Part IV

Mini projects

11 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].

11.1 Data

The data for the exercise is found in the file called `texture_data.zip` 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. Examples images are shown in Figure 11.1. You are also welcome to find your own data set for the exercise.

11.1.1 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 $\mathcal{L}$.

[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

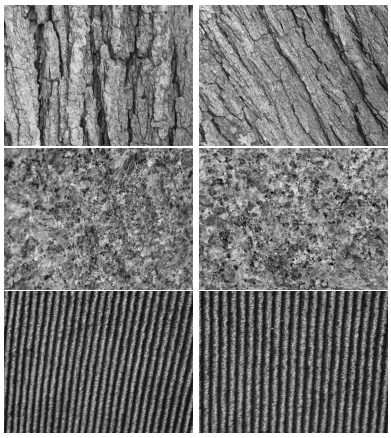


Figure 11.1: Examples of three classes of textured images.

[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 discard 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

$$\mathcal{B}(\mathbf{x}) = \sum_{i=0}^{3} (\mathcal{L}(\mathbf{x}; i) - 1)6^i \, . \tag{11.1}$$

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.

11.2 *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

11.3 *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

12 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

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t), \tag{12.1}$$

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[1] and the Horn-Shunck methods[2] that you will work with in this exercise. This exercise is based on the book Computer Vision: Algorithms and Applications[3], Chapter 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.

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

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

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

12.1 Optical flow

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

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v = -\frac{\partial I}{\partial t}, \tag{12.2}$$

where u and v are displacements in the horizontal and vertical directions respectively. $[u, v]^T$ is the velocity of the flow field.

Task: Derive (12.2) form (12.1) by using a first order Taylor approximation.

In (12.2) 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 (12.2) 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 images of 10×10 pixels called `composedIm_1.png` and `composedIm_2.png` are available in the `optical_flow_data.zip` file 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 shifted in the opposite direction as shown in Figure 12.1. You can use these two images to try simple experiments with optical flow.

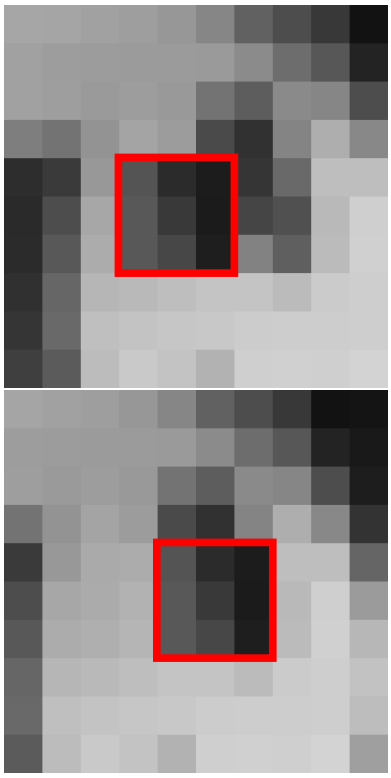


Figure 12.1: Two images of the same patten shifted one pixel to the left, whereas the center part marked in the red box is shifted on pixel to the right.

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

$$\arg\min_{\mathbf{u}} \|\mathbf{Au} - \mathbf{b}\|^2. \tag{12.3}$$

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

$$\mathbf{u} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}. \tag{12.4}$$

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 MATLAB function `quiver`.

12.2 *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 until now 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 the first part of this exercise 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 the first order derivative of a Gaussian.

Task: Test one or more differential filters.

Instead of treating all pixels in the window 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 wight 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.

12.2.1 *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

$$E = \int\int [(I_x u + I_y v + I_t) + \alpha^2(\|\nabla u\|^2 \|\nabla v\|^2)] \mathrm{d}x\mathrm{d}y. \tag{12.5}$$

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

$$u^{k+1} = \bar{u}^k - \frac{I_x(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \tag{12.6}$$

$$v^{k+1} = \bar{v}^k - \frac{I_y(I_x\bar{u}^k + I_y\bar{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2} \tag{12.7}$$

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

$$I_x = \frac{\partial I}{\partial x}\,, \quad I_y = \frac{\partial I}{\partial y}\,.$$

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.

