Anders Bjorholm Dahl
Vedrana Andersen Dahl

Advanced Image Analysis

SELECTED TOPICS

DTU Compute, March 11, 2020

Contents

Preface 5

1 *Introduction* 7

Part I Feature-based image analysis

2 *Scale-space* 23

3 *Feature-based segmentation* 29

4 *Feature-based registration* 35

Part II Image analysis with geometric priors

5 *Markov random fields* 43

6 *Deformable models* 53

Preface

This lecture note is a collection of topics for the students taking the course 02506 Advanced Image Analysis at Technical University of Denmark. The note provides background material for the course together with practical guidelines and advice for carrying out tasks in image analysis. The topics are selected to represent problems that you typically meet as an engineer that specialize in image analysis.

Image analysis is a rapidly growing field of research with a wealth of methods constantly being developed and published. This note is not intended to give a complete overview of the field. Instead, we focus on general principles for image analysis with the aim of giving students the skill-set needed for exploring new methods. General principles relate to identifying relevant image analysis problems, finding suitable methods for quantification, implementing image analysis algorithms and verifying their performance. This requires programming skills and the ability to translate a mathematical description into an efficient functioning program.

Image analysis methods published in scientific articles can be challenging to implement as a computer program, since they are described using mathematical notation, which my vary between articles. In some cases the notation can be very different from the code you need to write. One aim with this note is to guide the implementation of image analysis algorithms from descriptions in articles to the functioning programs. This is done through examples, practical tips, and advice on designing useful test to ensure that the obtained implementation gives the expected output.

There are several papers and book chapters that describe the methods to be implemented during the course. These are integral parts of the course curriculum, and should be read when working with the examples in this note.

The structure of the note is as follows. First comes a general introduction to a few central aspects relevant for image analysis along with the first introductory exercise, which has the purpose of refreshing basic image analysis concepts. The main text includes three compul-

sory exercises. Towards the end of the note we provide a number of examples for the final exercise in form of a mini-project.

During the course you may be implementing methods and algorithms that are already integrated in existing software libraries. These commercial or public implementations might be better than what you can achieve given the time available for the exercises. The reason to redo what other people have already done is to gain insight and understanding of how image analysis methods work, and to give you the skill-set needed for implementing or modifying advanced methods where there might not be an available implementation. It can however be a good idea to use existing implementations for evaluating your implementation.

1 Introduction

We refer to images as regularly sampled signals in 2D or 3D space that typically represent a measurement, e.g. a measure of light intensity. In image analysis, we use the image to obtain some information about the signal we have measured. There are aspects to consider when working with images regarding what they represent and how they were created, that we will discuss here.

We start with the mathematical notation of images. One way of representing an image using mathematical notation is as a function $I(x, y)$ with $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}$, which means that each coordinate (x, y) in the image domain Ω a scalar value $I(x, y)$ is assigned. Typically the image is sampled at integer values, e.g. running from 1, such that $x \in \{1, ..., X\}$ and $y \in \{1, ..., Y\}$. There can be some variation between scientific articles on the notation of an image, e.g. that it is implicitly assumed that the image lives in 2D space, and the notation would simply be a symbol like I.

A 3D image is typically termed a volume, but again we can model it as a function $I : \Omega \subset \mathbb{R}^3 \rightarrow \mathbb{R}$. Volumetric images are often reconstructed from projection data obtained using a scanner, e.g. a CT or an MRI scanner. Again the image is represented as a function that maps to a scalar value, but here from a 3D coordinate (x, y, z). In a volumetric image, the three spatial dimensions encode intensity information similar to a 2D image. This means that if we apply operators on a volumetric image, we would use a 3D operator, e.g. an averaging filter in three dimensions. In some cases the sampling is anisotropic, which is typically seen in e.g. medical CT images, and this can influence the applied analysis methods.

Spectral images have multiple measures in each pixel (sometimes represented as a vector) that encode the recorded spectral bands. A common example are the RGB images where $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^3$ encodes the red, green, and blue bands. If more spectral bands are recorded, we are typically talking about multispectral or hyper-spectral images where $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^n$ normally with $n > 3$. For 2D spectral images we would often apply operators in the spectral bands independently. Using

the smoothing example from before, but now in a RGB image, would be a 2D smoothing in the R-band, G-band, and B-band respectively.

Another common image-related representation is a movie. A movie is a set of consecutive images also called frames sampled over time, which we can model as $I(x, y, t)$ where $I : \Omega \subset \mathbb{R}^3 \to \mathbb{R}$ for a gray scale movie or $I : \Omega \subset \mathbb{R}^3 \to \mathbb{R}^3$ for an RGB movie. You can also have a multispectral movie ($I : \Omega \subset \mathbb{R}^3 \to \mathbb{R}^n$ with $I(x, y, t)$) or a volumetric movie ($I : \Omega \subset \mathbb{R}^4 \to \mathbb{R}$ with $I(x, y, z, t)$). For movies we would typically expect small changes between frames, and this can be utilized in the analysis.

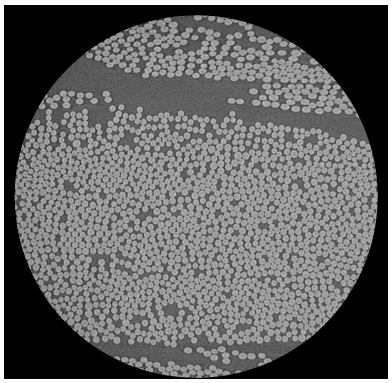

Figure 1.1: Slice of a CT image of glass fibers viewed orthogonal to the fiber direction.

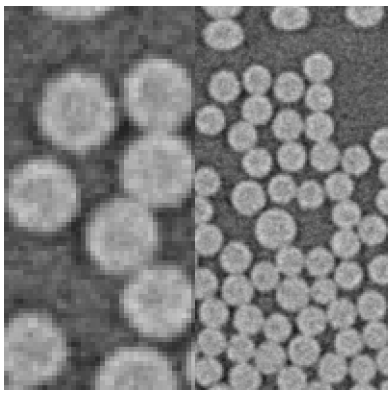

Figure 1.2: Two zoomed in images from image shown in Figure 1.1.

1.1 *Introductory exercise*

This exercise is aimed at refreshing or introducing concepts from basic image analysis curriculum and other related subjects. It contains some topics that will be useful at a later stage in the course. You are expected to carry out the first exercises, whereas the last exercises are not mandatory, but you are welcome to read or solve those exercises also.

1.1.1 *Image convolution*

Image convolution is a central tool in image analysis, and in this exercise you will investigate some properties of image convolution with a Gaussian kernel and its derivatives.

For two continuous functions convolution is defined as

$$(f * g)(x) = \int_{-\infty}^{\infty} f(x - \tau) g(\tau) \mathrm{d}\tau \, . \tag{1.1}$$

Convolution is commutative, but we usually distinguish between the signal and the kernel, and we say that the signal f is convolved with the kernel g.

For a discrete sampled signal we get

$$(f * g)(x) = \sum_{i=-l}^{l} f(x - i) g(i) \, . \tag{1.2}$$

A convolution with a square kernel in 2D is given by

$$(f * g)(x, y) = \sum_{i=-l}^{l} \sum_{j=-l}^{l} f(x - i, y - j) g(i, j) \, . \tag{1.3}$$

In image analysis, Gaussian kernel is often used used for image smoothing by filtering. The 1D Gaussian is defined by

$$g(x; t) = \frac{1}{\sqrt{2t\pi}} e^{-x^2/(2t)} \, , \tag{1.4}$$

where $t = \sigma^2$ is the variance of the Gaussian normal distribution. The 2D isotropic Gaussian is given by

$$g(x, y; t) = \frac{1}{2t\pi} e^{-(x^2+y^2)/(2t)} \ . \tag{1.5}$$

The Gaussian is separable, which means that we can convolve the image using two orthogonal 1D Gaussians, and obtain the same result as when convolving with 2D Gaussian of the same variance. This can speed up convolutions significantly, especially for large convolution kernels.

Another property of the Gaussian convolution is the so-called *semi-group structure*, which means that we get the same convolution using a single large Gaussian as we get using several small ones

$$g(x, y; t_1 + t_2) * I(x, y) = g(x, y; t_1) * g(x, y; t_2) * I(x, y) \ , \tag{1.6}$$

where I is an image. On the right part of equation, the order of convolution does not matter, as convolution is associative.

When computing various features for image $I(x, y)$, we often need to know a local change in intensity values for all positions (x, y). This can be achieved by taking the spatial derivative of the image. Since the image is a discretely sampled signal, we can only compute an approximation of the derivative. For example, we can take the difference between neighboring pixels. On the other side, we often want to smooth the image in connection with taking the derivative.

However, it turns out that if we want to convolve the image with a Gaussian and then take the derivative, we can instead convolve the image with the derivative of the Gaussian

$$\frac{\partial}{\partial x} (I * g) = \frac{\partial I}{\partial x} * g = I * \frac{\partial g}{\partial x} \ . \tag{1.7}$$

Since we can compute the derivative of the Gaussian analytically, we get an efficient and elegant approach to computing a smoothed image derivative.

The analytic 1D Gaussian derivative is given by

$$\frac{\mathrm{d}}{\mathrm{d}x} g(x) = \frac{x}{\sigma^3 \sqrt{2\pi}} e^{-x^2/(2\sigma^2)} \ . \tag{1.8}$$

The semi-group structure also holds for image derivatives, such that we get

$$\frac{\partial}{\partial x} g(x, y; t_1 + t_2) * I(x, y) = \frac{\partial}{\partial x} g(x, y; t_1) * (g(x, y; t_2) * I(x, y)) \ , \tag{1.9}$$

which implies that we can convolve with a large Gaussian derivative kernel or we can convolve with a smaller Gaussian and a smaller Gaussian derivative and get the same result.

Data For this exercise you will use an X-ray CT image of fibres `fibres_xcth.png`, shown in Figure 1.1 and Figure 1.2.

Tasks

1. Experimentally verify the separability of the Gaussian convolution kernel. Do this by convolving a test image with 2D kernel, and convolving the same image with two orthogonal 1D kernels. Subtract the result and verify that the difference is very small.
2. Investigate the difference between the derivative of the image convolved by a Gaussian and the image convolved with the derivative of the Gaussian as described in Eq. 1.7. Note that you can compute the derivative of the image by convolving with the kernel `k = [0.5, 0, -0.5]`.
3. Test if a single large convolution with a Gaussian of $t = 20$ is equal to ten convolutions with a Gaussian of $t = 2$.
4. Test if convolution with a large Gaussian derivative

$$I * \frac{\partial g(x, y; 20)}{\partial x} ,$$

is equal to convolving with a Gaussian with $t = 10$ and a Gaussian derivative with $t = 10$

$$I * g(x, y; 10) * \frac{\partial g(x, y; 10)}{\partial x} .$$

1.1.2 *Computing length of segmentation boundary*

Segmentation is one of the basic image analysis tasks, and we will also cover a few segmentation methods in the course. For an outcome of a segmentation, as for example shown in Figure 1.3, it may be important to measure some quality of the result. One relevant measurement is a length of the segmentation boundary.

Assume that segmentation is represented by an image $S(x, y)$ which takes n discrete values, i.e. $S : \Omega \to \{1, 2, \dots n\}$, where n is the number of segments. We define the length of the segmentation boundary as

$$L(S) = \sum_{(x,y)\sim(x',y')} d\left(S(x, y), S(x', y')\right) ,$$

where $(x, y) \sim (x', y')$ indicates two neighboring pixel locations, and d is a discrete metric

$$d(a, b) = \begin{cases} 0 & \text{if } a = b \\ 1 & \text{otherwise} \end{cases}$$

which in this case operates on pixel intensities. In other words, $L(S)$ counts all occurrences of two neighboring pixels having different labels.

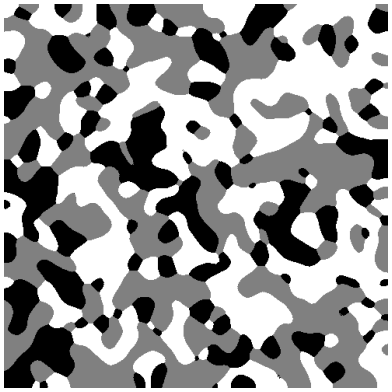

Figure 1.3: Image of a segmented fuel cell with three phases. Black represents air, grey is cathode, and white is anode.

