ANDERS BJORHOLM DAHL

VEDRANA ANDERSEN DAHL

NOTE FOR IMAGE ANALYSIS

April 25, 2018

Contents

Introduction

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 with speciality in image analysis.

Image analysis is a rapidly growing field of research with a wealth of methods constantly being developed and published. It is not the intention to give a complete overview of the field with this note. Instead, we focus on general principles for image analysis with the aim of giving students the skill-set needed for exploring new methods. General principals 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 to an efficient functioning program.

Image analysis methods published in scientific articles can be challenging to implement in 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 that needs to be written in order to implement the method. 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 pensum, 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 compulsory exercises. Towards the end of a note we provide a number of examples

for the final exercise.

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 give you the skill-set to implement or modify 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 Image representations

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 in the form of a quantaty. There are important 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 grey 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 utilised in the analysis.

1.1 *Exercise 1*

This exercise is aimed at refreshing or introducing concepts from basic image analysis curriculum and other related subjects, and contains some topics that will be useful at a later stage in the course. It is not expected that you will be able to carry out all exercises, so you can choose the ones you find most interesting.

Based on the outcome of this exercise, we will assess the general competence levels of the students, so that we can plan course based on this. Therefore, it is important that you hand in what you have finished by the end of the exercise.

Hand in You should hand in one page pdf telling which parts of the exercise you managed to finish, and showing one illustration from each of the finished exercise. Text should be limited to a short figure caption, title, labels, legends etc. Please also hand in a MATLAB script or a Jupyter notebook with code for one of the exercises.

1.1.1 *Measuring image smoothness*

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 smoothness measurement commonly used is the total variation defined for an image I as

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

where $x \sim x'$ indicates two neighbouring pixel locations. Implement a function which computes the total variation of a 2D grayscale image and test it on an 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.

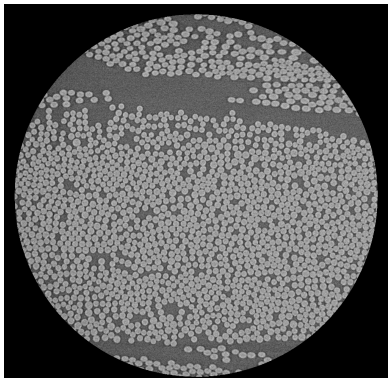

Figure 1.1: Slice of a CT image of glass fibres viewed orthogonal to the fibre direction.

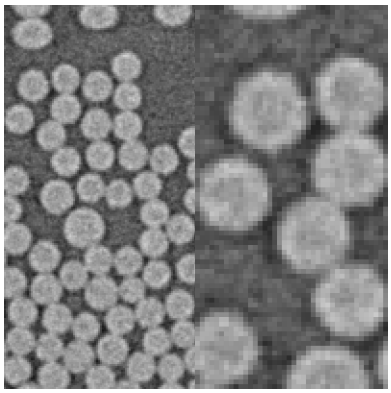

Figure 1.2: Two zoomed in images from image shown in Figure 1.1.

1.1.2 *Computing boundary length*

In case of segmentation, we are often interested in quantifying the length of the segmentation boundary. If the segmentation is represented by an image $S : \Omega \to \{1, 2, \dots n\}$, where n is the number of segments, we may define the length of the segmentation boundary as

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

where $x \sim x'$ again indicates two neighbouring pixel locations 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 the occurrences of two neighbouring pixels having different labels.

Implement a function which computes the length of a segmentation boundary and test it on provided segmentation images of a fuel cell, where one is shown in Figure 1.3. 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.

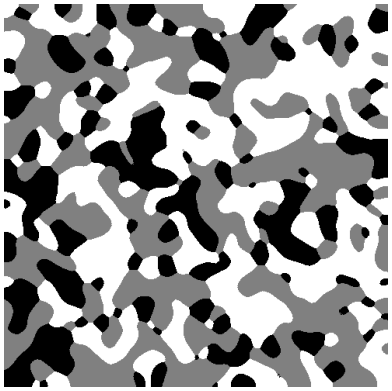

Figure 1.3: Image of a segmented fuel cell with three phases. Black represents air, grey is cathode, and white is anode.

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 2-times-N matrix $\mathbf{X}$ contains x and y coordinates of N points which define a closed curve, a so-called snake [1]. 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 neighbours, possibly iteratively. Point displacement can be seen as a result of filtering the snake with kernel $\lambda \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$, where λ is a parameter controling the magnitude of the displacement. For efficiency, we want to implement the snake-smoothing step as

$$\mathbf{X}^{\text{new}} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{X} \tag{1.1}$$

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 elswhere. Confirm that $\lambda = 0.5$ displaces every snake point exactly to the average of its

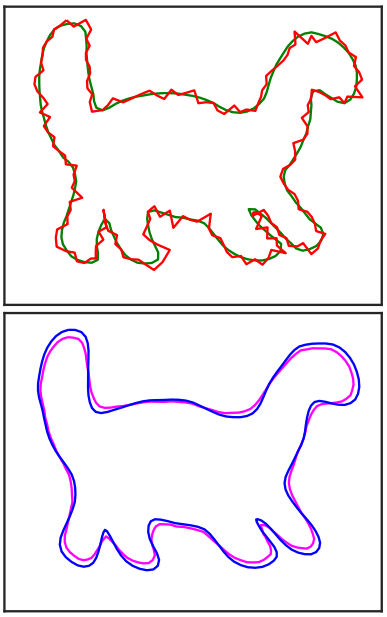

Figure 1.4: Top image shows the dinosaur curve in green, while red shows the curve with added noise. Bottom image shows two smoothing results with different α and β weights.

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

neighbours. 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 1.1 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

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

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.

Implement the two improvements to the snake smoothing: the extended kernel and the implicit approach. Note that your final implementation instead of $\lambda \mathbf{L}$ uses a matrix that combines the two contributions. Note also that this matrix is a (sparse) circulant matrix. Test your improved smoothing. Snake smoothing will be useful when we will be working with deformable models later in the course.

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

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`, that you can find on Campusnet.

1.1.4 *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 dataset, 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.

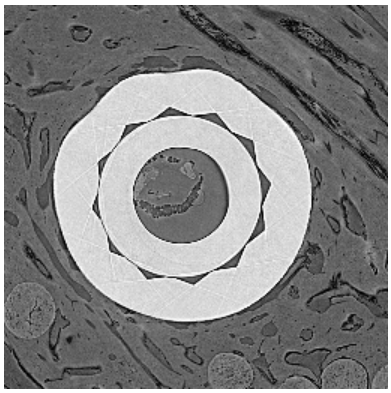

Figure 1.5: Image of a dental implant that should be unwrapped.

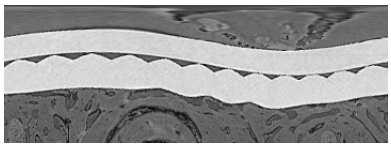

Figure 1.6: Unwrapped image of a dental implant.

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

1.1.5 *Working with volumetric image*

To give you a taste of working with 3D images, we have prepared a small dataset containing slices from an X-ray CT scan. By convention in X-ray imaging, dense structures (having a hight 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`.

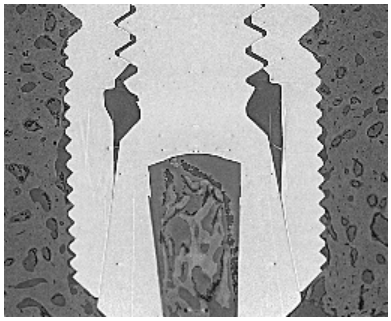

Figure 1.7: A longitudinal slice (an xz-plane) of the volumetric image of a dental implant.

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.

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

1.1.6 *PCA of multi-spectral 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 $I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^{18}$ where each channel corresponds to a wavelength, and channels cover the range from 410 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.

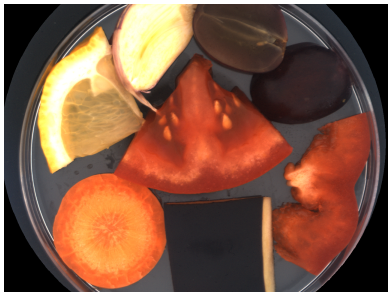

Figure 1.9: False colour image obtained from an 18 band VideometerLab image.

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

$$\mathbf{C}[i,j] = \frac{1}{N-1} \sum_{n=1}^{N} (\mathbf{X}[n,i] - \mu[i])(\mathbf{X}[n,j] - \mu[j]), \tag{1.2}$$

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

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

$$\mathbf{C} = \frac{1}{N-1}\overline{\mathbf{X}}^T\overline{\mathbf{X}}, \tag{1.3}$$

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 components can be computed as

$$\mathbf{Q} = \overline{\mathbf{X}}\mathbf{V}. \tag{1.4}$$

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

Tasks 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. What is the datatype of the images?

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.

1.1.7 *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 analyse 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 visualise bacterial growth by counting the cells.

The image quality is however not very good due to the low resolution and compression artefacts, 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.

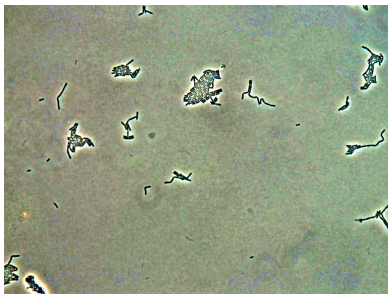

Figure 1.10: Example of microscopic image of listeria bacteria in a petri dish.

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

Tasks A suggested approach is to 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?

2 Feature-based image analysis

Analyzing images using feature-based representations is central to many applications. Here we will work with scale-space for detecting image features independently of scale. Based on ideas from scale-space theory, a large number of texture-based image features have been suggested, where scale-invariant feature transform (SIFT) was one of the pioneering ones, and we will work with that in the second part of the exercise for feature-based analysis in week three of the course.

2.1 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 hight measurements at regularly placed positions. In this landscape, an edge is a line where hight 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 for example can identify 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

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

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.

The idea of scale-space is to represent 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 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.

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

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 where for scale $t = 0$ we have $L(x, y; 0) = I(x, y)$.

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 Laplacian $\nabla^2 L = L_{xx} + L_{yy}$ 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

2.2 *Exercise 2 – part I*

This exercise is aimed at feature based image analysis, and will cover tree topics: scale-space analysis[4], scale-invariant feature transform (SIFT)[5] for image matching, and texture analysis for image segmentation. This exercise will run over three weeks, and the tasks carried out one in week will be used the next week. So, it is important that you carry out the exercise in the order, that it is presented.

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

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

Hand in After three weeks you should hand in a report. The report should have a technical character, and briefly illustrate the main results obtained in the exercise. Text should be limited to short explanations, and can e.g. be headlines and figure captions. Please also hand in a MATLAB script or a Jupyter notebook with code for one of the exercises in separate `.m` or `.py` files. This code should be able to reproduce your results in the report.

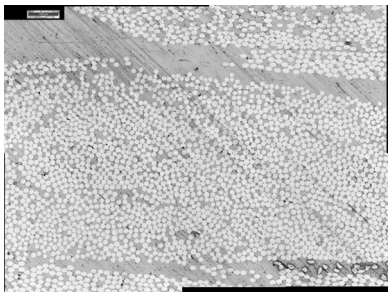

Figure 2.1: Example of fibre image acquired using an optical microscope.

2.2.1 *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.

2.2.2 *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

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

Tasks

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`.

2.2.3 *Detecting blobs on one scale*

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

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

Tasks

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.

2.2.4 *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).

Tasks

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`.

2.2.5 *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.

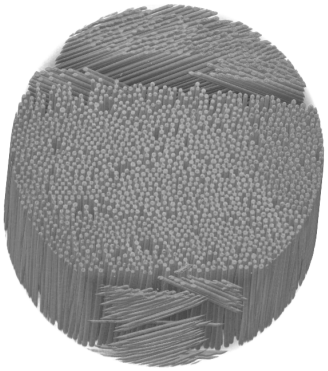

Figure 2.2: Visualization the 3D fibers scanned with the high resolution X-ray CT-scanner.

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

Tasks

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.

2.2.6 *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.

Tasks

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.

2.3 *Exercise 2 – part II*

Will be included in this note.

2.4 *Exercise 2 – part III*

You have now extracted blob-features in the images using scale-space blob detection and then used SIFT features to find correspondence between images. This allows us to match the blob-features between the images. In this exercise you should combine the results of the first two parts of the exercise and obtain a matching of the blobs, that allows a comparison of how well the blob-detection measures the size of the fibers.

