COMP9517 Assignment1 Report

Name: Xiaowei Zhu. Z Number: z5102903

Programming library Instructions:

```
# import the library
import numpy as np
```

Numpy is used to deal with the numpy array/matrix

```
import cv2
```

We use cv2(OpenCV) to deal with image reading, writing

```
import matplotlib.pyplot as plt
```

Plt is used to draw the image and show the histogram

```
import glob
```

Glob is used to get the files in the directory

```
from collections import Counter
```

Counter is used to count the frequency of element in the list

```
from statistics import mean
```

Mean is used to get the mean of a array

Task1:

Background Analysis:

We got 2 images and an equation. The equation could be used to swap the three (RGB)color-bands into one band.

$$I(x,y) = 0.299 * r(x,y) + 0.587 * g(x,y) + 0.114 * b(x,y)$$

Implementation:

Use OpenCV to read the image as matrix

Get a duplicated matrix of this image to implement in order to avoid changing the original image matrix

Get the shape(x, y coordinate values)

Retrieve all the pixel points of the matrix(for loop)

During the retrieving, perform the equation and swap the original pixel value with this equation answer.

So, we get a new matrix. (should be a grey-level image with identical RGB value)

Programming and comments:

According to the requirement of different tasks in this assignment, I define different functions

Here, Task1 function deal with logic in task1 and the right picture show how I call the images under a directory with a for loop.

```
# define the task1 function
def Task1(img_pixels,task1_img):
    #get the original image shape value
    img_shape = img.shape
    # max x coordinate value
    x_max = img_shape[0]
    # max y coordinate value
    y_max = img_shape[1]

    # double for loop to retrieve all the pixels in the image
    for x in range (0,x_max):
        for y in range(0,y_max):

            # get the original RGB band value
            px = img_pixels[x,y]
            # get the new band value through ths equation provided
            new_band = 0.299*px[0]+0.587*px[1]+0.114*px[2]
            # swap the original band value with this new_band
            # implement it in a copied original image matrix in order to avoid changing the origianl matrix
            task1_img[x,y]=new_band
    # return the updated image matrix
    return task1_img
```

```
# get all the image files in the directory
for file in glob.glob("*.jpg"):
    # read the img as a matrix
    img = cv2.imread(file,1)
    #cv2.imshow("image",img)
    # get the matrix shape
    shape = img.shape
    print(f"Loading the image:{file}, Shape:{shape} \n")
    print (f"Start Time:{time.ctime()}")

    # copy the image matrix
    img_pixels = np.copy(img)
    task1_img = np.copy(img)
    ####################################
    ## Call task1 function
    task1_answer = Task1(img_pixels,task1_img)
    print(task1_answer)
    # write down the task1 image and store in local directory
    # define the name
    t1_file_name = f"task1_{file}.jpg"
    cv2.imwrite(t1_file_name,task1_answer)
    print("Task1 finished")
```

Analysis of intermediate results:

```
Loading the image:light_rail.jpg, Shape:(1080, 1080, 3)

Start Time:Sun Oct  6 01:28:35 2019
[[[244 244 244]
  [244 244 244]
  [244 244 244]
  ...
  [244 244 244]
  [244 244 244]
  [244 244 244]]

 [[244 244 244]
  [244 244 244]
  [244 244 244]
  ...
```

```
Loading the image:dog.jpg, Shape:(2048, 1536, 3)

Start Time:Sun Oct  6 01:44:30 2019
[[[176 176 176]
  [176 176 176]
  [176 176 176]
  ...
  [134 134 134]
  [134 134 134]
  [133 133 133]]

 [[176 176 176]
  [176 176 176]
  [176 176 176]
```

Image I

We execute taks1 function under a loop (get all jpg image in the directory)
Write the return image(matrix) into local directory.
Left: is the grey-level light rail and dog image with identical RGB band value.

Task2:

Background Analysis:

In Task2, we need copy the matrix of image I from task1. We need to get the most frequent pixel value in a window around the pixel (the logic is similar to convolution). Then use the most frequent local pixel value to replace the central pixel in the corresponding window in image J. Do the retrieving.

Implementation:

I use two duplicated image matrixes of I, one is for replacing value, another is for getting most frequent value. This method can avoid the implementation of replacing value influence the most frequent value.

I define 3 different window sizes, 3X3, 9X9, 21X21.

For the edging pixel points, we only consider its meaningful neighbours(the neighbours are not out of index value).

I create a function to get the most frequent pixel value, the function converts all multi dimension array to single dimension array (with ravel method). The calculate the most frequent value with Counter function.

This most frequent value is easy to be identified from Histogram.

Assign the value to the central pixel.

