Exercises Lecture C in MATLAB

Contents

1 Image Processing with MATLAB **1**
- 1.1 Crash-Course MATLAB . . . 1
- 1.2 Reading in graphic files in MATLAB . . . 4
- 1.3 Data types and Colors . . . 4
- 1.4 Color Gradation . . . 6
- 1.5 Other Display styles for Image Analysis . . . 6

2 Simple Perceptrons **7**
- 2.1 "More left than right" Perceptron . . . 8
- 2.2 "Somehow O-shaped" Perceptron . . . 8
- 2.3 Backpropagation . . . 9

3 Data Preparation, especially for image processing **9**
- 3.1 Filters and filtering . . . 9
- 3.2 The Laplace Filter . . . 10
- 3.3 Smoothing Filters, Blurr and Compression . . . 11
- 3.4 Regionprops . . . 12

4 Convolutional Neural Network in MATLAB **13**
- 4.1 Convolutional Neural Network from CodingLikeMad . . . 13
- 4.2 Further Self Study . . . 14
- 4.3 DLARRAY, auto differentiation and gradient descent . . . 14

1 Image Processing with MATLAB

1.1 Crash-Course MATLAB

Matlab as programming language is in many respects similar to Python:

1. Both MATLAB and Pyton are interpreter-languages (like the original Basic and the original JAVA), and you work can work from a prompt or run a program.

2. While a compiler takes a whole program ("source code") which a human can understand and then translates it into a whole program ("object code") the computer can understand, an interpreter translates and executes line by line. The advantage is that when there is a programming mistake, the variables are all in the memory, except if they are in functions (sorry about that).
   (Since 2016, MATLAB actually does not work single lines but whole blocks, so that even nested loops with lots of if's are executed relatively fast blockwise instead of line by line.)

3. Both MATLAB and Python are case-sensitive.

4. Math operators `+,-,*,/`work in the same way, only the the power in MATLAB uses `^` instead of `**`: 5^4 is computed by `5^4` (not as `5**4` as in Python).

Differences between MATLAB and Python are:

1. One does not have to link any libraries in MATLAB, the whole language (including libraries, called toolboxes, if installed) is accessible without the need to declare function packages or header files.[1]

2. Graphics functions in MATLAB are builtin and made for visualizing data for mathematics.

3. All arithmetic is automatically complex: The square-root of a negative number is evaluated to complex values:

```
>> a=-16
a =
   -16
>> sqrt(a)
ans =
   0.0000 + 4.0000i
```

   When you get a complex result for a real datatype, a program in C or FORTRAN will crash: MATLAB will assume that you mean it and continue calculating with complex values.

4. The results of all operations which are not followed by a semi-colon ";" will be echoed on the screen, which is a bad idea if the result is a large matrix:

```
>> a=5;
>> b=3
b =
     3
>> c=a*b
c =
    15
>> A=rand(500)
```

5. Loops in MATLAB start with `for` and are terminated by `end`:

```
>> for k=1:4
k
end
k =
```

[1]Linking libraries and declaring header-files became necessary for C, because it was developed in the early 1970s on a PDP-11 minicomputer which came by default with only 4 kb (yes, kilobyte!) of memory. All libraries and header files never fitted to the computer memory. Because C-Programmers consider themselves nerds, newer programming languages for computers with much larger memories since that time have retained that bad habit, because it looks so professional. It is just amateurism: The purpose of computer usage is that the computer serves the programmer, not the other way round. One can leave it to the computer the task of figuring out which functions, libraries, headers or modules are needed. Therefore, ADA (airplanes get their certification only if their software is written in ADA), FORTRAN or MATLAB don't require the declaration of headers or libraries: You've got what you've got.

```
     1
k =
     2
k =
     3
k =
     4
```

6. Conditionals start with `if` and are also terminated by `end`. Be careful if you delete larger blocks of a program with several `end` in one go, deleting the wrong number of end's may foul up the control structure.

7. It is possible to overwrite variables and functions, for example, you can overwrite π,

```
>> format compact
>> format long
>> pi
ans =
   3.141592653589793
>> pi=3.2
pi =
   3.200000000000000
```

   which is not a good idea. Also `i` and `j` are predefined as the roots of -1, but overwriting them as loop variables is without risk:

```
>> i*i
ans =
    -1
>> for i=1:5
i
end
i =
     1
i =
     2
i =
     3
i =
     4
i =
     5
```

8. Additionally to **explicit loops** with `for`, MATLAB also knows implicit loops so that vectors or matrices are initialized and processed without the need to use a for-end-structure:

```
>> a=1:6
a =
     1     2     3     4     5     6
>> b=a*pi/13
b =
```

```
    0.2417    0.4833    0.7250    0.9666    1.2083    1.4500
>> c=cos(b)
c =
    0.9709    0.8855    0.7485    0.5681    0.3546    0.1205
```