The homography $\mathbf{H}$ obtained from the matching SIFT features gives correspondence between points in two images $\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i$. This allows us to transform the center coordinates of the blobs detected in one image to the coordinates in the other image. To compute this in a robust manner, you tried a symmetric matching, where the requirement is that a matched point in one image to the other, should also match from the other to the one. You also tried to use a criterion for the uniqueness of the match by using the ratio between the best and the second best match. Combining the two criteria gives a good match, and you should use that in this exercise.

Modeling the fibers using scale-space blob detection is done using the center coordinate and the diameter. The homography $\mathbf{H}$ gives the relation between the fiber's center coordinate in the two images, but it does not model the scale between the images. $\mathbf{H}$ models an affine transformation between the images so the scale is not directly obtainable from H, since it typically will be vary over the image. We will ignore this variation and compute one scale factor s that relates the diameters of fibers in one image to the diameters in the other image.

We will compute the scale parameter using principal component analysis (PCA). Recall that PCA can be obtained through the eigen-decomposition of the covariance matrix $\mathbf{C}$ given by $\mathbf{C}\mathbf{v}_i = \lambda\mathbf{v}_i$. The largest eigenvalue λ_1 is the direction of largest variance in the data. If we compute this in each of the two images for the points matched using SIFT features we obtain the variance of these point sets as $\lambda_{a,1}$ and $\lambda_{b,1}$ respectively, where a and b refers to the two images. The scale can be obtained from this as $s = \sqrt{\lambda_{b,1}/\lambda_{a,1}}$. The square root is needed, since the scale factor is proportional to the standard deviation and not the variance.

Tasks The following steps takes you through computing the scale factor s.

1. Obtain the corresponding coordinate sets $\mathbf{X}_a$ and $\mathbf{X}_b$ by matching SIFT features between images.
2. Compute the eigenvalues $\lambda_{a,1}$ and $\lambda_{b,1}$ for $\mathbf{X}_a$ and $\mathbf{X}_b$ respectively.
3. Compute the scale factor $s = \sqrt{\lambda_{b,1}/\lambda_{a,1}}$.

Now you have all the parts necessary to compare fibers detected in two images. We have measured the size of fibers in each image, we can compute the correspondence between points in the two images, and we have the scale relation between the images. This allows us to directly compare the position and diameter for each fiber and perform quantitative statistics. We will however not do an extensive statistical analysis now, but just make a visualization of fibers computed in two images but displayed in one.

Tasks You should now apply the transformation of the detected fibers in one image to display it in the other image.

1. Detect fibers in two images with their coordinates $\mathbf{P}_a$ and $\mathbf{P}_b$ and their diameters $\mathbf{d}_a$ and $\mathbf{d}_b$.
2. Compute the point transformation $\mathbf{P}'_a = \mathbf{H}\mathbf{P}_a$.
3. Compute the scale change in diameters $\mathbf{d}'_a = s\mathbf{d}_a$.
4. Plot $\mathbf{P}_b$ and $\mathbf{P}'_a$ in image b with diameters $\mathbf{d}_b$ and $\mathbf{d}'_a$ respectively.

If you obtain this visualization for one pair of images it is fine, but if you have time, you can try for more pairs and investigate which parameters gives you the best results.

3 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 deformable models useful for handling a distinctive geometry.

3.1 Markov random fields

Markov random fields (MRF) is a probabilistic framework that can be used for modeling while taking contextual information into consideration. MRF are very general and have many applications in image analysis. MRF are characterized by the Markov property, i.e. that the probability of a label is only dependent on the local neighborhood.

In Exercise 2 (part I and part III) will use the 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

$$E(f) = U(f|d) = U(d|f) + U(f) \tag{3.1}$$

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

3.1.1 *Example: Gender determination*

We start with the extremely small example with the aim of introducing terminology, demonstrating the modelling possibilities provided by MRF, and demonstrate 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. assign either a label M or F to each person by combining a data term and your knowledge of 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 each 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 and μ is defined as $\mu(M) = 181$, $\mu(F) = 165$. The likelihood energy 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 for labeling

$$f^D = \begin{bmatrix} M & M & M & F & M & F \end{bmatrix}, \tag{3.2}$$

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 you posses about the gender of the people standing along the bar counter. Your experience is that females usually stand next to other females, men stand next to other man, and while 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 in a configuration.

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

Obviusly, this prior energy is minimal for a configuration with all labels bein equal, while spacialy alterating labels yield maximal prior energy of 5β. The prior energy for the configuration which f^D minimizes the likelihood energy (3.2) 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 (3.1), 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 reducing the cost for the posterior term. Worth considering are two configurations

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

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

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 the data, 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).

3.1.2 *Exercise: 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 3.1. 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.

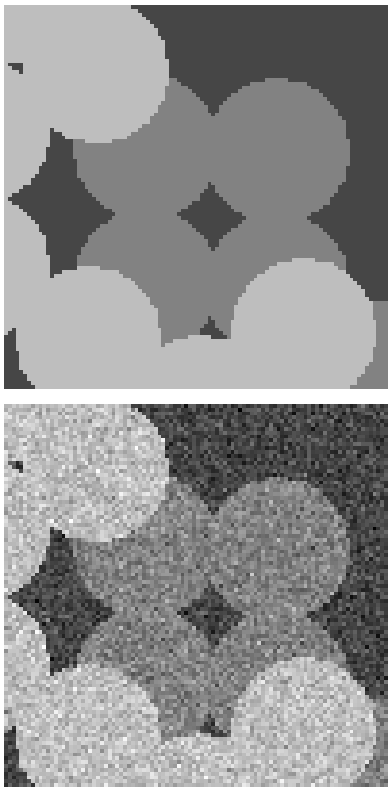

Figure 3.1: A ground truth (the desired segmentation should resemble ground truth) and a noisy image (input data).

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 3.2. 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?

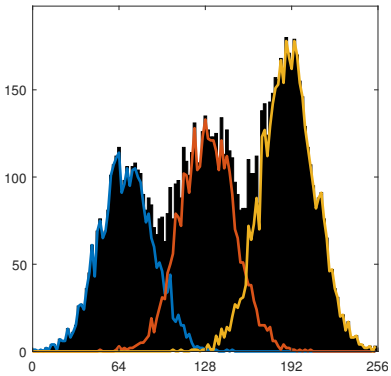

Figure 3.2: Histograms.

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 $\mu(1) = 70$, $\mu(2) = 130$, $\mu(3) = 190$. The parameter α will be used to weight the data term and may be set to for example $\alpha = 0.0005$. 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 (3.1).

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 and α, returns computed likelihood, prior and posterior energy. Use your function for computing energies of different configurations, 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 β 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.

3.1.3 *Exercise: Iterative optimization*

The interaction modelled by MRF prior makes optimization (finding an optimal configuration) of the MRF very hard. 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 we will 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. In the following exercise you will be given graph cut implementation which you will use for optimization. You may chose to first solve the problem using graph cuts, and then return to this exercise if time allows.

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

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

(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.

3.1.4 *Example: Graph cuts*

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

[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

`html`.

To begin with, we look again 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 3.3. 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 3.4. 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.

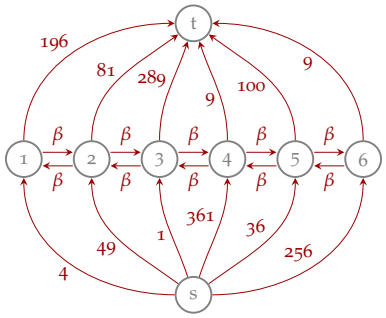

Figure 3.3: A sketch of a s-t graph for a gender labeling problem.

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

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

Figure 3.4: Representing a s-t graph using two matrices, one containing weights of terminal edges and one matrix for internal edges.

3.1.5 *Exercise: Binary segmentation*

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.

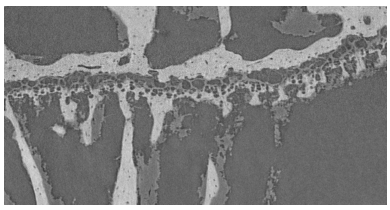

Figure 3.5: A bone image.

Tasks

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

3.1.6 *Exercise: Multilabel segmentation*

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 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.

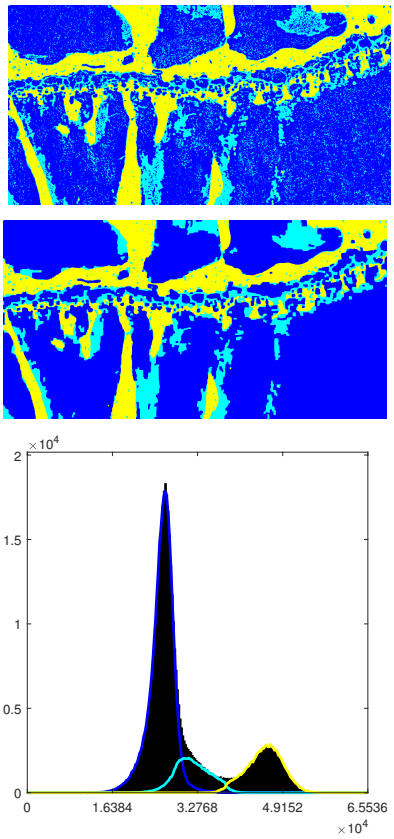

Figure 3.6: A segmentation of bone using maximum likelihood (top) and maximum posterior (middle). Histograms show intensity distributions for the three segmented classes.

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?

3.2 *Deformable models*

Typically, image segmentation involves modeling of both the image data and the desirable segmentation. For example, we have used Markov random fields to impose smoothness to the segmentation. 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* or *implicit* (in the context of image segmentation also called geometric), depending on the method used for representing the curve. Despite this fundamental difference in curve representation, the underlying principles of both methods are the same [5].

In this exercise we use parametric curve representation, often called *a snake* [6], $C(s) = (x(s), y(s))$ where parameter $s \in [0, 1]$ is 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 moving the curve to minimize the energy, and the essence of the approach is in deriving energy-minimizing curve deformation forces $F = -\nabla E$. To allow deformation, the curve is made dynamic (time-dependable), and 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}.

[5] 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

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

This exercise is inspired by the Chan-Vese algorithm [7], 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 [8], 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.

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

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

3.2.1 *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

$$F_{\text{ext}} = (m_{\text{in}} - m_{\text{out}}) \, (2I - m_{\text{in}} - m_{\text{out}}) \, N \, . \tag{3.3}$$

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}.

3.2.2 *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

$$F_{\text{int}} = \frac{\partial}{\partial s}\left(\alpha \frac{\partial C}{\partial s}\right) + \frac{\partial^2}{\partial s^2}\left(\beta \frac{\partial^2 C}{\partial s^2}\right) . \tag{3.4}$$

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.

3.2.3 *Final model*

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

$$\mathbf{C}^t = \mathbf{B}_{\text{int}}\left(\mathbf{C}^{t-1} + \tau \operatorname{diag}(\mathbf{f}_{\text{ext}})\,\mathbf{N}^{t-1}\right) . \tag{3.5}$$

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).

3.2.4 *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 [9] and water bear [10]. The same code can be used for both sequences, with only a minor adjustment in pre-processing step.

Tasks Steps for solving the problem are listed below. The exercise was previously (in 2017) intended for MATLAB users, and this is still reflected in hints about MATLAB implementation. Those hints were not removed, in hope that they will show useful both for MATLAB and Python users.

[9] 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

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

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.

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 circle center, r is radius and parameter α takes n values from $[0, 2\pi\rangle$. See Figure 3.7 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.

5. Compute the magnitude of the snake displacement given by Eq. (3.3). 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 3.8. 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`.

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 you can 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 exaggerated oscillations. This step corresponds to computing the expression in the parentheses in the Eq. (3.5).

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.

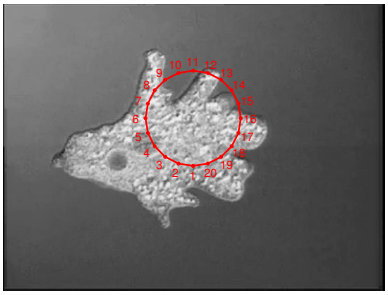

Figure 3.7: The first frame of a crawling amoeba and a circular a 20-point snake.