Tasks

1. Compute the length of the segmentation boundary for provided segmentation images of a fuel cell, where one is shown in Figure 1.3. For efficient and compact implementation, avoid loops and use vectorization provided by Matlab or numpy.

2. Collect your code in a function which takes segmentation as an input, and returns the length of the segmentation boundary as an output. You function will be useful when we will be working with Markov random fields later in the course.

Data In this exercise you should use the volume slice `fuel_cell_1.tif`, `fuel_cell_2.tif`, and `fuel_cell_3.tif` that you can find on Campusnet.

1.1.3 *Curve smoothing*

A segmentation boundary may be explicitly represented using a sequence of points connected by line segments, which typically delineates an object in the image. Assume that N-times-2 matrix $\mathbf{X}$ contains x and y coordinates of N points which define a closed curve, a so-called snake [1].

[1] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988

To impose smoothness to this representation, we will need to smooth the snake. This can be achieved in a simple way by displacing every snake point towards the average of its two neighbors, possibly iteratively. Point displacement can be seen as a result of filtering the snake with a kernel $\lambda \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$, where λ is a parameter controlling the magnitude of the displacement. For efficiency, we want to implement the snake-smoothing step as

$$\mathbf{X}^{\text{new}} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{X} \tag{1.10}$$

where $\mathbf{L}$ is a N-times-N matrix with elements 1, -2, and 1 in every row such that -2 is on the main diagonal, and 1 on its left and right (also circularly in the first and the last row), and zeros elsewhere.

Confirm that $\lambda = 0.5$ displaces every snake point exactly to the average of its neighbors. Try smoothing one of the provided contours, also shown in Figure 1.4. We have included both original and noisy curves.

Maybe you noticed two important limitations of our simple approach. First, for larger values of λ the curve will start oscillating, but using a small λ requires many iterations of the smoothing step for a noticeable result. Second, smoothing leads to the shrinkage of the curve.

Stability issues can be avoided by evaluating the displacement on the new snake $\mathbf{X}^{\text{new}}$. In other words, we can use an implicit (backwards

Euler) approach. Instead of Equation 1.10 where $\mathbf{X}^{\text{new}} = \mathbf{X} + \lambda \mathbf{L}\mathbf{X}$ we use $\mathbf{X}^{\text{new}} = \mathbf{X} + \lambda \mathbf{L}\mathbf{X}^{\text{new}}$ leading to

$$\mathbf{X}^{\text{new}} = (\mathbf{I} - \lambda \mathbf{L})^{-1}\mathbf{X}. \tag{1.11}$$

We can now choose an arbitrary large λ and obtain the desired smoothing in just one step. The price to pay is matrix inversion, but for many applications, this needs to be computed only once.

Shrinkage is caused by the kernel which minimizes curve length. Instead, we can use a kernel which minimizes the curvature, or even better, we can weight the elasticity (length minimizing) and rigidity (curvature minimizing) term. The kernel with the two contributions is

$$\alpha \begin{bmatrix} 0 & 1 & -2 & 1 & 0 \end{bmatrix} + \beta \begin{bmatrix} -1 & 4 & -6 & 4 & -1 \end{bmatrix}$$

as derived in [2], section 3.2.4, with α and β weighting the two terms.

[2] Chenyang Xu, Dzung L Pham, and Jerry L Prince. Image segmentation using deformable models. *Handbook of medical imaging*, 2:129–174, 2000a

Tasks

1. Implement curve smoothing as in Equation 1.10 and test it for various values of λ. Try using smoothing iteratively to achieve a visible result for small λ.

2. Implement curve smoothing as in Equation 1.11 (implicit smoothing) and test it for various values of λ. Do you need an iterative approach of this smoothing?

3. Implement implicit curve smoothing but with the extended kernel. This means that your implementation instead of $\lambda\mathbf{L}$ uses a matrix that combines elasticity and rigidity, and has two parameters instead of λ. Note also that this matrix is a (sparse) circulant matrix. Test smoothing with various values of α and β. What do you achieve when choosing a large β and small α?

4. Implement a function which returns a smoothing matrix needed for implicit smoothing with the extended kernel. You will be using this when working with deformable models later in the course.

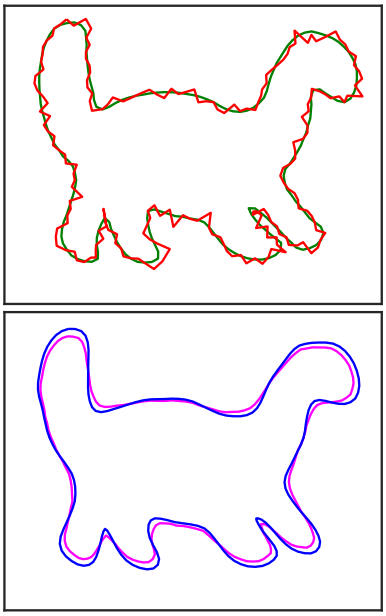

Figure 1.4: Top image shows the dinosaur curve in green, while red shows the curve with added noise. Bottom image shows two smoothing results with different α and β weights.

Data In this exercise you should use the curves given as text files containing point coordinates `dino.txt`, `dino_noisy.txt`, `hand.txt`, and `hand_noisy.txt`.

1.1.4 *Optional: Total variation*

In many image analysis applications, such as image denoising and image segmentation, we are interested in producing a result which has

a quality that we loosely call *smoothness*. A common way of estimating smoothness is by considering a total variation defined for an image I as

$$V(I) = \sum_{x \sim x'} |I(x) - I(x')|,$$

where $x \sim x'$ indicates two neighboring pixel locations. We expect smooth images to have a low total variation.

Implement a function which computes the total variation of a 2D grayscale image and test it on the image shown in Figure 1.1 and Figure 1.2. Use Gaussian smoothing to remove some of the noise from the image, and confirm that the smoothed image has a smaller total variation.

Data In this exercise you should use the volume slice `fibres_xcth.png` that you can find on Campusnet.

1.1.5 *Optional: Unwrapping image*

A solution to image analysis problem may involve geometric transformations. When working with spherical or tubular objects, we sometimes want to represent an image in polar coordinate system. Implement a function which performs such *image unwrapping* using a desired angular and radial resolution. Use your function to unwrap one of the slices from the dental data set, an example is shown in Figure 1.5 and 1.6. Unwrapping will be useful when we will be working with deformable models later in the course.

Data In this exercise you should use one of the central slices from the `dental` folder that you can find on Campusnet.

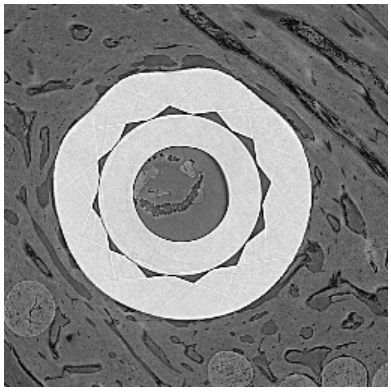

Figure 1.5: Image of a dental implant that should be unwrapped.

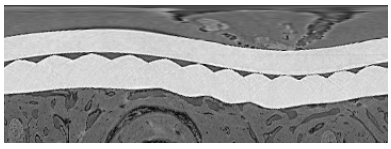

Figure 1.6: Unwrapped image of a dental implant.

1.1.6 *Optional: Working with volumetric image*

To give you a taste of working with 3D images, we have prepared a small data set containing slices from an X-ray CT scan. By convention in X-ray imaging, dense structures (having a height X-ray attenuation) are shown bright compared to less dense structures. Furthermore, the direction given by image slices is most often denoted z. The volume you are given contains a metal (very bright) object. Show orthogonal cross sections of the object, see Figure 1.7. Can you determine an optimal threshold for segmenting the object from the background?

Optionally, show a volumetric 3D rendering of the thresholded object using any available software. An example is shown in Figure 1.8. If you are using MATLAB, check a function `isosurface`.

Data In this exercise you should use the volumetric image stored as individual slices in the folder called `dental` that you can find on Campusnet.

1.1.7 *Optional: PCA of multispectral image*

Principal component analysis (PCA) is a linear transform of multivariate data that maps data points to an orthogonal basis according to maximum variance. A basic introduction in PCA is given in [3], and here we will apply it on a multispectral image.

We provided an image acquired with the VideometerLab, which is a multispectral imaging device, that uses coloured LED's to illuminate a material, in this case samples in a petri dish. This gives an 18 channel image $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^{18}$ where each channel corresponds to a wavelength, and channels cover the range from 410 nm to 955 nm, i.e. the visible and near-infrared spectrum. The image depicts vegetables on a dish and is shown in false colours in Figure 1.9.

The aim of this exercise is to carry out PCA and visualise the principal components as images. PCA can be done by eigenvalue decomposition of a data covariance matrix. In our analysis we view each pixel as an observation, so we rearrange I into a N-by-18 data matrix $\mathbf{X}$. Each row of $\mathbf{X}$ represents one pixel (observation), with the successive columns corresponding to wavelengths (variables).

Data covariance matrix $\mathbf{C}$ is defined as

$$\mathbf{C}_{ij} = \frac{1}{N-1} \sum_{n=1}^{N} (\mathbf{X}_{ni} - \mu_i)(\mathbf{X}_{nj} - \mu_j), \tag{1.12}$$

where $i, j \in \{1, ..., 18\}$, and μ is a 18 dimensional empirical mean vector computed for each variable.

Covariance matrix $\mathbf{C}$ can be computed as a matrix product

$$\mathbf{C} = \frac{1}{N-1} \overline{\mathbf{X}}^T \overline{\mathbf{X}}, \tag{1.13}$$

where $\overline{\mathbf{X}} = \mathbf{X} - \mathbf{1}_{n\times 1}\mu^T$ is a zero-mean matrix obtained by independently centering each row of $\mathbf{X}$ around its mean value. Convince yourself that is correct.

Principal components are given by the eigenvectors of $\mathbf{C}$, e.i. vectors such that $\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i$. Eigenvector corresponding to the largest eigenvalue gives the direction of the largest variance in data, the eigenvector corresponding to the second largest eigenvalue is the direction of the largest variance orthogonal to the first principal direction, etc. The projections of the data points onto principal directions $\hat{\mathbf{X}}\mathbf{v}_i$ can be rearranged back into image grid, and viewed as images.

If $\mathbf{V}$ is a matrix containing eigenvectors in it columns, all principal

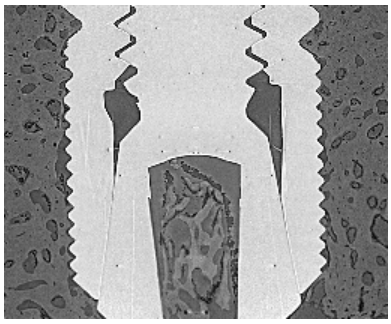

Figure 1.7: A longitudinal slice (an xz-plane) of the volumetric image of a dental implant.

Figure 1.8: 3D rendering of the thresholded volumetric image of a dental implant.

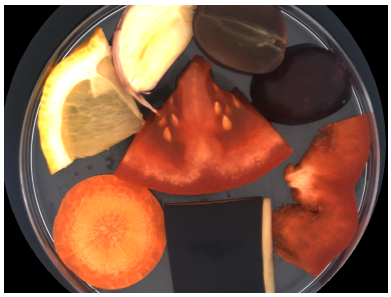

Figure 1.9: False colour image obtained from an 18 band VideometerLab image.

[3] Lindsay I Smith. A tutorial on principal components analysis. Technical report, 2002

components can be computed as

$$\mathbf{Q} = \overline{\mathbf{X}}\mathbf{V}. \tag{1.14}$$

Data In this exercise you should use the images in the folder called `mixed_green` which contains png-images. You find the file on Campusnet.

Suggested approach The following steps takes you through computing the principal components.

1. Write a script to read in the images and display them. Convince yourself that there is a difference between the spectral bands. Make sure to change the data type to float or double.
2. Rearrange the image into a matrix $\mathbf{X}$ as described above with one pixel in each row. Compute the column-wise mean μ and subtract this from $\mathbf{X}$ to get the zero mean $\overline{\mathbf{X}}$.
3. Compute the covariance matrix $\mathbf{C}$.
4. Compute the eigenvectors $\mathbf{V}$ and eigenvalues λ.
5. Compute the principal component loadings $\mathbf{Q}$.
6. Rearrange $\mathbf{Q}$ into images and display the result.

You can compare your implementation to an already implemented PCA function in e.g. MATLAB or Python.

