A picture containing text, metalware, gear

Description automatically generated

Department of Electronic & Telecommunication Engineering

University of Moratuwa

**Assignment 01**

**EN2550 – Fundamentals of Image Processing and Machine Vision**

|  |  |
| --- | --- |
| Name | K. D. S. D. Kuruppu |
| Index No. | 190338C |
| GitHub | <https://github.com/SupunDK/Fundamentals-of-Image-Processing.git> |

05th March 2022

**Question 1**

Chart, line chart

Description automatically generatedThe given intensity transformation has been applied on the Emma Watson’s image in this question. The graph of the supplied intensity transformation in as follows,

In the above graph, the intensity values from 0 to 50 and from 150 to 255 are unchanged. The middle intensity values from 50 to 150 is mapped linearly to 100 to 255. Therefore, we will be able to see a transformation in pixels with mid-intensities while pixels with high and low intensities remain unchanged. The resulting transformation on the image is as follows,

A collage of a person

Description automatically generated with low confidenceIn the original image, the mid-intensity pixels mostly lie in the shadows of Emma Watson’s face. Therefore, through the transformation, the portion of Emma Watson’s face which had a shadow is now illuminated in the transformed image. Most of the hair and the bright portion of Emma Watson’s face is unchanged in the transformed image.

A more natural-looking transformed image with illumination on the shadowed portion of Emma Watson’s face will be able to be achieved by fine-tuning the range that the mid-intensities map to in the intensity transformation look up table.

**Question 2**

In this question a brain proton density image has been provided for us. After studying about brain proton density images, white matter and gray matter in the brain, I noticed that the provided image is an inverted image of the usual brain proton density image format. Therefore, I inverted the given image as the first step.

When accentuating white matter of the image, I decided to enhance it while keeping the gray matter of the image intact. Therefore, the information in the white matter will be pronounced while the information at the boundaries of the white matter and the gray matter is preserved. The same strategy is used for accentuating the gray matter of the image.

A picture containing text

Description automatically generatedA picture containing text

Description automatically generatedIn white matter accentuation, it was found that pixel intensities of the white matter region in the inverted image is in the range of 50 to 200. Therefore, the intensity transformation for this purpose was designed to increase the intensities in the region from 50 to 200 while keeping the remaining of the intensities unchanged. The result of the transformation is shown below.

The intensities from 50 to 200 is mapped to 60 to 255. The implementation of the intensity transformation is as follows,

1. segment\_1 = np.array([i for i in range(0, 51)])
2. segment\_2 = np.array([1.3\*i - 5 for i in range(51, 201)])
3. segment\_3 = np.array([i for i in range(201, 256)])
4. white\_matter\_transform = np.concatenate((segment\_1, segment\_2, segment\_3), axis=0)

In the gray matter accentuation, the pixel intensities of the gray matter region in the inverted image are in the range from 0 to 75. Therefore, the intensities from 0 to 75 is mapped to 0 to 50 and the remaining intensities are unchanged in the implemented intensity transformation. The results are shown in the next page.

The implementation of the intensity transformation is as follows,

1. segment\_1\_gray = np.array([2\*i/3 for i in range(0, 76)]).astype(np.uint8)
2. segment\_2\_gray = np.array([i for i in range(76, 256)]).astype(np.uint8)
3. gray\_matter\_transform = np.concatenate((segment\_1\_gray, segment\_2\_gray), axis=0)

A picture containing text

Description automatically generated

**Question 3**

A picture containing indoor

Description automatically generatedIn this question, the given image was initially converted from the RGB colour space to the LAB colour space. Then, gamma correction on the L dimension of the image was carried out for different gamma values. The following diagram shows the effect of gamma correction on the image for two gamma values with respect to the original.

The images obtained for different gamma values shows that the images become brighter when the gamma value is less than 1 and the images become darker when the gamma value is greater than 1. This can be further observed by the intensity distributions of the images. For gamma values less than 1, the intensity distribution has expanded and shifted to higher intensity values while for gamma values greater than 1, it has contracted and concentrated on lower intensity values. Furthermore, the images become more brighter or darker when the gamma value increases or decreases from a gamma value of 1, respectively.

