TNM025 – Lab-Introduction

Matlab

Matlab is the recommended programming environment for this course. Here we describe what you need to know about Matlab programming to succeed with the other assignments. Useful hints about handling images, vectors and matrices will be provided. Even if this homework is voluntary (no one will check if you have read this), we highly recommended you to read the entire document.

If you are familiar with other programming languages (C++, C#, Java, etc.) you may choose another implementation language, but then you have to manage without support on programming details.

General comments

Using for-loops in Matlab code is usually very time consuming (mainly because of the interpreted programming language). Instead of lopping through pixels (or matrix values) you can often use Matlab functions directly on the entire image/matrix. Moreover, for example calculating similarity between one vector and thousand other vectors can be done with simple matrix multiplications instead of using for-loops (details further down in this document).

Basic image processing

Read an image file, and show the image

```
im=imread('testimage1.jpg');
figure; imshow(im);
```

Pixels values in the image are now represented with 8-bit integers. We can easily transform the image to other representations

```
imdouble=double(im); %Double
imuint16=uint16(im); %16-bit integer
```

The uint16 image cannot be shown on your monitor. To show the double image, divide the image by 255 (2^8 -1 = 255).

```
imdouble=imdouble/255;
figure; imshow(imdouble);
```

For displaying images you can also use the functions 'image' and 'imagsc' (see Matlab Documentation).

If we want to read a numbered image sequence we can dynamically create each file name by combining predefined text and a varying variable. In the example below, four images are read and displayed in the same figure. We also add the image number above each image as a title.

```
figure
for n=1:4
   imname=['testimage',num2str(n),'.jpg'];
```

```
    myim=imread(imname);

    subplot(1,4,n);
    imshow(myim);
    title(num2str(n));
end
```

Two other useful Matlab functions for displaying more than one image in a single figure are 'subimage' and 'montage' (see Matlab Documentation).

Changing the image size can be done in many ways. The Matlab function 'imresize' is a convenient choice. Here the new image size will be 200 rows (height) and 300 columns (width), and we use bilinear pixel interpolation.

```
im=imresize(im,[200 300],'bilinear');
```

We can also make the image smaller by sampling pixel positions. In the example below every third pixel in row direction and the first 200 pixels in column direction are gathered in a new matrix. Notice that we do the same sampling in all three channels (the ':' before the last parenthesis).

```
imsampled=im(1:3:end,1:200,:);
```

Calculating a 256 bins histogram (pixel value distribution) for a gray scale image can be done as follows. We simply create a gray value image from the green channel in 'im'. If we use the 'imhist' function we need to make sure that the image contains at least one black (pixel value 0) and one white (pixel value 255) pixel, otherwise bin locations will not be equally spaced between 0 and 255. Remember to normalize the histogram, otherwise the histogram count will depend on image size.

```
imgray=im(:,:,2);
imgray(1,1)=0; %Black pixel
imgray(1,2)=255; %White pixel
imhistogram=imhist(imgray,256); %256 bins
imhistogram=imhistogram./sum(imhistogram); %Normalization
```

Another solution is to use the 'hist' function. In this case we need to convert the image matrix to a vector, using the function 'reshape'. The advantage of using the 'hist' function is that the function can handle a vector as second input argument, describing exactly how many bins we want, and where they are located.

```
%First reshape to one column
reshapedim=reshape(im(:,:,2),[size(im,1)*size(im,2) 1]);
imhistogram=hist(reshapedim,[0:1:255])';
imhistogram=imhistogram./sum(imhistogram); %Normalization
```

The obtained histogram vector will be used for demonstrating basic vector and matrix manipulations in the following section.

Working with vectors and matrices

Multiplying one vector with thousand other vectors, contained in a matrix, can easily be done WITHOUT for-loops! We create a vector with random content and size 10, and a matrix containing 1000 random vectors with size 10.

```
myvector=rand(1,10);
mymatrix=rand(1000,10);
```

If we want to calculate the inner product between each vector (row) in 'mymatrix' and the vector 'myvector', we can avoid for-loops by multiplying 'mymatrix' with the transpose of 'myvector'.

```
myresult=mymatrix*myvector';
```

If we want to obtain point-wise multiplication between elements in each vector (row) in 'mymatrix' and elements in the vector 'myvector', we can avoid for-loops by transforming 'myvector' to a diagonal matrix.

```
myresult=mymatrix*diag(myvector);
```

Representing matrix values in different bit positions can be useful when calculating for instance histograms. Normally, image values are represented with 8 bits, bit 1 to 8. As an experiment we can represent the image with bit 9 to 16 instead. Before shifting bit values, we need to convert the original image to a 16 bits image.

```
im1=imread('testimage1.jpg');
im1=uint16(im1);
im1s=bitshift(im1,8); %Shift 8 steps
```

Now we can save another image in bit 1 to 8, and we end up with two images in the same matrix! Not particularly useful when showing images, but sometimes useful when we need to carry out calculations involving both images.

```
im2=imread('testimage2.jpg');
%Images must have the same size
im2=imresize(im2,[size(im1,1) size(im1,2)]);
im2=uint16(im2);
imresult=im2+im1s; %Combine im2 with the shifted image
```

Eigenvectors and Eigenvalues

Symmetric matrices and their eigenvectors and eigenvalues will play a central role in the course. Make sure you know the basic facts and how to compute them in Matlab.

Take four new vectors in the 100D space:

```
ovectors = rand(100,4);
ovectors(1:50,4)=0;ovectors(51:100,4)=1;
```

Generate a symmetrical matrix:

```
symmatrix = ovectors*ovectors';
[eigenvectors,eigenvalues]=eigs(symmatrix,3);
```

The eigenvectors form a new orthonormal coordinate system.

Compute the first three coordinate values of these vectors in the basis given by the eigenvectors:

```
newcoords = eigenvectors ‘* ovectors;
```

Now you can compute the 100D vectors by projecting back the coordinate vectors:

```
newvectors = eigenvectors*newcoords;
```

Plot ovectors and newvectors!

How to use ‘dispimarray.m’ and ‘dispclick.m’

Below is an example on how to use the functions ‘dispimarray.m’ and ‘dispclick.m’ for viewing search results, and use the result as user interface for the next query.

You may also have a look at montage from the Image Processing Toolbox in Matlab.

Display a set of images.

```
imarray=round(rand(4,5)*5000); %save image numbers in imarray
fighandle=figure; %Create figure, and save handle in 'fighandle'
dispimarray(imarray,fighandle); %Display images in figure 'fighandle'
```

Display a set of images, and if an image is clicked, the chosen image is used as next query.

```
[ret, fighandle]=dispclick(imarray,fighandle);
```

The returned value in 'ret' is the number of the clicked image.

```
while (fighandle)
   %Replace with similarity measurement
   imarray=round(rand(4,5)*5000);

   imarray(1,1)=ret; %The query image in the upper left corner

   %Display new images and wait for next click
   [ret, fighandle]=dispclick(imarray,fighandle);
end
```

Useful Matlab functions

Study Matlab help chapters for the following functions (most of the functions have been mentioned earlier in this document).

hist	reshape	input
imhist		
	uint8	eval
sort	uint16	
sortrows	double	num2str
sum	imshow	
mean	colormap	
min	imresize	
max		
var	imagesc	
cov	image	
	subplot	
bitshift	montage	
bitor	subimage	
	axis	
eigs		
Eig		