13 Nerve segmentation

This project is on the practical aspects of volumetric segmentation using geometric priors. This is demonstrated on a problem of segmenting myelinated axons from the volumetric data containing scans of the human posterior interosseous nerve.

There are various strategies for solving this problem. Here we demonstrate the use of two approaches we already presented: Markov random fields introduced in Chapter 5 and defromable models introduced in Chapter 6. Furthermore, an approach from Chapter 7 would be very suitable for segmenting nerves (but not explained in this project).

You will be working on the small part of the large data set. For better understanding of the goals of image analysis, we provide background information on the large study involving a complete data set.

13.1 X-ray tomography of human peripheral nerves

Nerve disorders caused by trauma or disease can have serious consequences for the affected people. One aspect of understanding nerve disorders involves a knowledge of the structure of peripheral nerves and their subcomponents. This is typically obtained trough microscopy.

Imaging using conventional light and electron microscopical techniques only allows a two-dimensional visualization of tissues such as peripheral nerves. With recent advances in synchrotron imaging techniques, we can now also obtain detailed three-dimensional images of tissue. This in turn allows extraction of 3D morphological information.

For a larger study[1], biopsies of the posterior interosseous nerve at wrist levels were taken from otherwise healthy subjects and from subjects with type 1 and 2 diabetes. The aim of the study was to determine whether diabetes influences the radius, trajectory and organization of myelinated axons in human peripheral nerves.

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-

[1] Lars B Dahlin, Kristian R Rix, Vedrana A Dahl, Anders B Dahl, Janus N Jensen, Peter Cloetens, Alexandra Pacureanu, Simin Mohseni, Niels OB Thomsen, and Martin Bech. Three-dimensional architecture of human diabetic peripheral nerves revealed by X-ray phase contrast holographic nanotomography. *Scientific reports*, 10(1):1–8, 2020

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 13.1 for a schematic drawing of a nerve) appear circular in the x-y slices through the volume, as shown in Figure 13.2.

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 13.2. 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`.

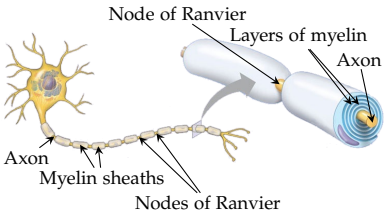


Figure 13.1: A schematic drawing of a nerve showing an axon, myelin sheaths and nodes of Ranvier.

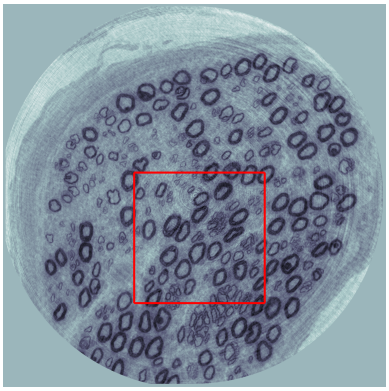


Figure 13.2: One slice from the volume showing peripheral nerves, and a region which was extracted for the exercise.

13.2 *Segmentation of myelinated nerves*

A good contrast between stained myelin sheaths and the background makes it possible to clearly distinguish individual nerve cells 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 cell by aligning a closed curve with the periphery of the myelin. For this, the circle can be manually initialized around the nerve cell, 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. In every slice the surface is to be attracted to the boundary of the myelin layer.

Try segmenting nerves using Markov random fields and deformable models. In Figure 13.3, 13.4, 13.5 and 13.6 we show results of image analysis performed for the original study. However, those results are obtained on a full-resolution volumes, and using a combination of deformable models(Chapter 6) and layered surfaces(Chapter 7). Your results might therefore be of poorer quality.

For this open assignment, we provide some tips, but you are free to investigate other approaches.

- 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

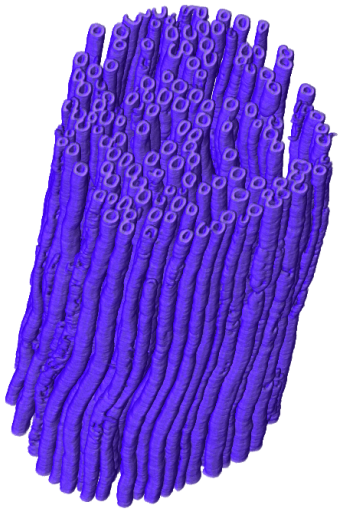


Figure 13.3: 3D visualization of segmented of myelinated nerves.

