

Reconocimiento de Escritura (2007)

Exercise Sheet 1

please present solutions until: 04.04.2007

You can work on these exercises in groups of two. If you choose to do that, both members of the group must be able to explain the work done.

Note that some of the exercises can be solved immediately and some need definitions that are discussed in the lecture before you can work on them. Alternatively, you can use definitions that you find in the literature or on the web (provided that they are reasonably close :-).

The exercises are set such that you can *choose* those that you would like to work on. The number of points the grade will be based on is 100. However, the exercises you find here are for much more than just 100 points. This means that any number of points equal to or larger than 100 will yield the best grade. The minimum number of points to pass this first part of the course is 50. The final grade will be determined by averaging the grades of this first part and the part presented by Alejandro Hector Toselli (weighted by the corresponding number of credits).

If you encounter an interesting question you would like to work on or an algorithm you would like to implement that is not on this list, you can propose it as an exercise as well.

Exercise 1.1 (Redundancy)

(5 Points)

Data compression is often based on using *redundancy* within the data stream. Digital images are one form of data that we encounter. For example, we can represent an image as a sequence of color or grey scale values.

Which types of redundancy are present in digital images and how could that be used for the compression of digital images? What is the difference between general photographs, screen shots, and scanned documents?

Please answer with a short text of about 300 words.

Exercise 1.2 (Rotation with 3 Shears)

(5 Points)

Rotating an image can be done by performing 3 shear operations instead of using a rotation matrix. Derive the shear parameters a, b, c for a rotation by α using the ansatz

$$\begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix} = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ b & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & c \\ 0 & 1 \end{pmatrix}$$

Exercise 1.3 (Morphological Processing)

(5 Points)

Prove that the following relation between morphological closing and opening holds:

$$(A \bullet B)^c = (A^c \circ \hat{B})$$

Exercise 1.4 (Histogram Properties)

(10 Points)

In the lecture, we discuss several statistical properties of the grayscale histogram of an image in the context of Otsu-thresholding. Show that the following equalities hold (cp. the notation used in the lecture):

a) $\sigma_b^2 = p_1 p_2 (\mu_1 - \mu_2)^2$

b) $\sigma^2 = \sigma_w^2 + \sigma_b^2$

Note: analogously to the quantities defined in the lecture, we have $\sigma_w^2 = p_1 \sigma_1^2 + p_2 \sigma_2^2$ with $\sigma_1^2 = \frac{1}{p_1} \sum_{g=0}^T (g - \mu_1)^2 h_g$, and $\sigma_2^2 = \frac{1}{p_2} \sum_{g=T+1}^{L-1} (g - \mu_2)^2 h_g$.

Exercise 1.5 (Connected Components)

(20 Points)

In your favorite programming language, write a program that given a binary image computes all connected components of value 1. (E.g., implement the algorithm discussed in the lecture.) Assume the binary image is given as a one-dimensional array with values for height and width. Output the result coded as an image. That is, the function should return an image of the same size as the input image in which each pixel is either 0 (when the corresponding pixel was 0 in the input) or has the label $l > 0$ of the connected component the pixel belongs to.

Exercise 1.6 (Features of Connected Components)

(20 Points)

You are given a binary image that contains exactly one connected component. (You can e.g. modify your program from the previous exercise such that it outputs one image per connected component.) Write functions that determine the following properties (or features) of the connected component:

- a) area
- b) perimeter/boundary length
- c) compactness
- d) bounding box and its aspect ratio (i.e. height/width)
- e) centroid
- f) orientation of axis of minimum inertia

Exercise 1.7 (Path-Finding)

(20 Points)

Consider a binary image of size $M \times M$. Assume the image is represented as a two-dimensional array `image[x][y]` where each element (or pixel) takes only the values 0 or 1. You are given the two points 'source' $s = (x_s, y_s)$ and 'target' $t = (x_t, y_t)$ in the image for which `image[xs][ys]==0` and `image[xt][yt]==0`.