1.1.8 *Optional: Bacterial growth from movie frames*

Image data with a temporal component can be stored in the form of a movie. The purpose of this exercise is to read in image frames from a movie and analyze them. The movie contains microscopic images of listeria bacteria growing in a petri-dish acquired at equal time steps. An example frame is shown in Figure 1.10. Your task is to make a small program that visualize bacterial growth by counting the cells.

The image quality is however not very good due to the low resolution and compression artifacts, making it difficult to separate the individual bacteria. So, we make a rough assumption that the number of pixels covered by bacteria is proportional to the number of bacteria. The task is therefore to make a plot of the number of pixels covered by bacteria as a function of time.

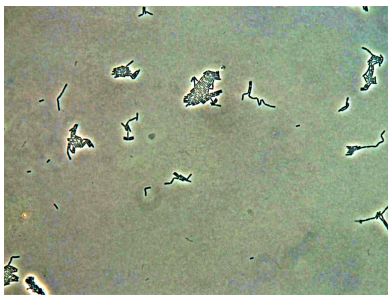

Figure 1.10: Example of microscopic image of listeria bacteria in a petri dish.

Data In this exercise you should use the movie `listeria_movie.mp4` that you can find on Campusnet.

Suggested approach You can for example first read in one representative frame from the movie and build an cell segmentation method. A simple threshold is not sufficient, but with a few processing steps the cells become distinguishable from the background. You can try the following steps:

1. Convert the image I to a grey scale image G.
2. Compute the gradient magnitude $M = \sqrt{(\partial G/\partial x)^2 + (\partial G/\partial y)^2}$ using an appropriate filter.
3. Smooth M using a Gaussian filter.
4. If the parameters have been chosen appropriately, the pixels covering bacteria can now be segmented by thresholding.

When you have made a functioning segmentation model, you can apply this to all images in the movie using the following steps for each image in the movie:

1. Apply the segmentation and sum the bacteria pixels.
2. Store this number in an array.
3. Plot the number of pixels as a function of time.

The obtained curve has a characteristic shape. Can you recognize the function that could describe this shape?

1.1 Information for 02506

The purpose of this exercise is to refresh a few concepts from image analysis.

Requirement to complete the exercise We will be giving a feedback on this exercises during the next exercise session. To make this efficient, you should prepare, such that you can quickly show us your results. This should take less than five minutes. You should give an overview of what you accomplished, and focus on elements that you would like to get feedback on.
We suggest that you prepare a script (e.g. a Jupyter Notebook or MATLAB script) which covers the exercise. Make sure that it runs fast and clearly shows your results.
We will use the same feedback format for other exercises in the course. If there are parts of your code that take long time to run, you should save the results in advance, and load the results for the feedback session.

Exam questions related to this exercise Later in the course, the questions for the oral exam will be given at the end of each exercises. But for this exercise, we have no specific questions. However, the tools used here are either expected to be known before the course or are the concepts used in other exercises. The introductory exercise may therefore be discussed during the examination.

Part I

Feature-based image analysis

Analyzing images using feature-based representations is central to many applications. We have chosen three topics that we will cover. First we will work with scale-space for detecting image features independently of scale. Specifically, we will focus on sale-space blob detection. Then we will investigate features as a basis for pixel classification as a method for segmentation. Finally, we will work with feature-based image registration.

Information for 02506

This part covers week 2 to 4 in the course, with a chapter for each week. For the first week you should read:

- Week 2: Lindeberg (1996): Scale-space: A framework for handling image structures at multiple scales

The reading material is uploaded to DTU Inside (Campusnet).

2 Scale-space

Methods from scale-space allow scale invariant detection of image structures. This means that we can find features like blobs (binary large objects), corners, ridges, edges, and other structures at different scale. When we talk about image features like corners and edges, it is not corners or edges of the physical depicted objects, but corners and edges in the image intensities. To visualize this, you can think of a 2D image as a landscape, with pixel intensities corresponding to height measurements at regularly placed positions. In this landscape, an edge is a line with high abruptly changes. A corner will be a height-change point where two (more or less) orthogonal edges meet, and other types of features can be described in the same way.

Using a feature-based image representation is convenient, because we break up the image into manageable parts that are more descriptive than the individual pixels. Scale invariance, which means that we characterize (make a mathematical description) the same feature shown at different scale in two images, is also very convenient. In e.g. computer vision where images of the same object are often captured from difference distance, it is typically a desired property to be able to measure the features independent of its scale. But it also allows us to measure image structures that are different in size for example from microscope images, as we will be working with here.

Here we will base our work on the article of Lindeberg[1] that gives an introduction to scale-space theory. Scale-space representation has made the basis for a range of image analysis methods and is extensively used in computer vision. In the exercise you will implement scale-space blob detection and use it for detecting and measuring the size of fibres that are imaged using X-ray CT.

[1] Tony Lindeberg. Scale-space: A framework for handling image structures at multiple scales. 1996

The computation of scale-space is done by representing image features at all scales at once and detect features based on criteria that is independent of the scale. Here we will be working with the Gaussian scale-space, and the analysis is in practice done by smoothing the image using a Gaussian filter. In Lindeberg[2] the scale-space representation is defined for a general N-dimensional signal $f : \mathbb{R}^N \to \mathbb{R}$. Here we

[2] Tony Lindeberg. Scale-space: A framework for handling image structures at multiple scales. 1996

will work with a 2D image $I : \mathbb{R}^2 \to \mathbb{R}$. For 2D image, its Gaussian scale-space representation is $L : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}$, which in practice becomes a 3D object, with the two spatial image dimensions (x, y) and the scale in the third dimension. Since scale is obtained by smoothing with a Gaussian, the variable determining the degree of smoothing is the variance t. Also the standard deviation $\sigma = \sqrt{t}$ is used in the article, but here we have simplified the notation and use only the variance t.

The Gaussian scale-space L is defined for N-dimensional signals by

$$L(x;t) = \int_{\xi \in \mathbb{R}^N} f(x-\xi) g(\xi;t) d\xi \tag{2.1}$$

with $g : \mathbb{R}^N \times \mathbb{R}_+ \to \mathbb{R}$ being the N-dimensional Gaussian kernel

$$g(x;t) = \frac{1}{(2\pi t)^{N/2}} e^{-(x_1^2 + \cdots + x_N^2)/(2t)}. \tag{2.2}$$

In practice we will work with the Gaussian scale-space for 2D images on a discrete set of pixels. Therefore, we can write the Gaussian scale-space (ignoring boundary issues) as

$$L(x,y;t) = \sum_{-\gamma}^{\gamma} \sum_{-\delta}^{\delta} I(x-\gamma, y-\delta) g(\delta, \gamma; t) \tag{2.3}$$

where $g : \mathbb{R}^2 \times \mathbb{R}_+ \to \mathbb{R}$ is the 2D Gaussian kernel

$$g(x,y;t) = \frac{1}{2\pi t} e^{-(x^2+y^2)/(2t)}. \tag{2.4}$$

Computing the scale-space is done at a discrete set of steps, and we have the start condition with $t = 0$ defined as $L(x, y; 0) = I(x, y)$.

For feature detection, we need to compute the derivatives of a scale-space representation. Note that this is conveniently achieved by convolving an image with a kernel that is a derivative of a Gaussian. Blob detection uses second order derivatives, more precisely the Laplacian of a Gaussian $\nabla^2 L = L_{xx} + L_{yy}$ which gives a high response where there is a blob in the image. To detect blobs we need to find local maxima and minima of the Laplacian.

What we still need to do is to ensure that we can detect blobs across different scales. The image in scale-space representation is increasingly smoothed, and with increasing scale t, pixels will change their intensity value towards the average value of the image. Therefore, the absolute values of derivatives will become smaller when increasing t. For blob detection, this means that the magnitude of the local maxima and minima in the scale-space of the Laplacian $\nabla^2 L$ will decrease and this smoothing must be compensated. The compensation factors for different features are given in Lindeberg[3] and for the blob feature it is t such that the scale normalized Laplacian of Gaussian is $t\nabla^2 L$.

[3] Tony Lindeberg. Scale-space: A framework for handling image structures at multiple scales. 1996

2.1 *Exercise 2 – part I, Scale-space blob detection*

In this part of the exercise you will implement scale-space blob detection with the purpose of detecting and measuring glass fibres from images of a glass fibre composite. An image example is given in Figure 2.1, that shows a polished surface of a glass fibre composite sample, where individual fibres can be seen. Since these fibres are relatively circular we will model them as circles. This means that we must find their position (center coordinate) and diameter, and for this we will use the scale-space blob detection. After having computed the fibres parameters, we will carry a statistical analysis of the results.

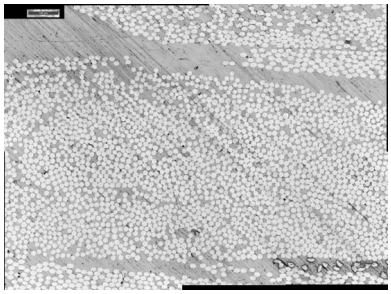

Figure 2.1: Example of fibre image acquired using an optical microscope.

2.1.1 *Computing Gaussian and its second order derivative*

We will approach this analysis in steps that lead to the final algorithm. First we will use synthetic data to develop and test our algorithm, and after that we will carry out the analysis on the real images. Since we focus on blob detection, we must have a Gaussian kernel and its second order derivative. Since the Gaussian is separable, we can employ 1D filters for our analysis. The 1D Gaussian is given by

$$g(x) = \frac{1}{\sqrt{2\pi t}} e^{\frac{-x^2}{2t}} \ . \tag{2.5}$$

Suggested procedure

1. Derive (analytically) the second order derivative of the Gaussian

$$\frac{\mathrm{d}^2 g}{\mathrm{d}x^2} \ .$$

2. Implement a function that takes the variance t as input and outputs a filter kernel of g and $\mathrm{d}^2 g / \mathrm{d}x^2$. You should use a filter kernel of at least $3t$. Why?

3. Try the function on the synthetic test image `test_blob_uniform.png`.

2.1.2 *Detecting blobs on one scale*

Blobs can be found as spatial maxima (dark blobs) or minima (bright blobs) of the scale-space Laplacian

$$\nabla^2 L = L_{xx} + L_{yy} \ . \tag{2.6}$$

Suggested procedure

1. Compute the Laplacian at one scale using the synthetic test image `test_blob_uniform.png`.

2. Build a function that detects the coordinates of maxima and minima in the Laplacian image (detect blobs).
3. Plot the center coordinates and circles outlining the detected blobs. The radius of the circles should be $\sqrt{2t}$.
4. Try varying t such that the blobs in `test_blob_uniform.png` are exactly outlined.

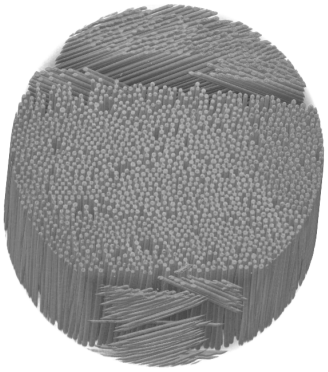

Figure 2.2: Visualization the 3D fibers scanned with the high resolution X-ray CT-scanner.

2.1.3 *Detecting blobs on multiple scales*

To find blobs at multiple scales, we must use the scale-space representation. This can conveniently be done by representing $\nabla^2 L$ as a 3D array (volumetric image).

Suggested procedure

1. Decide on scales at which the Laplacian must be computed. You could make it equal steps in the size of the blobs ($\sqrt{2t}$).
2. Compute the scale normalized scale-space Laplacian $t\nabla^2 L$ for the test image `test_blob_uniform.png`.
3. Find coordinates and scales of maxima and minima in this scale-space and plot the detected blobs on top of the image. What are the detected scales?
4. Detect blobs in the test image `test_blob_varying.png`.

2.1.4 *Detecting blobs in real data*

We will now continue with the real images of fibers. The fibre data is obtained using different scanning methods including scanning electron microscopy (`SEM.png`), optical microscopy (`Optical.png`), synchrotron X-ray CT (`CT_synchrotron.png`), and three resolutions of laboratory X-ray CT (`CT_lab_high_res.png`, `CT_lab_med_res.png`, `CT_lab_low_res.png`). The CT data is a single slice very close to the top, so we assume the data to be from the same part of the sample, and this allows us to directly compare the fibers. We will do this comparison in next exercise, but in this we will compute the fiber location and their diameter. In Figure 2.2 you can see a visualization of the fibre data from the high resolution X-ray CT scan.

We start by testing the blob-detection on this real data.