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 (similar to exercise 1.1.5) 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 13.1 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 13.6. Nodes are of a high interest for understanding nerve disorders. However, those might be challenging to capture due to the lack of myelin.

- Visualizing results in 3D usually provides a useful information on the segmentation results. Visualization options provided by MATLAB and python are good. Still, for large datasets, and advanced visualization you may want to use a specialized software, and we suggest trying ParaView. A few notes on 3D visualization using ParaView be found in ParaView notes.

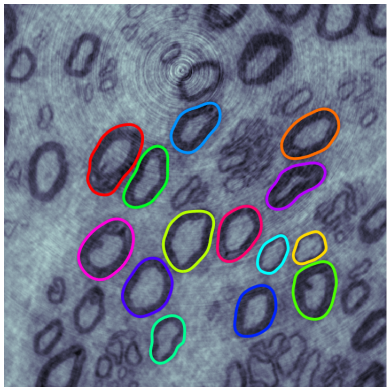


Figure 13.4: Axons segmented using deformable curves visualized on a single slice.

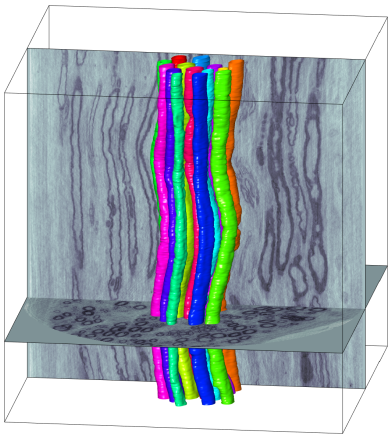


Figure 13.5: 3D visualization of axons segmented using deformable curves.

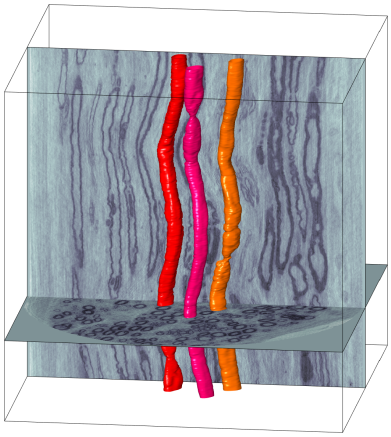


Figure 13.6: Three of the axons with visible node of Ranvier.

Tasks

1. Consider following microstructural measurements which can be extracted from volumetric data:

   Nerve density count: Number of axons per area of nerve-fibre cross-section. Measured in number per area.

   Myelin density: A fraction of nerve cross-section corresponding to myelin. Expressed as dimensionless fraction (a number between 0 and 1), or a percentage.

Average nerve area: Average area of nerves, broken down into average axon area, and average myelin area. This measure is very related to average nerve radius.

Average nerve radius: Average radius of nerves, broken down into average axon radius, and average myelin thickness. This measure is very related to average nerve area.

2. Perform a binary segmentation of the data using MRF. Which microstructural measurement can you extract from your MRF segmentation?

3. Perform a volumetric segmentation of few nerves using using deformable models. Which microstructural measurements can you extract from your segmentation?

14 Spectral segmentation 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 [1]. 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 14.1.

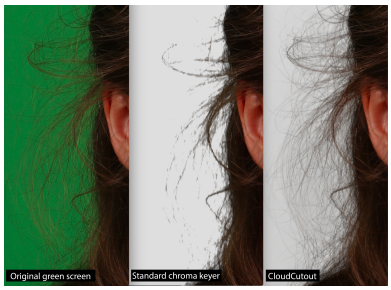


Figure 14.1: Spektral (formerly known as CloudCutout) green-screen product.

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 [2]. We will here briefly cover spectral clustering, and will then turn to its use in image segmentation.

Boiled down to four words, the essence of spectral clustering is: *Eigensolution gives graph partitioning*. To be able to understand spectral

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

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

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.

