PRACTICAL 2

IMAGE ENHANCEMENT USING NEIGHBOURHOOD OPERATORS

TASK 1: CONVOLUTION AND NOISE FILTERING

The pixel operations considered in the previous practical change a pixel's value independently of all other pixels. In this practical we consider neighbourhood operations in which a pixel's new value is calculated from its old value and the values of pixels in its vicinity. Neighbourhood operations are used to perform noise reduction and edge extraction.

Convolution is the mathematical operation used to perform neighbourhood operations. Convolution consists of filtering an image using a mask. The mask is a small image whose pixel values are called weights. The mask is placed over part of the image. Corresponding pixels are multiplied to give intermediate products. These intermediate products are added to give the new pixel value. Initially the mask is placed in the top left corner of the image. It is moved across the image one pixel at a time from left to right and top to bottom, a new pixel value being calculated at each position. The resulting image is called the convolved image.

Matlab provides 2 different but very similar functions that perform convolution for images: `conv2()` and `filter2()`.

STEP 1: Download the image `'boatnoise.jpg'`. Load the image and display it in as the first image in a figure with 2 subplots (1x2).

STEP 2: Create a mask B as a 3x3 matrix, where all the elements have the value 1/9 (so the sum of all of them gives 1).

[HINT:To make it more elegant, you can use the command ones() rather than introduce the values one by one]

STEP 3: Perform the convolution between the image and the mask B using both `conv2()` and `filter2()`. What is effect in the original image? Observe the difference between the outputs and explain it (check the documentation if needed). Which one do you prefer? Display the result in the second subplot.

STEP 4: Create a new mask 5x5 so that the sum of all elements is 1. Filter the image with this mask. What difference you observe regarding Step3?

STEP 5: Write a function:

```
function Iout = noiseReduction(I,N)
```

that calculates first any average filter mask of size NxN and applies it to the input image.

STEP 6 : Let's create our own convolution function by completing the provided function:

```
function Iout = myconvolution(Iin,B)
```

by implementing the convolution equation for every pixel in the image:

Please remember that the two hour duration of a practical class will probably not be enough time to complete all sections. There is an expectation that you will work outside of class as an independent learner.

$$Iout(k,l) = \sum_{i=k}^{k+M-1} \sum_{j=l}^{l+N-1} Iin(i,j) \times B(i-k+1, j-l+1)$$

As, you can notice, the function converts the image from uint8 to doubles as first instruction to avoid losing precision in the mathematical operations.

[HINT1:You will need four nested for's to obtain the final convolved image]

[HINT2: Be careful not to go out of the limits of the image or the mask when accessing the values in the matrices or you will get an error. You final size will be size(I)-size(B)+1]

TASK 2: NEIGHBOURHOOD OPERATION FOR EDGE EXTRACTION

As we will explain in week 6, neighbourhood operators can also be used to extract the edges or borders of an image. As main difference, rather than using a low-pass filter, we will need to define one or several high-pass filters. In particular, it is very common to use the following 2 filters together:

$$B1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad B2 = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Where the first filter extract the vertical edges and the second the horizontal edges. All edges can be extracted by combining the output of both previous convolutions as:

$$Edges = \sqrt{(Iver)^2 + (Ihor)^2} = \sqrt{(I * B1)^2 + (I * B2)^2}$$

STEP 1: Write the function that performs edge extraction as:

```
function [Edges, Ihor, Iver] = edgeExtraction(Iin,B1,B2)
```

where Edges is the combined edge image, `Ihor` and `Iver` are the images containing the horizontal and vertical gradients images respectively, and B1 and B2 are the masks defined above.

STEP 2: Apply the previous function to the boat image and display the 2 resulting image in a window. Observe and analyse the results. Do they have sense?

[HINT: the resulting images of edge extraction are made of doubles rather than uint8. Be sure of using the appropriate image function to represent this type of images (e.g. `imagesc`())]

PREPROCESSING AND NAÏVE SEGMENTATION

The first two steps of a usual automated image processing system imply performing first preprocessing and then segmentation (sometimes by thresholding). In this section we will aim to complete a very simple version of these 2 steps.

STEP 1 : Open a figure with 3x2 subplots. Load the given image `'vehicle.jpg'` and display this original image and its histogram below. Use the command `title` to label each plot with a suitable name

STEP 2: Analyse the image visually and apply the preprocessing step(s) you seem suitable (brightness enhancement, contrast enhancement, noise reduction,…) using the functions you wrote in Practical 1 and 2. Display and label the results (both preprocessed image and its histogram) in the figure, beside the original image.

Let's now apply a simple segmentation technique based on thresholding. Thresholding transforms a greyscale image into a binary image (Booleans or logical in Matlab). In a greyscale image each pixel takes a value in the range 0 to 255. In a binary image each pixel takes one of only two values, 0 or 1 representing black or white.

A threshold is applied to each input grey level. Input values that fall at or below the threshold `Th` are replaced by one of the output values, input values above the threshold are replaced by the other output value. By choosing an appropriate threshold we can create a binary image in which the object of interest appears as a white area on a black background. This will allow us to take measurements of the object at the later feature extraction stage.

STEP 3: In our loaded image the object is in general darker than the background. So by given the value 'true' or 1 to those pixels below a threshold `Th`, the van can be segmented. Using the following instruction, you can implement this operation taking advantage of Matlab matrix operation capabilities:

```
Ibinary = I <= Th;
```

[ALTERNATIVE: Since a thresholding is basically a point operation, we could implement this as al Look-up table, as we did in Practical 1, where the transfer function is:

```
if input value <= Th
    output value = true
else
    output value = false
```

that can then be applied using `intlut()`]

STEP 4: Experiment with different Values of `Th` until you are reasonably happy with the segmentation. Display the segmented image in the last subplot.

STEP 5: Load now the second image `'vehicle2.jpg'`. Apply exactly the same processing with the same parameter values. What is the result? Is still satisfactory? Why is that? If not, modify the parameter until both images are reasonably segmented with the same parameter values.