This behaviour is as expected because when the gamma value is greater than 1, the intensities from a wider range from the origin is mapped to a tighter range from the origin. Similarly, when the gamma value is less than 1, the intensities from a tighter range from the origin is mapped to a wider range from the origin. Therefore, as observed, the lightness of the image (brightness) will decrease and increase for the two cases. Moreover, when the gamma value moves further away from 1, the range that is subjected to change become wider and tighter for the cases of gamma being greater than 1 and gamma being less than 1, respectively. As a result, it will decrease the brightness and increase the brightness more, for the two cases respectively.

The following shows the important parts from the python implementation,

1. lady\_img = cv.imread(r"Images\highlights\_and\_shadows.jpg")
3. lady\_img\_orig\_lab = cv.cvtColor(lady\_img, cv.COLOR\_BGR2LAB)
4. lady\_img\_lab = cv.cvtColor(lady\_img, cv.COLOR\_BGR2LAB)
5. lady\_img\_lab\_2 = cv.cvtColor(lady\_img, cv.COLOR\_BGR2LAB)
7. gamma = 0.5
8. gamma\_transform = np.array([(i/255.0)\*\*gamma \* 255.0 for i in range(0, 256)]).astype(np.uint8)
10. gamma\_2 = 2
11. gamma\_transform\_2 = np.array([(i/255.0)\*\*gamma\_2 \* 255.0 for i in range(0, 256)]).astype(np.uint8)
13. lady\_img\_lab[:,:,0] = cv.LUT(lady\_img\_lab[:,:,0], gamma\_transform)
14. lady\_img\_lab\_2[:,:,0] = cv.LUT(lady\_img\_lab\_2[:,:,0], gamma\_transform\_2)

**Question 4**

The implemented histogram equalization function is as follows,

1. def histogram\_equalization(gray\_img, max\_val = 255):
2. gray\_img\_flat = gray\_img.flatten()
3. element\_count = np.array([np.count\_nonzero(gray\_img\_flat == i) for i in range(0, 256)])
4. coefficient = max\_val / gray\_img\_flat.shape[0]
6. sums = np.array([np.sum(element\_count[0:i]) for i in range(1, 257)])
8. hist\_equalization\_transform = ( coefficient \* sums ).astype(np.uint8)
10. gray\_img\_transformed = cv.LUT(gray\_img, hist\_equalization\_transform)
12. fig, ax = plt.subplots(1, 1, figsize = (5, 5))
13. ax.plot(hist\_equalization\_transform)
14. ax.set\_title("Histogram Equalization Transform")
15. ax.set\_xlabel("Input Intensity")
16. ax.set\_ylabel("Output Intensity")
17. ax.grid()
19. return gray\_img\_transformed

The results of the implemented histogram equalization function, the OpenCV histogram equalization along with the original is depicted below.

Calendar

Description automatically generated

**Question 5**

The implementations of the two zooming algorithms are as follows,

1. Nearest-Neighbour
2. def nearest\_neighbour\_zoom(img, scale):
3. num\_rows = round(img.shape[0] \* scale)
4. num\_columns = round(img.shape[1] \* scale)
6. scale = num\_rows / img.shape[0]
8. zoomed\_img = np.zeros((num\_rows, num\_columns, img.shape[2]), np.uint8)
10. for z in range(0, img.shape[2]):
11. for i in range(0, num\_rows):
12. for j in range(0, num\_columns):
13. zoomed\_img[i, j, z] = img[ min(round(i/scale), img.shape[0]-1), min(round(j/scale), img.shape[1]-1), z]
15. return zoomed\_img
16. Bilinear Interpolation
17. def bilinear\_interpolation(img, scale):
18. num\_rows = round(img.shape[0] \* scale)
19. num\_columns = round(img.shape[1] \* scale)
21. scale = num\_rows / img.shape[0]
23. zoomed\_img = np.zeros((num\_rows, num\_columns, img.shape[2]), np.uint8)

