Anders Bjorholm Dahl
Vedrana Andersen Dahl

Advanced Image Analysis

SELECTED TOPICS

DTU Compute, February 8, 2024

Contents

Preface

This lecture note is a collection of topics for the students taking the course 02506 Advanced Image Analysis at the Technical University of Denmark. The note provides background material for the course together with practical guidelines and advice for carrying out course tasks.

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 to give 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 computer programs since they are described using terminology and mathematical notation, which may vary between articles. In some cases, the description of methods can be very different from the code you need to write. One aim of this note is to guide the implementation of image analysis algorithms from descriptions in articles to 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.

During the course, you will be implementing methods and algorithms that are already integrated into existing software libraries. These commercial or public implementations are probably better than what you can achieve given the time available for the exercises. The reason to reimplement methods and algorithms is to foster deep understanding of how image analysis methods work. This process equips you with the skills necessary for implementing or modifying advanced methods where no existing implementation is available.

1 Introduction

An image is a regularly sampled signal, typically representing light intensity. In image analysis, we use the image to extract information about the signals we have measured.

Mathematically, a gray-scale image may be represented as a function $I(x, y)$ with $I : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$. Each coordinate (x, y) in the image domain Ω is assigned a scalar value $I(x, y)$. The signal is sampled at integer values, but some sources treat I as a continuous function.

In computer programs, 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 may need to consider specific details: 0 or 1 indexing, placement of the origin, and so on. For certain images (for example, filtering kernels) it is convenient to place the origin in the center.

A 3D image is typically termed a volume, and here we model it as a function $I : \Omega \subset \mathbb{R}^3 \to \mathbb{R}$. Volumetric images are often reconstructed from projection data obtained using a scanner, e.g. CT or MRI scanner. Operators on volumetric images are also volumetric, e.g. smoothing is achieved using filters that operate in all three dimensions. For some volumetric images, e.g. medical CT images, the sampling is anisotropic, and this can influence the applied method. Figures 1.1 and 1.2 show a slice of a CT image.

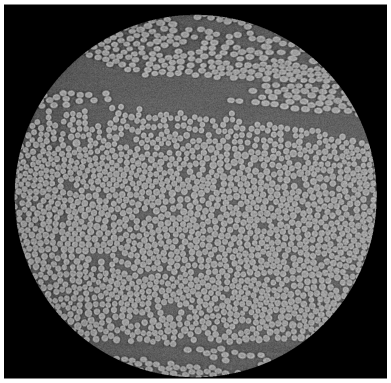
Figure 1.1: Slice of a CT image of glass fibers viewed orthogonal to the fiber direction.

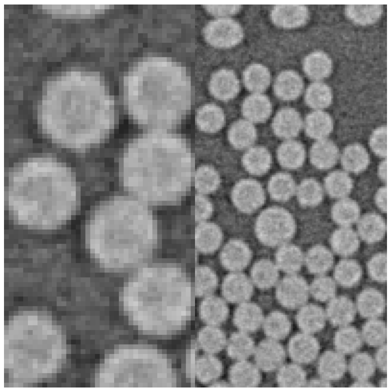
Figure 1.2: Two zoomed in regions from the image shown in Figure 1.1.

Spectral images have multiple measures (spectral bands) in each pixel. A common example is the RGB image 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. For spectral images, we often apply operators in each of the spectral bands independently. For example, we would perform 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 t is a time dimension. For movies, we would typically expect small changes between frames, and this can be utilized in the analysis.

Saving, loading and visualizing images Images may be stored in a variety of formats, and image readers from different packages may behave differently, leading to unexpected results. You should always check the image size and data type after reading an image. Also, be careful when visualizing images; choose an appropriate colormap and adjust the value range. A notebook provided in the course repository contains examples of reading and visualizing an image (shown in Figure 1.3) using `scikit-image` and `matplotlib`.

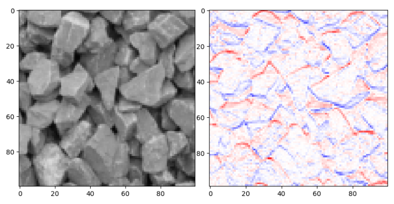

Figure 1.3: Adequate choice of colormap when visualizing a grayscale image and its gradient.

