NTNU
Norwegian University of
Science and Technology

Digital Image Processing Assignment 1

Marius Blom

Sondre Jensen

2017

TDT4195 - Visual Computing Fundamentals

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 THEORY QUESTIONS

1.1 Histogram equalisation

Histogram equalisation is a technique for adjusting image intensities to enhance contrast.

1.1.1 How can we see that an image has low contrast when looking at its histogram?

The frequencies are concentrated on the mid range intensities.

1.1.2 What effect does τ_{heq} have when applied on an image?

Improving the contrast by uniformly distributed difference between dark and light values.

1.1.3 What happens when histogram equalisation is applied multiple times on the same image in sequence?

Histogram equalisation enhance the contrast. Applying histogram equalisation multiplied times does not change anything since the intensities is already at the desired values.

1.2 Perform histogram equalisation

Table 1.1: Original image

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

Table 1.2: Histogram equalisation

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
f_n	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{2}{12}$	$\frac{3}{12}$	$\frac{1}{12}$	$\frac{0}{12}$	$\frac{1}{12}$	$\frac{0}{12}$	$\frac{0}{12}$	$\frac{1}{12}$	$\frac{0}{12}$	$\frac{0}{12}$	$\frac{1}{12}$	$\frac{0}{12}$	$\frac{0}{12}$	$\frac{1}{12}$
F_n	$\frac{1}{12}$	$\frac{2}{12}$	$\frac{4}{12}$	$\frac{7}{12}$	$\frac{8}{12}$	$\frac{8}{12}$	$\frac{9}{12}$	$\frac{9}{12}$	$\frac{9}{12}$	$\frac{10}{12}$	$\frac{10}{12}$	$\frac{10}{12}$	$\frac{11}{12}$	$\frac{11}{12}$	$\frac{11}{12}$	$\frac{12}{12}$
$F_n * 15$	$\frac{15}{12}$	$\frac{30}{12}$	$\frac{60}{12}$	$\frac{105}{12}$	$\frac{120}{12}$	$\frac{120}{12}$	$\frac{135}{12}$	$\frac{135}{12}$	$\frac{135}{12}$	$\frac{150}{12}$	$\frac{150}{12}$	$\frac{150}{12}$	$\frac{165}{12}$	$\frac{165}{12}$	$\frac{165}{12}$	$\frac{180}{12}$
floor	1	2	5	8	10	10	11	11	11	12	12	12	13	13	13	15

Table 1.3: Equalised image with different intensity on each pixel

13	8	2	12
8	1	10	8
5	11	15	5

1.3 Convolution operator being associative

The "Associative Laws" say that it doesn't matter how we group the numbers when we add or multiply. This property is beneficial for convolution since the grouping of the numbers does not affect the result.

1.4 A MxM kernel

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

(a) left figure

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

(b) right figure

Figure 1.1: Combined figure

1.4.1 Determine if the convolution kernel are separable by computing their rank

matrix a

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \overset{R3-R1}{\sim} \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 0 & 0 & 0\end{bmatrix}$$

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \overset{\frac{R2}{2}}{\sim} \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 1 & 2 & 1\\ 0 & 0 & 0\end{bmatrix}$$

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \overset{R2-R1}{\sim} \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

Which implies that $rank(a) = 1$. Hence the matrix is seperable. We thereby have that

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

matrix b

$$\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix} \overset{R3-R1}{\sim} \begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 0 & 0 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix} \overset{R2-R1}{\sim} \begin{bmatrix}1 & 1 & 1\\ 0 & -9 & 0\\ 0 & 0 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix} \overset{\frac{R2}{-9}}{\sim} \begin{bmatrix}1 & 1 & 1\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}$$

$$\begin{bmatrix}1 & 1 & 1\\ 1 & -8 & 1\\ 1 & 1 & 1\end{bmatrix} \overset{R1-R2}{\sim} \begin{bmatrix}1 & 0 & 1\\ 0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix}$$

The matrix is not separable since $rank(b) = 2$

1.4.2 Why is it helpful to have convolution kernels that are separable?

We are able to increase the computation speed by separating the matrix into vectors.