26. for i in range(0, num\_rows):
27. for j in range(0, num\_columns):
28. neighbours = []
30. if( (i/scale) == int(i/scale) and (j/scale) == int(j/scale) ):
31. for z in range(0, img.shape[2]):
32. zoomed\_img[i, j, z] = img[int(i/scale), int(j/scale), z]
34. continue
35. elif( (i/scale) == int(i/scale) ):
36. neighbours.append([int(i/scale), int(j/scale)])
37. neighbours.append([int(i/scale), min(int(j/scale)+1, img.shape[1]-1)])
38. elif( (j/scale) == int(j/scale) ):
39. neighbours.append([int(i/scale), int(j/scale)])
40. neighbours.append([min(int(i/scale)+1, img.shape[0]-1), int(j/scale)])
41. else:
42. neighbours.append([int(i/scale), int(j/scale)])
43. neighbours.append([min(int(i/scale)+1, img.shape[0]-1), int(j/scale)])
44. neighbours.append([int(i/scale), min(int(j/scale)+1, img.shape[1]-1)])
45. neighbours.append([min(int(i/scale)+1, img.shape[0]-1), min(int(j/scale)+1, img.shape[1]-1)])
47. viable\_neighbours = []
49. for k in range(len(neighbours)):
50. if (neighbours[k] not in viable\_neighbours):
51. viable\_neighbours.append(neighbours[k])
53. for z in range(0, img.shape[2]):
54. if ( len(viable\_neighbours) == 1 ):
55. intensity = img[viable\_neighbours[0][0], viable\_neighbours[0][1], z]
56. elif ( len(viable\_neighbours) == 2 ):
57. if ( viable\_neighbours[0][0] != viable\_neighbours[1][0] ):
58. intensity = img[ viable\_neighbours[0][0], viable\_neighbours[0][1], z ] \* abs(i/scale - viable\_neighbours[1][0]) + img[ viable\_neighbours[1][0], viable\_neighbours[1][1], z ] \* abs(i/scale - viable\_neighbours[0][0])
59. else:
60. intensity = img[ viable\_neighbours[0][0], viable\_neighbours[0][1], z ] \* abs(j/scale - viable\_neighbours[1][1]) + img[ viable\_neighbours[1][0], viable\_neighbours[1][1], z ] \* abs(j/scale - viable\_neighbours[0][1])
61. elif ( len(viable\_neighbours) == 4 ):
62. intensity\_1 = img[ viable\_neighbours[0][0], viable\_neighbours[0][1], z ] \* abs(i/scale - viable\_neighbours[1][0]) + img[ viable\_neighbours[1][0], viable\_neighbours[1][1], z ] \* abs(i/scale - viable\_neighbours[0][0])
63. intensity\_2 = img[ viable\_neighbours[2][0], viable\_neighbours[2][1], z ] \* abs(i/scale - viable\_neighbours[3][0]) + img[ viable\_neighbours[3][0], viable\_neighbours[3][1], z ] \* abs(i/scale - viable\_neighbours[2][0])
65. intensity = intensity\_1 \* abs(j/scale - viable\_neighbours[2][1]) + intensity\_2 \* abs(j/scale - viable\_neighbours[0][1])
67. zoomed\_img[i, j, z] = intensity
69. return zoomed\_img

The two algorithms performance was measured by calculating the normalised sum of squared differences (SSD) between the output of the algorithms and a given zoomed version of the input image. The implementation of the normalised SSD is as follows,

1. def ssd(img1, img2):
2. if img1.shape != img2.shape:
3. return -1
5. num\_pixels = 1
7. for i in range(len(img1.shape)):
8. num\_pixels \*= img1.shape[i]
10. return np.sum( ( (img1 - img2)\*\*2).flatten() ) / num\_pixels

The calculated normalised SSD values for the two algorithms are as follows,

|  |  |  |
| --- | --- | --- |
| Image | Nearest Neighbour | Bilinear Interpolation |
| Image 1 | 40.11174 | 39.25703 |
| Image 2 | 16.79297 | 16.21177 |

Therefore, from the above results, we can see that the error of the Bilinear Interpolation method is less than that of the Nearest Neighbour method. As a result, we can deduce that the Bilinear Interpolation is a better method of zooming compared to the Nearest Neighbour method.