1.1 Introductory exercise

This exercise serves to refresh concepts from the basic image analysis curriculum that will be useful at a later stage in the course.

1.1.1 Image convolution

Image convolution is a central tool in image analysis. In this exercise, you will investigate properties related to convolution using a Gaussian kernel and its derivatives. For further reading material regarding convolution and filtering, refer to [1], Chapter 5.[2]

[1] Wilhelm Burger, Mark James Burge, Mark James Burge, and Mark James Burge. *Principles of digital image processing*, volume 54. Springer, 2009

[2] This exercise will be useful in 2, when we will be working with scale space.

For continuous functions f ang g, 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 sometimes distinguish between the signal and the kernel, and we say that the signal is convolved with the kernel. In discrete settings, the equation becomes

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

In 2D, a convolution with a square kernel 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 for image smoothing. The 1D Gaussian with variance t is defined by

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

Sometimes we parametrize the Gaussian with the standard deviation σ, where $t = \sigma^2$. The 2D isotropic Gaussian is given by

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

The Gaussian is separable ([3] Section 5.3.1), which means that convolution with two orthogonal 1D Gaussians yields the same result as convolving with a 2D Gaussian of the same variance. This can significantly speed up convolutions, especially for large kernels.

[3] Wilhelm Burger, Mark James Burge, Mark James Burge, and Mark James Burge. *Principles of digital image processing*, volume 54. Springer, 2009

Another property of the Gaussian convolution is the so-called *semi-group structure*, stating that convolving an image I with a single large Gaussian is equivalent to convolving with several small ones

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

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

For an image I, we often need to know a local change in intensity values. This can be achieved by taking the spatial derivative. Since the image is a discretely sampled signal, we approximate the derivative, often by computing the difference between neighboring pixels.

When taking the derivative, it is often desirable to remove the noise by smoothing, e.g. using a Gaussian. It turns out, that instead of convolving with a Gaussian and then taking the derivative, we can convolve 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 expression for the 1D Gaussian derivative is

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

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

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

This implies that convolving with a large Gaussian derivative yields the same result as convolving with a smaller Gaussian and a smaller Gaussian derivative.

Exercise overview We will experimentally test the statements made in the text above. You may use an existing implementation of convolution e.g. `scipy.ndimage.convolve`. However, refrain from using an existing implementation of the Gaussian filtering or functions returning the Gaussian kernel and its derivative.

Data For this exercise you may use any grayscale image. We have provided an X-ray CT image of fibres `fibres_xcth.png`, shown in Figure 1.1 and Figure 1.2.

Tasks

1. Create Gaussian kernel. Gaussian kernels are usually truncated at the value between 3 and 5 times σ. You can create a kernel as follows:

   (a) Compute an integer kernel radius s approximately equal to 4σ.

   (b) Create an array x with integer values centered around 0, that is $x = [-r, \dots, 0, \dots, r]$.

   (c) Compute the kernel values using 1.4. You may initially compute the values without the normalization term, and normalize the kernel to ensure the sum of all values equals 1.

   Verify that your kernel is correct by plotting x against g as shown in Figure 1.4. A kernel with the derivative of the Gaussian can be created similarly, and you can see the plot in Figure 1.5.

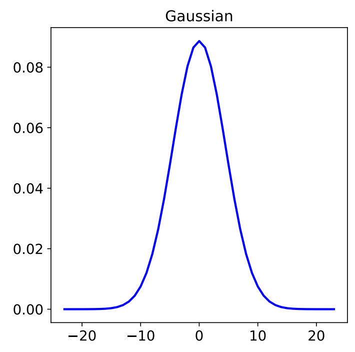

Figure 1.4: Plot of Gaussian with $\sigma = 4.5$.

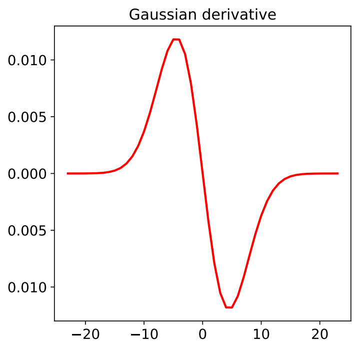

Figure 1.5: Plot of Gaussian derivative with $\sigma = 4.5$.

