Lab 4:
Automatical thresholding and simple OCR

Maria Magnusson, 2018,
Computer Vision Laboratory, Department of Electrical Engineering,
Linköping University, Sweden
Based on an older lab developed at the Department of Electrical Engineering.

1 Introduction

Read all instructions before the lab-exercise and repeat necessary theory from the lectures. Problem marked with a pointing hand are supposed to be solved as preparation for the lab-exercise.

A computer symbol indicates that a MATLAB -script has to be written and demonstrated during the lab-assignment.

Start by extracting the zip-archive containing the files `cells.tif, mid_way.m` into a suitable folder.

OCR (Optical Character Recognition) is used to optically read text with the help of image processing, interpret the text and translate it into a machine-readable form. It is quite easy to read typed text on a well-defined surface. On the other hand, read and interpret handwritten text in a noisy environment is a complex problem. When human beings read a text, we use the context of the text to make intelligent guesses about the letter or number to interpret. Such global information is not trivial to implement in an algorithm for a computer. The problem to recognize handwritten text can be resolved if we accept a certain uncertainty in the results.

We will in here implement a simple method to recognize handwritten numbers. Your task will be to write a MATLAB program which recognizes the number in a gray scale image.

2 Automatic thresholding

Automatic thresholding is an important part of many image analysis methods. We will work with the mid-way method and the least-error method.

2.1 The mid-way method

The mid-way method is quite a rough method. It divides the histogram in two parts, computes the means of the two parts and computes the threshold as the mean value of the two mean values. This procedure is then iterated.

Compute the histogram of the image **cells**, call the mid-way method and threshold the image accordingly by creating a file `threshCells.m` with the subsequent contents.

```
im = imread('cells.tif');
histo = hist(im(:),[0:255]);
T = mean(mean(im))
Tmid = mid_way(histo, T)
imTmid = im>Tmid;
figure(1)
colormap(gray(256))
subplot(2,2,1), imagesc(im, [0 255]);
axis image; colorbar
title('original image');
subplot(2,2,2), plot(0:255, histo, '.-r');
axis tight; grid
title('histogram');
subplot(2,2,3), imagesc(imTmid, [0 1]);
axis image; colorbar
title('mid-way thresholded image');
```

QUESTION 1: In the MATLAB script above, what is the starting value for the mid-way method?

The starting value is 77.0054

QUESTION 2: This time, the value used as starting value turned out to be the same as the value returned from the mid-way method. Vary the starting value to check how sensitive the mid-way method is to different starting values.

it's not very sensitive

QUESTION 3: The resulting threshold is 77 and it is not optimal. It leaves big holes in the cells. Suggest a better threshold by looking at the histogram.

50

In the next subsection we will try to obtain a similar value with the least-error method.

2.2 The least-error method

The least-error method is another method for thresholding. This method gives a better threshold than the mid-way method. It requires, however, a good starting value and it is more computational demanding.

QUESTION 4: How is the threshold T for the next iteration step determined in the least-error method? Tip: Check lesson 2.

P0 p0(f) = P1 p1(f)

The MATLAB function for the mid-way method is shown on the next page. The mean μ_0 (or rather the center of gravity) of the lower part of the histogram $p(f)$ up to the estimated threshold T is

$$\mu_0 = \frac{\sum_{f=0}^{T} (p(f) \cdot f)}{\sum_{f=0}^{T} p(f)}.$$

QUESTION 5: How is the mean μ_0 calculated by MATLAB code in the mid-way method on the next page?

```
lowersum1 = sum(histo(1:t0+1).*[0:t0]);
```

```
lowersum2 = sum(histo(1:t0+1));
if lowersum2 ~= 0
mean0 = lowersum1/lowersum2;
else
error('Cannot calculate new threshold');
end;
```

```
function outT = mid_way(histo, startT)
% Calculates a threshold value from a histogram using the mid-way method.
%
% histo: A 1D array histogram.
% The histogram bins must correspond to gray value 0, 1, 2,...
% startT: start threshold
% outT: the calculated threshold (integer)

num = length(histo);
t0 = floor(startT);
t1 = t0+2;

% Calculate the threshold
%------------------------
while abs(t0-t1) > 0.5

  t0 = t1;

  % Calculate mean for the lower part of the histogram
  %---------------------------------------------------
  lowersum1 = sum(histo(1:t0+1).*[0:t0]);
  lowersum2 = sum(histo(1:t0+1));
  if lowersum2 ~= 0
    mean0 = lowersum1/lowersum2;
  else
    error('Cannot calculate new threshold');
  end;

  % Calculate mean for the upper part of the histogram
  %---------------------------------------------------
  uppersum1 = sum(histo(t0+2:num).*[t0+1:num-1]);
  uppersum2 = sum(histo(t0+2:num));
  if uppersum2 ~= 0
    mean1 = uppersum1/uppersum2;
  else
    error('Cannot calculate new threshold');
  end;

  % Calculate new threshold
  %------------------------
  t1 = floor((mean0+mean1)/2);

end;

outT = t1;
```

In the least-error method, the total probability of the lower part of the histogram $p(f)$ up to the estimated threshold T is

$$P_0 = \frac{\sum_{f=0}^{T} p(f)}{\sum_{f=0}^{max} p(f)}.$$

QUESTION 6: How can P_0 be calculated by MATLAB code?