**Question 6**

The following is the result from using the Sobel filter with the existing filter2D method in OpenCV.

A collage of a person

Description automatically generated with low confidence

My implementation of the filter2D function in the OpenCV library is as follows,

1. def filter\_image(img, filter):
2. upper\_padding = int( (filter.shape[0] - 1) / 2 )
3. lower\_padding = filter.shape[0] - 1 - upper\_padding
4. left\_padding = int( (filter.shape[1] - 1) / 2 )
5. right\_padding = filter.shape[1] - 1 - left\_padding
7. img\_shape\_rows = img.shape[0]
8. img\_shape\_columns = img.shape[1]
10. img = np.append(img, np.zeros((img\_shape\_rows, left\_padding)), axis=1 )
11. img = np.append(np.zeros((img\_shape\_rows, right\_padding)), img, axis=1 )
12. img = np.append(img, np.zeros((upper\_padding, img\_shape\_columns+left\_padding+right\_padding)), axis=0)
13. img = np.append(np.zeros((lower\_padding, img\_shape\_columns+left\_padding+right\_padding)), img, axis=0)
15. start\_index\_column = left\_padding
16. end\_index\_column = img.shape[1] - 1 - right\_padding
17. start\_index\_row = upper\_padding
18. end\_index\_row = img.shape[0] - 1 - lower\_padding
20. filtered\_img = np.zeros((img.shape[0], img.shape[1]))
22. for i in range(start\_index\_row, end\_index\_row+1):
23. for j in range(start\_index\_column, end\_index\_column+1):
24. for filter\_i in range(filter.shape[0]):
25. for filter\_j in range(filter.shape[1]):
26. filtered\_img[i, j] += filter[filter\_i, filter\_j] \* img[i+(filter\_i-upper\_padding), j+(filter\_j-left\_padding)]
28. return filtered\_img

The Sobel filtered images using my implementation of the filter2D function is as follows,

Calendar

Description automatically generatedA collage of a person

Description automatically generated with low confidenceTherefore, from the above results, it is visible that the output given by the filter2D function in the OpenCV library, and the output given by my implementation are identical. Then, the Sobel filtering using the convolution property was carried out. The filters and were used for the stage 1 and 2 of Sobel horizontal filtering respectively. Similarly, for Sobel vertical filtering, the filters and were used for the two stages in the mentioned order. The results are depicted on the right.

The gradient magnitude image obtained via the above Sobel filtered images is shown on the next page.

A picture containing text

Description automatically generatedTherefore, by the obtained results, it is clear that the Sobel filtering done via the convolution property is identical to that of the directly filtered version. The reason for the above observation is the associative property of the convolution operation.

**Question 7**

The final segmentation mask, foreground image and the background image are as follows,

A group of yellow flowers

Description automatically generated with low confidence

Gaussian blur was applied to the extracted background image with a kernel size of (9,9) and a standard deviation of 10. Then, the resulting background was added with the extracted foreground image to obtain an enhanced image. The resulted enhanced image is as follows,

A group of yellow flowers

Description automatically generated with medium confidenceThe reason for the background just beyond the edge of the flower being quite dark in the enhanced image is as follows,

The extracted background image has black pixels in the places where the foreground appeared in the original image. Therefore, when gaussian blur was applied to the background, the pixels at the edges of those completely black pixels obtained intensities closer to 0, resembling the black colour. As a result, when the extracted foreground image is added with this modified background image, the resulting enhanced image obtained a darkness at the edges of the flowers.