As a by-product, all functions are defined in a vector- or matrix-sense. A single random number can be generated with `rand`, while `rand(4,5)` creates a 4×5 matrix of random numbers (`rand(5)` creates a 5×5 matrix), and in the next step, we compute here the elementwise exponential of each element:

```
>> A=rand(4,5)
A =
    0.9480    0.7722    0.3801    0.4723    0.8463
    0.0596    0.4754    0.2133    0.3334    0.4081
    0.2687    0.6809    0.3829    0.9758    0.4620
    0.9867    0.4169    0.0297    0.5554    0.8263
>> B=exp(A)
B =
    2.5805    2.1645    1.4625    1.6037    2.3310
    1.0615    1.6086    1.2377    1.3957    1.5039
    1.3083    1.9757    1.4666    2.6534    1.5873
    2.6823    1.5173    1.0301    1.7427    2.2849
```

1.2 Reading in graphic files in MATLAB

Before we can use filters or other graphic processing tools, we first have to read in graphics files into MATLAB. MATLAB can read in conventional graphics formats (`.gif`, `.jpeg`, `.png`, `.bmp` with various different spellings of the extensions, including capitalization). Some extended file formats (Postscript and pdfs) may create problems and have to be converted first to more "basic" formats. The program `ReadinConvertHattedLady.m` reads in the png-graphics for the "hatted lady" in the directory and converts it to grayscale:

```
Acolordata=imread('HattedLady.png');
Agraydata=rgb2gray(Acolordata);
```

Using `imshow`, the original image, its grayscale conversion and the three RGB ("red, green blue") color channels are displayed. With `imread`, color graphics are converted "three dimensional" MATLAB matrices.

1.3 Data types and Colors

The command `whos` shows the size and class (in MATLAB, class means the datatype). MATLAB's standard data type is double precision real (actually, floating point) numbers. Additionally, we can have "strings", vectors (or arrays) of characters:

```
>> a='abcdef'
a =
    'abcdef'
>> c=rand(2,5)
c =
    0.8147    0.1270    0.6324    0.2785    0.9575
    0.9058    0.9134    0.0975    0.5469    0.9649
```

```
>> whos
  Name      Size             Bytes  Class     Attributes

  a         1x6                 12  char
  c         2x5                 80  double
```

When we analyze from the previous program `ReadinConvertHattedLady.m` data with MATLAB's `whos` function, we see that the color values are not of `double` type:

```
>> whos Acolordata
  Name              Size                Bytes  Class    Attributes

  Acolordata      322x289x3            279174  uint8
```

where `uint8` means "unsigned integer 8 bit", which allows entries from 0 to 255, as there are not higher number values for the 8-bit/channel RGB (red-green-blue) graphics format.[2]

```
>> a=uint8(253)
a =
  uint8
   253
>> a=a+1
a =
  uint8
   254
>> a=a+1
a =
  uint8
   255
>> a=a+1
a =
  uint8
   255
```

Additionally, the information of

```
>> whos Acolordata
  Name              Size                Bytes  Class    Attributes

  Acolordata      322x289x3            279174  uint8
```

means that the first dimension 322 is the vertical direction (rows of pixels), the second is the horizontal direction (columns of pixels), beware, that is exactly the opposite of what you are used to when you define a function as $f(x, y)$, where the first argument is the horizontal direction. T the last 3 entries are the RGB-color dimensions, so that `(:,:,1)` contains the "red", `(:,:,2)` the "green" and `(:,:,3)` the "blue" values.

[2]There is also a 32-bit/channel "HDR" (high dynamic range) format. When you get error messages reading in graphics with `imread` , you have to use `hdrread`.

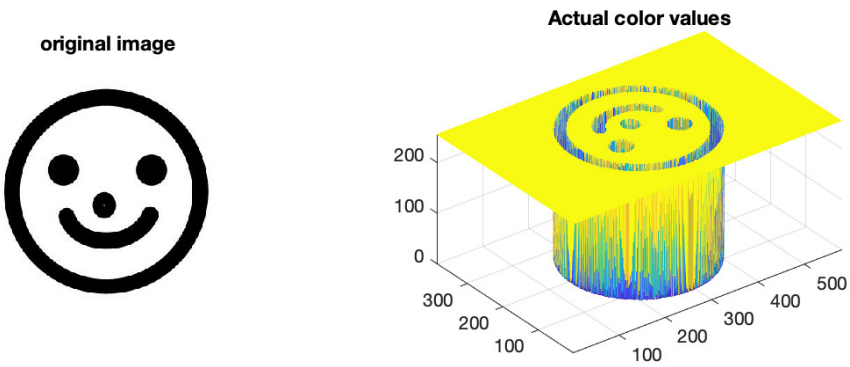

Figure 1: Original (left) of a Smiley which looks as if the black color were the same everywhere, compared with the actual color values (right), displayed with `mesh`: The color values at the boundary are not "100% black". Because the color values are a 3D-graphics, the lines appear in color.