Suggested procedure

1. Run your blob-detection function from above on a cut-out example one of the images. It is important that you tune your parameters to get the best possible results.

2.1.5 *Localize blobs*

It is difficult to detect blobs in the scale-space Laplacian, such that all fibers are found. To overcome this, we will detect the fibers as maxima in a Gaussian smoothed image. Since the fibers are almost the same size, we can use a single scale of the Gaussian to detect the fiber centers.

Suggested procedure

1. Smooth an image of fibers with a Gaussian and visualize the result.
2. Find locations of maxima in this image and plot the positions on top of the original image.
3. Compute the scale-space Laplacian for the image.
4. Find the scale of each fibre as the minimum over scales at the fiber locations.
5. Plot circles according to the found scale on top of the original image.
6. Detect fibers in all six fiber images. Save the locations and diameters.

In the next exercise, where you will work with image matching based on SIFT features, you will use the results obtained in this exercise. So, next time it will be possible to continue working on the parts that you did not finish here.

2.1 Information for 02506

This is the first part of the exercise in Part I, and will be continued the following two weeks. Please prepare a short (5 minutes presentation) of your code and results.

Exam questions related to this exercise

- Explain scale-space blob detection. This includes basic theory of scale-space blob-detection, explanation of the algorithm you made, and discussion of results.

3 Feature-based segmentation

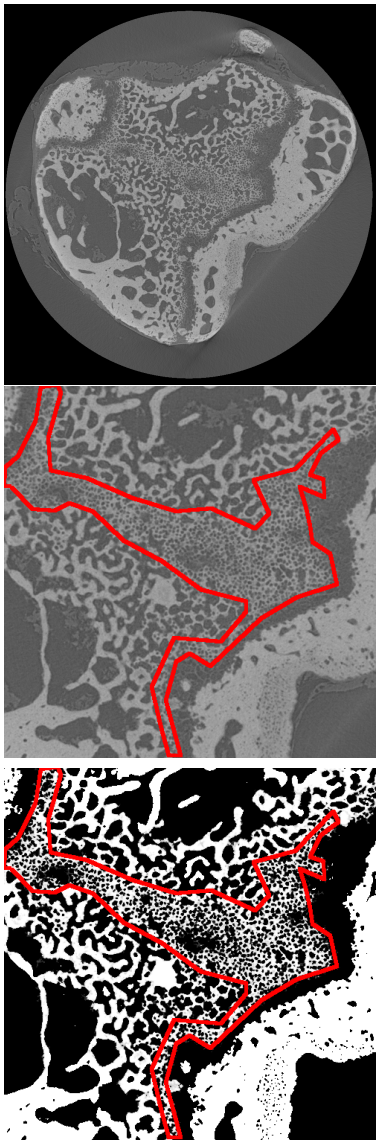

Figure 3.1: Slice of a CT-scan of a trabecular mouse bone. Top image shows the full image, middle is a zoom on the central part with the fine trabecular structure outlined by the red line, and bottom shows the same with a threshold.

Segmentation of an image is done by partitioning the image. This can be described by a function $g(x, y) \rightarrow \ell$ that for each position (x, y) in the image I maps to a label $\ell \in 1, \ldots, L$. Here we will do segmentation by assigning a label to each pixel in the image.

The simplest pixel labeling is obtained by classifying each pixel according to its intensity. In a gray-scale image, this will typically be one or more threshold values that separates the intensities into groups. For many image segmentation problems, this is not sufficient for obtaining a suitable segmentation as illustrated by the bone example in Figure 5.5. Here the fine structures cannot be separated by the pixel intensity alone, but we must utilize information in the image patterns such as size and shape of the depicted structures.

Instead of using the intensity in one pixel, we can use a neighborhood around a pixel to obtain our segmentation. Hereby, we capture the local appearance of the image, and we will try to use that for labeling each pixel in the image. The local appearance is also known as image texture.

3.1 *Supervised feature-based segmentation*

We will approach the segmentation as a supervised labeling problem, where we learn the parameters of our segmentation model from training data. The training data is obtained from labeled training images, which e.g. can be done manually. Now the segmentation problem is to transfer the labels from this image to an unknown test image of the same type, meaning an image with similar appearance.

3.1.1 *Image features*

The local appearance of the image can be computed as image features, and in this exercise we will compute two image features. The first feature is obtained by computing a set of image derivatives by convolving with Gaussians and its derivatives. The second is obtained as

patches centered on a pixel. There are many other features that we could choose, e.g. SIFT[1], SURF [2] or HOG [3] features, but here we choose relatively simple features that are easy to compute.

Features from a Gaussian and its derivatives Similar to the previous exercises, we can smooth and compute image derivatives by convolving with a Gaussian kernel and its derivatives. The area over which we want to compute the Gaussian is determined by the scale of the Gaussian kernel.

The Gaussian features are obtained as a stack of Gaussian derivatives. Let us use the same notation of Gaussian derivatives as we did in the scale-space exercise, such that we have

$$L = I * g(t) \ ,$$

where L is an image I convolved with a Gaussian

$$g = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$$

at scale t, where $t = \sigma^2$. The images obtained from convolution with Gaussian derivatives are given by

$$L_x = I * \frac{\partial g}{\partial x} \ , \ L_y = I * \frac{\partial g}{\partial y} \ , \ L_{xx} = I * \frac{\partial^2 g}{\partial x^2} \ , \ L_{xy} = I * \frac{\partial^2 g}{\partial x \partial y} \ ,$$

etc. In this exercise, we will compute the higher order derivatives until the fourth order, which will result in a 15-dimensional descriptor that captures the local appearance of the image. The descriptor is formed by stacking the Gaussian convolved images, such that each pixel position has an associated 15-dimensional feature vector. That is, we have a $r \times c \times 15$ feature image

$$\begin{aligned} F \quad = \quad & [L, L_x, L_y, L_{xx}, L_{xy}, L_{yy}, L_{xxx}, L_{xxy}, L_{xyy}, L_{yyy}, \\ & L_{xxxx}, L_{xxxy}, L_{xxyy}, L_{xyyy}, L_{yyyy}] \ . \end{aligned}$$

Patch-based features Another way of computing image features is by extracting small patches centered on a pixel and concatenating the pixels into a feature vector. If we e.g. have a 9×9 image patch, this will result in an 81 dimensional feature vector for each pixel position.

3.1.2 *Probabilistic clustering-based segmentation*

The basic idea in the segmentation model that we will use, is that parts of the image that have similar appearance should have the same label. Since the features encode the local appearance of the image, it means that we can give features that are similar the same label.

[1] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2): 91–110, 2004

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005

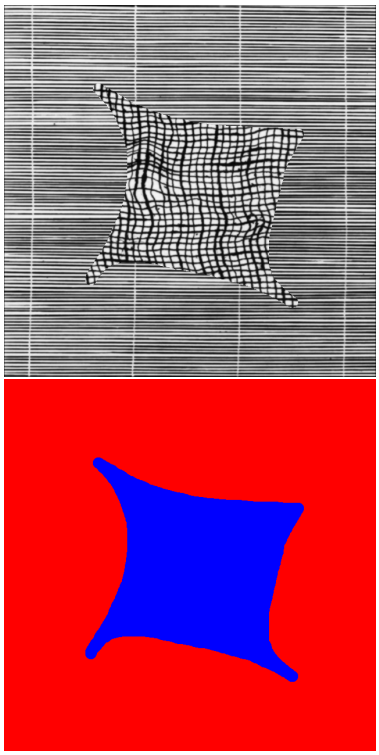

Figure 3.2: Top shows a training image with ground truth labeling. Red is one label and blue is another. Typically, this will be two scalar values, e.g. $\{0, 1\}$.

Using an already labeled image, we can learn the desired labels of the features. In practice, we typically have labels given as a separate image of the same size as the training image. This is shown in Figure 3.2 for one of the two-label images that you will work with in this exercise. Since the training image and the label image are of the same size, for each feature from the training image we can look up the label at a corresponding position in the label image.

By computing image features in both a training and a test image, we can transfer the labels from training image to the test image. This is done by using a similarity measure of the image features, and here we will use Euclidean distance

$$d(f_p, f_q) = \sqrt{\sum_{i=1}^{n} (f_p(i) - f_q(i))^2} \,,$$

where f_p and f_q are two n-dimensional feature vectors.

A direct approach for labeling the image would be to compute the Euclidean distance from each feature vector in the test image to each feature vectors in the training image (or a random subset of the features in the training image). By selecting the label of one or more of the nearest feature vectors, we could obtain a labeling. If we select more than one feature, we can e.g. label according to majority vote. This would be using k-nearest neighbor classifier, which might not always be robust, because outliers could introduce noise.

A more robust approach is using k-means clustering, where each cluster center makes up a representative feature vector. This is sometimes referred to as a dictionary-based approach, where each cluster center is referred to as a *visual word*, and the collection of cluster centers make up a dictionary. Each of the clusters is made up of a number of feature vectors from the training image. This allows us to compute the probability of a given label for each feature cluster

$$p_C(\ell = \lambda) = \frac{1}{m} \sum_{f \in C} \delta(\ell(f) - \lambda) \,, \tag{3.1}$$

where C is a feature cluster with m elements f, $\ell(f)$ is the label of feature element f, and

$$\delta(x) = \begin{cases} 1 & \text{if} \quad x = 0 \\ 0 & \text{otherwise} \end{cases} \,. \tag{3.2}$$

3.2 *Exercise*

You should implement a probabilistic clustering-based segmentation using Gaussian features. If time allows, you should also implement the same using image patches.

There is a set of images available for training and testing named `train_{A,B,C}_image.png` and `test_{A,B,C}_image.png` and `ground_truth.png`, which is the label image. Furthermore, there is a set of images of a bone, which only has labels for training but not for test.

3.2.1 *Gaussian features*

The function for computing the Gaussian feature image is available as the functions `get_gauss_feat_im.m` for MATLAB and `get_gauss_feat_im` function in the `feature_based_segmentation.py` file for Python.

Suggested procedure

1. Read in the images and display them.
2. Compute the feature image.
3. Transform the feature image into a matrix where each column is a feature vector.

3.2.2 *Clustering for building the dictionary*

You should use k-means clustering for building the dictionary. Using all feature vectors would be too time consuming, and it si sufficient to select a random subset of features. You should select both features and image labels from the same pixel positions.

Suggested procedure

1. Select a random subset of feature vectors with corresponding labels (you can use random permutation). If you choose e.g. 5000-10000 vectors, it should be sufficient for clustering.
2. Cluster the feature vectors into a number of clusters. You can choose e.g. 100-1000 clusters.
3. Make a $2 \times n$ matrix to store label probabilities, where n is the number of cluster centers. Compute the probability of a cluster belonging to each of the two labels using Eq. 3.1 and 3.2, and store the probabilities in the matrix.

3.2.3 *Computing probability image and segmenting*

You should now extract image features from the test image. For each of the features you should find the nearest cluster center and assign the label probability to that cluster to the pixel position of the feature. This will result in a probability image of size $r \times c \times 2$, where each pixel has a probability of belonging to one of the two labels.

Suggested procedure

1. Compute a feature image from the test image.
2. Use a nearest neighbor algorithm (`knnsearch` in MATLAB or `Nearest Neighbors` from Scikit Learn in Python) and find the nearest cluster.
3. Build a probability image based on the nearest cluster for each pixel.
4. Obtain a segmentation by selecting the most probable label in each pixel. (You can obtain an improved segmentation by smoothing the probability image).

3.2.4 *Segmentation with patch-based features*

Do the same as above using image patches instead of Gaussian features. This is a little more difficult because of boundary effects. You can extract image patches using the `im2col` function in MATLAB, and we have provided an `im2col` function found in the `feature_based_segmentation.py` file for Python, that has the same functionality.

3.1 Information for 02506

This is the second part of the exercise in Part I, and will be continued next week. Please prepare a short (5 minutes presentation) of your code and results.

Exam questions related to this exercise

- Explain feature-based segmentation using k-means clustering (dictionary-based segmentation).

4 Feature-based registration

Image registration is the process of transforming one image (the moving image) to another (the target image). We will do this by matching image features. Here we will use SIFT features that is described in detail in [1].

The concept of image features from interest points is essential in image analysis. The basic idea is to identify salient positions in an image (unique key-points) and use the surrounding image context to describe the image locally in the form of a descriptor vector. This principle has been extensively used for solving many problems including object recognition, image retrieval (image search), image stitching, geometric reconstruction, etc. Solutions to many of these problems have however been improved by machine learning methods and especially convolutional neural networks have replaced feature-based analysis in a range of applications.

