Anders Bjorholm Dahl
Vedrana Andersen Dahl

Advanced Image Analysis

SELECTED TOPICS

DTU Compute, February 1, 2022

Preface

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

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

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

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

The structure of the note is as follows. First comes a general introduction to a few central aspects relevant for image analysis along with the first introductory exercise, which has the purpose of refreshing basic image analysis concepts. The main text includes three compulsory exercises. Towards the end of the note we provide a number of examples for the final exercise in form of a mini-project.

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

1 Introduction

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

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 a scalar value $I(x, y)$ is assigned to each coordinate (x, y) in the image domain Ω. Sometimes we consider the image as digital signal sampled at integer values, i.e. running from 1, such that $x \in \{1, ..., X\}$ and $y \in \{1, ..., Y\}$. Still, many papers will ignore the discrete nature of the images, and treat I as a continuous function. In general, scientific articles show substantial variation on the notation of an image, e.g. it is often implicitly assumed that the image lives in 2D space, and the notation would simply be a symbol like I.

In the computer program, a gray-scale image is represented as a 2D array of numbers. Here, the indexing of the array elements is implicitly related to image space (x, y), however, one needs to consider program-specific details: 0 or 1 indexing, placement of the origin, and so on. For mathematical treatment of certain topics it is convenient to considered images as if centered around origin. For example, when we talk about filtering kernels. Also here it is important to be aware of the difference between the notation, and the actual representation of the kernel.

A 3D image is typically termed a volume, and again here we can model it as a function $I : \Omega \subset \mathbb{R}^3 \rightarrow \mathbb{R}$. Similar to before, I is a function that maps to a scalar value, but here from a 3D coordinate (x, y, z). Volumetric images are often reconstructed from projection data obtained using a scanner, e.g. a CT or an MRI scanner. 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. This means that each pixel value may be represented as a 1D array of values that encode the recorded spectral bands. A common example are the RGB images where $I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$ encodes the red, green, and blue band. If more spectral bands are recorded, we are typically talking about multispectral or hyper-spectral images where $I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^n$ normally with $n > 3$. For 2D spectral images we would often apply operators in each of the spectral bands independently. Using the smoothing example from before, but now in a RGB image, we would preform 2D smoothing in the R-band, G-band, and B-band individually.

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

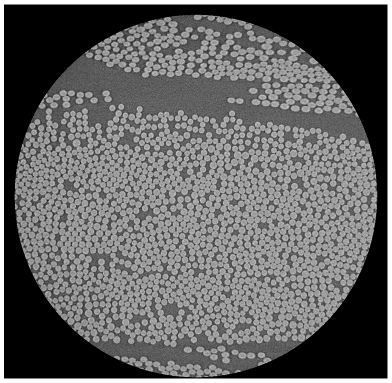


Figure 1.1: Slice of a CT image of glass fibers viewed orthogonal to the fiber direction.

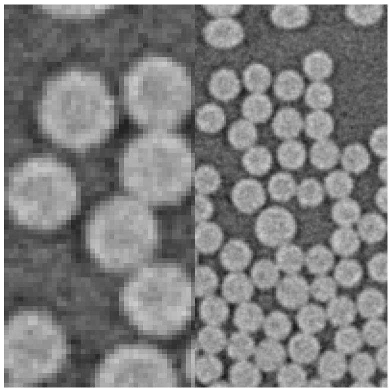


Figure 1.2: Two zoomed in images from image shown in Figure 1.1.

1.1 Introductory exercise

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

1.1.1 Image convolution

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

For two continuous functions, convolution is defined as

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

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

For a discrete sampled signal we get

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

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

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

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

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

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

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

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

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

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

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

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

It can, however, often be desirable to smooth the image in connection with taking the derivative, e.g. to remove nosie. It turns out that if we want to convolve the image with a Gaussian and then take the derivative, we can instead convolve the image with the derivative of the Gaussian

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

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

The analytic 1D Gaussian derivative is given by

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

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

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

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

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

Tasks

1. Experimentally verify the separability of the Gaussian convolution kernel. Do this by convolving a test image with a 2D kernel, and convolving the same image with two orthogonal 1D kernels. Subtract the result and verify that the difference is very small.

2. Investigate the difference between the derivative of the image convolved by a Gaussian and the image convolved with the derivative of the Gaussian as described in Eq. 1.7. Note that you can compute the derivative of the image by convolving with the kernel `k = [0.5, 0, -0.5]`.

3. Test if a single large convolution with a Gaussian of $t = 20$ is equal to ten convolutions with a Gaussian of $t = 2$.

4. Test if convolution with a large Gaussian derivative

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

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

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

1.1.2 *Computing length of segmentation boundary*

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

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

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

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

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

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

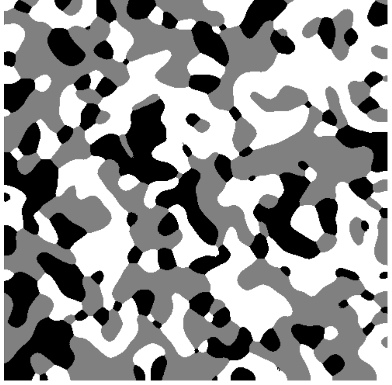


Figure 1.3: Image of a segmented fuel cell with three phases. Black represents air, grey is cathode, and white is anode.