In the least-error method, the variance σ_0^2 of the lower part of the histogram $p(f)$ up to the estimated threshold T is

$$\sigma_0^2 = \frac{\sum_{f=0}^{T} \left(p(f) \cdot (f - \mu_0)^2\right)}{\sum_{f=0}^{T} p(f)}.$$

QUESTION 7: How can σ_0^2 be calculated by MATLAB code?

Regard the answers on the last four questions and sketch a **preliminary** MATLAB programs for the least-error method.
Start with the program for the mid-way method and modify it.
Tip: The MATLAB command `roots` might useful.

DEMO A: Type in your MATLAB program according to the home exercise and modify it until it works. Try your implementation on the image **cells**. First call the mid-way method to get a preliminary threshold. Then use this threshold as a starting value for for the least-error method.

QUESTION 8: What about your resulting threshold and the resulting image quality?

Also try your implementation on the image **nuf4a**. Since the object is dark in a brighter background, you will need to change the thresholding symbol from > to <=.

QUESTION 9: What about the threshold values for the mid-way and the least-error method? What about the corresponding image quality?

QUESTION 10: Also try your implementation on the image **nuf0b**. The least-error method will probably crash, but check the result for the midway method. What is the problem with this image?

2.3 Local thresholding

For the image **nuf0b** we will try local thresholding. Local thresholding can be achieved with help of an adjustable threshold that follows the local mean. This can be solved practically by producing a background image by low-pass filtering the image with a large filter so that the object disappear. The backgound image is then subtracted from the original image. Also, it is good to add ≈ 128 to approximately get back to the original interval [0 255]. The image received in this way can then be thresholded by the mid-way method and the least-error method.

Create a file `localThresh.m` with the subsequent content and execute it.

```
% Read the image and cast it to double
% ====================================
nuf = double(imread('nuf0b.tif'));

% Extend the image
% ================
tmp = [nuf(:,64:-1:1) nuf nuf(:,128:-1:65)];
nufextend = [tmp(64:-1:1,:); tmp; tmp(128:-1:65,:)];

% Make a Gaussian filter
% ======================
sigma=10;
lpH=exp(-0.5*([-64:64]/sigma).^2);
lpH=lpH/sum(lpH); % Horizontal filter
lpV=lpH';         % Vertical filter

% Convolve in the horizontal and vertical direction
% =================================================
nufblur = conv2(nufextend, lpH, 'valid');
nufblur = conv2(nufblur, lpV, 'valid');

% Make a new image
% ================
nuf = nuf - nufblur + 128;
```

```
figure(1)
colormap(gray(256))
subplot(2,2,1),
imagesc(nufextend, [0 255])
axis image; colorbar
title('extended image')
subplot(2,2,2),
plot(-64:64, lpH, '.-r')
axis tight; title('Gaussian kernel')
subplot(2,2,3), imagesc(nufblur, [0 255])
axis image; colorbar
title('blurred image image')
subplot(2,2,4), imagesc(nuf, [0 255])
axis image; colorbar
title('new image')
```

QUESTION 11: Before the image is convolved with the Gaussian filter it is extended. Describe how this extension is made. Why is it needed to extend the image in this way?

The image is extended with copies of itself, in order to make the edges similar to the image itself, instead of zeroes (black)

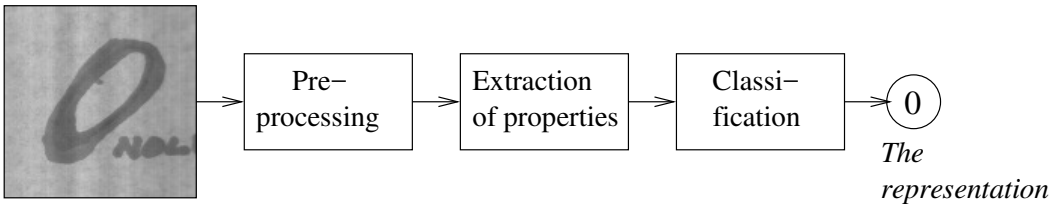
QUESTION 12: Adjust `sigma` of the Gaussian filter so that the object disappear from the background image. To which value do you adjust `sigma`?

sigma is 20 around

DEMO B: Complete `localThresh.m` by calling the mid-way method and the least-error method and show that **nuf0b** can be properly thresholded.