1.4 Color Gradation

Unfortunately, when we look at a graphics and see something "black", this does not mean that the numerical values represent "exactly black", which would be an RGB-value of (0,0,0). Additionally, in graphics formats which allow compression (.jpeg, .gif, .png), the edges of objects may be smeared out. So even what looks a "black and white" image may actually be "grayscale". The program `Dr_bw_gradation.m` with the output in Fig. 2 shows such a case, together with the possibility to display graphics data not only as image, but actually with `mesh` to see the color values.

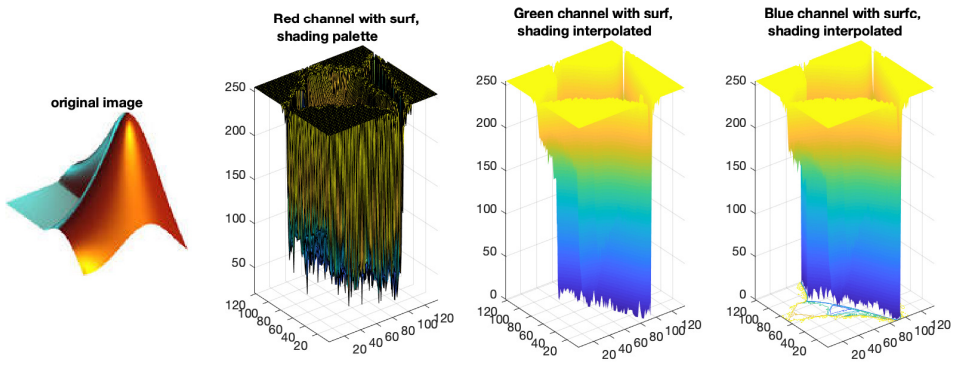

Figure 2: Original graphics (left) and different color channes plotted with `mesh` and `meshc`.

1.5 Other Display styles for Image Analysis

Instead of "assuming" properties, it is always better to have sharp look a the image content: That is true for both the original as well as the processed images. Other graphics styles of MATLAB than `mesh` may be better suited for image analysis. In `Dr_other_plottingstyles.m`, `mesh` and `meshc` are used: Usually, for graphics with reasonably high resolution, one has to remove the black facet lines with `shading interpolated`.

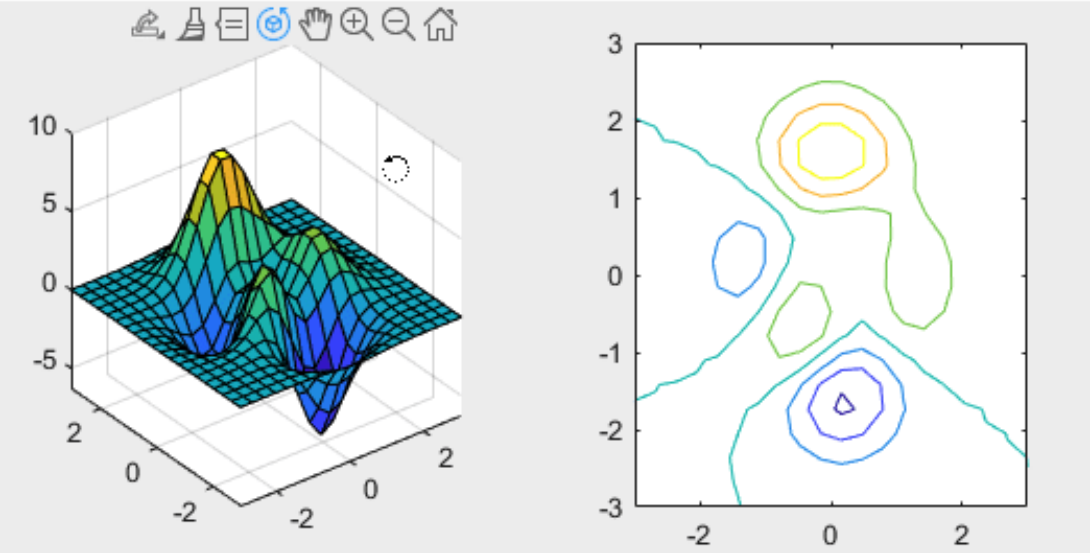

Figure 3: When you go with the cursor onto the graphics window, a menue appears: You can click the operation: In the left window, the rotation symbol is selected, so that you can rotate the graphics in three dimensions.), Original graphics (left) with `surf` and on the right with `contour`.

Structure may be hidden in the default view direction of MATLAB, therefore one can change the direction with `view` (have a look at `help view`) or rotate the view interactively, see Fig. 3.