Tasks

1. Compute the length of the segmentation boundary for provided segmentation images of a fuel cell, where one is shown in Figure 1.3. You can consider avoiding loops and instead using vectorization provided by Matlab or numpy, which will ensure an efficient and compact implementation.

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

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

1.1.3 *Curve smoothing*

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

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

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

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

where $\mathbf{L}$ is a N-times-N matrix with elements 1, -2, and 1 in every row such that -2 is on the main diagonal, and 1 on its left and right (also circularly in the first and the last row), and zeros elsewhere. See Figure 1.4 for an example.

$$\mathbf{L} = \begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

Figure 1.4: Matrix $\mathbf{L}$ for N=6. Notice value 1 in the upper right and lower left.

Using this step iteratively, where we use t for iteration number (i.e. not transpose or potential), gives

$$\mathbf{X}^{(t+1)} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{X}^{(t)}\,. \tag{1.11}$$

Confirm that one step with $\lambda = 0.5$ displaces every curve point exactly to the average of its neighbors. Try smoothing one of the provided contours, also shown in Figure 1.5. 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 curve $\mathbf{X}^{\text{new}}$. In other words, we can use an implicit (backwards Euler) approach. Instead of Equation 1.10 where $\mathbf{X}^{\text{new}} = \mathbf{X} + \lambda\mathbf{L}\mathbf{X}$ we use $\mathbf{X}^{\text{new}} = \mathbf{X} + \lambda\mathbf{L}\mathbf{X}^{\text{new}}$ leading to

$$\mathbf{X}^{\text{new}} = (\mathbf{I} - \lambda \mathbf{L})^{-1}\mathbf{X}\,. \tag{1.12}$$

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. While outside the scope of this course, an interested student may read on implicit smoothing of triangle meshes in an influential paper by Desbrun et al. [2].

[2] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, 1999

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

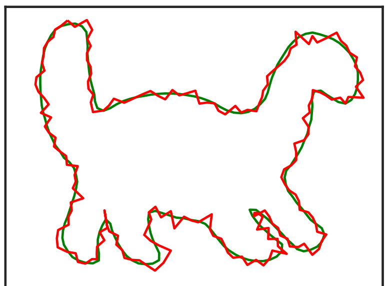
$$\alpha \begin{bmatrix} 0 & 1 & -2 & 1 & 0 \end{bmatrix} + \beta \begin{bmatrix} -1 & 4 & -6 & 4 & -1 \end{bmatrix}$$

with α and β weighting the two terms. See Hanbook of medical imaging [3], section 3.2.4, for the derivation of the kernels.

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

Smoothing is now obtained as

$$\mathbf{X}^{\text{new}} = (\mathbf{I} - \alpha\mathbf{A} - \beta\mathbf{B})^{-1}\mathbf{X}\,, \tag{1.13}$$



where **A** is identical to **L** we used before, and **B** is a very similar matrix, but having other values on the diagonals, e.g. -6 on the main diagonal. Note that $\alpha\mathbf{A} + \beta\mathbf{B}$ is also a (sparse) circulant matrix.

Tasks

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

2. Implement curve smoothing as in Equation 1.12 (implicit smoothing) and test it for various values of λ. Do you need an iterative approach of this smoothing?

3. Implement implicit curve smoothing but with the extended kernel. This means that your implementation instead of $\lambda\mathbf{L}$ uses a matrix that combines elasticity and rigidity, as in Equation 1.13. Test smoothing with various values of α and β. What do you achieve when choosing a large β and small α?

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

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

1.1.4 *Optional: Total variation*

In many image analysis applications, such as image denoising and image segmentation, we are interested in producing a result which has a quality that we loosely call *smoothness*. A common way of estimating smoothness is by considering a total variation defined for an image I as

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

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

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

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

1.1.5 *Optional: Unwrapping image*

A solution to image analysis problem may involve geometric transformations. When working with spherical or tubular objects, we sometimes want to represent an image in polar coordinate system. Implement a function which performs such *image unwrapping* using a desired angular and radial resolution. Use your function to unwrap one of the slices from the dental data set, an example is shown in Figure 1.6 and 1.7. Unwrapping will be useful when we will be working with deformable models later in the course.

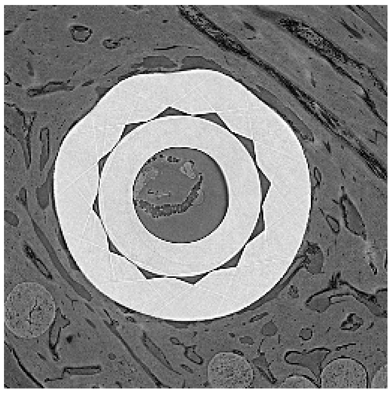


Figure 1.6: Image of a dental implant that should be unwrapped.