Interest point features are often referred to as hand-crafted features whereas neural networks learn the image features. Since the end-to-end optimization employed in neural networks often results in an improved performance, much work now focuses on neural network methods. There are, however, still problems where interest point features will be a good choice, e.g. for matching images (feature-based image registration), i.e. finding point-wise correspondence between images. Some deep learning based alternatives to hand-crafted features have been suggested[2], showing superior performance. But due to the simplicity of hand-crafted features, where their design make them easy to compute without the time consuming learning part, we will here work with interest point features.

Scale invariant feature transform (SIFT) described in[3] is a widely used interest point feature. Here we focus on SIFT. But bear in mind that many other interest point features have similar properties[4].

We will use SIFT for feature-based image registration. The problem we address is that we are given two images that depict the same object and we want to find an image transformation that aligns the two images. This allows us to compare structures found in one image

[1] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2): 91–110, 2004

[2] Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision*, pages 467–483. Springer, 2016

[3] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2): 91–110, 2004

[4] Tinne Tuytelaars, Krystian Mikolajczyk, et al. Local invariant feature detectors: a survey. *Foundations and trends® in computer graphics and vision*, 3(3):177–280, 2008

with structures found in the other image. In some cases the difference between images can be modeled by a rotation $\mathbf{R}$, translation $\mathbf{t}$ and scale s. This is e.g. the case in microscopy or CT, where pixels have a fixed physical size. A more general affine transformation can be computed using a homography, but in cases where the imaging system can only transform the image by a rotation, translation and scale, these will be the appropriate parameters to compute. Here we will work with sets of corresponding 2D points, where the correspondence is found by matching SIFT features.

Fitting two sets of 2D points in least squares sense. Given two 2D point sets $\hat{\mathbf{p}}_i$ and $\mathbf{q}_i$ (2×1 column vectors), $i = 1, \ldots, n$, we are interested in finding a rotation $\mathbf{R}$, a translation $\mathbf{t}$ and a scale s that minimizes the squared distance between the two point sets. Scale s can be found e.g. by computing the ratio between average distance to the centroid of each point set

$$s = \frac{\sum_{i=1}^{n} ||\mathbf{q}_i - \boldsymbol{\mu}_q||}{\sum_{i=1}^{n} ||\hat{\mathbf{p}}_i - \boldsymbol{\mu}_p||},$$

where $\boldsymbol{\mu}_p = \frac{1}{n}\sum_{i=1}^{n} \hat{\mathbf{p}}_i$ and $\boldsymbol{\mu}_q = \frac{1}{n}\sum_{i=1}^{n} \mathbf{q}_i$ are the centroids of $\hat{\mathbf{p}}_\mathbf{i}$ and $\mathbf{q}_i$ respectively. From this we can compute the scaled point set

$$\mathbf{p}_i = s(\hat{\mathbf{p}}_i - \boldsymbol{\mu}_p) + \boldsymbol{\mu}_p \,.$$

Note that the centroid for the scaled point set is the same as for the original. Now we want to find the rotation and translation that minimize

$$\sum_{i=1}^{n} ||\mathbf{q}_i - \mathbf{R}\mathbf{p}_i - \mathbf{t}|| \,.$$

One way of least-squares fitting 2D point sets involves singular value decomposition of the 2-by-2 covariance matrix

$$\mathbf{C} = \frac{1}{n-1}\sum_{i=1}^{n} (\mathbf{p}_i - \boldsymbol{\mu}_p)(\mathbf{q}_i - \boldsymbol{\mu}_q)^T \,,$$

obtained as

$$\mathbf{U}\Sigma\mathbf{V}^T = \mathbf{C} \,.$$

Here, $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular matrices respectively and Σ is a diagonal 2-by-2 matrix containing the singular values. From this we obtain the rotation

$$\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^T \,.$$

In rare cases this computation can result in a reflection instead of a rotation. If the determinant of $\det(\hat{\mathbf{R}}) = 1$ it is a rotation and if $\det(\hat{\mathbf{R}}) = -1$ it is a reflection. This computation will typically only

result in a reflection e.g. if there are only two points for determining the rotation. Therefore, we can compute the rotation taking this into account by

$$\mathbf{R} = \hat{\mathbf{R}}\mathbf{D} \,,$$

where

$$\mathbf{D} = \begin{bmatrix} 1 & 0 \\ 0 & \det(\hat{\mathbf{R}}) \end{bmatrix} \,.$$

Finally, we find the translation as the average vector from points in $\mathbf{q}$ to the rotated points in $\mathbf{p}$

$$\mathbf{t} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{q}_i - \mathbf{R}\mathbf{p}_i) = \boldsymbol{\mu}_q - \mathbf{R}\boldsymbol{\mu}_p \,.$$

See e.g. Arun et al. [5] which covers a more general 3D case.

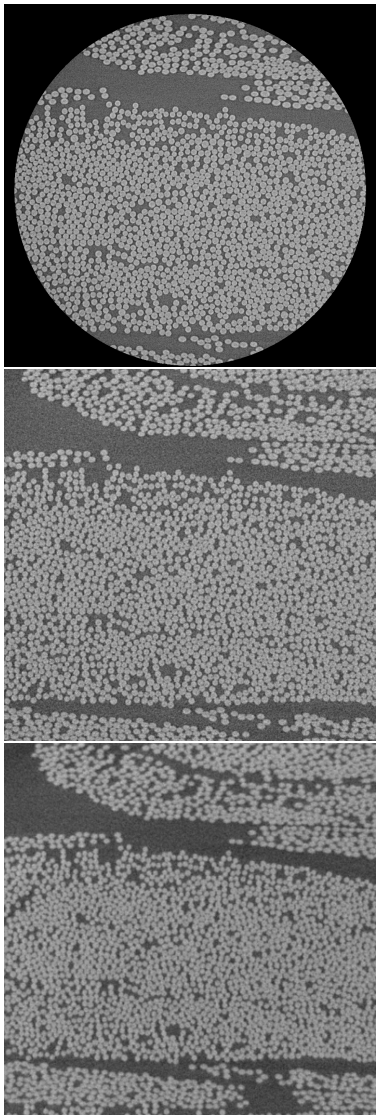

Figure 4.1: Example of an image of the same fiber sample acquired using a CT scanner at three resolutions.

[5] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):698–700, 1987

4.1 *Exercise on feature-based registration*

This exercise aims at finding correspondence between images by matching SIFT features and computing the transformation, i.e. the rotation, translation and scale between the two images. For computing the SIFT features you can use `vlFeat` for MATLAB or `OpenCV` for Python. In the extra exercise you can use the transformation obtained here to compare the fiber diameters that you got from using blob detection for fibers. But for now, the purpose is to compute the transformation.

4.1.1 *Rotation, translation and scale*

You should implement a function that takes two point sets as input and returns the rotation, translation and scale. To ensure that you have the correct implementation, you can make a set of random 2D points that you rotate and translate with known parameters. By applying the transformation to the points, you can verify that you obtain the correct transformation parameters.

4.1.2 *Compute and match SIFT*

Here you should compute SIFT features in two images and match them using Euclidean distance between their descriptor vectors. You should use the criterion by Lowe where a correct match is found if the fraction between the closest feature vector and the second closest feature vector is less than a certain value, e.g. 0.6. There are functionality for matching features in both `vlFeat` and `OpenCV` that you can use. You can also implement you own matching function where you take e.g. scale and rotation into the matching criterion.

To make sure that you compute and match the SIFT features, you can make two corresponding images by rotating and scaling one image and matching it to the original. After that, you can try to match the CT-images of fibers at the three resolutions shown in Figure 4.1. Visualize the matching feature points by drawing lines between them e.g. as shown in Figure 4.2. This allows you to visually evaluate the matching.

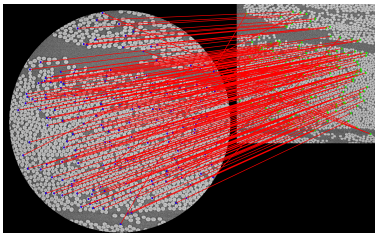

Figure 4.2: Matching SIFT features illustrated by red lines.

4.1.3 *Transform the matched features*

Now you should combine the function for computing the transformation parameters and the SIFT feature matching. The matching will not be perfect and you will most likely see some wrongly matched features. The larger the difference in appearance or scale of the image that is being matched, the more wrongly matched features can be expected. If the majority of the correspondences are correct then the least squares fit will give a relatively good result.

Since we are computing the transformation by a least squares fit the outliers will affect the result to some extent. Outliers can however be removed relatively easy. You can compute the Euclidean distance between the two point sets after you have aligned them. There you will see that most of the distances are relatively small. And if you remove matching points with a distance larger than a certain threshold, you can repeat the computation of the transformation and obtain higher precision in the matching. You should implement a function that makes this two step computation of the transformation and choose a good criterion for a threshold.

Illustrate your transformed feature points by plotting the two point sets on top of each other in the image e.g. as illustrated in Figure 4.3. You should be able to see a difference in the precision of the matching after removing the outliers.

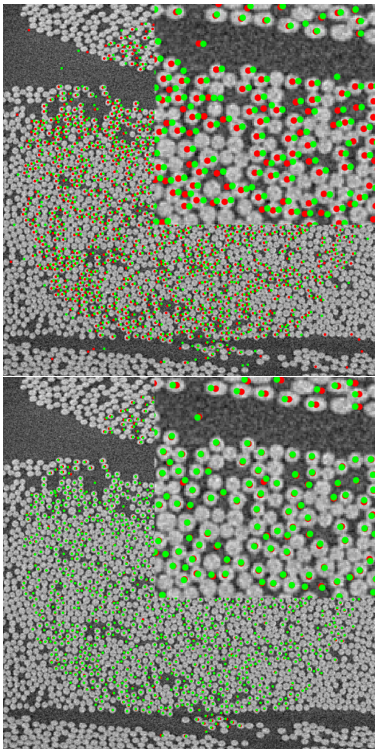

Figure 4.3: Matching features shown in red and green (zoom in upper right corner). Top is after computing least squares of all matching features and bottom is a recomputed match after removing outliers.

4.1 Information for 02506

Exam questions related to this exercise

- Explain the principle of feature-based image registration and how SIFT can be used for feature matching.

Part II

Image analysis with geometric priors

In the context of image analysis, the term *prior knowledge* refers to all the information about problem that is available in addition to the image data. There are numerous ways of incorporating priors when solving image analysis problems, and here we look into Markov random fields which are used to model local contextual information, and two models for handling a distinctive geometry: parametric deformable curves and layered surfaces.

Information for 02506

The second part of the course, image analysis with geometric priors, is covered in three weeks of the course. Counting from the start of semester, these are weeks 5, 6, and 7.

- Week 5: Markov random fields (MRF) modeling and optimization using graph cuts. In addition to lecture notes, the optional reading material is a book by Stan Li, which is freely available for download at DTU Findit. Exercises in MRF is on bone segmentation.
- Week 6: Mumford-Shah functional, Chan-Vese algorithm and snakes. In addition to lecture notes, the optional reading material is an article by Chan and Vese, and a book chapter on snakes. For exercises in deformable models you will track a simple object in a sequence of images.
- Week 7: Layered surface detection. In addition to lecture notes, the reading material is an article by Li et al. For exercise you will detect layers in bone data. Additionally, you can combine layered surface detection with deformable model and segment nerves from volumetric data.

5 Markov random fields

Markov random fields (MRF) is a probabilistic framework that can be used for various image analysis tasks, where contextual information needs to be considered. All MRF formulations involve labeling, i.e. assigning labels to some image entity, typically assigning labels to pixels. MRF are characterized by the Markov property, i.e. that the probability of a (pixel) label is only dependent on the local neighborhood (of a pixel).

In exercises on MRF, we will use MRF model for image segmentation. A segmentation can be solved by assigning a discrete label to each pixel in the image according to the pixel intensity. Often, we would like the segmentation to be smooth, meaning that the probability of a pixel label being different from label of neighboring is low. Provided an image, we aim at finding a label configuration that maximizes the *a posterior* (MAP) probability which is a combination of a likelihood (data) term and the term modelling a smoothness prior.