2. Experimentally verify the separability of the Gaussian kernel. For this, convolve the test image with a 2D Gaussian kernel. Note that you can get a 2D Gaussian kernel as the outer product of two 1D kernels. Then, convolve the same test image with two orthogonal 1D kernels. You should get the same result, i.e. after subtracting the two images, the image difference should be small. Tip: use `'bwr'` colormap to visualize the difference image. Look at the sketch in Figure 1.6 to understand what you are testing.

3. Experimentally verify 1.7. It is enough to test the 1D case. That is, obtain one result by convolving the image with 1D Gaussian, and take the derivative in the same direction. You can compute the derivative by convolving the image with the kernel $[0.5, 0, -0.5]$. Obtain the second result by convolving the image with the derivative of the Gaussian. Verify that the difference between the two results is small. Hint: compute the average absolute difference between the two images.

4. Verify that a convolution with a Gaussian of $t = 20$ is equal to ten convolutions with a Gaussian of $t = 2$. Remember that $\sigma = \sqrt{t}$. You can again compare the two images by showing the difference.

5. Verify that a convolution with a Gaussian derivative of $t = 20$ is equal to convolving with a Gaussian of $t = 10$ and a Gaussian derivative of $t = 10$.

After completing the exercise, you may compare your result with `scipy.ndimage.gaussian_filter`. Read the documentation to understand how the function parameters `order` and `truncate` relate to your implementation.

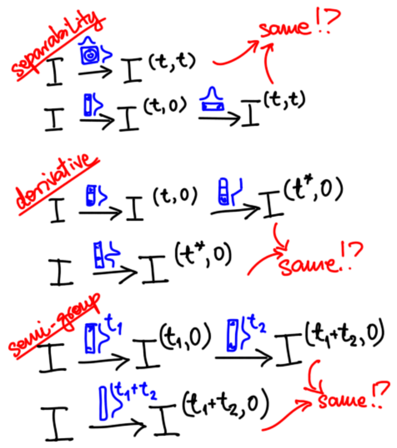
separability
same!?
$I \rightarrow I^{(t,t)}$
$I \rightarrow I^{(t,0)} \rightarrow I^{(t,t)}$
derivative
$I \rightarrow I^{(t,0)} \rightarrow I^{(t^*,0)}$
$I \rightarrow I^{(t^*,0)}$
same!?
semi-group
t_1
t_2
$I \rightarrow I^{(t_1,0)} \rightarrow I^{(t_1+t_2,0)}$
t_1+t_2
$I \rightarrow I^{(t_1+t_2,0)}$
same!?

Figure 1.6: A sketch of the Gaussian properties which you should investigate in this exercise. The arrows symbolize the convolution and the kernel is sketched over the arrow. The superscripts indicate how much smoothing was applied for each dimension, while star indicates derivation.

1.1.2 *Computing length of segmentation boundary*

Segmentation is one of the basic image analysis tasks. The length of the segmentation boundary may be used as a measure of segmentation quality.[4]

[4] This will be useful when we are working with Markov random fields later in the course.

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. The image shown in Figure 1.7 has $n = 3$ values.

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 metric

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

In other words, $L(S)$ counts all occurrences of two neighboring pixels having different labels. Note that every pair of pixels should be counted once. For neighborhood we use 4-connectivity, i.e. we consider only the pixels to the left, right, up, and down.

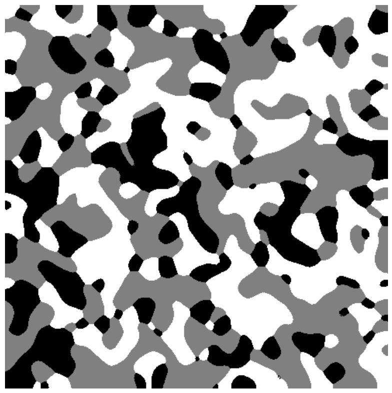

Figure 1.7: Image of a segmented fuel cell with three phases. Black represents air, grey is the cathode, and white is the anode.

Data You may use any image with discrete pixel values. We have provided the segmentation images `fuel_cell_1.tif`, `fuel_cell_2.tif`, and `fuel_cell_3.tif`.