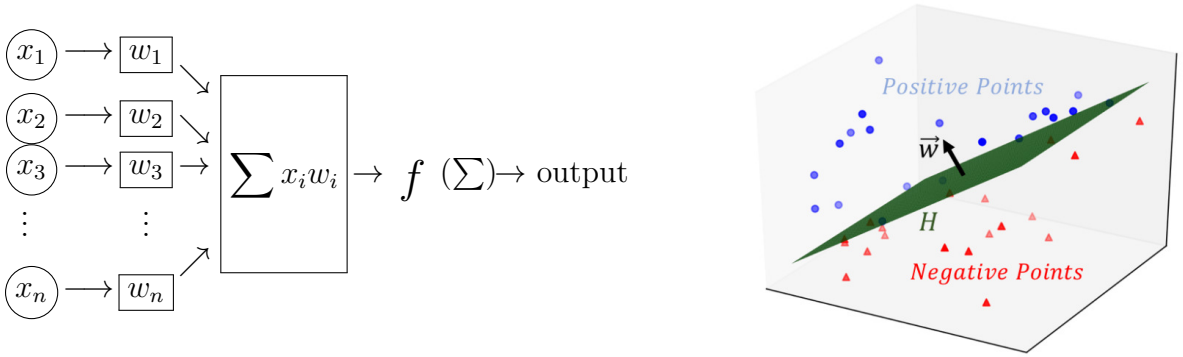

Figure 4: Schematic algorithm of a Perceptron which outputs +1 or -1 (left) and (right) representation of the "hyperplane" of a Perceptron which discriminates the solution space which gives +1 from the solution space which gives -1.

2 Simple Perceptrons

For Deep Learning algorithms, usually we first to deal with various hierarchies of data preparation algorithms: We have to understand perceptrons and prepare initial data with filtering and various techniques of image processing. A simple perceptron is an algorithm which takes some input data x_i and computes with pre-determined weights w_i as weighted sum, which can then be processed with a so-called "activation function"

f which outputs a 0 or 1 based on e.g. the sign, a sigmoid (s-shaped) function[3] or a theta-function ("Heaviside step function"[4]), see Fig. 4, left.
Many texts define the "hyperplane"[5] of the perceptron, which in the simplest case is that "boundary" where the sum $\sum x_i w_i$ outputs 0 or 1, see Fig. 4, right.
In the following, we will play with some simple perceptrons, which are probably not of practical interest, but should serve as simple examples how certain properties of input patterns can be decided with the help of perceptrons.

2.1 "More left than right" Perceptron

The program `Moreleftthanright.m` determines whether a 8×8 input array of "pixels" 0 and 1 called `xin_1` contains more entries on the left than on the right. There are other `xin_2`, `xin_3` defined so that you can play around more easily, just change the variable names. The actual working part is the weight function, which is defined via

```
w =
    -1    -1    -1    -1     1     1     1     1
    -1    -1    -1    -1     1     1     1     1
    -1    -1    -1    -1     1     1     1     1
    -1    -1    -1    -1     1     1     1     1
    -1    -1    -1    -1     1     1     1     1
    -1    -1    -1    -1     1     1     1     1
    -1    -1    -1    -1     1     1     1     1
```

(Probably, the "activation function" should not be defined as the sign of the resulting product with `w` as minus the sign.) In the same way, one can define "more right than left", "more up than down", and so on. It is clear that something has to be done when the number of columns becomes odd, because the middle column is neither left nor right: Probably the best strategy will be to define the weight for the middle column of odd number of columns as zero.
If the input data are not limited to values of 0 and 1, and one wants to compare the number of non-zero elements, one can instead first take the `sign` of the input data.

2.2 "Somehow O-shaped" Perceptron

The program `SomehowOshape.m` uses a somehow arbitrary weight function

```
w=[-1  1   1  1  1  1  1 -1
    1  1   1  1  1  1  1  1
    1  1  -4 -4 -4 -4  1  1
    1  1  -4 -4 -4 -4  1  1
    1  1  -4 -4 -4 -4  1  1
    1  1   1  1  1  1  1  1
   -1  1   1  1  1  1  1 -1]
```

[3] `https://en.wikipedia.org/wiki/Sigmoid_function`
[4] `https://en.wikipedia.org/wiki/Heaviside_step_function`
[5] Because we have n input variables x_n, we have an n-dimensional space, and so the high-dimensional plane which separates the results is called a "hyperplane".

so that it is possible that an input pattern has still some bits in the middle, as would be the case for Θ to be recognized as "O". If one wants to forbid this, the values in the middle should be changed from -4 to some values of larger negative magnitude. The corners are weighted with `-1`, and again, it depends if one would like to prohibit a sharp corners so that the pattern can be counted as "O" or not.