Programming and comments:

I define two function, one is used to collect the most frequent neighbourhood value under a certain window size. The other is used to swap the most frequent value to original value(ffollowing task2 requirements

```python
# define a sub matrix to get the neighbours around one point/pixel
# x, y is the coordinate
# window_size is the neighbourhood size that we need to define
# the copied image matrix from original image
def get_most_freq_neighbour(x,y,window_size,image_matrix):

    # get the winsow_size matriax
    s = (window_size-1)//2

    # avoid out of index
    if y-s < 0:
        row_s = 0
    else:
        row_s = y-s
    if x-s < 0:
        col_s = 0
    else:
        col_s = x-s
    # get the subbmatrix based on current location
    sub_matrix = image_matrix[row_s:y+s+1 ,col_s:x+s+1]
    # change 3D to 1D
    oneD_matrix = sub_matrix.ravel()
    # cuonter the elements
    ele_counts = Counter(oneD_matrix)
    # get the most frequent one element
    most_freq = ele_counts.most_common(1)[0][0]
    # return the most frequent pixel value
    return most_freq
```

```python
# define a function to deal with task2
# get_most_freq_neighbour function is used here
def Task2(window_size,task1_answer):
    # get a copied image matrix of task1
    task1_img = task1_answer
    image_matrix=np.copy(task1_img)
    # another copied image matrix of task1
    task2_matrix = np.copy(task1_img)
    # get the shape of matrix and (x,y)
    shape2 = task2_matrix.shape
    y2_max = shape2[0]
    x2_max = shape2[1]
    # retrieve task2_matrix
    for x in range (0,x2_max):
        for y in range(0,y2_max):
            # get the frequent neighbour
            most_freq_neigh = get_most_freq_neighbour(x,y,window_size,image_matrix)
            # swap the current pixel value with the most frequency neighbour
            task2_matrix[y,x] = most_freq_neigh
    # return task2_matrix
    return task2_matrix
```

Analysis of intermediate results :

The same as Task1, we perform Task2 function under the same loop, and write the answer/image into local directory. However, one important thing, here we add 3 different window size with a for loop. That means every matrix from task1 should be perform under 3, 9, and 21 window size.

With the list loop, I assign 3 different window size to Task2 function automatically, can it will return 3 different image matrixes based on the window size, I also write the matrix as an image.

Btw, I also print the histogram of sub matrix (consists by all the pixels in the window size)

```
#####################################
## TASK2
# try three different window size
for window_size in [3,9,21]:
    # call task2 function
    task2_answer = Task2(window_size,task1_answer)
    # define the filename
    t2_file_name = f"task2_{file}_{window_size}X{window_size}.jpg"
    # write down the image of task2
    cv2.imwrite(t2_file_name,task2_answer)
    print("Task2 finished")
```

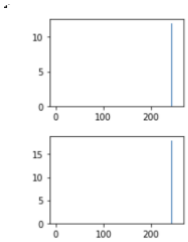
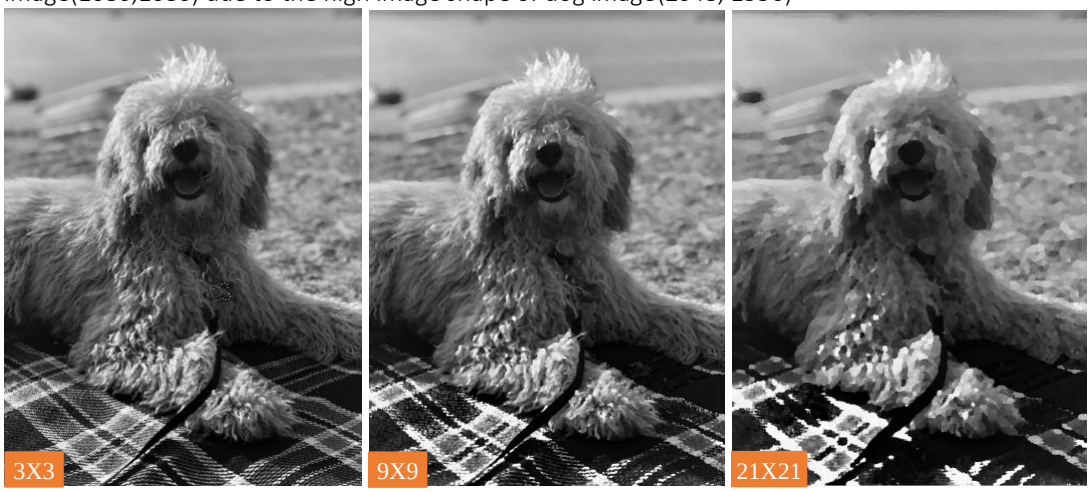


Image J: We can find the obvious difference between these 3 images (from clear to blurred) due to different window sizes.

Image J: The same logic in dog's image, but the difference is not so obvious comapred to light rail image(1080,1080) due to the high image shape of dog image(2048, 1536)



Task3

Background Analysis:

We make a copy of the original image A and the image matrix from task2 (J). In task3 we need to mirror the same window size from copied image J to copied image A matrix, because they have the same matrix logic.

In addition, we should get the average RGB channel value in the mirrored window size in image B. and replace B(x, y) with the average intensities in each corresponding band.

Implementation, Programming and comments:

Following task3's requirements, I define two function here, one is used to get the list of pixels coordinates in certain window size who have the same value as current pixel value.

Another one is used to calculate the average RGB band value of the pixel points (in original image) under the list we get from last function (mirrored window size area).