One characteristics of MRF is that the probability of the MRF configuration is an exponential of the negative configuration energy. Maximizing the posterior probability is therefore equivalent to minimizing the posterior energy of the configuration f given by

$$E(f) = U(f|d) = U(d|f) + U(f) \tag{5.1}$$

or, in terms of clique potentials

$$E(f) = \sum_{\{i\} \in \mathcal{C}_1} V_1(f_i) + \sum_{\{i,i'\} \in \mathcal{C}_2} V_2(f_i, f_{i'})$$

where $\mathcal{C}_1$ is the set of one-cliques, V_1 is a one-clique potential used for modeling the likelihood term, $\mathcal{C}_2$ is the set of two-cliques, and V_2 is a two-clique potential used for modeling the prior term.

As discussed in the book by Li[1], Chapter 1, Introduction, first paragraph, the main concerns of the MRF framework are how to define an objective function, i.e. clique potentials (modelling part), and how to find the optimal solution for a given objective function (optimization part).

[1] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009

5.1 *Example: Gender determination (optional, written for students wanting an easy introduction to MRF)*

We start with the extremely small 1D example with the aim of introducing MRF terminology, demonstrating the modelling possibilities provided by MRF, and the use of the data term and likelihood. A student comfortable with these MRF concepts may skip the example and proceed with the exercises.

Imagine entering a bar and observing 6 persons standing along the counter. You estimate persons heights (in cm) and record this data as

$$d = \begin{bmatrix} 179 & 174 & 182 & 162 & 175 & 165 \end{bmatrix}.$$

You want to estimate the persons gender, i.e. you want to assign either a label M or F to each person by combining a data term and your knowledge of the contextual information. You decide to pose the problem as a MRF with the neighbourhood given by the first neighbor (person to the left and person to the right).

We first consider the likelihood (data) term. You know that the average male height is 181 cm, and the average female height is 165 cm, and that height for each gender may be described as following a normal distribution where you assume the standard deviation for both genders being the same. For this reason you define the likelihood terms as one clique potentials

$$V_1(f_i) = (\mu(f_i) - d_i)^2$$

where d_i is the height of person i, f_i is a label assigned to person i, and μ is defined as $\mu(M) = 181$, $\mu(F) = 165$. The likelihood energy of a configuration $f = [f_1 \dots f_6]$ is the sum of all one-clique potentials

$$U(d|f) = \sum_{i=1}^{6} V_1(f_i).$$

To find the configuration which minimizes the likelihood energy you can consider the one-clique potentials for all i and both labels

$$\begin{array}{lcccccc} (\mu(M) - d_i)^2 & : & 4 & 49 & 1 & 361 & 36 & 256 \\ (\mu(F) - d_i)^2 & : & 196 & 81 & 289 & 9 & 100 & 0 \end{array}$$

Obviously, the minimal likelihood energy is obtained if we choose a label which minimizes the cost for each i, resulting in a labeling

$$f^D = \begin{bmatrix} M & M & M & F & M & F \end{bmatrix}, \tag{5.2}$$

and giving $U(d|f^D) = 99$. Another thing to notice is that additional cost for deviating from this labeling varies, depending on which label we change. For example, it costs additional 352 to label the forth person

as male, while it costs only additional 32 to label the second person as female.

Now you want to incorporate the contextual (prior) information about the gender of the people standing along the bar counter. Your experience is that people tend to group by gender, and that a configuration with a man standing next to a woman occurs less frequently. For this reason, you decide to incorporate a cost β which penalizes a less-frequent configuration. For prior energy you define 2-clique potentials as

$$V_2(f_i, f_{i+1}) = \begin{cases} 0 & \text{if } f_i = f_{i+1} \\ \beta & \text{otherwise} \end{cases}$$

The prior energy is the sum of all 2-clique potentials for all 2-cliques (all pairs of neighbors) in a configuration.

$$U(f) = \sum_{i=1}^{5} V_2(f_i, f_{i+1})$$

Obviously, this prior energy is minimal for a configuration with all labels being equal, while spacialy alternating labels yield maximal prior energy of 5β. The prior energy for the configuration f^D which minimizes the likelihood energy (5.2) is $U(f^D) = 3\beta$.

Last modelling choice involves setting a suitable parameter β. This choice depends on your confidence in the prior, compared to the data. You choose to use $\beta = 100$. According to (5.1), the posterior energy of configuration f^D is

$$U(f^D|d) = U(d|f^D) + U(f^D) = 99 + 300 = 399\,.$$

The question is, can we find another configuration which yields a smaller posterior energy? And finally, which configuration minimizes posterior energy?

For our small problem, we can simply try different configurations, two are worth considering

$$f^P = \begin{bmatrix} M & M & M & M & M & F \end{bmatrix},$$

$$f^O = \begin{bmatrix} M & M & M & F & F & F \end{bmatrix}.$$

Relatively easy we can confirm that configuration f^O with $U(f^O|d) = 163 + 100 = 263$ is an optimal configuration.

Note again that the prior knowledge encodes our assumptions of how the solution is supposed to look like, and it also influences the result such that what we find is what we expect to find. Our experience (prior knowledge) about people standing in a bar might have motivated another prior energy which encourages configurations where males

and females stand next to each other. For example a prior

$$V_2(f_i, f_{i+1}) = \begin{cases} \beta & \text{if } f_i = f_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

This prior would yield in another optimal configuration.

Note also the distinction between modeling (setting up the problem by defining a likelihood term and a prior term) and optimization (finding the configuration which minimizes the posterior energy).

5.2 *Exercise: MRF modelling*

In this exercise we define an objective function for segmenting a noisy image, similar to the problem in Li Section 3.2.2. Here we will compute the energy of different configurations to confirm that minimizing an objective function leads towards the desired solution. The model we use is very similar to the model used for gender determination. In this exercise we use synthetic data (i.e. we produce our input data by adding noise to a ground truth image) shown if Figure 5.1. This allows us to evaluate the quality of our objective function. In the text the input image is denoted D and ground truth segmentation S_{GT} where elements of S_{GT} are from the set $\{1, 2, 3\}$ corresponding to the darkest, medium gray, and brightest class.

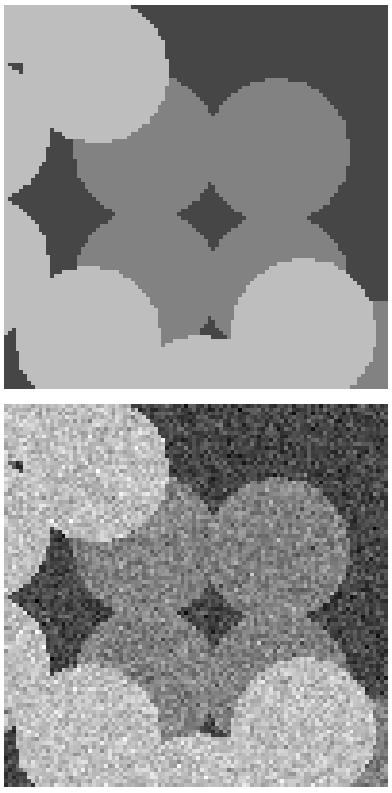

Figure 5.1: A ground truth (the desired segmentation should resemble ground truth) and a noisy image (input data).

Write a function that given D and a segmentation, for example S_{GT}, produces a histogram of the pixel intensities and histograms of the pixel intensities divided into segmentation classes. An example is shown in Figure 5.2. What does this histogram say about the chances of obtaining a reasonable segmentation of D using a method which considers only pixel intensities, for example thresholding?

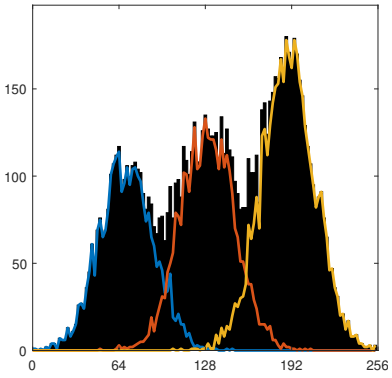

Figure 5.2: Histograms.

Now we pose image segmentation as a MRF. Sites are pixels, labels are from $\{1, 2, 3\}$, and we choose a first-order neighborhood (four closest pixels). As in the previous example, we define the one-clique potentials for the likelihood energy as the squared distance from the class mean

$$V_1(f_i) = \alpha \left(\mu(f_i) - d_i\right)^2$$

where d_i are intensities of the (noisy) image, f_i are pixel labelings given by the configuration, and μ is estimated from the histogram and set to $\mu(1) = 70$, $\mu(2) = 130$, $\mu(3) = 190$. In this example, we will weight the data term with the parameter α (unlike the previous example, where we weight the prior term). You may use $\alpha = 0.0005$. The likelihood energy is defined similar to before

$$U(d|f) = \sum_i V_1(f_i)\,,$$

where summation covers all image pixels. Similarly to the previous example, we define 2-clique potentials for discrete labels which penalizes neighbouring labels being different

$$V_2(f_i, f_{i'}) = \begin{cases} 0 & f_i = f_{i'} \\ 1 & \text{otherwise} \end{cases},$$

and prior energy

$$U(f) = \sum_{i \sim j} V_2(f_i, f_{i'})$$

where summation runs over all pairs of neighbouring pixels. The posterior energy is now given by (5.1).

We want to check that our optimal function leads to the desired result. That is, we want to make sure that posterior energy gets smaller when we approach the desired result. Therefore we want to compute the likelihood, prior and posterior for some reasonable segmentations (MRF configurations). For the purpose of this testing, produce at least two segmentations of the noisy image D. The first segmentation S_T is obtained by thresholding D at intensity levels 100 and 160 (valleys of the histogram). The second segmentation S_M is computed by median filtering S_T using an appropriate kernel. You are welcome to produce additional configurations, e.g. by applying a Gaussian filter to D prior to thresholding, or by using morphological operations. For all candidate configurations you should take a look at the intensity histograms of three classes, similarly as for the S_{GT} earlier.

Write a function which given an image D, a configuration S and the MRF parameters μ (intensities for segmentation classes) and α (weighting of the data term), returns computed likelihood, prior and posterior energy. Use your function for computing energies of different configurations, and also the ground truth S_{GT}. If we consider only the likelihood, which configuration is the most probable? If we consider only the prior energy, which configuration is the most probable? What if we consider the posterior energy? Would you expect that minimizing the optimal energy leads to a good segmentation? If not, try adjusting α and improve the posterior.

Tasks

1. Implement a function which produces histograms, as explained in the text.
2. Implement a function which computes segmentation energies, as explained in the text.
3. Produce a number of configurations. Apply your two functions to all configurations, and answer the questions from the text.

5.3 *Exercise: Iterative optimization for MRF (optional, for students wanting additional challenges)*

The interactions modelled by MRF prior make optimization (finding an optimal configuration) of the MRF very difficult. Standard MRF optimization methods may be very slow, but efficient graph cut algorithms can be used for a subset of problems. To gain a better understanding of MRF you may first implement an standard MRF optimization called iterated conditional modes (ICM), briefly sketched in Li Section 3.2.2. and elaborated in Section 9.3.1. Later, in the exercises following this one, you will be given graph cut implementation which you will use for optimization. You may also chose to first solve the problem using graph cuts, and then return to this exercise.

The general principle of ICM is the following. Every pixel contributes to the overall energy only locally, so for each pixel we can find a label that locally minimizes the energy (i.e. maximizes the conditional probability given all other labels). The iterative process continues until convergence. In our case, for a pixel i we need to compute

$$V(f_i|d_i, f_{\mathcal{N}_i}) = \alpha\,(\mu(f_i) - d_i)^2 + \#\{f_i \neq f_{i'}|i' \in \mathcal{N}_i\}$$

(see Li Equation (9.15) from Section 9.3.1) for three possible labels $f_i \in \{1, 2, 3\}$ and choose the label yielding the lowest value. The symbol # denotes the number of neighbors of i which have a label different from f_i.

To implement ICM, write a function which takes as input a segmentation S, the data term D (the noisy image), and MRF parameters μ and α. The function should output conditional local potential, i.e. values $V(f_i|d_i, f_{\mathcal{N}_i})$ for all pixels and all labels. For our purpose the output should have three layers, each layer as big as the image, such that the k-th layer gives a pixel-wise local energy for label k. You can can use your function to iteratively improve the configuration by labeling each pixel with the locally optimal label.

The convergence of ICM is guaranteed only for serial updating (updating labels one after another). To see why, we will first try parallel update (updating all labels at once), where we in each iteration at once overwrite all labels of the current configuration with locally optimal labels. Start for example with S_T and run for 10 or 20 iterations. What do you observe? Does the algorithm converge?