- a) Write a function in your favorite programming language that determines the shortest path between s and t that only uses image points with value 0 if such a path exists. (Otherwise return e.g. the empty sequence.) More specifically, the function should determine a sequence p_1, \dots, p_N , such that $p_1 = s$ and $p_N = t$ and for each $n = 1, \dots, N : \text{image}(p_n) = 0$ and for each $n = 1, \dots, N-1 : p_{n+1} - p_n \in \{(0, 1), (1, 0), (0, -1), (-1, 0)\}$ (4-neighborhood), and for which no shorter sequence with the same attributes exists.
- b) What is the computational complexity of your approach?

- c) How does the function change if you consider the 8-neighborhood instead of the 4-neighborhood? Will the paths be shorter or longer in general? What is the relation between the existence of a path in the two cases?
- d) What are possible uses for this function (within or outside of document image processing)?

Exercise 1.8 (Skew Correction)

(25 Points)

One simple way to perform skew correction on a given a binary document image is the following:

- for a set of skew angles α do
 - rotate the document by α (note that this requires some care)
 - in each image row count the number of black pixels, yielding a sequence of counts
 - determine the variance of these counts over all rows
- choose the angle that corresponds to the largest variance

Instead of actually rotating the image, it will probably be faster to directly determine the projection onto a rotated y -axis.

- a) In your favorite programming language, write a program that performs skew correction.
- b) Why (if at all) does this method work? What are advantages and disadvantages of the method? How can it be improved? What are alternative approaches?

Exercise 1.9 (Evaluation of OCR)

(35 Points)

If you want to work with commercial software, use a legal copy of a commercial OCR system (maybe you have bought a scanner and that ships with an OCR software?). You should also try to gain some experience with Open-Source OCR systems. There are three major Open-Source OCR systems available: Tesseract, GOCR, and Ocrad. In our experience, out of these Tesseract performs best. You can also use the web-service at <http://demo.iupr.org/ocropus> to try OCRopus before it is available as source code. (If you're interested, you can obtain an executable for linux as well.)

- a) Evaluate the results of a commercial system on a small set of your own document images and determine the OCR errors and the error rate. Give examples of errors that are made by the system and try to explain their causes.
- b) To evaluate a large number of images, we need the ground truth of the text present in the image and an automatic method to determine the error rate. Implement the edit-distance (also called Levenshtein-distance) as an executable in your favorite programming language. The program should take two ASCII files as input and output the edit distance. You should include an option to show the positions at which the errors occur.
- c) Run the OCR system on the pages available at <http://www.iupr.org/~keyzers/res> and determine the error rate using your implementation of the Levenshtein-distance. What are the most common errors made? Give examples of errors that are made by the system and try to explain their causes.

- d) Download Tesseract from <http://code.google.com/p/tesseract-ocr/> and compile it. Evaluate the results on a small set of your own document images and determine the OCR errors and the error rate.
- e) Compare the errors made by at least two OCR systems.

Exercise 1.10 (Character Segmentation) (80 Points)

Do a literature survey of methods for segmentation of touching printed character strings and write a short report about the most promising methods (about five pages).


Exercise 1.11 (Handwritten Digits/Tangent Distance) (90 Points)

In this exercise you will conduct some experiments in the field of handwritten digit recognition. You will run your tests on the US-Postal-Service (USPS) database, which is an often used standard database consisting of 16×16 pixel grey-scale images of handwritten digits. There are 7291 digits for training and 2007 digits for testing. For some testing experiments it may be useful to use only a subset of the data to make your programs terminate faster. The data can be found together with simple input-routines for C (not necessarily perfect, but working) at <http://www-i6.informatik.rwth-aachen.de/~keyzers/usps.html>