2 GREYSCALE CONVERSION

2.1 Implement two functions that manually convert a RGB colour image to a greyscale representation

$$grey_{i,j} = \frac{R_{i,j} + G_{i,j} + B_{i,j}}{3} \tag{2.1}$$

```
def average(rgb):
        for i in range(len(rgb)):
                for j in range(len(rgb[i])):
                        grey = 0
                        grey += rgb[i, j, 0]
                        grey += rgb[i, j, 1]
                        grey += rgb[i, j, 2]
                        avgColor = grey/3
                        rgb[i, j] = [avgColor, avgColor, avgColor]
        return rgb
```

$$grey_{i,j} = 0.2126R_{i,j} + 0.7152G_{i,j} + 0.0722B_{i,j} \tag{2.2}$$

```
def weightedAverage(rgb):
        for i in range(len(rgb)):
                for j in range(len(rgb[i])):
                        grey = 0
                        luminance = [0.2126, 0.7152, 0.0722]
                        grey += rgb[i, j, 0]*luminance[0]
                        grey += rgb[i, j, 1]*luminance[1]
                        grey += rgb[i, j, 2]*luminance[2]
                        rgb[i, j] = [grey, grey, grey]
        return rgb
```

2.2 Apply functions on colour images

We can see clearly from figure 2.1 that the averaging method lack some shades of grey. The averaging method is having trouble with the color red and the color green, since they seem to get the same shade of grey in the picture.

Figure 2.1: Averaging method, luminance-preserving method and original image

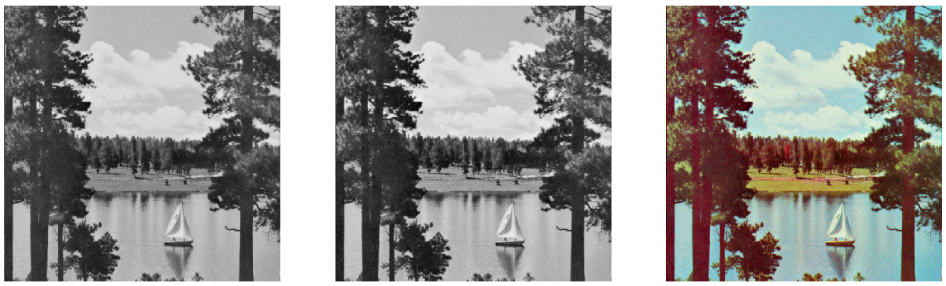

Figure 2.2: Averaging method, luminance-preserving method and original image

3 Intensity Transformations

3.1 Intensity transformation greyscale

```
#       intensity transformations
def intensity(grayscale):
#       loop through the colors
        for i in range(len(grayscale)):
                for j in range(len(grayscale[i])):
                #       save original image
                        p = grayscale[i, j]
                #       p_k is the highest value a pixel can have
                        p_k = 255
                #       use formula T(p)=p_k-p and update the colors
                        grayscale[i, j] = p_k - p
        return grayscale
```

3.1.1 Explain what this transformation does

The transformation inverts the colors of the image.

3.1.2 Apply the function on an image of your choice and show the result

Figure 3.1: subplot of Intensity

3.2 Gamma transformation greyscale

```
#       gamma transformations
def gammaTransform(rgb,gammaValues):
        for i in range(len(rgb)):
                for j in range(len(rgb[i])):
                #       normalize the image
                        p = rgb[i, j]/255
                        rgb[i, j] = (p**gammaValues)*255
        return rgb
```

3.2.1 Explain what happens with the image intensity values when $\gamma > 1$ and $\gamma < 1$

The image gets lighter when $\gamma < 1$

3.2.2 Different γ values on a greyscale image

```
def subplotImage(filepath):
        _, ax = plt.subplots(1, 3, figsize=(16, 8))
#       store gamma values in array
        gammaValues = [.2, .5, .9]
#       read from array and create plot for each value
        for i in range(len(gammaValues)):
                image = misc.imread(filepath)
                gammaTransformed = gammaTransform(image,gammaValues[i])
                ax[i].imshow(gammaTransformed, cmap=plt.cm.gray)
                ax[i].set_axis_off()
        plt.show()
        return None
```