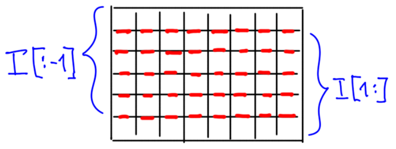

Figure 1.8: A sketch for vectorization when computing the length of the segmentation boundary. We produce two slices (image crops) with the same size but shifted for one row. By performing element-wise comparison, we can check whether the two pixels on each side of the red edge have different values. The sum of all such comparisons gives the length of the x-component of the boundary.

Tasks

1. Compute the length of the segmentation boundary for all images.

2. Try avoiding loops and instead use vectorization provided by `numpy` for an efficient and compact implementation. See Figure 1.8 for a sketch of the vectorization approach.

3. Collect your code in a function that takes segmentation as an input and returns the length of the segmentation boundary as an output.

1.1.3 *Curve smoothing*

A segmentation boundary may be explicitly represented using a sequence of points connected by line segments. Let's assume that an N-times-2 matrix $\mathbf{X}$ contains x and y coordinates of N points, defining a closed curve, often referred to as snake [5].

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

Curve smoothing may be achieved by displacing every curve point towards the average of its two neighbors, possibly iteratively. Point displacement can be considered 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. To displace all points simultaneously, we can use matrix multiplication

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

Here, $\mathbf{L}$ is a N-times-N matrix (often called Laplacian 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 1.9 for an example.

The same matrix may be used for iteratively

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

This approach has two limitations. 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 (oscillations) 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.

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 length minimizing (elasticity) and curvature minimizing (rigidity) 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 [6], section 3.2.4, for the derivation of the kernels.

Smoothing is now obtained as

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

where $\mathbf{A}$ is identical to $\mathbf{L}$ we used before, and $\mathbf{B}$ is similar, but containing values from the other kernel.

Data In this exercise you should use the curves given as text files containing point coordinates `dino_noisy.txt` (shown in Figure 1.10), and `hand_noisy.txt`. We have also provided curves without the noise. The curves may be loaded using `numpy.loadtxt`.

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

Figure 1.9: Matrix $\mathbf{L}$ for N=6. Notice value 1 in the upper right and lower left.

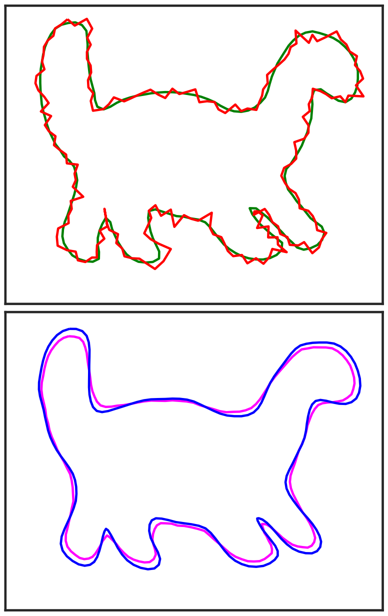

Figure 1.10: Top image shows the dinosaur curve in green, while red shows the curve with added noise. Bottom image shows two smoothing results with different α and β weights.

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

Hints To create matrices as in Figure 1.9, you can use the function `circulant` from `scipy.linalg`. To invert a matrix, you can use `numpy.linalg.inv`.

Tasks

1. Implement curve smoothing as in Equation 1.10. Visually confirm that one step with $\lambda = 0.5$ displaces every curve point exactly to the average of its neighbors. Try iteratively smoothing one of the provided noisy contours. Try visualizing every itereation, see the provided notebook for inspiration. What happens when you increase λ? How many iterations do you need 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 $\alpha\mathbf{A} + \beta\mathbf{B}$. 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 given N, α and β returns $(\mathbf{I} - \alpha\mathbf{A} - \beta\mathbf{B})^{-1}$ to be used for smoothing.[7]

[7] You will be using this when working with deformable models later in the course.

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, and we use 4-neighborhood (up, down, left, right). 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`.

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 1.11. Unwrapping will be useful when we will be working with deformable models later in the course.

Hint There are many functions to choose from, when interpolating image values. If you are in doubt which interpolation to choose, use `scipy.interpolate.RectBivariateSpline`.

Data In this exercise you should use a slice from the provided `dental` folder. The same data is used in the next exercise.