14.1 *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 14.2 consisting of 12 nodes. This graph can be represented in terms of a 12×12 adjacency matrix, as illustrated in Figure 14.3.

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 14.4 shows three graph cuts, while Figure 14.5 lists the costs associated with the three cuts given by different measures. You may confirm that the values are correct, and notice the different balancing properties of the cost measures.

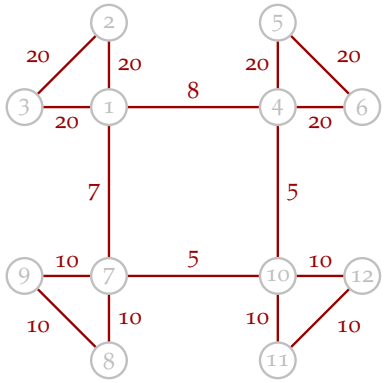


Figure 14.2: An example of an edge-weighted undirected graph.

	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 14.3: An adjacency matrix of a graph shown in Figure 14.2.

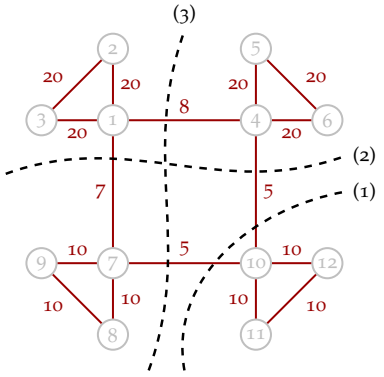


Figure 14.4: Three different partitioning of a graph.

	(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 14.5: Values of the different cuts shown in Figure 14.4.

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 (corresponding to smallest eigenvalues) yield solution relaxed versions of finding the normalized cut. The relaxed solution is subsequently transformed into an approximate discrete solution to the original problem, 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 $\mathbf{A}$.

14.2 *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.

14.3 *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.

15 Probabilistic Chan-Vese

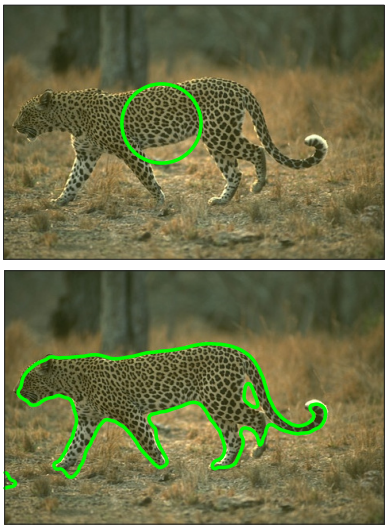


Figure 15.1: A deformable model used for segmenting forward-background image where two regions are characterized by different textures.

Chan-Vese segmentation algorithm alternates between two updates: update for mean intensities of the two regions given region boundaries, and update of the boundary between the two regions given mean intensities. The use of mean intensities implies that the two regions are distinguished by having different mean intensities. There are situations, where this is not the case, see Figure 15.1. Regions can be characterized by distributions of intensities or other features.

In this mini-project you will investigate generalizations of Chan-Vese algorithm, which allow for segmenting more challenging situations than with the original Chan-Vese. Some inspiration can be found in paper by Dahl and Dahl [1], which proposes a intensity-distribution approach Figure 15.2 and patch-distribution approach Figure 15.3. The approach is illustrated in Figure 15.2. Once the curve is initialized, instead of computing mean intensities for the inside and outside region, the distributions of intensities are collected for the inside and the outside region. For every pixel value we now have an information on how often it is in the inside and outside region, which can be translated into a probability that this pixel value is inside or outside. Computing such probabilities for all image pixels leads to probability image which can be used to deform the curve. Alternating, in a Chan-Vese manner, between computing probability image given the curve, and deforming the curve given probability image leads to segmentation.

For even more general case, instead of working with distributions of pixel intensities, distributions of image features may be used to compute the probability image. Figure 15.3 shows an example of using dictionary of image patches. For every patch from the dictionary we can compute the probability of it occurring in the inside or outside region.