Figure 3.2: gammaValues from .2, .5, .9

Figure 3.3: gammaValues from .2, .5, .9

4 SPATIAL CONVOLUTION

4.1 Implementation of convolution method

The code used to implement the convolution is listed bellow:

```
def spatialConvolution(inputImage, kernel):
        #Makes a copy of the image to write over
        outputImage = np.array(inputImage)
        #Find height and width of image to help us navigate
        (imageHeight, imageWidth) = inputImage.shape

        #Find height, width and center of kernel to help us navigate
        kernelWidth = len(kernel[0])
        centerKernelWidth = int(np.floor(kernelWidth / 2))
        kernelHeight = len(kernel)
        centerKernelHeight = int(np.floor(kernelHeight / 2))


        #traverse through every pixel in the image
        for y in range(imageHeight):
                for x in range(imageWidth):
                        #Include a summation of the intensety of the pixel
                        #set to 0 whenever we move to the next pixel
                        sum = 0

                        #Traverse through the kernel
                        for ky in range(kernelHeight):
                                for kx in range(kernelWidth):
                                        #Find the position of the current kernel square
                                        #the center of the kernel get position 0.0
                                        #everything above and to the left of the kernel
                                        #get negative position
                                        #The rest are positive
                                        nx = kx - centerKernelHeight
                                        ny = ky - centerKernelWidth

                                        #kernel positions are both negative and positive
                                        #traverse the image by adding the kernel
                                        #position and the position of the current pixel
                                        currentPixelX = x + nx
                                        currentPixelY = y + ny

                                        #Since we don't have padding are all values
                                        #that layes outside the image excluded
                                        if 0 <= currentPixelX < imageWidth and 0 <= currentPixelY < imageHeigh
                                                #Get the intensity of the pixel and multiply with the
                                                #value in the corresponding square in the kernel
                                                level = inputImage[currentPixelY][currentPixelX] * kernel[ky]

                                                #We summarize the intensity of all the kernel pixels
                                                sum = sum + level

                        #set the corresponding pixel equal to the new intensity
                        outputImage[y][x] = sum

        return outputImage
```

We simply navigate through every pixel in any imported image. For every pixel we navigate around an imported kernel with centre in the current pixel. This works with any matrix consisting of odd numbers smaller than the actual image. We then summarise every intensity level and multiply it with the corresponding kernel value, for so to apply that new value to the new image.
We implement the same algorithm for colour image, with the change that it takes every RGB colour and summarise them independent of each other.

4.2 Smoothening function

4.2.1 Smoothening greyscale image

On figure 4.1 have we implemented the two kerneles from equation 5 in the assignment. The figure to the left is the original image. The image smoothed by the averaging kernel is in the middle and the image smoothed by the by the Gaussian kernel is the image to the right.

Figure 4.1: Smoothening greyscale image

We can easily see a difference between the original image and the smoothed ones. In both the smoothed images does it look like the original image is blurred. It is hard to see a difference between the image smoothed by the average kernel and the Gaussian kernel. If we had an image with Gaussian noise would the difference been significant.

4.2.2 Smoothening colour image

The arrangement of the images in 4.2 is the same as in 4.1. The original image to the left, the averaging image in the middle and the Gaussian image on the right.

Figure 4.2: Smoothening colour image

On figure: 4.2 can we see the same result as in section: 4.2. Both the smoothed images seams to be blurred, but it is not easy to differentiate between them.

4.3 Gradient

4.3.1 Horizontal and vertical gradient

On figure 4.3 is the original image placed to the left, the vertical gradient in the middle and the horizontal gradient to the right.

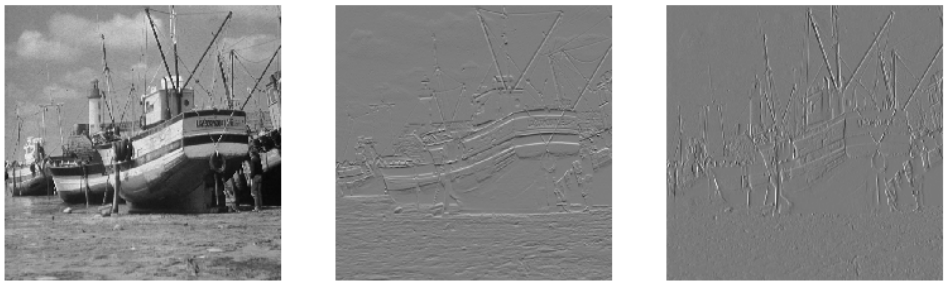

Figure 4.3: Smoothening colour image

We can easily see that both the vertical and horizontal gradient amplifies the changes in the vertical and the horizontal direction.

4.3.2 Compute the magnitude of the gradients $|\nabla|$

On figure: 4.4 and figure: 4.5 do we have the original image to the left and the magnitude image on the right.

Figure 4.4: Magnitude image before Gaussian filter

On the magnitude image can we easily see the lines representing the edges of the image. Since the gradient amplifies the changes in the image is it very vulnerable to noise. We implemented therefore the 5x5 Gaussian kernel from section: 4.2.

Figure 4.5: Magnitude image after Gaussian filter

We can see on figure: 4.5 that some of the less visible lines are gone, but the ones remaining is ticker and easier to see.

4.3.3 What $|\nabla|$ tell us about the image

Implementing the magnitude of the gradient ($|\nabla|$) gives us a indication of where all the edges in the images are. This simplifies the process of detecting objects in a image.