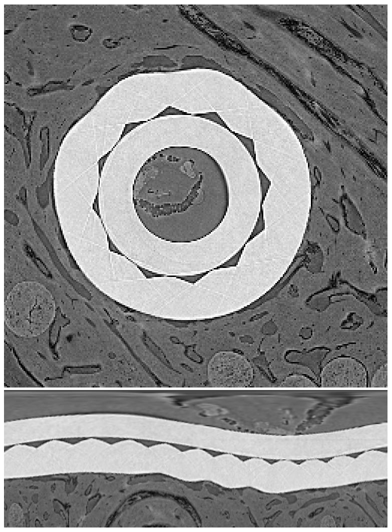
Figure 1.11: Image of a dental implant that should be unwrapped (top) and the unwrapped image (bottom).

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, similar to the top images in Figure 1.11 and 1.12. 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 1.12.

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

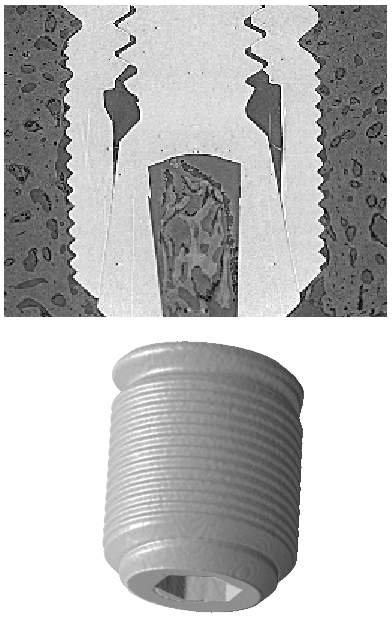
Figure 1.12: A longitudinal slice (*xz*-plane) of the volumetric image of a dental implant and a 3D rendering of the implant thresholded from the volume.

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 to PCA is given in [8], and here we will apply it to a multispectral image.

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

The image is captured using VideometerLab, a device that uses colored LEDs to illuminate samples. This device produces an 18-channel image, where each channel corresponds to a wavelength ranging from 410 nm to 955 nm, covering the visible and near-infrared spectrum.

The image, depicting vegetables on a dish, is shown in false colors in Figure 1.13.

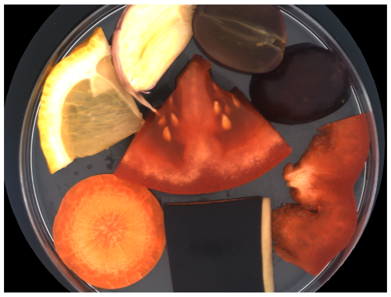
Figure 1.13: False color image obtained from an 18-band VideometerLab image.

The aim of this exercise is to carry out PCA and visualize 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 successive columns corresponding to wavelengths (variables).

The elements of the 18×18 data covariance matrix $\mathbf{C}$ are defined as

$$c_{ij} = \frac{1}{N-1} \sum_{n=1}^{N} (x_{ni} - \mu_i)(x_{nj} - \mu_j), \tag{1.14}$$

where μ_j is an empirical mean for each variable

$$\mu_j = \frac{1}{N} \sum_{n=1}^{N} x_{nj} \,.$$

The 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 $\overline{\mathbf{X}}$ 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 $\overline{\mathbf{X}}\mathbf{v}_i$ 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 You should use the images in the folder `mixed_green`.

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

1. Read 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_j and subtract this from rows of $\mathbf{X}$ 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 components **Q**.
6. Rearrange **Q** into images and display the result.

You can compare your implementation to an already implemented PCA function in 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 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 1.14. Your task is to make a small program that visualizes bacterial growth.

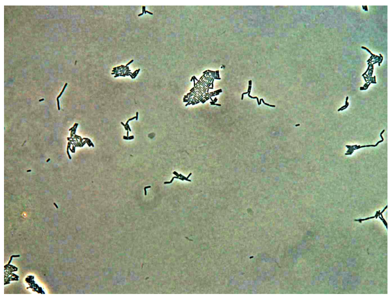
Figure 1.14: Example of microscopic image of listeria bacteria in a petri dish.

The image quality is however not sufficient to distinguish the individual bacteria. So, we make a simplifying 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`.

Hint To read the move, use `imageio` library.

Suggested approach You can first read in one representative frame from the movie and build a 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 gray 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 and plot the number of segmented pixels as a function of time. The obtained curve has a characteristic shape. Can you recognize the function that could describe this shape?