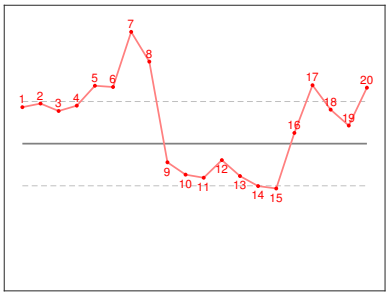

Figure 3.8: Red curve shows image intensities along the snake in Figure 3.7. Dashed gray lines indicate m_{in} and m_{out}, while gray line indicates the mean of m_{in} and m_{out}. Signed length of the curve displacement is given by the difference between the red curve and the gray line.

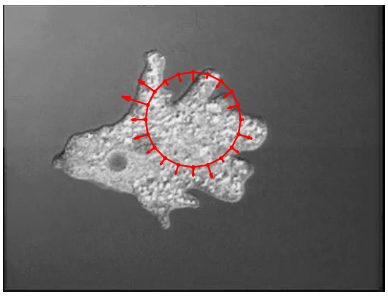

Figure 3.9: Force on the curve indicated by arrows. Displacement is in the normal direction and the length of the displacement is given by the values shown in Figure 3.8.

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. (3.5).

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 can 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, in MATLAB implemented in function `interp1`.
   - Constrain snake to image domain.
   - Apply heuristics for removing crossings from the snake. We provide a MATLAB function which detects curve self-intersections, identifies two curve segments separated by intersection, and reverses ordering of the smallest segment.

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 if the size of the snake and the regularization if fixed, which is a typical case. It is therefore sufficient to precompute $\mathbf{B}_{\text{int}}$ prior to looping. Figure 3.10 shows our 20-point snake during evolution.

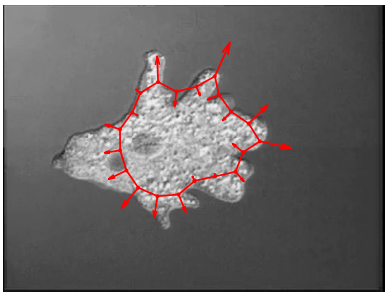

Figure 3.10: The curve and the forces after 20 iterations.

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.

11. Process additional frames of the image sequence.

3.3 *Assignment on geometric priors*

Assignment on geometric priors should demonstrate the use of two different approaches: Markov random fields and defromable models. You will be working on the small part of the volumetric data containing scans of the human posterior interosseous nerve. For better understanding of the goals of image analysis, we provide background information on the large study involving a complete data set.

3.3.1 *X-ray tomography of human peripheral nerves*

Understanding nerve disorders caused by trauma and disease requires a knowledge of the structure of peripheral nerves and their subcomponents. Conventional light and electron microscopical techniques only allow a two-dimensional visualization of tissues such as peripheral nerves. Recent advance in synchrotron imaging technique provides detailed three-dimensional images of tissue, allowing extraction of morphological information. For a larger study [11], biopsies of the posterior interosseous nerve at wrist levels were taken from otherwise healthy subjects and from subjects with type 1 and 2 diabetes. Biopsies were stained in osmium (a heavy metal used for staining lipids) which provides contrast to the image, and embedded in Epon (epoxy resin) for stability. The samples were then imaged using X-ray phase contrast zoom tomography at the European Synchrotron Radiation Facility (ESRF, Grenoble, France) with an isotropic voxel size of 130nm. In the obtained volumetric data, the nerve fibers are aligned with the z direction, and the stained myelin sheaths around axons (see Figure 3.11 for a schematic drawing of a nerve) appear circular in the x-y slices through the volume, as shown in Figure 3.12.

Complete data-set contains more than 10 samples, each resulting in a volume of a size $2048 \times 2048 \times 2048$ voxels. For the exercise, we extracted a small region from one volume, as indicated in Figure 3.12. Furthermore, extracted volume has been downsized by a factor 2, which yields a volume of size $350 \times 350 \times 1024$. The extracted volume is saved as a stacked tiff image `nerves_part.tiff`.

[11] Dahlin L B, Rix K R, Dahl V A, Dahl A B, Jensen J N, Cloetens P, Pacureanu A, Mohseni S, Thomsen N O B, and Bech M. X-ray phase contrast zoom tomography to visualize human diabetic peripheral nerves. *Nature Methods (in submission)*

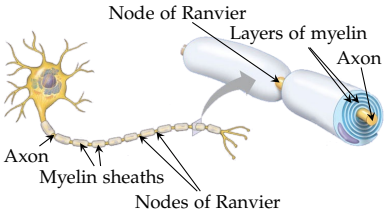

Figure 3.11: A schematic drawing of a nerve showing an axon, myelin sheaths and nodes of Ranvier.

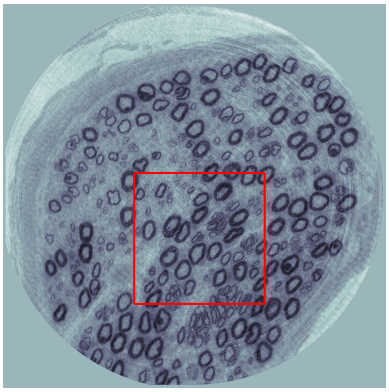

Figure 3.12: One slice from the volume showing peripheral nerves, and a region which was extracted for the exercise.

3.3.2 *Segmentation of myelinated nerves*

A good contrast between stained myelin sheaths and the background makes it possible to clearly distinguish peripheral nerves in the volume. For this reason, a reasonable segmentation strategy would utilize dark appearance of the myelin. Segmentation may be improved by incorporating a prior knowledge about the directionality of the nerves.

Furthermore, circular appearance of myelin sheaths allows a segmentation of a single nerve by aligning a closed curve with the periphery of the myelin. For this, the circle can be manually initialized around the nerve, and automatically moved to the boundary of the myelin. For segmenting a whole nerve, the curves are automatically propagated trough the volume, such that the surface moves only slightly between the slices, and is in every slice attracted to the boundary of the myelin layer.

Try segmenting nerves Markov random fields and deformable models. In Figure 3.13, 3.14, 3.15 and 3.16 we show results of image analysis performed for the original study. However, those results are obtained

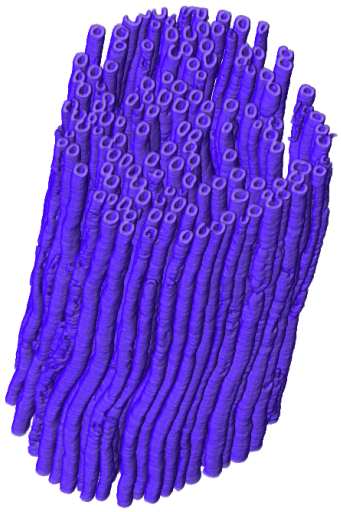

Figure 3.13: 3D visualization of segmented of myelinated nerves.

on a full-resolution volumes, and using more advanced image analysis techniques than covered in the lessons. Your results might therefore be of poorer quality.

For this open assignment, we provide some tips, but you are free to investigate other approaches. The tips listed below will be extended and elaborated during the lesson on Wednesday.

- For MRF segmentation you might consider further downsizing the data or using only a subset of slices.
- When using MRF segmentation you can process the volume slice-by-slice. This corresponds to a situation where MRF-modelled smoothness prior has a parameter β for two neighbouring pixels in x and y direction, while the change of labels for neighbours in z direction is not penalized. However, note that nerves are elongated in the direction orthogonal to image slices. For this reason, it would be more appropriate to set MRF-modelled smoothness especially high along the z direction, and this calls for a full 3D implementation of the MRF segmentation.
- When using deformable models note that assumption of Chan-Vese about a object of different intensity than the background does not apply for nerves. This is because a nerve consists of a dark myelin and a bright axon. Instead of allowing for the automatic estimation of the parameters m_{in} and m_{out} by averaging, it might be better to fix those parameters using the values estimated from the images. Alternatively, values m_{in} and m_{out} may be estimated from a thin band inside and outside the curve.
- The robustness of the deformable models might be improved by moving the curve towards the point where the change of intensity in the normal direction is high. This can be implemented by unwrapping the image following the curve normals. The gradient in normal direction can than be computed for the unwrapped image, and curve moved to the point where gradient is high.
- A node of Ranvier (see Figure 3.11 for schematic drawing of a nerve) can be seen on a few of the nerves in the volume to be analysed, as shown in Figure 3.16. Nodes are of a high interest for understanding nerve disorders. However, those might be challenging to capture due to the lack of myelin. Your method is not expected to capture the nodes of Ranvier, and if you succeed in segmenting those, remember to document it in your report.
- Visualizing results in 3D usually provides a useful information on the segmentation results. However, there is not a required for your report.

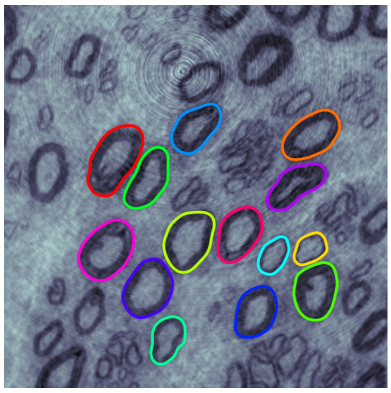

Figure 3.14: Axons segmented using deformable curves visualized on a single slice.

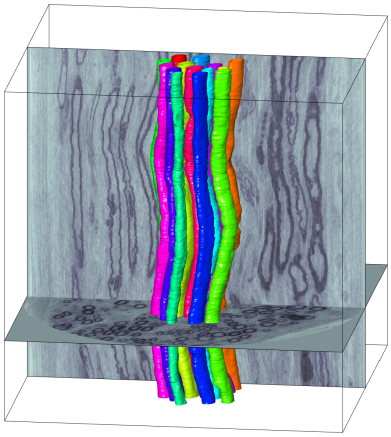

Figure 3.15: 3D visualization of axons segmented using deformable curves.

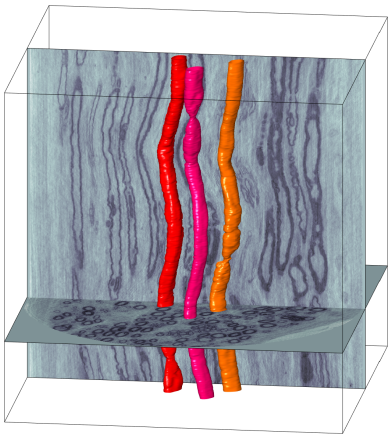

Figure 3.16: Three of the axons with visible node of Ranvier.

Tasks

1. Perform a binary segmentation of the data using MRF and a segmentation of myelinated nerves using deformable models.
2. Make a report containing a brief (a paragraph or two) description of your approaches and a visualization of your final results. Write also a paragraph discussing your result and the possible improvements of your approach.
3. Collect your code in a zip file and submit together with the report.

4 Neural networks

Neural networks are very useful for a range of image analysis tasks including segmentation, detection, etc. Neural networks are often easy to adapt to a specific problem and they allow approximating an unknown function f^* that based on some input $\mathbf{x}$ can predict the output $\mathbf{y}$ even without *a priori* knowing the relation between $\mathbf{x}$ and $\mathbf{y}$. This is done by learning a set of parameters θ from a training set, i.e. of corresponding input values $\mathbf{X}$ and predictions $\mathbf{Y}$. In image analysis problems the input will typically be an image or a part of an image, and the output is a scalar vector or an image.

A range of high-performance libraries for neural networks exists, that are very well suited for solving a number of problems also in image analysis. The aim here is however to understand the basic elements of neural networks and get experience with their functionality. This will be done by implementing a feed forward neural network, a Multilayer Perceptron (MLP). The first task is to separate simple point sets, and this is chosen both to ensure that the implementation is correct, and to get some experience with various parameters. Later this will be applied to image classification and image segmentation.

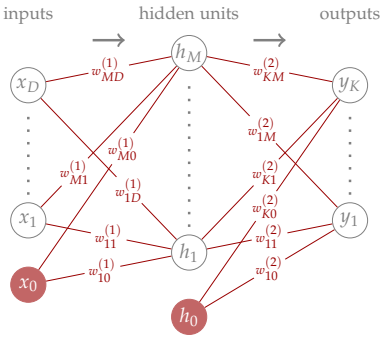

Figure 4.1: Example of a neural network with an input layer, one hidden layer, and an output layer. This is termed a one layer network, since it has one hidden layer. Typically, there will be many hidden layers.

4.1 Feed forward neural network

A neural network is modeled as a directed graph as shown in Figure 4.1. The input layer is shown on the left, hidden layers are in the middle, and the output layer is to the right. This exercise is based on the description in the Deep Learning book[1] but also Chapter 5 in the book Pattern Recognition and Machine Learning[2] gives a good introduction to neural networks.