[1] Vedrana Andersen Dahl and Anders Bjorholm Dahl. A probabilistic framework for curve evolution. In *Scale Space and Variational Methods in Computer Vision*, pages 421–32. Springer, 2017

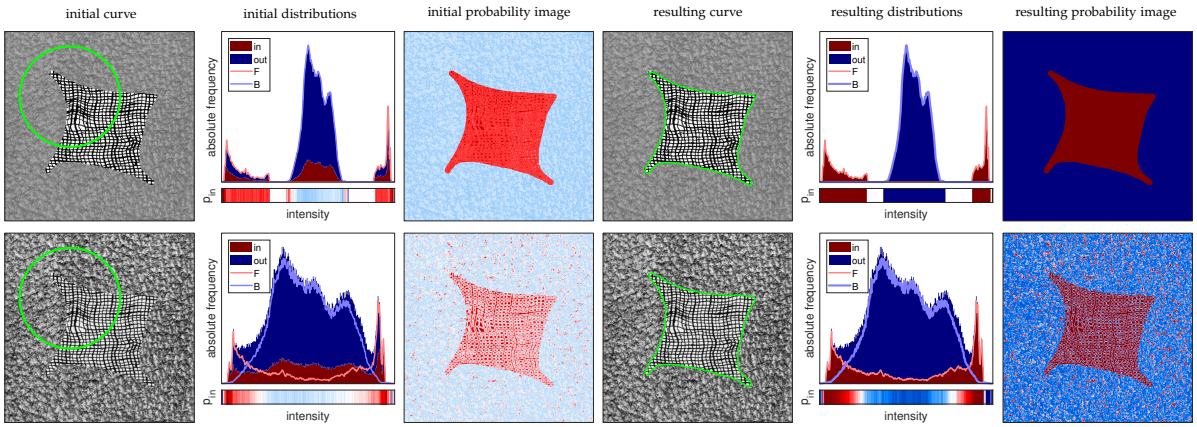


Figure 15.2: A probabilistic Chan-Vese approach when regions inside and outside are characterized by different distributions of pixel intensities. Top row shows easier problem of non-overlapping distributions, bottom row shows two overlapping distributions. Colums 1–3 show initialization, columsn 4–6 show result after iterating.

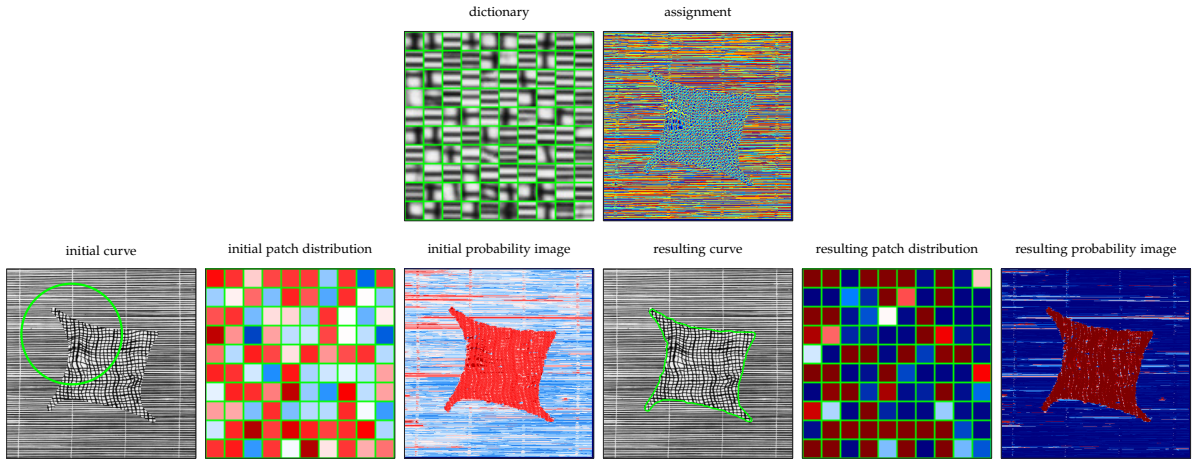


Figure 15.3: A probabilistic Chan-Vese approach when regions inside and outside are characterized by different texture. Top row shows used dictionary of image patches, and an assigmnent of image pixels to dictionary. In the second row, Images 1–3 show initialization and images 4–6 show result after iterating.

