestColor(pixel, colors)
      ditherImg[y, x] = newPixel
      err = pixel - newPixel

      if x < w - 1:
        oriImg[y, x + 1] = oriImg[y, x+1] + err * 7 / 16
      if x > 0 and y < h - 1:
        oriImg[y + 1, x - 1] = oriImg[y+1, x-1] + err * 3 / 16
      if y < h - 1:
        oriImg[y + 1, x] = oriImg[y+1, x] + err * 5 / 16
      if x < w - 1 and y < h - 1:
        oriImg[y + 1, x + 1] = oriImg[y+1, x+1] + err * 1 / 16
```

F. Compute the color quantization MSE error by comparing the original and dithered images.

The MSE in case 3-bit is roughly about 76.6354, and the MSE in the case 6- bit is about 40.4803

```
MSE for 3 bits:  76.63539496875501
MSE for 6 bits:  40.48026491481066
```

(error diffusion dithering 3-bits color)

(error diffusion dithering 6-bits color)

4. Discussion

I can get less score of MSE by median-cut. To compare image result, I found the pictures through error diffusion dithering would have some grains, it might be the noise produced by error. On the other hand, median-cut algorithm may let image lost some detail, it's significantly be found in higher bits result.

Question 2. Interpolation

In this question, I need implement two different algorithms to implement image upsample interpolation.

1. Preprocess

Initially, I use opencv(cv2) library to read the image.

```
img = cv2.imread('./img/bee.jpeg')
```

2. Nearest-neighbor interpolation

These are the step I implement the dithering method:

A. Extract the height, width and depth of the original image.

```
for dd in range(d):
  for y in range(newh):
    for x in range(neww):
      oriy = int(np.floor(y/4.0))
      orix = int(np.floor(x/4.0))
      upsampleImg[y, x, dd] = img[oriy, orix, dd]
```

B. Create a new np array with four times the height and width of the original image.

C. calculate the corresponding pixel location in the original image with np.floor(x/4.0) and np.floor(y/4.0), repeat it for each pixel and each channel.

3. Bilinear interpolation

A. Extract the height, width and depth of the original image.

B. Create a new np array with four times the height and width of the original image.

C. Calculate the closest four pixel in original image

```
def bilinearInterpolation(img, x, y, d):
  h, w, dep = img.shape
  x1, y1, x2, y2 = int(np.floor(x)), int(np.floor(y)), int(np.floor(x))+1, int(np.floor(y))+1

  if x1<0 or x2>=img.shape[1] or y1<0 or y2>=img.shape[0]:
    return img[max(0, min(h-1, int(np.round(y)))), max(0, min(w-1, int(np.round(x)))), d]

  q11, q12, q21, q22 = img[y1, x1, d], img[y2, x1, d], img[y1, x2, d], img[y2, x2, d]
  w1 = (x2 - x) * (y2 - y)
  w2 = (x - x1) * (y2 - y)
  w3 = (x2 - x) * (y - y1)
  w4 = (x - x1) * (y - y1)
  interpolationValue = (w1*q11 + w2*q21 + w3*q12 + w4*q22)
  return interpolationValue
```

D. Calculate four weight corresponding to the pixel distance to these 4 pixels.

E. Calculate the new pixel value by multiple weight and original pixel value.

F. Repeat step C to E until all pixels visited.

4. Discussion

Nearest neighbor upsampling simply duplicates each pixel in the original image, resulting in a blocky appearance. This method is fast and easy to implement, but it can produce a low-quality output. On the other hand, bilinear upsampling is a more sophisticated technique that produces smoother results.

(NN interpolation)

(bilinear interpolation)

Question 3. Photo enhancement

In the question, I need to convert the image color space, and implement the gamma transform.

1. Preprocess

Initially, I use matplotlib to read the image.

```
img = plt.imread('./img/lake.jpeg')
```

2. RGB2YIQ and Histogram of Y channel

A. Create a transformation matrix and dot matrix and image.

```
def RGB2YIQ(img):
  transform = np.array([[0.299, 0.587, 0.114],
                        [0.596, -0.274, -0.322],
                        [0.211, -0.523, 0.312]])
  shape = img.shape
  yiq = np.dot(img.reshape(-1,3), transform.T).reshape(shape)
  return yiq
```

B. Count each pixel in Y channel

```
def Hist(channel):
  histogram = np.zeros(256)
  for i in range(channel.shape[0]):
    for j in range(channel.shape[1]):
      intensity = math.floor(channel[i][j])
      histogram[intensity] += 1
  return histogram
```

C. Plot the histogram

3. Gamma Transform

A. Define the gamma value as 4.0

```
yGamma = util.GammaTransform(y, 4.0)
```

B. Normalization the image

```
imgNorm = img.astype(float) / 255
```

C. Gamma transform with power normalized image and gamma

```
imgGamma = np.power(imgNorm, gamma)
```

D. Denormalized the image

```
imgGamma *= 255.0
```

4. YIQ2RGB

A. Create a transform matrix with the inverse matrix of the subproblem 1.

B. Dot the matrix and the image

```
def YIQ2RGB(img):
  transform = np.linalg.inv(np.array([[0.299, 0.587, 0.114],
                                      [0.596, -0.274, -0.322],
                                      [0.211, -0.523, 0.312]]).transpose())
  shape = img.shape
  rgb = np.dot(img.reshape(-1,3), transform).reshape(shape)
  return rgb
```

C. Plot the image

5. Discussion

After the gamma transformation, the histogram distribute in the larger range and the summit become lower. Moreover, the image become darker and show more detail.