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016

[2] CM Bishop. Pattern recognition and machine learning (information science and statistics), chapter 3, pages 138–147, 2006

4.1.1 Forward model

The fundamentals for the deep learning method are explained in the Deep Learning book[3], but here we will give a brief introduction to a simple feed forward network. You will later implement a more general version of a feed forward network, but first we will focus on the basic

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016

elements. The network shown in Figure 4.1 contains an input layer of three nodes (neurons), where two are the input variable (x_1, x_2) (independent observations) and $x_0 = 1$ is the bias node. The hidden layer contains four nodes including three nodes connected to the input layer (h_1, h_2, h_3) and the bias node $h_0 = 1$, and the output layer contains the two predicted values (y_1, y_2). The weights connecting the nodes are termed $w_{ij}^{(l)}$ connecting node j from layer $l-1$ to node i in layer l.

The values of the nodes in the hidden layers are computed by first computing a weighted linear combination z_i of the node values and the edge weights followed by an activation function, and we we use the max function $a(z_i) = \max(z_i, 0)$, which in deep learning is called the rectified linear units function to obtain h_i. We have

$$z_i = \sum_{d=0}^{D} w_{id}^{(1)} x_d \,, \tag{4.1}$$

$$h_i = a(z_i) = \max\{0, z_i\} \,, \tag{4.2}$$

$$\hat{y}_j = \sum_{m=0}^{M} w_{jm}^{(2)} h_m \,. \tag{4.3}$$

We use the softmax function to get the values of y

$$y_j = \frac{e^{\hat{y}_j}}{\sum_{k=1}^{K} e^{\hat{y}_k}} \,. \tag{4.4}$$

We use the cross entropy loss function

$$L = -\sum_{k=1}^{K} t_k \log y_k \,, \tag{4.5}$$

where t_k is the target value of the predictionwhere

$$t_k = \begin{cases} 1 & \text{if class label is } k \\ 0 & \text{otherwise} \end{cases} \,. \tag{4.6}$$

4.2 *Backpropagation*

We now want to adjust the weights of the network to minimize the loss over a training set of inputs and the associated target values. This is done trough the backpropagation algorithm which uses an iterative optimization method called stochastic gradient descent. Just as in gradient descent the loss is minimized by taking steps proportional to the negative gradient. However, we do not consider loss function for all inputs and targets at once. Instead, we consider one input (or a smaller sample of inputs) in a random order. The derivation below computes update for one iteration of the stochastic gradient descent.

We will evaluate derivatives for a certain (fixed) input in order to determine how change in each $w_{ij}^{(l)}$ affects L, and then we will use an update

$$w_{ij}^{(l)\text{new}} = w_{ij}^{(l)} - \eta \frac{\partial L}{\partial w_{ij}^{(l)}},$$

where η is a user-chosen learning rate.

Change in each $w_{ij}^{(l)}$ contributes to the change in L only trough $z_i^{(l)}$ and using the chain rule the derivative may be separated into two elements

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \overbrace{\frac{\partial L}{\partial z_i^{(l)}}}^{\text{difficult}} \overbrace{\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}}^{\text{easy}}.$$

Since $z_i^{(l)}$ is a linear function of $w_{ij}^{(l)}$ the second (easy) derivative evaluates to $h_j^{(l-1)}$ (or, in the case of the first layer, input values x_j). This is valid regardless of the non-linear activation. The first (more difficult) derivative needs to be evaluated for each $z_i^{(l)}$ and we denote those values by $\delta_i^{(l)}$, such that we have

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} h_j^{(l-1)}. \tag{4.7}$$

Notice the simplicity: the update for weight $w_{ij}^{(l)}$ is a product of two values, the first value depends only on the *to*-node and the second value depends only on the *from*-node.

We still need to evaluate $\delta_i^{(l)}$, i.e. establish how a change of $z_i^{(l)}$ affects L. This depends only on what happens in the layers further down the pipeline, and on the choice of the non-linear activation used on $z_i^{(l)}$. We distinguish between the last layer (where we use the softmax function) and the internal layers.

For the last layer we express L as a function of $z_k^{(l^*)}$

$$\begin{aligned} L &= -\sum_k t_k \ln \frac{\exp z_k^{(l^*)}}{\sum_j \exp z_j^{(l^*)}} = \text{using the properties of ln and the distributive rule} \\ &= -\sum_k t_k z_k^{(l^*)} + \overbrace{\sum_k t_k}^{=1} \overbrace{\ln \sum_j \exp z_j^{(l^*)}}^{\text{equal for all } k}. \end{aligned}$$

The derivative of L with respect to $z_i^{(l^*)}$ is therefore

$$\delta_i^{(l^*)} = -t_i + \frac{1}{\sum_j \exp z_k^{(l^*)}} \exp z_i^{(l^*)} = y_i - t_i. \tag{4.8}$$

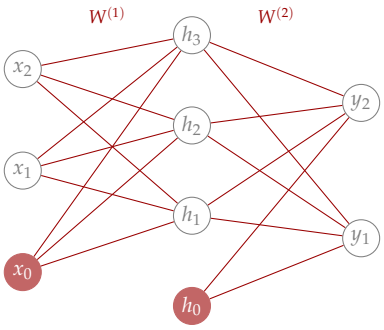

Figure 4.2: Simple three layer neural network.

Now consider the internal layers. The change in $z_i^{(l)}$ may change all z_k^{l+1}, and any of these changes may affect L. The chain rule gives

$$\frac{\partial L}{\partial z_i^{(l)}} = \sum_k \underbrace{\frac{\partial L}{\partial z_k^{(l+1)}}}_{\text{we have}} \underbrace{\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}}}_{\text{we need}}.$$

The first set of derivatives are $\delta_k^{(l+1)}$ for the layer further down the pipeline. We already evaluated those for the last layer in (4.8), and this is why we compute the update backwards trough the network. The only remaining is to determine how the change of $z_i^{(l)}$ affects $z_k^{(l+1)}$. From definition (4.1) we see that $z_k^{(l+1)}$ is a linear function of $a(z_i^{(l)})$ which gives

$$\frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} = w_{ki}^{(l+1)} a'\left(z_i^{(l)}\right),$$

where a' denotes the derivative of the activation function, which for ReLU function takes a value zero for arguments smaller than zero, and one otherwise. This is easy to determine by assessing whether $h_i^{(l)}$ is zero or larger. The final expression for internal layers is

$$\delta_i^{(l)} = a'\left(z_i^{(l)}\right) \sum_k w_{ki}^{(l+1)} \delta_k^{(l+1)}. \tag{4.9}$$

4.3 *Implementation*

You are now ready to implement your neural network.

Data preprocessing The recommended preprocessing is to center the data to have mean of zero. For features of different scale, it is advised to normalize the scale along each feature.

Initialization Weights should be initialized with small numbers. Initializing with zeros is not a good idea, as it introduces no asymmetry between neurons. The current recommendation in the case of neural networks with ReLU neurons is to draw the weight from the Gaussian distribution with standard deviation of $\sqrt{\frac{2}{n}}$, where n is the number of inputs to the neuron.

Batch optimization A variant of stochastic gradient descend splits the training data into smaller subsets (minibatches) and updates weight according to the average of the gradients for the minibatch. One cycle through the entire training dataset is called a training epoch.

Matrix multiplications The layered structure makes it efficient to evaluate and train neural networks using matrix vector operations. The

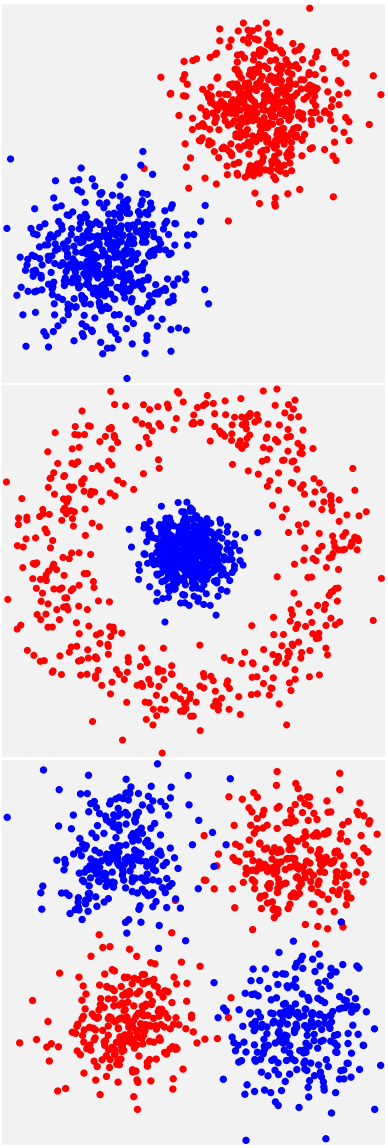

Figure 4.3: Scatter plots of point sets to test neural network

forward propagation trough layers is evaluated using a product

$$\mathbf{h}^{(l)} = a\left(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)}\right), \tag{4.10}$$

where vectors $\mathbf{h}^{(l)}$ and $\mathbf{h}^{(l+1)}$ contain values $h_i^{(l)}$ and $h_i^{(l+1)}$ (without a bias), $\mathbf{W}^{(l)}$ is a matrix containing weights and a is activation. Vectorized expression (4.10) is also valid when passing multiple inputs trough the network: a $2 \times m$ input results with a $2 \times m$ output. Likewise, during training, values $\delta_i^{(l)}$ can be represented as vectors, such that computation (4.9) utilizes a matrix multiplication, while (4.9) becomes an outer product of two vectors. When working with batches (4.9) is a matrix-matrix multiplication.

4.3.1 *Setting up the problem*

You should implement the network shown in Figure 4.2 using the description given above. This network contains a single hidden layer and takes a two dimensional input and classifies the input to a two dimensional output. Before you get there you should construct some test data for running your method. You can create test point-sets similar to the ones shown in Figure 4.3. In this experiment you will use the same points for training and testing. This first experiment is to ensure that you build the neural network in a correct way, and later you will test the performance of your method on an independent validation dataset.

4.3.2 *Simple three layer network*

You should start with a three layer network with a single hidden layer with five neurons, i.e. four neurons that are connected to the input layer and a bias variable. It is a good idea to start making a hand drawing of the network you should implement will the nodes, edges and notation for the parts of the network.

Tasks

1. Start by implementing the forward propagation step with random weights. Make sure that it works as expected. It is important to check that the obtained results make sense. This is easy when working with 2D data and can be done by sampling points on a regular grid, which can be displayed as an image where each pixel takes the label number.

2. Make a script that computes the classification results on a regular grid. You can look at the results from your forward propagation with random numbers.

3. Implement the backpropagation algorithm. Note that both the forward and backward propagations can be implemented efficiently using matrix operations. Display the loss as you iterate to ensure that your algorithm is converging.

4. Test your implementation on the generated data. What should the learning rate be? How is performance affected by noise? Does it converge to the same result each time you run it?

4.3.3 *Variable number of layers and hidden units*

In the exercise above you have obtained a neural network with a fixed architecture, i.e. the number of neurons and hidden layer. The architecture is however central in modeling with neural networks, and therefore you should make the number of layers and the number of neurons in each of the hidden layers a part of your input choice. You will also be needing this flexibility in the later exercises for classifying the MNIST handwritten digits.

Tasks

1. Implement a neural network keeping the single hidden layer and with variable number of hidden units.

2. Implement a neural network with variable number of hidden layers but with variable number of hidden units. Now you can play around with your model and from here it is easy to modify it to include other elements like other activation functions.

3. Again test your implementation on the given data.

The expected output is shown in Figure 4.4. The coloring of the pixels in the background is obtained by running all the coordinates of the pixels in an through the neural network, and coloring the result in dark or bright gray depending on the classification result.

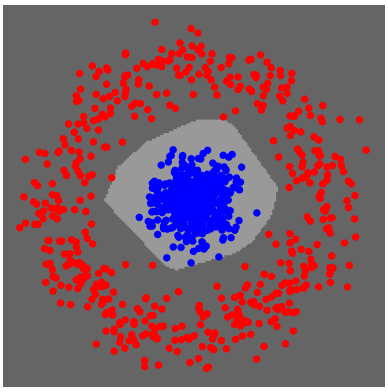

Figure 4.4: Result of training on the input data shown in colored points, and the test result is shown in the pixel colors in the background.

4.4 *MNIST classification*

In this exercise you should build a neural network for classifying the MNIST image data. Performance will be measured in number of misclassified images, and the goal is to obtain the least result.