2.3 Backpropagation

The above two examples show that simple geometries ("where are the data?") can be implemented rather simply, while the recognition of shapes ("is it an O or not?") with a "simple" perceptron is a rather hand-waving affair. Because the shape itself and the deviation from it is not well define, Deep Learning uses "perceptrons with backpropagation", where the weight functions for the pixels which "should be there" or "should not be there" are computed, instead of conjectured as in our previous "Somehow O-shaped" example. Such backpropagation with many alternative input data or input shapes allows the "fitting" of the weight function and recognition of various shapes, in contrast to Hopfield Neural networks, where only a single shape can be implemented.

3 Data Preparation, especially for image processing

For Deep Learning algorithms, usually we first deal with various hierarchies of data preparation algorithms: Especially for image processing, various methods for filtering, shape recognition, decomposition into sub-shapes etc. are explained examples, but they may not be the most efficient ones to in practical applications. Many applications for Deep learning are for the fields of image recognition. Accordingly, a look in the literature for image processing, be it with MATLAB[6] or without[7] will always be helpful. Here, like everwhere, internet searches instead of the use of textbooks give results of rather random quality. We start with discussing some basics of image processing methods in MATLAB for "preprocessing" the graphics.

3.1 Filters and filtering

"Filters" in image processing are not something you can brew a coffe with, but they are functions to enhance (in whatever sense, and not necessarily esthetically) an image or help to extract certain features. For convolutional networks, we want to "coarsen" the information of an image (let's say from 40×40 pixels to 20×20 or 16×16 pixels), which is also a task for filters. In the following, we will run different filters to see how they work and how they can be used to prepare images for further processing. (Similar filters can be employed for e.g. one-dimensional data audio streams.)
Filters can be expressed with matrices, so for a pixel $x_{i,j}$ and its surrounding pixels $x_{i+1,j}$, $x_{i-1,j}$, $x_{i,j+1}$, $x_{i,j+1}$, $x_{i+2,j}$, ... some mathematical operations with a "weight function" $w_{i,j}$

[6] E.g. O. Marques, Practical Image and Video Processing Using MATLAB. Wiley (2011); R. C. Gonzalez, R. E. Woods, Richard, S. L. Eddins, Digital image processing using MATLAB, Prentice Hall (2004); A. McAndrew, Introduction to digital image processing with MATLAB, Thomson (2004)

[7] E.g. B. Jähne, Digital Image Processing. Springer (2005) D. Sundararajan, Digital Image Processing - A Signal Processing and Algorithmic Approach, Springer (2017)

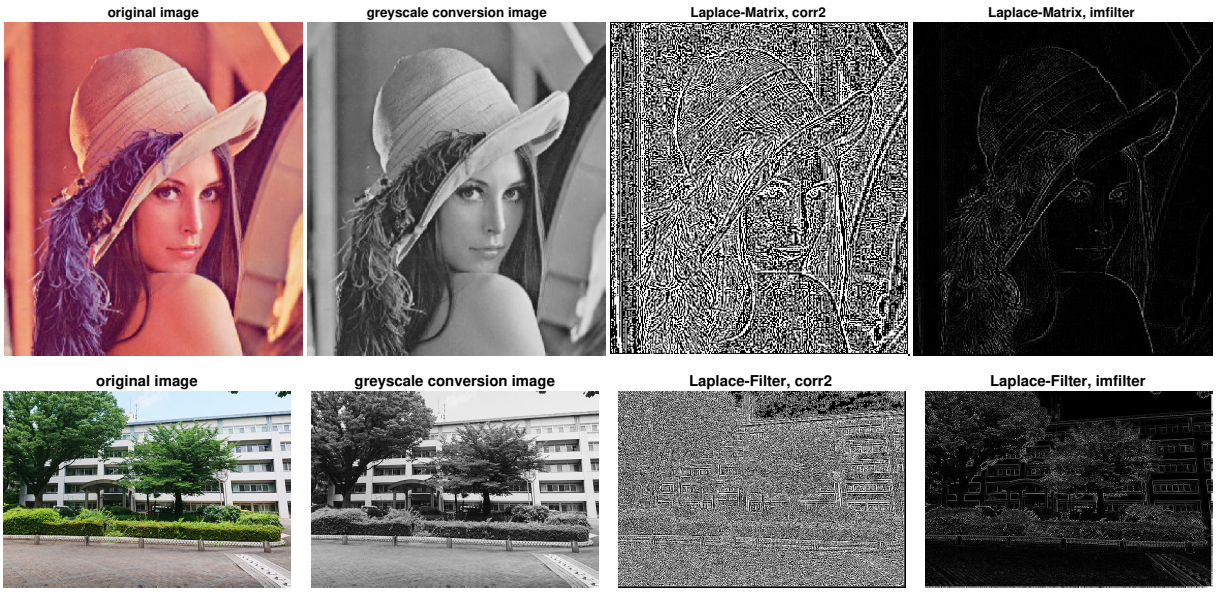

Figure 5: From left to right: Original, grayscale and Laplace-filtered (first with `conv2`, then with `imfilter`) e hatted lady (upper row) and UEC's main building (lower row).