Instead of a fully serial update, we can in parallel update a set of labels where no two sites are neighbors. In our case, this can be obtained by dividing all pixels in two sets using a checkerboard pattern. Why can we update such sets in parallel, and why use a checkerboard pattern in our case? Modify the algorithm such that you in each iteration compute conditional local potentials (using the function you wrote) and update

half of the pixels according to checkerboard pattern, then compute local potentials again and update the other half of the pixels. Does the algorithm converge?

Compute the posterior energy for the configuration obtained using ICM, and compare with the energies for configurations given by ground truth and all the segmentation configurations. Did you come closer to the optimal configuration? Try also starting with a random initialization of the labels. Do you obtain the same result regardless of the initialization? Try reducing and increasing the smoothness by changing the weighting of likelihood and prior. For example, make α 10 times bigger or 10 times smaller. What do you observe?

Optionally, you can try implementing another popular and well-known optimization algorithm. The Gibbs sampling algorithm (Li Section 7.1.6) is a randomized sampling algorithm for finding the optimal configuration of the MRF. It is based on changing the labeling f_i with a probability which is proportional to the probability of the labelings f_i. The sketch of the Gibbs sampling algorithm is as follows. Initialized based on the maximum likelihood. Iterate for a number of times. In each iteration compute the local probability of each label every pixel. In our case this is easily obtained from the output of your function by taking an exponential of negative energy and normalizing probabilities to sum to 1. For each pixel, divide the interval $[0, 1]$ according to probabilities, then choose a random number from $[0, 1]$ and determine which subpart it belongs to – this indicates the label which should be assigned for the pixel. The Gibbs sampler mig can be further improved using simulated annealing (Li Section 10.1). An easy way of implementing simulated annealing is to multiply conditional local potentials with a increasing value, for example iteration number.

5.4 *Example: Graph cuts for MRF*

A binary (two label) MRF problem with submodular second order energy (loosely speaking an energy favoring smoothness) can be exactly solved by finding a minimum *s*-*t* cut of a graph constructed from the energy function [2,3,4]. A minimum *s*-*t* graph cut can be found e.g. using the Ford and Fulkerson algorithm, or an efficient freely available graph cut implementation by Boykov and Kolmogorov. A multiple-label discrete MRF problem can also utilize graph cuts via iteratively solving multiple two-label graph cuts, e.g. by using α expansion.

In the following exercises we are using graph cuts to optimize discrete MRF. MATLAB users may use the provided code, in particular the `GraphCutMex` function. This is a slightly modified version of the older Boykov implementation, the newest version can be found at `http://pub.ist.ac.at/~vnk/software.html`. Python users may use `PyMaxflow`

[2] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001

[3] V Kolmogorov and R Zabih. What energy functions can be minimized via graph cuts? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2): 147–159, 2004

[4] Y Boykov and V Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004

package documented at http://pmneila.github.io/PyMaxflow/index.html.

To begin with, we look back at the small example with gender labeling. Recall that the heights (in cm) of 6 persons are

$$d = \begin{bmatrix} 179 & 174 & 182 & 162 & 175 & 165 \end{bmatrix}$$

and we want to estimate the persons gender. For likelihood we use squared distance from the means $\mu(M) = 181$, $\mu(F) = 165$. For the prior we use $\beta = 100$ as a penalty for neighbouring labels being different.

We want to s-t graph corresponding to this problem. The solution for this is not unique. The focus is often on constructing a graph with fewest edges, an approach suggested in Li book Section 10.4.2. However, you might prefer drawing a more intuitive graph despite a higher number of edges. This approach is sketched in Figure 5.3. Terminal edges (linking to source and sink) are used for the likelihood energy terms, while internal edges model the prior energy terms. Confirm that a cost of an s-t cut in this graph equals to the posterior energy of the corresponding configuration.

To be able to compute the optimal configuration using the MATLAB `GraphCut` function, we need to create two matrices which contain edge weights to be passed to the function. The matrix containing terminal weights and the matrix containing weights between internal nodes for gender assignment example are shown in Figure 5.4. Python wrapper has slightly different manner of passing graph weights to the function, as explained in the package documentation.

Find the minimum s-t cut by calling `[Scut,flow] = GraphCutMex(N,Et,Ei)`, with first inputs being the number of internal nodes in the graph, followed by the two weight matrices. What is given in the outputs? Which configuration is optimal? Change $\beta = 10$ and solve again. Which configuration is optimal now? Try also $\beta = 1000$.

Tasks

1. Download and test the provided software for graph cuts. For further help, we have provided small scripts which show a way of achieving this.

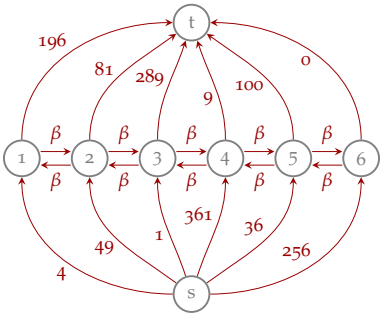

Figure 5.3: A sketch of a s-t graph for a gender labeling problem.

$$E_{\text{terminal}} = \begin{bmatrix} 1 & 4 & 196 \\ 2 & 49 & 81 \\ 3 & 1 & 289 \\ 4 & 361 & 9 \\ 5 & 36 & 100 \\ 6 & 256 & 0 \end{bmatrix}$$

$$E_{\text{internal}} = \begin{bmatrix} 1 & 2 & \beta & \beta \\ 2 & 3 & \beta & \beta \\ 3 & 4 & \beta & \beta \\ 4 & 5 & \beta & \beta \\ 5 & 6 & \beta & \beta \end{bmatrix}$$

Figure 5.4: Representing a s-t graph using two matrices, one containing weights of terminal edges and one matrix for internal edges.

5.5 *Exercise: Binary segmentation using MRF*

Take a look at the bone image `V12_10X_x502.png`. The image is a slice from a CT scan of a mouse tibia. You can visually distinguish air (very dark), bone (very bright) and cartilage (dark). The task in this exercise is to segment the image in two segments: air and bone. Cartilage should

be segmented together with air. The model we use is still the same as in the previous exercises, with the likelihood as the sum of squared distances, and the prior penalizing neighbouring labels being different. The bone image is of type `uint16`, and should be converted into double precision before any computation. You may also want to divide image intensities with $2^{16} - 1$, as this might simplify the weighting between the likelihood and the prior term.

Produce the histogram of the image to determine the mean intensities of the air and bone classes. Formulate the likelihood energy. Construct the matrices containing edge weights of the corresponding graph. Choose a parameter β and compute the optimal configuration using the `GraphCutMex` function. Adjust β to obtain a visually pleasing segmentation with reduced noise in air and bone. Produce a figure showing histogram of the entire image, and on top of that the intensity histograms of the air and bone classes, similar to how it was done in the modelling exercise.

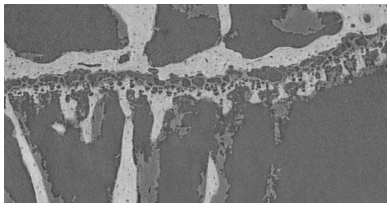

Figure 5.5: A bone image.

Tasks

1. Segment bone image of circles into two classes. Check how changing β affects the segmentation.

5.6 *Exercise: Multilabel segmentation usign MRF (optional)*

Multilabel segmentation is obtained using an iterative α expansion algorithm. A MATLAB function `multilabel_MRF` implements α expansion. Read the help text of the function for explanation on input and output variables. Python users may try the `maxflow.fastmin` which is a part of `maxflow` package.

We will first verify the quality of the solution provided by the α expansion algorithm by returning to the segmentation of the synthetic image. How does the energy of the graph cut solution compare to the energies of the configurations found previously? Try changing β to see how it affects the result.

Use the α expansion algorithm to segment the bone image into air, cartilage and bone class. The challenge here is to distinguish between air and cartilage. You should aim at producing a visually pleasing result with cartilage as solid as possible (without noisy pixels segmented as air) and air as clean as possible (without noisy pixels segmented as cartilage). A good result can be obtained by tweaking two parameters: the mean value for the cartilage class and the smoothness weight β. As means for air and bone you can use the values estimated from the histogram, and you may assume the same standard deviation for all three classes (so there is no need for additional weighting of

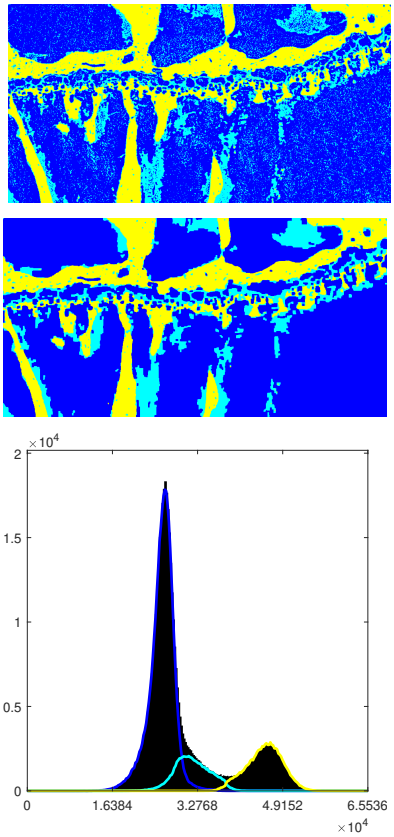

Figure 5.6: A segmentation of bone using maximum likelihood (top) and maximum posterior (middle). Histograms show intensity distributions for the three segmented classes.

the likelihood terms). When adjusting a mean value for cartilage, choose no smoothing ($\beta = 0$) and try to obtain a reasonable (but noisy) segmentation. Then increase β to remove the noise.

After you have tuned the parameters and obtained a nice segmentation, try segmenting the other bone image (`V8_10X_x502.png`). Can we use the same parameters? Why?

5.1 Information for 02506

Exam question related to this exercise

- Explain the use of MRF for image segmentation. Hints: Explain the concepts of likelihood and prior. Explain the difference between modelling and optimization. Explain how graph cuts are used to for MRF optimization.

6 Deformable models

Typically, image segmentation involves a combination of two terms: one dealing with the image data and the other describing a desirable segmentation. For example, we have used Markov random fields to impose smoothness on the segmentation. Using deformable models for image segmentation is another strategy which combines two contributions: the first originating from the image, and the second imposing smoothness.

The basic principle of deformable models is to perform image segmentation by evolving a curve in an image. The curve moves under the influence of *external forces*, which are computed from the image data, and *internal forces* which have to do with the curve itself. Deformable models are generally classified as either *parametric* (also called explicit) or *implicit* (in the context of image segmentation also called geometric), depending on the method used for representing the curve, see Figure 6.1. Despite this fundamental difference in curve representation, the underlying principles of both methods are the same [1].

In this exercise we use parametric curve representation, often called *a snake* [2], $C(s) = (x(s), y(s))$ where parameter $s \in [0, 1]$ is an arclength. In a discrete setting this reduces to a sequence of points, and parameter s becomes a discrete index $s = \{1, \ldots, N\}$ indicating ordering of the points. We consider an image where the task is to separate the foreground from the background, and we use subscripts F and B for the corresponding image entities, such as for the image domain Ω consisting of Ω_F and Ω_B. At the same time, a curve C divides the image into inside and outside region, and for those regions we use subscripts in and out.

Deformable models are guided by the segmentation energy E, which should be defined such that the desired segmentation has a minimal energy. A segmentation is obtained by iteratively moving the curve to minimize the energy, and the most challenging part of the approach is deriving energy-minimizing curve deformation forces $F = -\nabla E$. To allow deformation, the curve is made dynamic (time-dependable), and

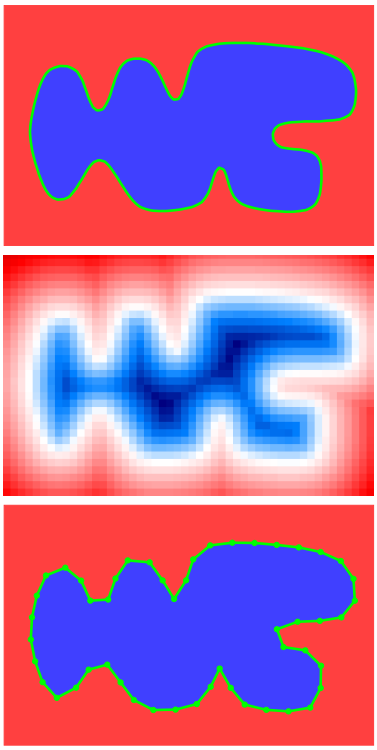

Figure 6.1: A curve (top) and its two discrete representations: implicit (middle) and parametric (bottom).