16 Learning snake deformation

Snakes, covered in 6, provide a very strong model for segmenting one simple object (foreground) from the background. The deformation forces used to align the snake with the boundary of the object may be defined to solve a special problem. The forces may also be learned, as attempted by a few recent research papers (see also Figure 16.1): Learning active contours, CVPR 2018; Fast Curve, CVPR 2019; Deep Snake, CVPR 2020; Learned Snakes, Signal Processing 2021.

In this project, we can attempt learning snake deformation using either the edge-based or the region-based approach. We could learn deformation from images with available ground-truth labels, for example cells or pets, or we could attempt an approach learning deformation without ground truth.

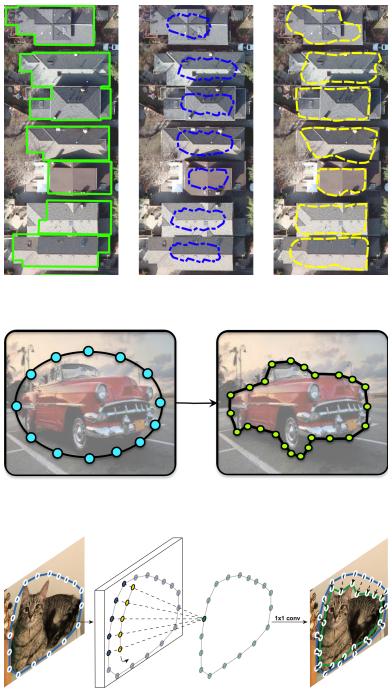


Figure 16.1: Approaches for learning snakes deformation, images taken from recent CVPR papers.

17 Orientation analysis

Analyzing orientations of images structures is often needed if we want to visualize, quantify, or elsewhere utilize orientation information obtainable from images. A common tool for orientation analysis is a structure tensor.

In this mini-project you will be working with structure tensor and orientation analysis. You may decide to focus on computation of structure tensor, visualization of the orientation information, quantification of orientation, or some similar aspect.

17.1 Computating structure tensor

In the context of volumetric (3D) image analysis, a structure tensor is a 3-by-3 matrix which summarizes orientation in a certain neighborhood around a certain point.

For example, consider volumetric data showing a bundle of roughly parallel fibers. If we extract two cubes from this volume, mutually displaced along the predominant fiber orientation, the two cubes will have very similar intensities. For two cubes displaced along other orientations the cube intensities would be more different. For this reason, measuring the change of intensities between slightly displaced cubes may be used for determining predominant orientation of imaged structures.

It turns out that, given an initial point and a size of the cube, squared change of intensities may be expressed as $\mathbf{u}^T \mathbf{S}\, \mathbf{u}$, where $\mathbf{u}$ is the direction of the displacement and $\mathbf{S}$ is 3-by-3 symmetric positive semi-definite matrix – a structure tensor. Finding predominant orientation now amounts to finding $\mathbf{u}$ which minimizes $\mathbf{u}^T \mathbf{S}\, \mathbf{u}$.

For a more formal derivation, consider a volumetric data V defined on the domain $\Omega \subset \mathbb{R}^3$, where $V(\mathbf{p})$ denotes voxel intensity at the point $\mathbf{p} = [p_x \quad p_y \quad p_z]^T$. Consider an arbitrary but fixed neighborhood N around a point $\mathbf{p}$, such that $N(\mathbf{p}) \subset \Omega$. We want to measure

$$D = \sum_{\mathbf{p}' \in N(\mathbf{p})} \left(V(\mathbf{p}' + \mathbf{u}) - V(\mathbf{p}') \right)^2 .$$

Assuming a small displacement we use first order Taylor expansion and arrive to

$$D = \sum_{\mathbf{p}' \in N(\mathbf{p})} \left(\begin{bmatrix} V_x(\mathbf{p}') & V_y(\mathbf{p}') & V_z(\mathbf{p}') \end{bmatrix} \mathbf{u} \right)^2$$

where we use notation $V_x = \frac{\partial V}{\partial x}$, and correspondingly for partial derivatives in y and z direction. Finally, exploiting commutativity of the inner product leads to

$$D = \mathbf{u}^T \sum_{\mathbf{p}' \in N(\mathbf{p})} \begin{bmatrix} V_x(\mathbf{p}') \\ V_y(\mathbf{p}') \\ V_z(\mathbf{p}') \end{bmatrix} \begin{bmatrix} V_x(\mathbf{p}') & V_y(\mathbf{p}') & V_z(\mathbf{p}') \end{bmatrix} \mathbf{u} \,.$$