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

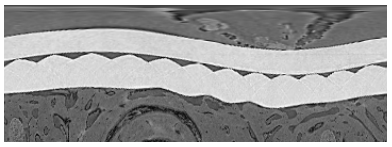


Figure 1.7: Unwrapped image of a dental implant.

1.1.6 *Optional: Working with volumetric image*

To give you a taste of working with 3D images, we have prepared a small data set containing slices from an X-ray CT scan. By convention in X-ray imaging, dense structures (having a height X-ray attenuation) are shown bright compared to less dense structures. Furthermore, the direction given by image slices is most often denoted z. The volume you are given contains a metal (very bright) object. Show orthogonal cross sections of the object, see Figure 1.8. Can you determine an optimal threshold for segmenting the object from the background?

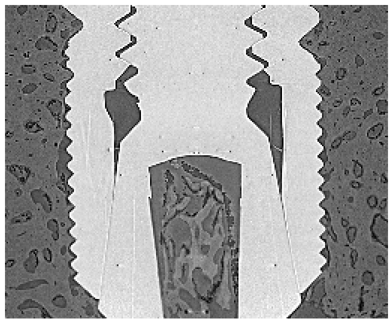


Figure 1.8: A longitudinal slice (an xz-plane) of the volumetric image of a dental implant.

Optionally, show a volumetric 3D rendering of the thresholded object using any available software. An example is shown in Figure 1.9. If you are using MATLAB, check a function `isosurface`.

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

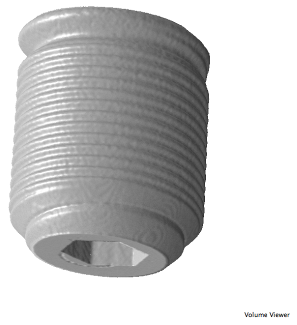
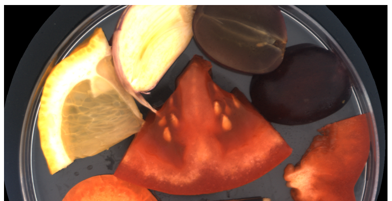


Figure 1.9: 3D rendering of the thresholded volumetric image of a dental implant.

1.1.7 *Optional: PCA of multispectral image*

Principal component analysis (PCA) is a linear transform of multivariate data that maps data points to an orthogonal basis according to maximum variance. A basic introduction in PCA is given in [4], and here we will apply it on a multispectral image.

We provided an image acquired with the VideometerLab, which is a multispectral imaging device, that uses coloured LED's to illuminate a material, in this case samples in a petri dish. This gives an 18 channel image $I : \Omega \subset \mathbb{R}^2 \rightarrow \mathbb{R}^{18}$ where each channel corresponds to a wavelength, and channels cover the range from 410 nm to 955 nm, i.e. the visible and near-infrared spectrum. The image depicts vegetables on a dish and is shown in false colours in Figure 1.10.



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

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

$$\mathbf{C}_{ij} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{X}_{ni} - \mu_i)(\mathbf{X}_{nj} - \mu_j), \tag{1.14}$$

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

$$\mathbf{C} = \frac{1}{N-1}\overline{\mathbf{X}}^T\overline{\mathbf{X}}, \tag{1.15}$$

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

$$\mathbf{Q} = \overline{\mathbf{X}}\mathbf{V}. \tag{1.16}$$

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

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

1. Write a script to read in the images and display them. Convince yourself that there is a difference between the spectral bands. Make sure to change the data type to float or double.

2. Rearrange the image into a matrix $\mathbf{X}$ as described above with one pixel in each row. Compute the column-wise mean μ and subtract this from $\mathbf{X}$ to get the zero mean $\overline{\mathbf{X}}$.

3. Compute the covariance matrix **C**.

4. Compute the eigenvectors **V** and eigenvalues λ.

5. Compute the principal component loadings **Q**.

6. Rearrange **Q** into images and display the result.

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

1.1.8 *Optional: Bacterial growth from movie frames*

Image data with a temporal component can be stored in the form of a movie. The purpose of this exercise is to read in image frames from a movie and analyze them. The movie contains microscopic images of listeria bacteria growing in a petri-dish acquired at equal time steps. An example frame is shown in Figure 1.11. Your task is to make a small program that visualize bacterial growth by counting the cells.

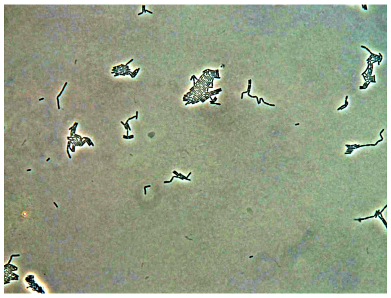


Figure 1.11: Example of microscopic image of listeria bacteria in a petri dish.

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

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

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

1. Convert the image I to a grey scale image G.

2. Compute the gradient magnitude $M = \sqrt{(\partial G/\partial x)^2 + (\partial G/\partial y)^2}$ using an appropriate filter.

3. Smooth M using a Gaussian filter.

4. If the parameters have been chosen appropriately, the pixels covering bacteria can now be segmented by thresholding.

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

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

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