[1] Chenyang Xu, Anthony Yezzi Jr, and Jerry L Prince. On the relationship between parametric and geometric active contours. In *The Asilomar Conference on Signals, Systems,and Computers*, volume 1, pages 483–489. IEEE, 2000b

[2] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1988

its change in time, often denoted *evolution*, is given by

$$\frac{\partial C}{\partial t} = F(C)\,.$$

We use E_{ext} to denote external energy, which is a contribution to the segmentation energy determined by image data. We use E_{int} for internal energy, which has to do with the curve itself. Correspondingly, deformation forces on the curve are divided into external and internal F_{ext} and F_{int}.

This exercise is inspired by the Chan-Vese algorithm [3], a deformable model for image segmentation which minimizes a piecewise-constant Mumford-Shah functional. Chan-Vese uses a implicit (level-set) curve representation and a two-step optimization. We will use the solution of the Chan-Vese approach, but will combine it with a parametric curve representation. For this reason our model uses an external energy similar to Chan-Vese algorithm, and an internal energy similar to snakes. The detailed explanation on how external forces are derived from external energy can be found in the Chan-Vese article. How internal forces are derived is explained in Chapter 3 of the Handbook of Medical Imaging, Image Segmentation Using Deformable Models [4], subsection 3.2.1 and 3.2.4. Recall that you already implemented curve smoothing as one of the introductory exercises during the first week of the course.

[3] Tony F Chan and Luminita A Vese. Active contours without edges. *IEEE Transactions on image processing*, 10(2):266–277, 2001

[4] Chenyang Xu, Dzung L Pham, and Jerry L Prince. Image segmentation using deformable models. *Handbook of medical imaging*, 2:129–174, 2000a

6.1 *External energy, Chan-Vese*

An external energy closely related to the two-phase piecewise constant Mumford-Shah model is

$$E_{\text{ext}} = \int_{\Omega_{\text{in}}} (I - m_{\text{in}})^2 \, d\omega + \int_{\Omega_{\text{out}}} (I - m_{\text{out}})^2 \, d\omega$$

where I is an image intensity as a function of the pixel position, while m_{in} and m_{out} are mean intensities of the inside and the outside region. This energy seeks the best (in a squared-error sense) piecewise constant approximation of I. An evolution that will deform a curve toward an energy minimum is derived as

$$F_{\text{ext}} = (m_{\text{in}} - m_{\text{out}})\,(2I - m_{\text{in}} - m_{\text{out}})\,N\,. \tag{6.1}$$

where N denotes an outward unit normal.

In other words, curve deforms in the normal direction, and for every point on the curve we only need to evaluate the (signed) length of the displacement. We will denote the scalar components of the force as $f_{\text{ext}} = (m_{\text{in}} - m_{\text{out}})\,(2I - m_{\text{in}} - m_{\text{out}})$. Note that this can be written as $f_{\text{ext}} = (m_{\text{in}} - m_{\text{out}})\left(I - \frac{1}{2}(m_{\text{in}} + m_{\text{out}})\right)$, i.e. the signed length of displacement is proportional to the difference between the image intensities under the curve and the mean of m_{in} and m_{out}.

6.2 *Internal forces, snakes*

The internal energy is determined solely by the shape of the curve. In the classical snakes formulation internal forces discourage stretching and bending of the curve

$$E_{\text{int}} = \frac{1}{2} \int \alpha \left| \frac{\partial C}{\partial s} \right|^2 + \beta \left| \frac{\partial^2 C}{\partial s^2} \right|^2 ds \,,$$

with weights α and β controlling the elasticity (first-order derivative) and the rigidity (second-order derivative) term. Corresponding deformation forces are

$$F_{\text{int}} = \frac{\partial}{\partial s} \left(\alpha \frac{\partial C}{\partial s} \right) + \frac{\partial^2}{\partial s^2} \left(\beta \frac{\partial^2 C}{\partial s^2} \right) . \tag{6.2}$$

Those regulatory forces are the key to success of deformable models, as they provide robustness to noise.

Since our snake is discrete, the derivatives should be approximated by finite differences. The regularization now corresponds to filtering (smoothing) the curve with filters for the first and the second derivative, (i.e. a filter $\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$ and a filter $\begin{bmatrix} -1 & 4 & -6 & 4 & -1 \end{bmatrix}$. Those contributions, weighted by parameters α and β are now used to regularize the curve. In efficient implementation this is done by a matrix multiplication, and for better stability we use a backward Euler scheme. For slightly more detail, you can revise the introductory exercise on curve smoothing 1.1.3.

6.3 *Final model*

For a snake consisting of n points and represented using an $n \times 2$ matrix $\mathbf{C}$, a final discrete update step is, adapted from Handbook of Medical Imaging, Eq. (3.22),

$$\mathbf{C}^t = \mathbf{B}_{\text{int}} \left(\mathbf{C}^{t-1} + \tau \, \text{diag}(\mathbf{f}_{\text{ext}}) \, \mathbf{N}^{t-1} \right) . \tag{6.3}$$

In this expression τ is the time step for displacement, while $\mathbf{B}_{\text{int}}$ is the $n \times n$ matrix used for regularizing the curve and taking the role of the internal forces. Curve normals are represented as $n \times 2$ matrix $\mathbf{N}$, and pointwise displacement is obtained by multiplying $\mathbf{N}$ with a $n \times n$ diagonal matrix containing the displacement lengths (this is in principle a row-wise multiplication).

6.4 *Exercise: Segmentation and tracking*

We will use a deformable model to segment and track a simple organism in a sequence of images. You are provided with two image sequences:

crawling amoeba [5] and water bear [6]. The same code can be used for both sequences, with only a minor adjustment in a pre-processing step.

Tasks Steps for solving the problem are listed below, with the hints for MATLAB and python users.

1. Read in and inspect the movie data. In MATLAB you may use `VideoReader`. You may save the image sequence as a multi-dimensional array, or as a movie object using `im2frame` conversion. In python you may use function `get_reader` from `imageio` package.

2. Process movie frames. For our segmentation method to work, movie frames need to be transformed in grayscale images with a significant difference in intensities of the foreground and a background. For the movie showing the crawling amoeba (which is white on a dark background), it is enough to convert movie frames to grayscale. Transforming intensities to doubles between 0 and 1 is advisable, as it might prevent issues in subsequent processing. For the movie of the echinicsus, we want to utilize the fact that foreground is yellow while background is blue. A example of suitable transformation is $(2b - (r + g) + 2)/4$, with r, g, b being color channels (with values between 0 and 1).

3. Choose a starting frame and initialize a snake so that it roughly delineates the foreground object. You may define a circular snake with points $(x_0 + r\cos\alpha, y_0 + r\sin\alpha)$, where (x_0, y_0) is a circle center, r is a radius and angular parameter α takes n values from $[0, 2\pi\rangle$. See Figure 6.2 for example, but use approximately 100 points along the curve.

4. Compute mean intensities inside and outside the snake. In MATLAB you can use `poly2mask` function. In python use `polygon2mask` from package `skimage.draw` starting with version 0.16.

5. Compute the magnitude of the snake displacement given by Eq. (6.1). That is, for each snake point, compute the scalar value giving the (signed) length of the deformation in the normal direction. This depends on image data under the snake and estimated mean intensities, as shown in Figure 6.3. A simple approach evaluates the image intensities under the snake by rounding the coordinates of the snake points. A more advanced approach involves interpolating the image at the positions of snake points for example using bilinear interpolation, which is in MATLAB implemented in function `interp2`, and is in python available under the same name in `scipy.interpolate` package.

[5] The video of crawling amoeba is from Essential Cell Biology, 3rd Edition Alberts, Bray, Hopkin, Johnson, Lewis, Raff, Roberts, & Walter, https://www.dnatube.com/video/4163/Crawling-Amoeba

[6] The video of water bear is from Olympus microscopy resources, https://www.olympus-lifescience.com/ru/microscope-resource/moviegallery/pondscum/tardigrada/echiniscus

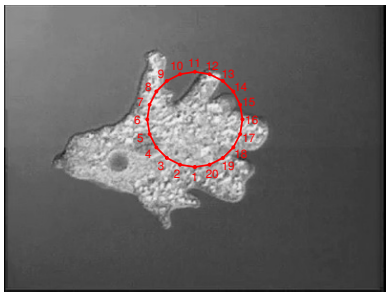

Figure 6.2: The first frame of a crawling amoeba and a circular a 20-point snake.

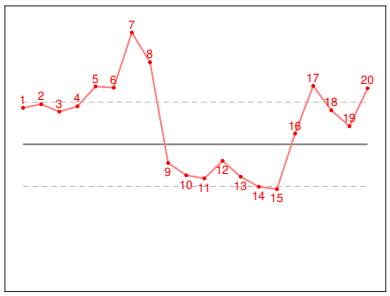

Figure 6.3: Red curve shows image intensities along the snake in Figure 6.2. Dashed gray lines indicate m_{in} and m_{out}, while gray line indicates the mean of m_{in} and m_{out}. Signed length of the curve displacement is given by the difference between the red curve and the gray line.

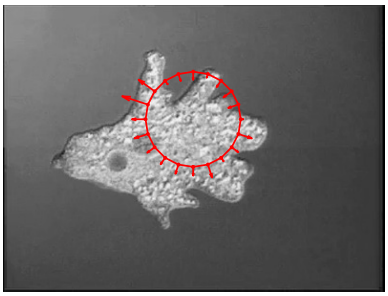

Figure 6.4: Force on the curve indicated by arrows. Displacement is in the normal direction and the length of the displacement is given by the values shown in Figure 6.3.

6. Write a function which takes snake points $\mathbf{C}$ as an input and returns snake normals $\mathbf{N}$. A normal to point c_i can be approximated by a unit vector orthogonal to $c_{i+1} - c_{i-1}$. (Alternatively, and slightly better, you may average the normals of two line segments meeting at c_i.) Displace the snake. Estimate a reasonable value for the size of the update step by visualizing the displacement. You should later fine-tune this value so that the segmentation runs sufficiently fast, but without introducing exaggerated oscillations. This step corresponds to computing the expression in the parentheses in the Eq. (6.3).

7. Write a function which given α, β and n constructs a regularization matrix $\mathbf{B}_{\text{int}}$. Your code from introductory exercise could be used. Apply regularization to a snake. Estimate a reasonable values for the regularization parameters α and β by visualizing the effect of regularization. You should later fine-tune these values to obtain a segmentation with the boundary which is both smooth and sufficiently detailed. This step corresponds to matrix multiplication on the right hand side of the Eq. (6.3).

8. The quality of the curve representation may deteriorate during evolution, especially if you use a large time step θ and/or weak regularization, i.e. small α and β. To allow faster evolution without curve deterioration, you may choose to apply a number of substeps (implemented as subfunctions) which ensure the quality of the snake:

   - Distribute points equidistantly along the snake. This can be obtained using 1D interpolation (function `interp1`).
   - Constrain snake to image domain.
   - Apply heuristics for removing crossings from the snake. For example, if you detect self-intersection, identify two curve segments separated by the intersection and reverse the ordering of the smallest segment.

   We provide a few functions for improving the quality of the snake, in MATLAB and in python.

9. Repeat steps 4–8 until a desirable segmentation is achieved. Note that the regularization matrix only depends on regularization parameters and a number of snake points. This is constant when the size of the snake and the regularization are fixed, which is a typical case. It is therefore sufficient to precompute $\mathbf{B}_{\text{int}}$ prior to looping. Figure 6.5 shows our 20-point snake during evolution.

10. Read in the next frame of the movie, and use the results of the previous frame as an initialization. Evolve the curve a few times by repeating steps 4–8.

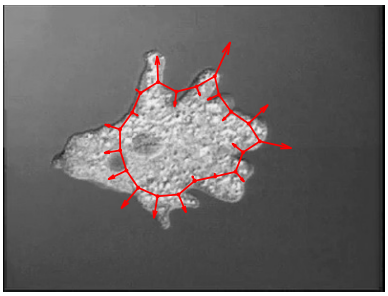

Figure 6.5: The curve and the forces after 20 iterations.

11. Process additional frames of the image sequence.

6.1 Information for 02506

Exam questions related to this exercise

- Explain the use of deformable models for image segmentation. Hints: Explain the piecewise-constant Mumford-Shah functional and how it is minimized using Chan-Vese algorithm? Explain the difference between the external and the internal forces. Explain curve representations used for deformable models, and their advantages/ disadvantages.