(the filter matrix) is computed. MATLAB has built-in functions `filter` (for 1 dimension), `filter2` (for 2 dimensions) to compute the filtering.

3.2 The Laplace Filter

The matrix of the Laplace filter[8] is

$$A_{Laplace} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

It is a simple approach to "sharpen" pictures or find edges. For the program with the hatted lady, we can add

```
A_Laplace=-[0  1 0
            1 -4 1
            0  1 0]
Afiltered=conv2(Agraydata,A_Laplace);
```

where `conv2` is the function which "runs" the filter matrix `A_Laplace` as convolution[9] over the whole input (two-dimensional) matrix data of `Agraydata`. An alternative in MATLAB's `imfilter` which can also filter color images, which takes additional inputs, get documentation with `help imfilter` at the MATLAB-prompt to see the options.

[8]The name comes from the fact that it is the discrete version ("finite difference approximation") of the Laplace-operator

$$\Delta f(x,y) = \frac{\partial f^2}{\partial x^2} + \frac{\partial f^2}{\partial y^2},$$

the "second derivative" (curvature) in two dimensions, see `https://en.wikipedia.org/wiki/Discrete_Laplace_operator`.

[9]`https://en.wikipedia.org/wiki/Convolution`, Scroll down to "Discrete convolution"

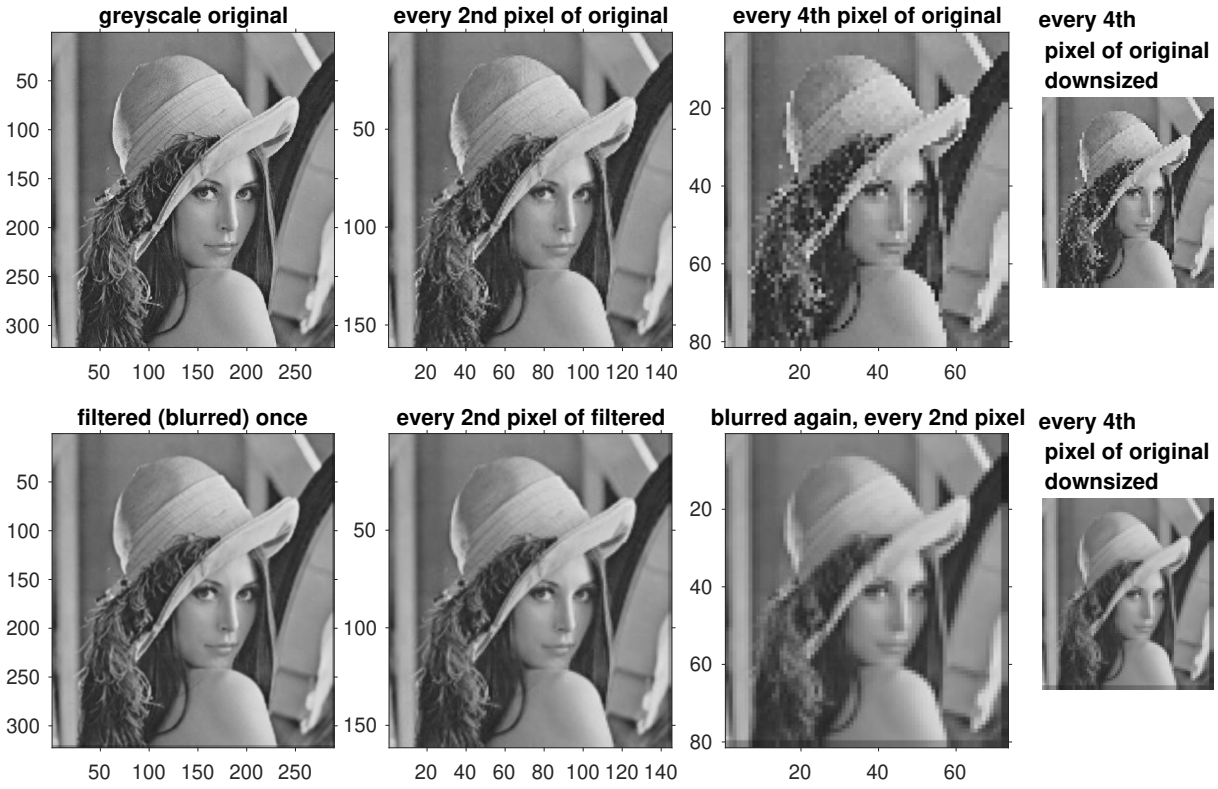

Figure 6: Upper row, from left to right: Original picture (left), every second pixel (middle) and every fourth pixel (right). Upper row, from left to right: Smoothed graphics (left), every second pixel (middle) and every second pixel from the smoothed again image. (Note the decreasing number of pixels from left to right)

The results from are different: Fig. 5 shows the results for running Laplace filters over the hatted Lady and the UEC main building: Lines become indeed more visible, while the rest of the image becomes rather unreconizable. To obtain actual lines from an image, there are better, so called edge-detection algorithms. Typically they carry the name of their inventors: Look for Sobel, Prewitt, Roberts and Canny, see MATLAB's help on `edge` [10]. Unfortunately, the `edge`-commands are much better in creating edges than in locating actual closed curves which can be used to extract features from a graphics.