The MNIST dataset contains images of handwritten digits of 28×28 pixels as shown in Figure 4.5. Ground truth class labels are given together with the images. The ground truth is 10 dimensional vectors with 1 in the dimension representing the class containing the digit and zeros elsewhere. MNIST contains 60000 images for training your network. If you use all 60000 images for training your network, you

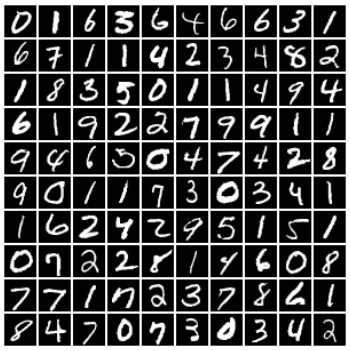

Figure 4.5: Example of the MNIST images.

might overfit your model, and to choose when to stop training you can split the data into a training and validation set. You can for example use 50000 images for training the network and reserve 10000 for validating it. By classifying the validation images, you can measure if you have overfitted your model, which is seen by a drop in classification performance of the validation data. In addition there are 10000 images for testing, but these should only be used for evaluating the performance of your networks after they have been trained.

The rules for the competition are:

1. Implement your own neural network for classifying the MNIST images.
2. Train the neural network on a part of the training data (e.g. 50000 images) and validate it on another part (e.g. the remaining 10000 images).
3. When you are satisfied with the obtained result – upload the trained network together with Matlab or Python code for running it.
4. Hand in a description of your network and a guide on how to run your code
5. Be a fair player and do not use the MNIST test data for training your network (can be found on the internet, but do not use it!).
6. Do an effort in making the code efficient.

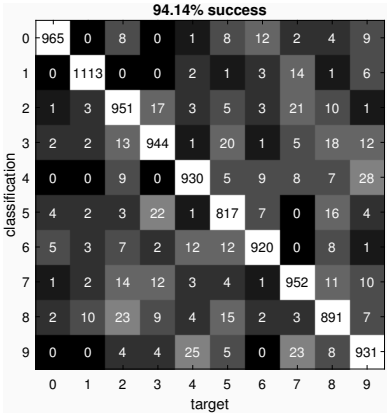

Figure 4.6: Table showing a classification performance example of a classification of the MNIST handwritten dataset.

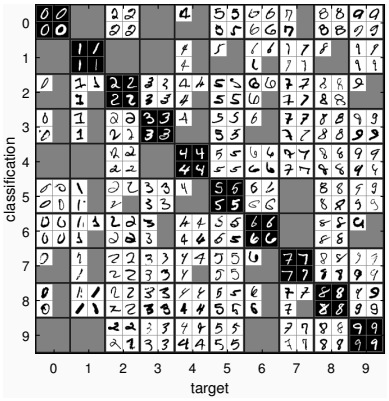

Figure 4.7: Examples of classified handwritten digits and misclassified digits.

4.4.1 *Modifications of your network for MNIST*

The classification will be using a fully connected feed forward neural network, similar to the one you implemented last exercise. But in contrast to last exercise, where data was points in two dimensions, you now have images. We treat these as vectors, so even though MNIST images are only 28×28 pixels the vector representation is 784 dimensions. Therefore, the network should take in 784 dimensional vectors and return a 10 dimensional vector for classifying the digits 0 to 9. You can use the book on deep learning by Goodfellow et al. [4] to get ideas for this exercise.

Obtaining a strong classifier requires many iterations of the backpropagation algorithm using 50000 training images. Therefore, it is a good idea to utilize vectorized code in your implementation. This can be done by computing the gradients for subsets of the training data using minibatches. You can have a minibatch as a matrix and compute the forward propagation and gradients using matrix operations. By averaging the gradients obtained from the minibatch, the backpropagation can be carried out in the same way as you would do when

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016

training with one sample at a time. Due to the averaging, the obtained gradients are less affected by noise and it is typically possible to have higher learning rates.

A part of optimizing the neural network is by changing its architecture. Therefore, it is recommended that you implement your network such that you can change the number of layers and the number of neurons in each layer.

1. Implement a fully connected neural network for MNIST classification. The data should be normalized and centered, i.e. vectors of unit length using the 2-norm and with zero mean. Besides vectorizing the code it is also worth considering the data type. Single is faster than double, so you should consider if you want faster computations, at the cost of lower precision. You can experimentally evaluate if the high precision is necessary.

2. Train the network and plot the training and validation error for each iteration (epoch). The dataset could be split into 50000 images for training and 10000 for validation.

4.4.2 *Optimizing the neural network*

A large number of techniques for optimizing the performance of the neural network has been proposed. Neural networks are typically initialized with random numbers, but the performance depends on the choice of the initialization strategy. You can therefore optimize the network by experimenting with different initialization strategies.

Optimization with stochastic gradient decent can be slow, but the updating the gradients using momentum can accelerate the learning rate. Momentum is obtained by computing the gradients as a weighted combination of the previous gradient and the new gradient. Hereby, the gradient is computed as a moving average with exponential decay. Another way of ensuring convergence of the gradient decent is by adapting the learning rate. Here you can adapt the learning rate to the individual gradient estimates.

1. Implement one or more optimization strategies and document how it affects the obtained result.

4.4.3 *Regularization methods*

It is important that the neural network generalizes well such that it can classify new unseen data. Since neural networks often have many parameters it is easy to overfit the model, especially on small datasets where a very low training error can be obtained, but the validation error will be high. One way to overcome small datasets is through dataset

augmentation, where fake data is fabricated by small modifications of the input data. This can be done by small permutations or by adding small amounts of noise. Hereby a much larger dataset can be obtained, which can help the training.

Instead of adding noise to augment the training data, small amounts of noise may be added to the hidden units in the network. You can add random noise in each minibatch iteration. Noise can also be added to the output targets for obtaining better performance.

Dropout is another method for regularizing the neural network. Here a random selection of neurons are set to zero during each minibatch iteration leaving out these neurons in that iteration. Setting the neurons to zero resembles having a number of different neural networks and is inspired by ensemble methods.

1. Try experimenting with regularization methods. You can also get inspired by architectures that other people have had success with.

4.5 Convolutional neural networks

Convolutional neural networks (CNNs) have many aspect in common with multilayer perceptrons (MLPs – fully connected feed forward neural networks) such as being a directed acyclic graph with weighted edges and non-linear activations. Instead of having unique weights for all connections, the CNNs share their weights, which means that only one edge weight is learned for many edges. This results in a significant reduction in number of model parameters, and therefore makes it possible to have much larger input data compared to MLP networks. Furthermore, the shared weights can efficiently be implemented as convolutions, which for images are 2D convolutions. The parameters that must to be learned are the weights of the convolutional kernels, which similar to MLPs, are learned using backpropagation.

Working with images in 2D makes it possible to apply additional operations to the hidden layers. This includes for example a pooling step typically combined with a down-scaling step. Max-pooling is an example of a widely used pooling method, where pixels are replaced by the local maximum in a local neighborhood. Max-pooling ensures that only the important features are kept, and makes the analysis robust to small translations. There is a number of other operations that can be applied, and an overview is given in chapter 9 in Goodfellow et al.[5].

Despite that CNNs have many aspects in common with MLPs, they are typically more complicated and therefore not as simple to implement. A large number of software frameworks are available, and training neural networks for many engineering applications will involve GPU processing. This is however implemented in a user friendly

[5] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016

way in many of the software packages and highly sophisticated neural networks can be developed using high-level programming using e.g. Python that makes these frameworks easy to use.

4.5.1 *Exercise*

In this exercise we will work with existing software frameworks and we will use a pre-trained network, namely the VGG19[6]. We will use a network that is trained for the 1000 categories in the ImageNet challenge[7], but instead of classifying photographs we will use it for classifying microscope images of histopathology tissue samples.

The data originates from the CAMELYON17 challenge[8] for breast cancer diagnosis. In clinical practice, a pathologist will spend long time inspecting images of tissue that has led to much research in automating this inspection task, which is also the goal of the CAMELYON17 challenge. Here the aim is to detect and classify breast cancer metastasis from so-called whole-slide images of stained histological lymph node sections. In this exercise you will do the core part of this analysis, namely classifying samples to healthy or diseased tissue.

In Figure 4.8 some examples of the histopathology images are shown. These are 224 by 224 pixels color images cropped from whole-slide images, and you should note that the visual difference between normal and diseased tissue is small. The basis for the exercise is to classify these images using pre-trained convolutional neural networks.

Data is provided in file called `histo_images.mat` where the images are stored in a 4D array called `histo_images` and each image is labeled given in a 1D array called `labels` with 1 for normal tissue and 2 for diseased tissue. There are 992 images in all where 629 are from normal tissue and 363 are from diseased tissue.

Task

1. Load in the data, visualize and get familiar with the format

4.5.2 *Setting up the CNN framework*

If you use MATLAB you should use MatConvNet[9] and if you use Python you should use Keras with Tensorflow[10]. You should use the VGG19 model pre-trained for the ImageNet ILSVRC classification task.

Tasks

1. Download the pre-trained VGG19 model for the ImageNet challenge.

2. Get MatConvNet or Keras to work on your computer.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014

[7] http://www.image-net.org/

[8] https://camelyon17.grand-challenge.org/

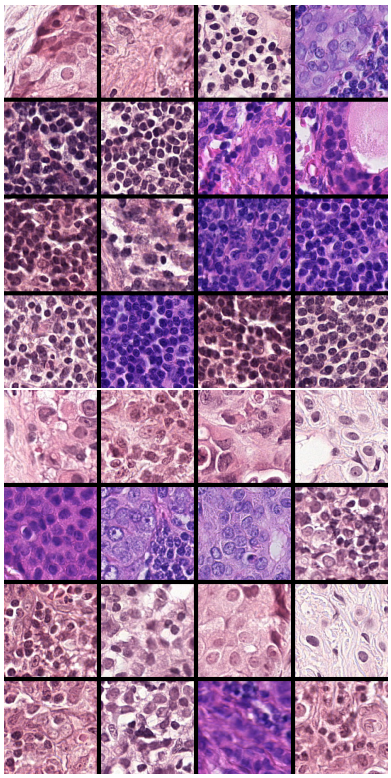

Figure 4.8: Example images of histological tissue samples from healthy tissue in the top and breast cancer metastasis in the bottom. These are from the CAMELYON17 challenge.

[9] http://www.vlfeat.org/matconvnet/

[10] https://keras.io/

3. Make a forward pass on your computer and investigate the output of the network including access to the hidden layers.

4.5.3 Classifying images

You should do a simple classification of the images using k-nearest neighbors, based on features extracted from the downloaded CNN. A part of this is to investigate the features extracted in the hidden layers in the network. You should not expect improved performance by looking at the pooling or ReLU layers, since there is no information introduced in these layers. The features that you should use for classification might not be the last layer, since it is trained for a very specific task, but still the chosen layer should be at a deep layer in the network. Therefore, it is suggested that you look a the fully connected layers, and determine which is suitable for the classification task.

Tasks

1. Extract and store features for the histopathology images provided for the exercise. Be careful to preprocess the images, such that they are normalized and fed into the model.
2. Do a k-nearest neighbor classification based on the extracted features by measuring the Euclidean distance between image features. You should take one out and measure the distance to 991 other images and classify to the most frequent occurring label among k neighbors. Investigate how k should be chosen.
3. Make an overview of classification performance with different choices of k and layer from the CNN. Which combination performs best?

5 Free exercise

In this exercise you are free to choose the image analysis case that you find most interesting. It is expected that you will find a relevant image analysis problem, implement a suitable analysis method, set up test data that verifies the correctness of your implementation, and report and present your results on a poster for the last day at the course.

You are welcome to come up with your own idea for a project, but you are also welcome to choose from the suggested projects in the following.

5.1 Texture Analysis

Image texture is important for a range of image analysis problems like object classification and quality control. Also a number of image processing problems like denoising and inpainting are based on principles of texture analysis. Here you will solve a texture classification based on the Basic Image Features described in Crosier and Griffin[1] and an inpainting problem described in the paper by Efros and Leung[2].

[1] M. Crosier and L.D. Griffin. Using basic image features for texture classification. *International Journal of Computer Vision*, 88 (3):447–460, 2010. ISSN 0920-5691. DOI: 10.1007/s11263-009-0315-0

[2] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038 vol.2, 1999

5.1.1 Data

The data for the exercise is found in a directory called `texture_data` that contains images with a wide variety of textures for BIF characterization used in the first part and corrupted images to be used during the texture synthesis part of the exercise. You are also welcome to find your own data set for the exercise.