And we need to copy two image matrixes from original image A. One is used for retrieving all average RGB band value, the other one is used for swapping value. (this method can avoid the implementation of last step affect the average value of next window, because the matrix will be changed by every step)

Task3 function is used to combine previous two functions through retrieving all the points(coordinates) in image matrix. In addition, the RGB band values are swapped during this process.The logic of executing Task3 function is the same as Task2. Under a loop of 3, 9 and 21 window sizes.

```
# get a list that contains the coordinators of the point whch has the same value
# (x,y) is the coordinate
def get_same_value_list(x,y,window_size,image_J):

    # get the value of current point
    current_value = image_J[y,x][0]
    # get the winsow_size matriax
    s = (window_size-1)//2
    # get the subbmatrix based on current location
    if y-s < 0:
        row_s = 0
    else:
        row_s = y-s
    if x-s < 0:
        col_s = 0
    else:
        col_s = x-s
    # create the list used to store coordinate
    common_list = []
    # use two for loops to retrieve all the points
    for yy in range (row_s, y+s+1):
        for xx in range(col_s,x+s+1):
            # avoid out of index
            if yy >= image_J.shape[0]:
                yy = image_J.shape[0]-1
            if xx >= image_J.shape[1]:
                xx = image_J.shape[1]-1
            # if neighbour's value equal to current value
            if  image_J[yy,xx][0] == current_value :
                # append to common list
                common_list.append((yy,xx))
    # return this common list
    # the common list will be used in the calculate_averge_pixel function
    return common_list
```

```
# this function is used to calculate the avergae RGB band of one piexl point
# image_B is the ori
def calculate_averge_pixel(image_B,common_list):

    # create a list used to store RGB band value
    avg_outcome=[]
    # create RGB band list used to store the neighbour's RGB value
    R_average_list = []
    G_average_list = []
    B_average_list = []

    # for loop to append RGB band value to respective list
    for (y,x) in common_list:
        R_average_list.append(image_B[y,x][0])
        G_average_list.append(image_B[y,x][1])
        B_average_list.append(image_B[y,x][2])

    #average_list=np.array(average_list)
    # if R_average_list is not empty
    if R_average_list:
        # get the mean of the R band list
        R_avg_value = mean(R_average_list)
        # append the value to avg_outcome
        avg_outcome.append(R_avg_value)
    # if G_average_list is not empty
    if G_average_list:
        # get the mean of the G band list
        G_avg_value = mean(G_average_list)
        # append the value to avg_outcome
        avg_outcome.append(G_avg_value)
    # if B_average_list is not empty
    if B_average_list:
        # get the mean of the B band list
        B_avg_value = mean(B_average_list)
        # append the value to avg_outcome
        avg_outcome.append(B_avg_value)

    return avg_outcome
```

```
# combine get_same_value_list and calculate_averge_pixel functions
# swap all the original image pixel values
def Task3(task2_answer,img,w_size):
    # get the copied image matrix from original image
    image_B = np.copy(img)
    # copy task2 outcome
    image_J = np.copy(task2_answer)
    image_T3 = np.copy(img)
    # get the image shape and it's max coordinate values
    shape2 = image_J.shape
    y2_max = shape2[0] -1
    x2_max = shape2[1] - 1
    # for loop, retrieve the point in task2 outcome and original image
    for x in range (0,x2_max):
        for y in range(0,y2_max):
            # call get_same_value_list function, get the common list
            common_list = get_same_value_list(x,y,w_size,image_J)
            # call function, calculate the avergae RGB value
            avg_list = calculate_averge_pixel(image_B,common_list)
            R_avg_value = avg_list[0]
            G_avg_value = avg_list[1]
            B_avg_value = avg_list[2]
            #swap the RGB value in copied original image matrix
            image_T3[y,x][0] = R_avg_value
            image_T3[y,x][1] = G_avg_value
            image_T3[y,x][2] = B_avg_value
    # return matrix
    return image_T3
```

```
####################################
## TASK3
    # call the task3 function
    task3_answer = Task3(task2_answer,img,window_size)
    # define the filename
    t3_file_name = f"task3_{file}_{window_size}X{window_size}.jpg"
    # write down the image of task3
    cv2.imwrite(t3_file_name,task3_answer)
    print("Task3 finished")
    print (time.time())
```

Analysis of intermediate results :

Oil painting-like image B, performance becomes better from 3X3 to 21X21 window size. And the performance of light rail is better than Dog due to the different shapes of image matrix. For example, the shape of dog image is 2048, 1536. The shape of light rail is 1080,1080. 21X21 implementation area is smaller in dog image compared to light rail image. Actually, the oil painting implementation is combining the RGB band values in the window size area. If the image shape is bigger which means the original RGB bands value in this window are more similar. So, combining the neighbour RGB band value doesn't make sense. The same logic for different window sizes. The RGB band values become more similar in smaller window sizes.
In conclusion, this is the reason why the performance of light rail is better than dog with the same window size and big window size contribute to better performance for the same image.

The original outputs are stored in Google Drive, you can also click this link to check:
https://drive.google.com/drive/folders/1R6UPvL7-nfKJln_XOuJbW0fvk2pgr366?usp=sharing