3.3 Smoothing Filters, Blurr and Compression

More important that sharpening for deep learning algoritms is smoothing or blurring: It allows to reduce the picture size in coarser resolution without that the pixelation becomes obvious. Fig.6 shows the coarsening and the grainy textture (conspicious single pixels and regions) without smoothing / blurring. Typical smoothing filters have all positive entries like

$$A_{blurr,1} = \frac{1}{256}\begin{pmatrix} 21 & 31 & 21 \\ 31 & 48 & 31 \\ 21 & 31 & 21 \end{pmatrix} \text{ or } A_{blurr,1} = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

[10] `https://www.mathworks.com/help/images/ref/edge.html`

or also "Gaussian blur"[11] which can be defined in different sizes like

$$A_{3\times3} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}, A_{5\times5} = \frac{1}{273}\begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}, A_{7\times7} = \frac{1}{1003}\begin{pmatrix} 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ 2 & 22 & 97 & 159 & 97 & 22 & 2 \\ 1 & 13 & 59 & 97 & 59 & 13 & 1 \\ 0 & 3 & 13 & 22 & 13 & 3 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \end{pmatrix}.$$

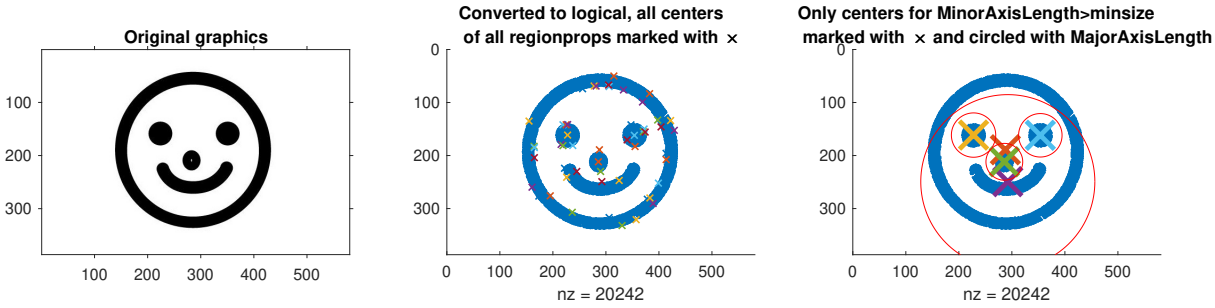

Figure 7: Image analysis for the smiley via `regionprops`. Left the original graphics, in the middle all centers of regionprops marked with a cross × and on the right the largest regions marked with the radii: Eyes, nose and mouth come out quite well, while the center of the face is and the mouth are a bit dubious

3.4 Regionprops

If you have a complex picture which is made up of several elements, you may first have to identify the elements. Let's say you want to identify eyes, nose and mouth in a face. In MATLAB, you can use `regionprops`, which tries to define convex regions in a two-dimensional array of logical data (0 and 1, true or false). The first hurdle is obviously to convert the image into suitable logical data. The program `Dr_bw_regionprops.m` tries to analyze our smiley from Fig.1. [12]
As can be seen in the middle graphics of Fig.7, where all centers are plotted where MATLAB's regionplots finds regions, the recognition of open and non-convex shapes (i.e. clusters of existing or missing pixels) may lead to results which are rather meaningless. Too many small regions detected by `regionprops` are potential candidates for preprocessing of images if we want to automatically analyse faces and extract shapes for eyes, noses and mouths. That said, it is not necessarily easy to find these features in images also with other methods. An attempt to copy eyes and mouth can be found in `Dr_pierrot_regionprops.m`, with the result in Fig.8, which uses

```
minsize=10 % minimal size in pixels
.....
logicalBW=(sum(double(BW),3)<400);
```

[11] `https://en.wikipedia.org/wiki/Gaussian_blur`
[12] For the many options which can be used with `regionprops`, see `https://www.mathworks.com/help/images/ref/regionprops.html`