5.1.2 Basic Image Features

This section will provide a small summary of how BIFs are estimated. The purpose of BIF is to go from an image of high dimensionality to a lower dimensional vector representation of the image texture. This representation uses simple geometric image features and will enable differentiation between different textures. The following recipe is used to estimate BIF:

1. Convolve with six Gaussian filters to get scale-normalized filter responses $(s, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$.
2. Calculate the flat, slope, blob (2×), line (2×) and saddle feature responses using the formulas from [3] and $(s, s_x, s_y, s_{xx}, s_{yy}, s_{xy})$ from Step 1.
3. Classify each pixel as flat=0, slope=1, dark blob=2, white blob=3, dark line=4, white line=5, saddle=6, by finding the label index of the maximum feature responses of Step 2. Denote the resulting label image as $\mathcal{L}$.

[3] M. Crosier and L.D. Griffin. Using basic image features for texture classification. *International Journal of Computer Vision*, 88 (3):447–460, 2010. ISSN 0920-5691. DOI: 10.1007/s11263-009-0315-0

For a fixed scale, a seven bin histogram can now be formed by counting how many times a pixel is classified as one of the classes. This histogram is the BIF of an image for scale σ. Please note that if we disregard the flat pixels, a six bin histogram is used per scale, however we are generally interested in a histogram that also models scale.

Four scales: How to get a histogram? When extending this BIF representation to multiple scales the process of forming a histogram becomes increasingly complicated. If we choose four scales $\sigma = (1, 2, 4, 8)$ we need to run steps 1 - 3 four times. This will lead to a four channel label image $\mathcal{L}(\mathbf{x}; i)$, where $\mathbf{x}$ is the position in the image and $i = 0, ..., 3$ for the four scales. To get the texture characterization, we have to convert these four channels of the label image into a histogram. Each bin of this histogram counts how often a specific label configuration occurs across the four scales. If we ignore flat BIF regions, the pixels can be classified as one of the labels $\mathcal{L}(\mathbf{x}; i) \in \{1, \ldots, 6\}$. First we want to translate this to a number between 0 and 1295 (i.e. $6^4 = 1296$ unique combinations). This is done in all pixels resulting in an image $\mathcal{B}(\mathbf{x})$ by converting the four BIF classes to one number

$$\mathcal{B}(\mathbf{x}) = \sum_{i=0}^{3} (\mathcal{L}(\mathbf{x}; i) - 1) 6^i \, . \tag{5.1}$$

This means that the label combination $[1, 1, 1, 1]$ is a pixel classified as slope on all the scales, and similarly $[1, 3, 4, 6]$ is a pixel that is classified as a slope at the first scale, a blob on the second scale, a line on the third scale, and a saddle on the fourth scale.

5.1.3 *Texture classification*

In this exercise you will experiment with *Basic Image Features* (BIF) for texture description.

The BIF features are computed from the following equations:

Classify according to the largest of the features:

Flat: ϵs

Slope: $2\sqrt{s_x^2 + s_y^2}$
Blob: $\pm\lambda$
Line: $2^{-\frac{1}{2}}(\gamma \pm \lambda)$
Saddle: γ

where

$\lambda = s_{xx} + s_{yy}$
$\gamma = \sqrt{(s_{xx} - s_{yy})^2 + 4s_{xy}^2}$

Suggestions for experiments:

- Illustrate the BIF response in some images using color codes similar to how this is done in Crosier and Griffin [4].
- Show the BIF histogram (set $\epsilon = 0, \sigma \in \{1, 2, 4, 8\}$) for an example image.
- Compare BIF histograms for a total of 30 images (6 texture classes, 5 images per class). Construct a 30×30 *confusion matrix* containing the histogram distances based on the L_1-norm (sum of absolute difference). Show the histogram as an image and explain the pattern.

[4] M. Crosier and L.D. Griffin. Using basic image features for texture classification. *International Journal of Computer Vision*, 88 (3):447–460, 2010. ISSN 0920-5691. DOI: 10.1007/s11263-009-0315-0

5.1.4 *Texture synthesis: Task 2*

In this exercise you will synthesize image texture using a method similar to the one presented in [5]. This method is based on fitting partly overlapping image patches to an image with holes by sampling (randomly) from a distribution of similar image patches. The distribution is approximated from the image itself by measuring distances to patches from the image itself.

Suggestions for experiments:

- Choose or construct a simple test example with repeated texture and a small hole and fill in the missing part.
- Choose a natural image and fill in a hole. Try varying number of patches, patch size, hole size, type of image, etc.
- Can the method be used for noise reduction? Experiment with e.g. salt and pepper noise.

[5] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038 vol.2, 1999

5.2 *Optical flow*

Small movements between two consecutive frames of an image series can be modeled as optical flow. The problem of optical flow is to determine local translations between two frames as a vector field such that the brightness constancy constraint

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t), \tag{5.2}$$

is fulfilled. Here, x and y are spatial coordinates and t is time. In this exercise you will implement methods for computing optical flow of the movement between two images.

A number of algorithms have been suggested for solving the flow problem, and a simple solution is to match patterns locally using block matching, where a window around a point in the first image is translated and compared to a window in the other image using e.g. sum of squared differences. This is however a time consuming task, so other methods based on computing differentials have been suggested. These include the Lucas-Kanade [6] and the Horn-Shunck methods [7] that you will work with in this exercise. This exercise is based on the book Computer Vision: Algorithms and Applications, Chapter 8 [8] (can be downloaded from `http://findit.dtu.dk/`). But you may also find relevant information from the internet and the two original papers.

In this exercise, different approaches for solving the optical flow problem are given, and it is suggested that you start by working with the basic elements of *Optical flow* and then try working with the Lucas-Kanade method or the Horn Shunck method and preferably both. These methods have a number of common elements, so when one is implemented it is relatively easy to implement the other.

[6] B D Lucas and T Kanade. An iterative image registration technique with an application to stereo vision. 1981

[7] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981

[8] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010

5.2.1 *Optical flow*

The assumption behind optical flow is that the movement between frames is small. Therefore, the optical flow can be computed by

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v = -\frac{\partial I}{\partial t}, \tag{5.3}$$

where u and v are displacements in the horizontal and vertical directions respectively.

Task: Derive (5.3) form (5.2) by using a first order Taylor approximation.

In (5.3) two unknown parameters (u, v) must be computed, but in a single pixel there is just one equation, so the problem is underdetermined. If a small neighbourhood around a pixel is assumed to have the same displacement, it will be possible to solve (5.3) as a linear least squares problem, where we have $\mathbf{Au} = \mathbf{b}$, where $\mathbf{u} = [u, v]^T$.

Task: Write up the elements of $\mathbf{A}$ and $\mathbf{b}$ for a 3×3 neighbourhood.

Two small composed images of 10×10 pixels called `composedIm_1.png` and `composedIm_2.png` are available on Campusnet. They are made from an image by extracting two patches shifted by one pixel. Furthermore a 3×3 region of the same pattern is placed in the image, but

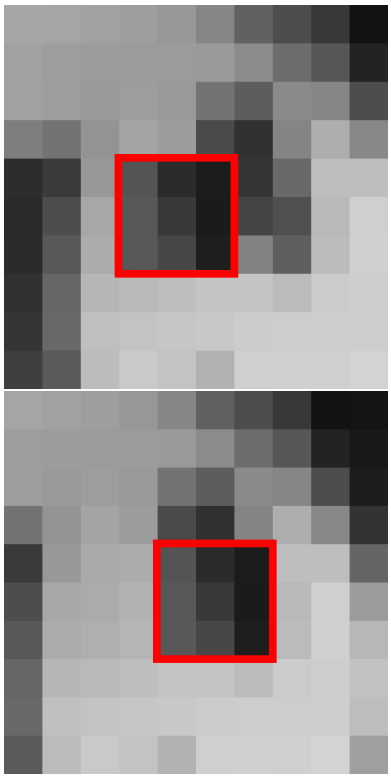

Figure 5.1: Two images of the same patten shifted one pixel to the left, whereas the center part marked in the red box is shifted on pixel to the right.

shifted in the opposite direction as shown in Figure 5.1. You can use these two images to try simple experiments with optical flow.

Task: Compute the optical flow vector for a window of 3×3 pixels centered at $(r, c) = (2, 6)$ and $(r, c) = (5, 4)$, where r is the row and c is the column. You should use a simple pixel differences to compute the differential by using the central difference filters $[-1, 0, 1]$ and $[-1, 0, 1]^T$. You can also use $[-1, 1]$ and $[-1, 1]^T$, but then the derivative is computed between pixels. You can ignore this and compare the result to the central difference filter.

The least squares solution to $\mathbf{Au} = \mathbf{b}$ is found by solving the minimization problem

$$\arg\min_{\mathbf{u}} \|\mathbf{Au} - \mathbf{b}\|^2. \tag{5.4}$$

Taking the derivative with regards to $\mathbf{u}$ and setting to zero yields

$$\mathbf{u} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \tag{5.5}$$

This allows us to precompute $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ as a number of sums over the image that can be obtained efficiently by filtering.

Task: Write up the elements of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$.

Task: Precompute the input needed for $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ and implement a filtering scheme that sums the elements for the small test images `composedIm_1.png` and `composedIm_2.png`. Compute the flow vectors for the full images.

Task: Display the flow vectors on top of the images using the MATLABfunction `quiver`.

5.2.2 *Lucas-Kanade method*

In this and the next part you should experiment with larger images. There are some benchmark images available from the Middlebury benchmark homepage `http://vision.middlebury.edu/flow/`. Some images from here have been uploaded to CampusNet that you can use for this exercise. But you are also welcome to experiment with your own images.

What you implemented in part 1 is a simple version of the Lukas-Kanade method. When you are working with larger images, there are however some practical aspects to consider. The linear system $\mathbf{Au} = \mathbf{b}$ may be ill posed such that there are no vector $\mathbf{u}$ that fulfills the equation. If this is the case, the 2×2 matrix $\mathbf{A}^T\mathbf{A}$ does not have an inverse.

Task: Find a way to check if there is a solution to the equation $\mathbf{Au} = \mathbf{b}$, i.e. that $\mathbf{A}^T\mathbf{A}$ has an inverse. Implement that in your solution for computing the optical flow vector field.

Task: Compute and display the flow field in two larger images from e.g. the Middlebury dataset.

In part 1 you implemented the image differential using pixel differences. There are also other methods for computing image differentials like central differences using the filter $[-1, 0, 1]$ and its transpose. You may also use a differential of a Gaussian.

Task: Test one or more differential filters.

Instead of treating all pixels from a window around a point in the image equally better results can be obtained by weighting the pixels using a weight matrix $\mathbf{WAu} = \mathbf{Wb}$ where $\mathbf{W}$ is a diagonal weight matrix, resulting in the least squares solution $\mathbf{u} = (\mathbf{A}^T\mathbf{WA})^{-1}\mathbf{A}^T\mathbf{Wb}$. This can efficiently be implemented using a weighted filter e.g. a Gaussian.

Task: Implement a weighted sampling window as a filter operation. Display the result on test images using different filter size.

Task: Comment on performance and processing time for the Lucas-Kanade method.

5.2.3 *Horn-Shunck method*

A problem with the Lucas-Kanade method is that it operates locally, so in regions with no texture it is not possible to compute the flow vectors. This is solved in the Horn-Shunck method where the Laplacian over the vector field is minimized in addition to ensuring that the brightness is constant by minimizing

$$E = \int\int [(I_x u + I_y v + I_t) + \alpha^2(\|\nabla u\|^2 \|\nabla v\|^2)]\mathrm{d}x\mathrm{d}y. \tag{5.6}$$

The energy E is minimized by iteratively updating the flow vectors using the update rules

$$u^{k+1} = \bar{u}^k - \frac{I_x(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \tag{5.7}$$

$$v^{k+1} = \bar{v}^k - \frac{I_y(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \tag{5.8}$$

where $\bar{u}^k$ is the average flow over a window in the x-direction.

Task: Implement the Horn-Shunck method and test it on two larger images from e.g. the Middlebury dataset. Display the flow vectors.

The choice of the α parameter and the number of iterations influence the smoothness of the obtained result. Also the choice of how the image is differentiated will affect the performance.

Task: Display results with varying parameter choices. Display results of a good and a bad choice of α. Do the same for number of iterations and size of averaging.

