Visual Computing Fundamentals
Image Processing
Assignment 1

Adrian Opheim

October 23, 2017

Task 1: Theory questions

a *Histogram equalisation is an algorithm that creates a transformation τ_{heq} by exploiting an image histogram. Explain in your own words:*

(a) *How can we see that an image has low contrast when looking at its histogram?*
An image with low contrast will have a narrow histogram, reflecting the fact that there is little difference between light and dark areas in a scene, see Figure 1

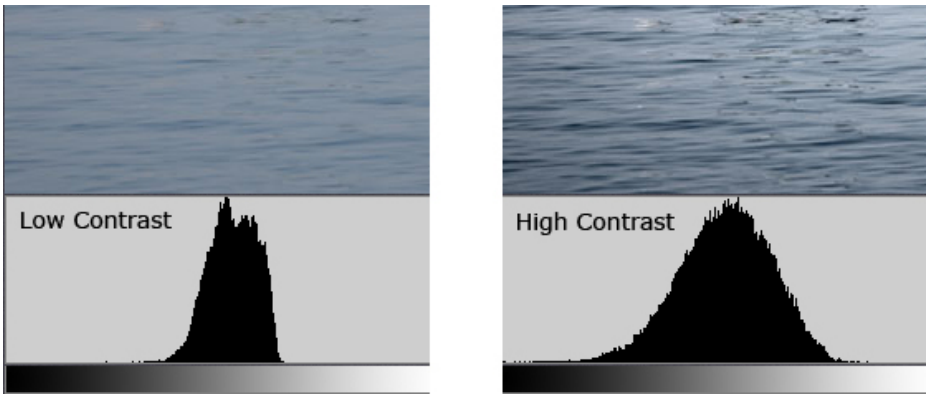


Figure 1: Histogram differences in high and low contrast

(b) *What effect does τ_{heq} have when applied on an image?*
Histogram equalization normally "stretches" the image's histogram, by this improving the contrast of the image.

(c) *What happens when histogram equalisation is applied multiple times on the same image in sequence?*

b *Histogram equalization by hand on 4-bit image*

1b) 4-bit 3×4 image:

12	3	1	9
3	0	4	3
2	6	15	2

r_k intens.	$n_k = H(r_k)$ occurrences
0	1
1	1
2	2
3	3
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	1
13	0
14	0
L-1 → 15	1
	12

Probability Density Function:

$$f(r_k) = \frac{n_k}{m \times n} = \frac{n_k}{3 \cdot 4}$$

Cumulative probability distribution:

$$F(r_k) = \sum_{j=0}^{k} f(r_k)$$

Scaling to desired range of output gray levels:

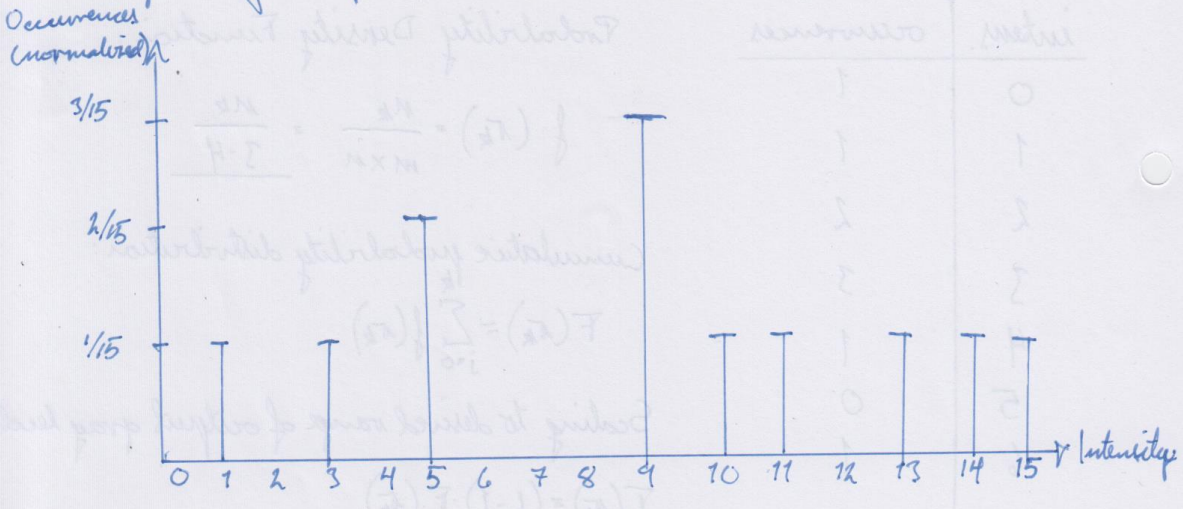
$$T(r_k) = (L-1) \cdot F(r_k)$$

r_k	n_k	$f(r_k)$	$F(r_k)$	$T(r_k)$
0	1	1/12	1/12	15/12 = 1
1	1	1/12	2/12	30/12 = 3
2	2	2/12	1/3	15/3 = 5
3	3	3/12	7/12	35/4 = 9
4	1	1/12	2/3	10
5	0	0	2/3	10
6	1	1/12	3/4	45/4 = 11
7	0	0	3/4	11
8	0	0	3/4	11
9	1	1/12	5/6	25/2 = 13
10	0	0	5/6	13
11	0	0	5/6	13
12	1	1/12	11/12	55/4 = 14
13	0	0	11/12	14
14	0	0	11/12	14
15	1	1/12	1	15

The transformed image:

14	9	3	13
9	1	10	9
5	11	15	5

Corresponding histogram:



c *The convolution operator is associative. What does this mean, and how is this property beneficial for convolution?*

The convolution operator is associative, meaning that

$$(f * g) * v = f * (g * v) \tag{1}$$

This is beneficial becuase it means that changing the order of subsequent convolution operations does not change the overall result. [1]

d Determine if the convolution kernel is separable computing their rank. If a kernel is separable, also provide the separated row and coloumn vector.

(a) Gaussian kernel. Performing Gaussian elimination. R_i meaning "row i":

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

$$R_2 - 2 \cdot R_1, R_3 - R_1 :$$

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

Number of non-zero rows remaining after Gaussian elimination, corresponding to the rank of the matrix: 1. This means that the kernel is separable.
The separated row and coloumn vectors:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\end{bmatrix} \quad \text{and} \quad \frac{1}{16}\begin{bmatrix}1\\ 2\\ 1\end{bmatrix}$$

The outer product $\otimes$ of the two becomes the original kernel:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\end{bmatrix} \otimes \frac{1}{16}\begin{bmatrix}1\\ 2\\ 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

(b) Kernel approximating the Laplacian:

$$\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$R_2 = R_2 - R_1, \quad R_3 = R_3 - R_1 : \begin{bmatrix}1 & 1 & 1\\ 0 & -9 & 0\\ 0 & 0 & 0\end{bmatrix}$$

$$-\frac{1}{9} \cdot R_2 : \begin{bmatrix}1 & 1 & 1\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}$$

[1] Taken from http://onlinelibrary.wiley.com/doi/10.1002/9781118393550.app1/pdf

Rank of the kernel: 2 $\implies$ not separable.

Why is it helpful to have convolution kernels that are separable?
Having separable convolution kernels is advantageous because of the computional time. Filtering an $M \times N$ image with an $P \times Q$ kernel requires roughly $MNPQ$ multiplies and adds. Having a separable kernel means you can do filtering in two steps: The first requiring MNP multiplies and adds, and the second requiring MNQ multiplies and adds, giving a total of $MN(P+Q)$. [2]

Task 2: Greyscale Conversion

a *Implement two functions that manually convert a RGB colour image to a greyscale representation. The first function must implement the averaging method, while the second must implement the luminance-preserving method*

The averaging method:

$$grey_{i,j} = \frac{R_{i,j} + G_{i,j} + B_{i,j}}{3} \tag{2}$$

The luminance-preserving method:

$$grey_{i,j} = 0.2126R_{i,j} + 0.7152G_{i,j} + 0.0722B_{i,j} \tag{3}$$

The implemented functions can be seen below. They all take in an `img` object

```
1  def conv2Grey_avg(img):
2      img_arr = np.array(img) # convert from image object to array
3      r = img_arr[:,:, 0] # equivalent to img[...,0]
4      g = img_arr[:,:, 1]
5      b = img_arr[:,:, 2]
6
7      grey = np.zeros((len(r), len(r[0])), np.uint8)
8
9      for row in range(0, len(r)-1):
10         for col in range(0, len(r[0])-1):
11             grey[row][col] = (int(r[row][col]) + int(g[row][col]) +
                   int(b[row][col])) / 3          # grey tone with the
                   average of the RGB values
12
13     img = Image.fromarray(grey) # converting back to image object
14
15     return img
```

```
1  def conv2Grey_lum(img):
2      img_arr = np.array(img) # convert from image object to array
3      r = img_arr[:,:, 0] # equivalent to img[...,0]
4      g = img_arr[:,:, 1]
5      b = img_arr[:,:, 2]
6
7      grey = np.zeros((len(r), len(r[0])), np.uint8)
8
9      for row in range(0, len(r)-1):
10         for col in range(0, len(r[0])-1):
11             grey[row][col] = int(0.2126*r[row][col] + 0.7152*g[row][
                   col] + 0.0722*b[row][col])
```

[2] Taken from `https://blogs.mathworks.com/steve/2006/10/04/separable-convolution/#7`

```
12
13      img = Image.fromarray(grey) # converting back to image object
14      return img
```

b *Apply your functions on a couple of colour images and show the result in your report. Briefly discuss the perceived differences between the output of your two functions.*

(a) Original color image 1 (b) Original color image 2

Figure 2: The original color pictures to apply greyscale conversion to.
Image 1 is a 768×512 pixel, 32-bit, .png file converted from .tiff
Image 2 is a 6000×4000 pixel 24-bit, .jpg file taken directly from a DSLR camera, chosen in order to do image processing on a high-resolution image.

(a) Greyscale conversion using the averaging method (2) (b) Greyscale conversion using the luminance-preserving method (3)

Figure 3: Comparison of greyscale conversion techniques for image 1

(a) Greyscale conversion using the averaging method (2)

(b) Greyscale conversion using the luminance-preserving method (3)

Figure 4: Comparison of greyscale conversion techniques for image 2

It is hard to notice any significant differences between the two methods. In image 1 it is hardly any difference, except a small change in the area around the lamp, which has become slightly brighter using the luminance-preserving method. For image 2, it seems like areas that are relatively dark in the averaging method becomes brighter in the luminance-preserving method. This can be seen in the green areas of the image - the trees in the background.

Task 3: Intensity Transformations

a *Implement a function that takes a greyscale image and applies the following intensity transformation:* $\tau(p) = p_k - p$

- *In your own words, explain what this transformation does. What would you call it?*
  This transformation inverts the greyscale values. Meaning that a value of 0 becomes 255, 1 becomes 254, and so on. Meaning that black pixels become white, and white pixels become black. The image becomes an inverted version of itself. So you call it an inversion function.

(a) Greyscale image using the averaging method (2)

(b) Applying the intensity transformation.

Figure 5: Applying an inversing transformation

b *A basic definition of the gamma transformation can be seen in Equation 4. It adjusts the overall intensity of an image by squeezing pixel intensities to either low or high intensities.*

$$\tau(p) = cp^{\gamma}, \quad c > 0, \gamma > 0 \tag{4}$$

In your own words, explain what happens with the image intensity values when $\gamma > 1$ and $\gamma < 1$?

For $\gamma > 1$, we see that the images appear darker than the original, giving more contrast. Opposite, for $\gamma < 1$, the image appear brighter, and more "washed out" than the original, reducing contrast.

The implemented γ-correction function:

```
1  def gamma_tran(img, c, gamma):
2      img_arr = np.array(img)
3
4      #Normalizing values to [0,1]:
5      max = np.amax(img_arr)
6      img_arr_float = img_arr.astype(float)
7      conv_dict={}
8      for i in range(0, max + 1):
9          conv_dict[i] = i / max
10     print("Performing gamma transformation...")
11     for i in range(0, len(img_arr)):
12         for j in range(0, len(img_arr[0])):
13             img_arr_float[i][j] = conv_dict[img_arr[i][j]]
                   # using the dictionary
14             img_arr_float[i][j] = c * math.pow(img_arr_float[i][j],
                   gamma) #applying the gamma function
15             img_arr[i][j] = img_arr_float[i][j] * max       #
                   converting back to original range for saving the
                   image correctly.
16
17     img = Image.fromarray(img_arr)
18     return img
```