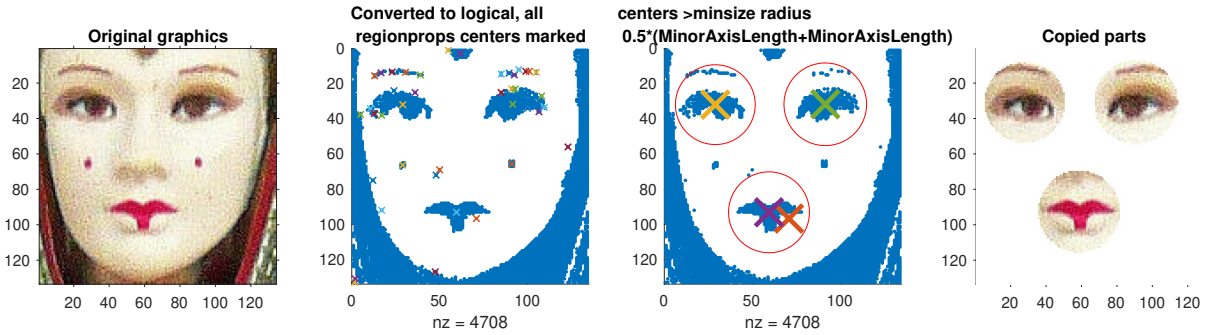

Figure 8: Successful extraction of mouth and eyes with `regionprops` from `Dr_pierrot_regionprops.m`.

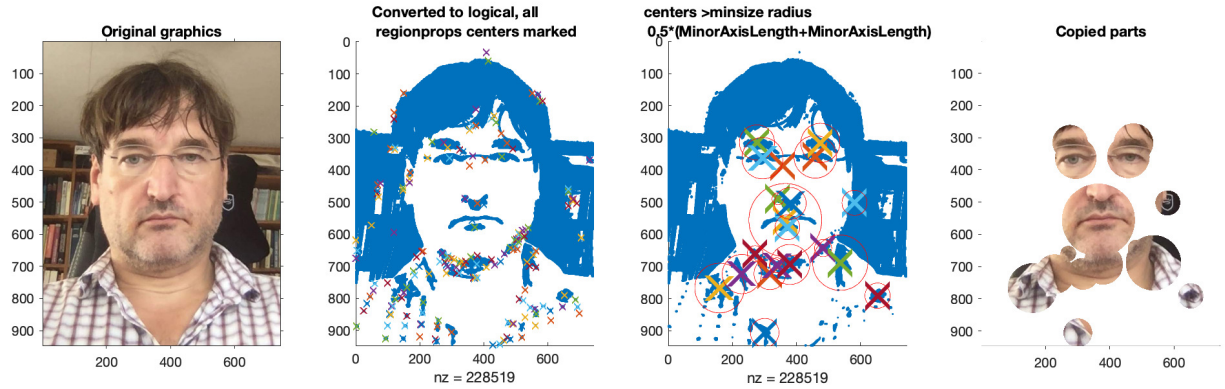

Figure 9: Extremely useless attempt to extract mouth and eyes with `regionprops` from a real photograph in `Dr_Matuttis_regionprops.m`, as even the eyebrows, details of the shirt, shadows and the crest on the headpiece of the gamer chair (looks like an earring, but is not!) ends up in the extraction.

which is not bad. Unfortunately, for real photos, a lot can go wrong, as can be seen in Fig.9, even with modified parameters

```
minsize=15 % minimal size in pixels
.....
logicalBW=(sum(double(BW),3)<350);
```

This shows that, depending on the problem, black box algorithms can sometimes be useful, and sometimes not.

4 Convolutional Neural Network in MATLAB

Up to now, the methods explained here served only for processing images, whole or in parts, to extract the information which should serve as the basis of the learning. In the following, proper algorithms for machine learning are run.

4.1 Convolutional Neural Network from CodingLikeMad

The file `cnnScript_example.m` runs the example from the youtube-movie
`https://www.youtube.com/watch?v=lK9YyX-q32k&t=917s` of a convolutional neural network with 28×28 pixels, i.e. $28^2 = 784$ input parameters. When you run the program, you see that finding the best weights costs time: This amount of data is not fitted in a trice, which justifies the use of dedicated hardware ("GPUs for deep learning", "AI & Deep-Learning workstation") for such kinds of operations. Depending on your version of MATLAB, different outputs of the patterns at the end are possible.

4.2 Further Self Study

The Deep Neural Network from MATLAB described on
`https://www.youtube.com/watch?v=hfA6wlDriMw&t=163s`
is from an e-book "Practical Deep Learning Examples with MATLAB", documented on
`https://www.mathworks.com/content/dam/mathworks/ebook/gated/deep-learning-practical-examples-ebook.pdf`
in English and on
`https://jp.mathworks.com/content/dam/mathworks/ebook/gated/jp-deep-learning-practical-examples-ebook.pdf`
in Japanese.

4.3 DLARRAY, auto differentiation and gradient descent

To optimize a function in linear algebra, one usually needs the gradients. Typically, in Machine Learning, auto differentiation (automatical numerical differentiation on a computer without approximation by finite differences) is used. The youtube movie
`https://www.youtube.com/watch?v=GEQMOopCeZk`
explains the principle of auto differentiation for use with the gradient descent algorithm.