Task: Experiment with different choice of differentiation method.

Task: Comment on performance and processing time for the Horn-Shunck method.

5.3 *Layered surfaces*

A volumetric segmentation problem can often be constrained in terms of topology. We may for example be interested in segmenting a roughly spherical object, tubular objects or surfaces. Such topological constraints strongly reduce the solution space, and may turn an otherwise challenging problem into a solvable problem, an example is shown in Figure 5.2.

In this mini-project we focus on optimal net surface detection via graph search originally suggested by Wu and Chen [9] for segmenting terrain-like surfaces. They construct a graph on a set of sample points from a volume, such that the roughness of possible solutions is constrained. The optimality of the solution is defined in terms of a volumetric cost function derived from the data. The approach has been extended for finding multiple interrelated layered terrain-like and tubular surfaces [10], which made it applicable for medical image segmentation and led to further extensions. One extension involves the volumetric cost function which determines an optimal placement of the surface. This was originally defined only in terms of on-surface appearance, and has been extended to incorporate appearance of the regions between surfaces [11].

We will here review an algorithm for finding optimal layered surfaces, with the focus on the inputs and the outputs. For details on *how* this algorithm works, the reader is referred to article by Li et al. We also very briefly cover the principle of transforming the data into volumetric cost, which is the input to the layered surface detection algorithm.

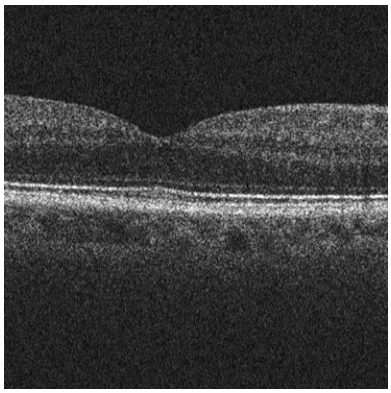

Figure 5.2: OCT (optical coherence tomography) image of retina. Quantifying the thickness of retinal layers is of clinical importance.

[9] Xiaodong Wu and Danny Z Chen. Optimal net surface problems with applications. In *Automata, Languages and Programming*, pages 1029–1042. Springer, 2002

[10] Kang Li, Xiaodong Wu, Danny Z Chen, and Milan Sonka. Optimal surface segmentation in volumetric images – a graph-theoretic approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):119–134, 2006

[11] Mona Haeker, Xiaodong Wu, Michael Abràmoff, Randy Kardon, and Milan Sonka. Incorporation of regional information in optimal 3-d graph search with application for intraretinal layer segmentation of optical coherence tomography images. In *Information Processing in Medical Imaging*, pages 607–618. Springer, 2007

5.3.1 *Layered surface detection*

In a discrete volume $x \in \{1,\dots,X\}$, $y \in \{1,\dots,Y\}$, $z \in \{1,\dots,Z\}$, a terrain-like surface s defined by $z = s(x,y)$ meets a smoothness constraint (Δ_x, Δ_u) if

$$|s(x,y) - s(x-1,y)| \leq \Delta_x \quad \text{and} \quad |s(x,y) - s(x,y-1)| \leq \Delta_y \,. \tag{5.9}$$

For a cost volume $c(x,y,z)$, an *on-surface* cost of s is defined as

$$C_{\text{on}}(s,c) = \sum_{x=1}^{X} \sum_{y=1}^{Y} c(x,y,s(x,y)) \,. \tag{5.10}$$

The *optimal net surface problem* is concerned with finding a terrain-like surface with a minimum cost among all surfaces satisfying smoothness constraint.

The polynomial time solution presented in the work by Wu and Chan transforms the optimal net surface problem into that of finding a *minimum-cost closed set* in a node-weighted directed graph with nodes representing volume voxels. This is further transformed into a problem of finding a *minimum-cost s-t cut* in a related arc-weighted directed graph. Minimum-cost *s-t* cut can be solved in polynomial time and efficiently found using algorithm of Boykov and Kolmogorov [12], a well known tool for many image segmentation tasks. While the optimal net surface problem is ultimately solved using the minimum-cost *s-t* cut algorithm, it should be noted that the graph constructed for surface detection is rather different than when used for Markov random fields.

The extension to multiple surfaces developed in [13] may be exemplified by considering two terrain-like surfaces s_1 and s_2. The surfaces are said to meet an overlap constraint $(\delta_{\text{low}}, \delta_{\text{high}})$ if

$$\delta_{\text{low}} \leq |s_2(x,y) - s_1(x,y)| \leq \delta_{\text{high}} \,. \tag{5.11}$$

Given two cost volumes c_1 and c_2 the total cost associated with surfaces s_1 and s_2 is

$$C_{\text{on}}(s_1,c_1) + C_{\text{on}}(s_2,c_2)$$

and the optimal surface detection will return a pair of surfaces with a minimum cost among all surfaces satisfying overlap and smoothness constraint. Depending on the problem at hand c_1 and c_2 may be different or identical, and likewise smoothness constrains may vary or be the same for the two surfaces.

Finally, two *layered* (i.e. non-intersecting and ordered) surfaces give rise to an *in-region* cost corresponding to the volume between two surfaces, and defined by

$$C_{\text{in}}(s_1,s_2,c_{1,2}) = \sum_{x=1}^{X} \sum_{y=1}^{Y} \sum_{z=s_1(x,y)+1}^{s_2(x,y)} c_{1,2}(x,y,z) \,. \tag{5.12}$$

[12] 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

[13] Kang Li, Xiaodong Wu, Danny Z Chen, and Milan Sonka. Optimal surface segmentation in volumetric images – a graph-theoretic approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):119–134, 2006

This cost, together with the cost for the region under the surface s_1 and the region over the surface s_2, can be incorporated into the minimization problem [14]. We use notation $c_{0,1}$ and $c_{2,3}$ for the cost volumes in connection with the boundary regions.

To summarize, for finding K cost-optimal layered surfaces we need to define

- K on-surface cost volumes c_k, $k = 1, \ldots, K$, and/or
- $K+1$ in-region cost volumes $c_{k,k+1}$, $k = 0, \ldots, K$.

The set of feasible surfaces is given by

- K smoothness constraints $\left(\Delta_x^k, \Delta_y^k\right)$, $k = 1, \ldots, K$ and
- $K-1$ overlap constraints $\left(\delta_{\text{low}}^{k,k+1}, \delta_{\text{hight}}^{k,k+1}\right)$, $k = 1, \ldots, K-1$.

Layered surface detection has found an immediate use for detecting tubular surfaces. The main principle is the fact that a circle $x^2 + y^2 = \rho^2$ appears as a straight line $r = \rho$ when represented in polar (r, θ) coordinates. Detecting a tubular surface is achieved by representing the volumetric data in a cylindrical coordinate system (r, θ, z) with the longitudinal axis $r = 0$ roughly aligned with the center of the tube. We call this transformation *unwrapping* the volume, and we also say that the volume is sampled along the radial rays. Important practical parameters for unwrapping are the radial and the angular resolution. In the unwrapped representation, the tubular surface is terrain-like and can be defined as $r = s(\theta, z)$. When using layered surface detection for detection of tubular surfaces, additional constraints are added to ensure a smooth transition over $\theta = 0$.

[14] Mona Haeker, Xiaodong Wu, Michael Abràmoff, Randy Kardon, and Milan Sonka. Incorporation of regional information in optimal 3-d graph search with application for intraretinal layer segmentation of optical coherence tomography images. In *Information Processing in Medical Imaging*, pages 607–618. Springer, 2007

5.3.2 *Constructing cost volumes*

The surfaces returned by the layered surface detection algorithm are optimal in terms of the volumetric cost. Therefore, to detect a surface we need to define a cost volume c_k which takes small values where a data $V(x, y, z)$ supports the surface k. This modelling step, crucial for the performance of the algorithm, is fully dependent on the data.

If the surface to be detected is characterized by a certain voxel intensity v_s, then the cost volume may be defined as $(V - v_s)^2$. More often, the surface divides two regions of different intensities, so cost volume needs to be defined in terms of change of intensity. When computing intensity changes for tubular surfaces, the best approach is to first unwrap the volume, and then compute the change in the r direction.

5.3.3 *Examples*

Figure 5.3 demonstrates the use of the layered surface detection.

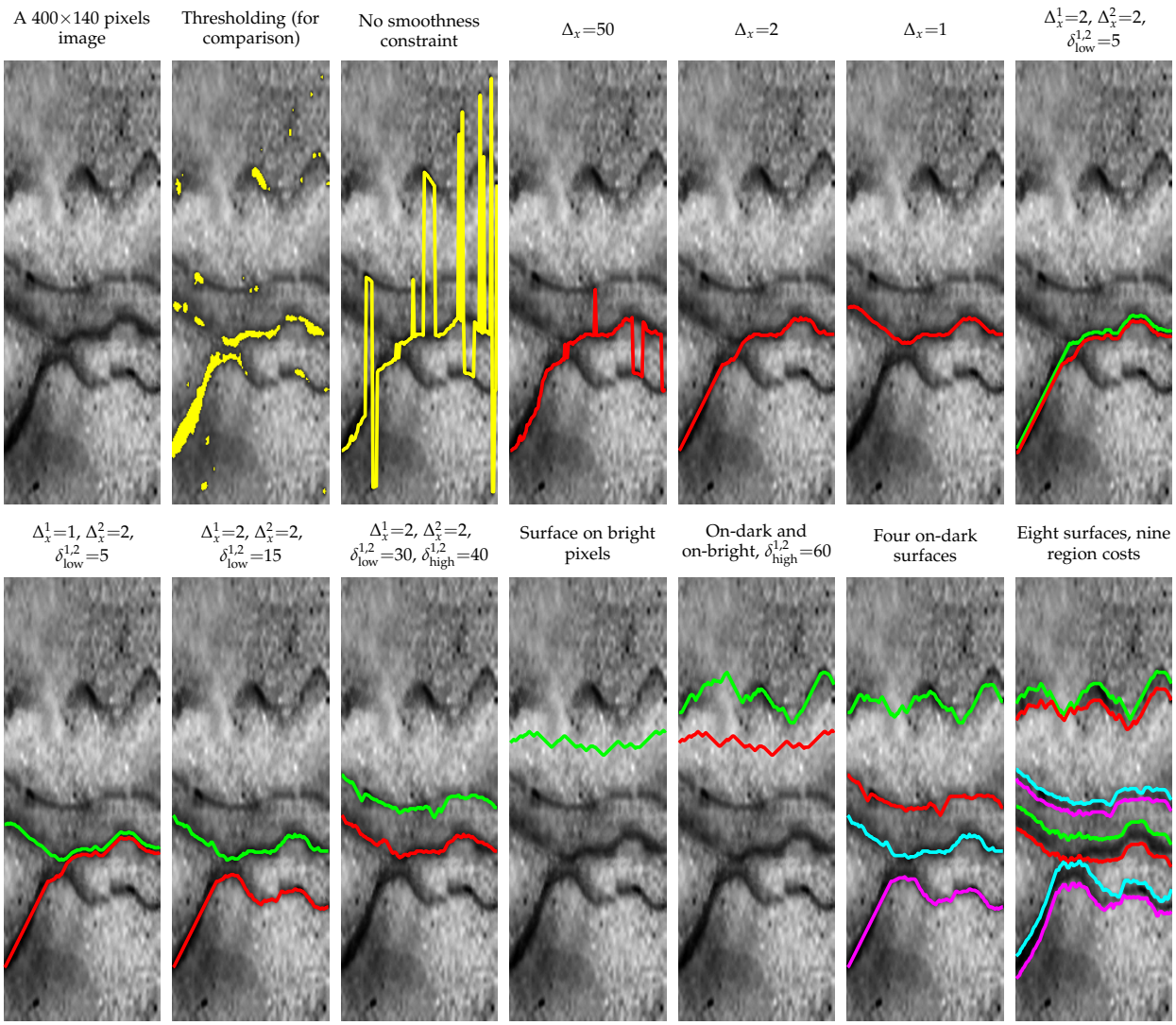

Figure 5.3: Output of layered surface detection. First three images serve to illustrate the problem). Images 4–6 show how changing smoothness constraint influences the result. Images 7–10 demonstrate the use of the overlap constraint. Images 11–12 demonstrate the use of different cost functions. Image 13 is a four-surface detection, while image 14 uses region costs.

5.3.4 *Proposed mini-projects for layered surface detection*

You may work with the layered surface detection in a number of ways. Contact the teacher to clarify the possibilities.