Results from the γ transformation

(a) $\gamma = 0.2$ (b) $\gamma = 0.6$ (c) $\gamma = 2.5$

(d) $\gamma = 0.2$ (e) $\gamma = 0.6$ (f) $\gamma = 2.5$

Figure 6: Applying Gamma transformations

Task 4: Spatial Convolution

a *Implement a function that takes a greyscale image and an arbitrary linear convolution kernel and performs two-dimensional spatial convolution. Assume that the size of the convolution kernel is odd numbered, e.g.* $3\times3, 5\times5,$ *or* 7×7*. You must implement the convolution procedure yourself from scratch. You are not required to implement procedures for adding and removing padding. In your own words, explain how you implemented convolution your report.*

```
1  def convolution(kernel, img):
2      kernel_height = len(kernel)
3      ker_h_2 = int(kernel_height/2)
4
5      kernel_width = len(kernel[0])
6      ker_w_2 = int(kernel_width/2)
7
8      img_arr = np.array(img)
9
10     if (img_arr.ndim == 3):      #color picture
11         print("Applying convolution filtering for color image...")
12         img_conv = np.zeros((len(img_arr), len(img_arr[0]), img_arr.
               shape[2]), dtype=np.uint8)
13         for i in range(ker_h_2, len(img_arr) - ker_h_2):
14             for j in range(ker_w_2, len(img_arr[0]) - ker_w_2):
                   #Skipping the outer row/coloumn to not go out of
                   index when applying convolution filter.
15                 for k in range(0, img_arr.shape[2]):        # shape
                       returns e.g. (50, 50, 3)
16                     img_conv[i, j, k] = np.sum(np.multiply(kernel,
                           img_arr[i - ker_h_2 : i + ker_h_2 + 1, j -
                           ker_w_2 : j + ker_w_2 + 1, k]))
17
18     else:       #greyscale picture
19         img_conv = np.zeros((len(img_arr), len(img_arr[0])), dtype=
               np.uint8)
20         print("Applying convolution filtering for greyscale image...
               ")
21         for i in range(ker_h_2, len(img_arr) - ker_h_2):
22             for j in range(ker_w_2, len(img_arr[0]) - ker_w_2):
                   #Skipping the outer row/coloumn to not go out of
                   index when applying convolution filter.
23                 img_conv[i, j] = np.sum(np.multiply(kernel, img_arr[
                       i - ker_h_2 : i + ker_h_2 + 1, j - ker_w_2 : j +
                        ker_w_2 + 1]))
24
25     img = Image.fromarray(img_conv)
26     return img
```

The implemented function `convolution(kernel, img)` works for both color image objects `img`, and greyscale images. The function finds the dimensions of the smoothing kernels, and from this determines if the image is greyscale or color image. For every pixel, it then sums the element-wise multiplication of the kernel and the neighbours of the current pixel (8-pixel neighbourhood).

Two convolution kernels used for smoothing images:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{5}$$

This is a 3×3 averaging kernel.

$$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\ 4 & 16 & 24 & 16 & 4\\ 6 & 24 & 36 & 24 & 6\\ 4 & 16 & 24 & 16 & 4\\ 1 & 4 & 6 & 4 & 1\end{bmatrix} \tag{6}$$

This is a 5×5 Gaussian kernel approximated using binomial coefficients.

Convolving a greyscale image with the 3×3 averaging kernel:

(a) Original image (b) Result of applying the averaging kernel

Figure 7: Applying smoothing kernels on greyscale image

Convolving a color image with the 5×5 Gaussian kernel

(a) Original image (b) Result of applying the Gaussian kernel

Figure 8: Applying an smoothing kernels on color image

b *The Sobel operator consists of a pair of directional convolution kernels and is used to measure the gradient of an image. The convolution kernels for the horizontal and vertical direction can be seen in Equation 7*

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{7}$$

This can be combined to measure the magnitude of the gradient at each point:

$$|\nabla| = \sqrt{I_x^2 + I_y^2} \tag{8}$$

Approximating the horisontal and vertical gradient of an image

(a) Original image

(b) Resulting gradient in x direction

(c) Resulting gradient in y direction

Figure 9: Applying Sobel operators to measure the gradient of an image

Computing the magnitude of the gradients

(a) Original image

(b) Resulting gradient in x direction

(c) Resulting gradient in y direction

(d) Magnitude of the gradients

Figure 10: Applying Sobel operators to measure the gradient of an image

What does $|\nabla|$ tell us about the image?

As the gradient ∇f gets us the direction of the greatest rate of change at every pixel position. The magnitude, $|\nabla f|$ then gives us the value of rate of change in the direction of the gradient vector.