3 A simple program for optical character recognition (OCR)

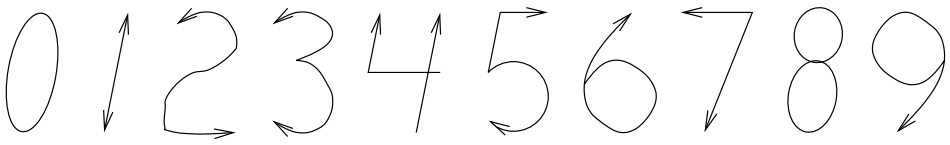
The input to our OCR system is a set of gray scale images containing handwritten numbers. The size of the images is 128×128 pixels with gray values in the interval $[0, 255]$. Since the images were obtained with a simple video camera, they contain rather much noise. A principal sketch of the OCR system is shown in the figure below.



The image of a 0

The representation of number 0

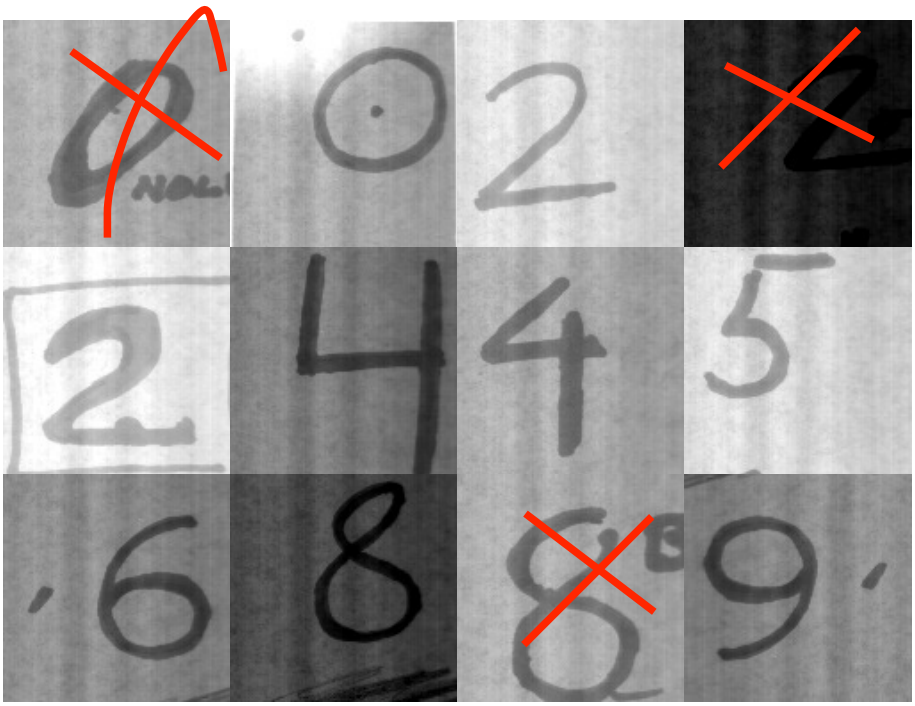
The given function **ocrdecide** will carry out the classification. The simple strategy is to study the slopes of the line ends in the skeleton of the number to distinguish between the different numbers. Also, the number of holes in the image is utilized. The image is scanned from left to right and from top to bottom until two end-points are found. For these end-points, the directions are calculated. The line ends and holes that **ocrdecide** are looking for are shown in the figure below.



The numbers 6 and 9 have only one end-point, 0 and 8 have no end-points and the other numbers have two end-points. 0 and 8 are separated by the number of holes within them. The given function **ocrdecide** assumes a clean skeleton without spurs.

As a hint for the OCR program, see the figure below. To the left is the original image, in the middle is the thresholded image and to the right is the image when it is ready to send to **ocrdecide**. Between each image, several computation steps have been performed. Note that the two binary images have been inverted to be more readable in print.

DEMO C: Write an OCR MATLAB program. All MATLAB commands that you need are mentioned in this lab or in the previous lab. Test your program on all the following images shown below; **nuf0a**, **nuf0b**, **nuf2a**, **nuf2b**, **nuf2c**, **nuf4a**, **nuf4b**, **nuf5**, **nuf6**, **nuf8a**, **nuf8b**, **nuf9**.



Modify your program until it recognizes 8-10 of these images. You can choose whether you want to use global or local thresholding. (If you choose the simpler global thresholding, you cannot recognize **nuf0b**, of course.)

QUESTION 13: Why is it almost impossible to recognize **nuf2b** and **nuf8b**?