The USPS training and test data is given as an ASCII file in the following format:

```
Number_of_Classes Number_of_Dimensions(=D)
Classlabel_1 Feature_1_1 Feature_1_2 ... Feature_1_D
Classlabel_2 Feature_2_1 Feature_2_2 ... Feature_2_D
...
Classlabel_N Feature_N_1 Feature_N_2 ... Feature_N_D
-1
```

There are $D = 256$ feature entries on each line corresponding to the 16×16 double greyscale values $\in [0, 2]$ of the pixels. The class labels are in the range of $\{0, \dots, K - 1\} = \{0, \dots, 9\}$

In the following exercise you will be asked to visualize some example digits. Do not write an image viewer! Write a routine to output some of the images of the digits in .pgm ASCII format files and view them with e.g. xv (xv -expand 20 -raw example.pgm \rightarrow ). .pgm format and example:

P2	P2
N_columns N_rows greyscale_range	4 3 255
11 12 ... 1N	0 255 0 255
...	255 0 255 0
M1 M2 ... MN	50 100 50 100

The pixel values should be integers between 0 and greyscale_range, which is usually 255. Scale the original float values appropriately before writing them to the file.

- a) Implement a Euclidean distance nearest-neighbor classifier. Write a *separate function* that calculates the distance (use the squared distance)! You should keep the entire training data in memory to reduce runtime. What is the classification error rate you get?
- b) Compute a confusion-matrix (i.e. a matrix or table m where the entry m_{ij} corresponds to the number of times a sample from class i has been classified into class j). Which digit is recognized most reliably? Which are the most frequent confusions?

- c) Visualize some examples of correct and incorrect classification as .pgm images, showing the test sample and the nearest-neighbor.

Use the original USPS data sets again to investigate the following improvements.

- d) Implement a k -nearest-neighbor classification using a simple majority vote. Construct a table showing how the error rate depends on k . The program can be written such that values for all k in a given range are computed in one pass!
- e) Augment your training data set by shifting all training samples in a ± 1 pixel neighborhood \Rightarrow 1 original training sample and 8 additional “virtual” ones. Repeat your single nearest neighbor experiment on this data.
- f) Think of other possible (simple) ways of improving the results and implement one of your ideas.

Tangent Distance

- g) You can obtain an open-source version to determine tangent vectors for an image at <http://www-i6.informatik.rwth-aachen.de/~keyzers/td/>. Use this function to visualize the tangent vectors of the first training data vector from each of the ten classes of the USPS data, yielding 70 images. (Note that the tangent vectors may have negative values!) You may do this inside the nearest-neighbor program (e.g. if called with the option `-visualize`).

To determine the distance of a feature vector to the linear subspace spanned by the tangent vectors there exist different methods. One is to calculate an orthonormal basis for the subspace and then use projections. A function that determines an orthonormal basis of the subspace, given the tangent vectors is included with the source code. Here, again, there exist several methods and the one used is the Gram-Schmidt orthogonalization. Make sure that the routine operates correctly by checking that:

- the output vectors have length 1
 - two different output vectors have a dot product (scalar product) of 0
 - the output vectors span the same subspace as the input vectors (choose one of the different possibilities to check this)
- h) Implement a function that returns the (one-sided) (squared) tangent distance of two given input vectors with respect to a given set of orthonormalized vectors. For the use in your classifier, replace the Euclidean distance with this function. Choose the tangents of the test sample as basis for the subspace. Orthonormalize these vectors using the function you implemented. Store the resulting vectors in memory (as they are needed in more than one distance calculation). What is the resulting error rate of the k -nearest-neighbor classifier? In another experiment, choose the tangents of the training samples. What error rate do you obtain? Why is it computationally more expensive to use the two-sided tangent distance? (Optionally, implement the two-sided tangent distance.)
- i) Try the improvements from the Euclidean distance classifier with the tangent distance. What result do you obtain, and why?

Exercise 1.12 (Improving OCR)

(100 Points)

Find a way to meaningfully improve the open source OCR system Tesseract. (It might be beneficial to discuss your idea before starting any implementation.)