1. Use the provided algorithm for layered surface detection – only MATLAB users. [Effort: easy to difficult.]
2. Implement the algorithm for layered surface detection. For finding minimal *s-t* cut use the same implementation as previously used for Markov random fields exercise. [Effort: medium.]
3. Adapt the layered surface detection algorithm so that it operates on the triangular mesh. [Effort: difficult.]

5.4 *Spectral Clustering and Normalized Cuts*

Spectral clustering is an approach to data clustering problem, and it includes a number of related techniques. Spectral clustering is used in machine learning, computer vision and signal processing, with applications in processing speech spectrograms, DNA gene expression analysis, document retrieval and computation of Google page rank. The name *spectral* originates from the mathematical term *spectrum* (a set of the egenvalues of a given matrix), and this is because spectral clustering utilizes eigenvalues and eigenvectors of the data similarity matrix.

Spectral clustering is one of the fundamental data clustering approaches, it is easy to implement and solve by standard linear algebra software, there is no assumptions on the nature of the clusters, and the techniques have been mathematically rigorously proved. Disadvantages include high computational and memory requirements of the direct implementation. For this reason the practical use of spectral clustering often involves computational simplifications and significant pre- or postprecessing.

Spectral approach has gained a great popularity for image segmentation following the seminal paper on normalized cuts by Shi and Malik [15]. Recent uses of spectral approach is in the superpixel segmentation. Another intriguing example is the Copenhagen-based company Spektral (formerly known as CloudCutout) where spectral segmentation is a part of the successful green-screen removal product Figure 5.4.

Spectral methods can be applied to the data which is represented using a similarity matrix, and is often described in terms of graph partitioning. For a comprehensible coverage of (general) spectral clustering we recommend an excellent tutorial by von Luxburg [16]. We will here briefly cover spectral clustering, and will then turn to its use in image segmentation.

[15] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000

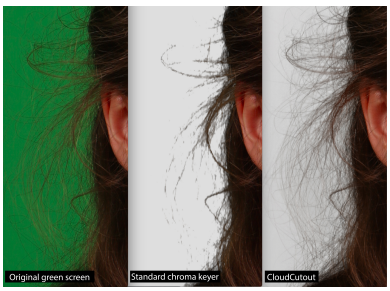

Figure 5.4: Spektral (formerly known as CloudCutout) green-screen product.

[16] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17 (4):395–416, 2007

Boiled down to four words, the essence of spectral clustering is: *Eigensolution gives graph partitioning*. To be able to understand spectral clustering, you need to be acquainted with the concept of graph cuts in order to describe the problem we want to solve. Furthermore, we need to represent a graph using an adjacency matrix and a closely related Laplacian matrix. Then we can see how eigensolution provides a solution to a graph cut problem.

5.4.1 *Graph cuts, graph representations and eigensolutions*

Recall that a graph consists of nodes and edges, and in general may be node-weighted, edge-weighed, directed or undirected. In context of spectral image segmentation, each pixel will correspond to a graph node, and pairs of pixels define graph edges – we will get back to image segmentation after covering the general case. For spectral clustering we work with edge-weighted undirected graphs which we represent using an adjacency matrix W with elements w_{ij} being the weight of the edge connecting the node i and a node j. Consider for example a graph in Figure 5.5 consisting of 12 nodes. This graph can be represented in terms of a 12×12 adjacency matrix, as illustrated in Figure 5.6.

A graph cut is a partitioning of a graph. The graph partitioning problems are concerned with finding a graph cut with the least cost. The simplest way of defining the cost of a cut is to consider all edges between the two partitions

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} w_{ij},$$

but that might lead to unbalanced cuts. For this reason we might prefer using some other measure of the cut cost, which also consider the size of the partitions. Commonly used are normalized cuts

$$\text{Rcut}(A, B) = \frac{\text{cut}(A, B)}{|A|} + \frac{\text{cut}(A, B)}{|B|},$$

where we use the notation $|A|$ for a number of vertices in subset A, but one could also consider ratio cut and min-max cuts

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{vol}(A)} + \frac{\text{cut}(A, B)}{\text{vol}(B)},$$

$$\text{MMcut}(A, B) = \frac{\text{cut}(A, B)}{\text{cut}(A, A)} + \frac{\text{cut}(A, B)}{\text{cut}(B, B)},$$

where $\text{vol}(A)$ is weight of all edges associated with the subset, i.e. $\text{vol}(A) = \sum_{i \in A} d_i$ and $d_i = \sum_j w_{ij}$ is a degree of ith node. Figure 5.7 shows three graph cuts, while Figure 5.8 lists the costs associated with the three cuts given by different measures. You may confirm that the

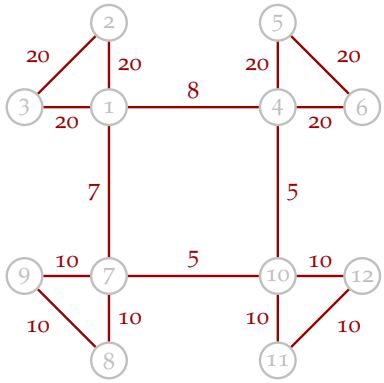

Figure 5.5: An example of an edge-weighted undirected graph.

	1	2	3	4	5	6	7	...
1	0	20	20	8	0	0	7	...
2	20	0	20	0	0	0	0	...
3	20	20	0	0	0	0	0	...
4	8	0	0	0	20	20	0	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Figure 5.6: An adjacency matrix of a graph shown in Figure 5.5.

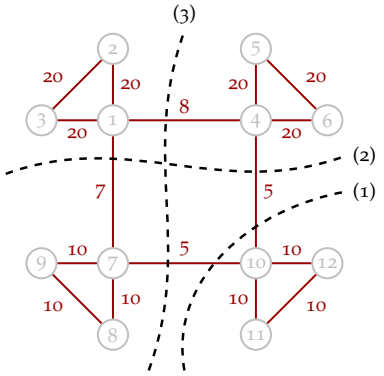

Figure 5.7: Three different partitioning of a graph.

	(1)	(2)	(3)
cut	10	12	13
Rcut	4.4̇	4.0	4.3̇
Ncut	0.1723	0.1293	0.1268
MMcut	0.3939	0.2784	0.2709

Figure 5.8: Values of the different cuts shown in Figure 5.7.

values are correct, and notice the different balancing properties of the cost measures.

It turns out that the solution to the graph cut problem may be found by considering the eigensolution of the matrix closely related to the adjacency matrix – the graph Laplacian. Depending on the cost measure, the derivation will be slightly different, leading to the different (unnormalized and normalized) versions of the graph Laplacians. To normalize the Laplacian, we first define a diagonal degree matrix $\mathbf{D}$ with diagonal elements being node degrees $d_i = \sum_j w_{ij}$.

We define unnormalized graph Laplacian

$$\mathbf{L} = \mathbf{D} - \mathbf{W},$$

and two normalized graph Laplacians

$$\mathbf{L}_{\text{sym}} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2},$$

$$\mathbf{L}_{\text{rw}} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}.$$

These matrices are closely related to each other, and so are their spectra. All three matrices have real-valued eigenvalues with 0 being the smallest eigenvalue, see tutorial by von Luxburg for a more complete list of properties. For our purposes, the most important are eigenvectors of $\mathbf{L}_{\text{rw}}$, i.e. vectors satisfying $\mathbf{L}_{\text{rw}}\mathbf{u} = \lambda\mathbf{u}$. The smallest eigenvectors (i.e. corresponding to smallest eigenvalues) yield solution to finding the normalized cut. The eigenvectors provide the representation of the date which enhances the cluster-properties, so that clusters can be trivially detected for example using k-means clustering. In particular, the second smallest eigenvector gives the partitioning of the data in two subsets – the very smallest eigenvector (corresponding to 0) is constant.

It can be shown that eigenvectors of $\mathbf{L}_{\text{rw}}$ are generalized eigenvectors of $\mathbf{L}$ and $\mathbf{D}$, see againg von Luxburgs tutorial, Proposition (3) part 3. Therefore, to fint the solution to normalized cut, one may also seek solution to generalized eigenproblem $\mathbf{L}\mathbf{u} = \lambda\mathbf{D}\mathbf{u}$. This is the formulation of normalized spectral clustering according to the original paper by Shi and Malik. In practice, solving a standard eigenproblem requires less computation.

We can utilize other matrix algebra identities to make computation of the spectra more accurate and/or efficient. Many eigensolvers are more accurate when working of symmetric matrices. A strategy exploiting matrix symmetry would involve computing eigenvectors of the (symmetric) $\mathbf{L}_{\text{sym}}$, and transforming those to eigenvectors of $\mathbf{L}_{\text{rw}}$ by multiplying with $\mathbf{D}^{-1/2}$, see tutorial by von Luxburg, Proposition (3) part 2. Furthermore, if only a subset of eigenvectors is to be found, some eigensolvers are more efficient when finding eigenvectors corresponding to largest eigenvalues. This may be utilized by noticing that that smallest eigenvectors of $\mathbf{I} - \mathbf{A}$ are largest eigenvectors of $\mathbf{A}$.

5.4.2 *Clustering 2D points*

You should implement two functions. One function should take and affinity matrix and return eigenvectors corresponding to solution for normalized cuts. The second function should perform discretization of the eigenvectors using k-means.

You should first test your implementation on a small 2D point set. You can use points previously used for neural networks, but we also provide five point sets in a mat file `points_data.mat`. For each of the point sets you should:

- Visualize the point set, and identify the clusters (there are 2 clusters in set 3 and 5, and 3 clusters in set 1, 2 and 4).
- Construct the affinity matrix W. Use the fully connected graph and Gaussian similarity function (Luxburg, Section 2.2). You should initially estimate parameter σ so that it reflects the distance between the neighbouring points of the point cloud.
- Compute eigenvectors and clustering given by the normalized cut. Visualize the clustering.
- Determine ordering (permutation) of the points according to the clustering (so that points from the first cluster come first, followed by points in the second cluster, etc.)
- Visualize the values of the second eigenvector, first for unsorted points, then for sorted points.
- Visualize affinity matrix, and the affinity matrix for sorted points.
- Estimate the parameter σ which results in a meaningful clustering. You will need to change the parameter σ between point sets.

5.4.3 *Image segmentation*

Now we use spectral clustering on a pixels of a small image. Consider some of the provided test images. You might want to (drasticaly!) reduce the size of your images, to avoid memory problems.

You should:

- Construct the affinity matrix W using Equation (11) from article by Shi and Malik. Initially estimate parameters σ_I, σ_X and r. Instead of setting a radius r, for our small example you may use a fully connected graph (i.e. ignore if–otherwise condition of Equation (11) which sets affinity of distant points to 0).
- Visualize the spatial part of W, the brightness part of W and the final W.

- Compute eigenvectors and clustering given by the normalized cut.
- Visualize the values of the second eigenvector on the image grid.
- Visualize the segmentation results.
- Estimate the model parameters to obtain meaningful segmentation.

Start by the grayscale image. Use 2 clusters for plane and 5 clusters for vegetables. For the similarity (brightness, color) part of W treat an RGB value of each pixel as a vector to compute the Euclidian distance between the pair of pixels. Try also clustering the pixels using k-means clustering by treating RGB pixels values as vectors.

Consider adapting spectral methods to be able to handle larger images.

6 Bibliography

CM Bishop. Pattern recognition and machine learning (information science and statistics), chapter 3, pages 138–147, 2006.

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.

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.

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

M. Crosier and L.D. Griffin. Using basic image features for texture classification. *International Journal of Computer Vision*, 88(3):447–460, 2010. ISSN 0920-5691. DOI: 10.1007/s11263-009-0315-0.

A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1033–1038 vol.2, 1999.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Mona Haeker, Xiaodong Wu, Michael Abràmoff, Randy Kardon, and Milan Sonka. Incorporation of regional information in optimal 3-d graph search with application for intraretinal layer segmentation of optical coherence tomography images. In *Information Processing in Medical Imaging*, pages 607–618. Springer, 2007.

Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

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

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.

Kang Li, Xiaodong Wu, Danny Z Chen, and Milan Sonka. Optimal surface segmentation in volumetric images – a graph-theoretic approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):119–134, 2006.

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

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

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

B D Lucas and T Kanade. An iterative image registration technique with an application to stereo vision. 1981.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22 (8):888–905, 2000.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Xiaodong Wu and Danny Z Chen. Optimal net surface problems with applications. In *Automata, Languages and Programming*, pages 1029–1042. Springer, 2002.

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

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.