So, as earlier claimed, we arrived to expression $D = \mathbf{u}^T \mathbf{S}\, \mathbf{u}$, where $\mathbf{S}$ is a 3-by-3 matrix – a structure tensor computed in a point $\mathbf{p}$ and using a neighborhood N. That $\mathbf{S}$ is symmetric and positive semi-definite follows directly from the construction of $\mathbf{S}$ (note that $D \geq 0$).

Using a compact notation for gradient $\nabla V = \begin{bmatrix} V_x & V_y & V_z \end{bmatrix}^T$, structure tensor is

$$\mathbf{S} = \sum \nabla V \, (\nabla V)^T \,.$$

Her we imply that structure tensor is computed for every voxel of the volume, i.e. structure tensor is a matrix-valued function over Ω. The summation is conducted over a neighborhood of each voxel, and result will be the same (up to the multiplicative factor) if summation is replaced by an averaging filter.

Two Gaussian filters are usually involved in computing structure tensor, see [1] for detailed description of 2D case. The one Gaussian filter has to do with averaging orientation information in the neighborhood. This can be achieved using a convolution with a Gaussian K_ρ, where parameter ρ, called *integration scale*, reflects the size of the neighborhood. Now we have

$$\mathbf{S} = K_\rho * \left(\nabla V \, (\nabla V)^T \right) .$$

The second Gaussian has to do with computing partial derivatives in gradient ∇V. To make differentiation less sensitive to noise we may convolve the volume with a Gaussian prior to computing derivatives. More efficiently, utilizing derivative theorem of convolution, partial derivatives can be computed by convolving with derivatives of Gaussian. We denote such gradient $\nabla_\sigma V$. The parameter σ is called *noise scale*. Expression with both Gaussians is

$$\mathbf{S} = K_\rho * \left(\nabla V_\sigma \, (\nabla V_\sigma)^T \right) .$$

In summary, computing structure tensor for each voxel of a volume V involves three steps:

[1] Joachim Weickert. *Anisotropic diffusion in image processing*, volume 1. Teubner Stuttgart, 1998. URL https://www.mia.uni-saarland.de/weickert/Papers/book.pdf

1. Convolve V with derivatives of Gaussian with standard deviation σ to obtain V_x, V_y and V_z. For efficiency, use separability of Gaussian kernel.

2. Using element-wise multiplication compute six volumes V_x^2, V_y^2, V_z^2, V_xV_y, V_xV_z and V_yV_z.

3. Convolve each of the six volumes with the Gaussian kernel with standard deviation ρ. For efficiency, use separability of Gaussian kernel. The resulting volumes contain per-voxel elements s_{xx}, s_{yy}, s_{zz}, s_{xy}, s_{xz} and s_{yz} of the structure tensor

$$\mathbf{S} = \begin{bmatrix} s_{xx} & s_{xy} & s_{xz} \\ s_{xy} & s_{yy} & s_{yz} \\ s_{xz} & s_{yz} & s_{zz} \end{bmatrix} .$$

17.2 *Computing orientations*

Given structure tensor $\mathbf{S}$, predominant orientation is found by minimizing Rayleigh coefficient $\mathbf{u}^T\mathbf{S}\,\mathbf{u}$ trough eigendecomposition of $\mathbf{S}$. Being symmetric and positive semi-definite $\mathbf{S}$ yields three positive eigenvalues $\lambda_1 \leq \lambda_2 \leq \lambda_3$ and mutually orthogonal eigenvectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$. The eigenvector $\mathbf{v}_1$ corresponding to the smallest eigenvalue is an orientation leading to the smallest variation in intensities, which indicates a predominant orientation in the volume. Note that $\mathbf{v}_1$ is an orientation, and we usually represent it using an unit vector, but this is still not an unique representation since there are two opposite unit vectors sharing the orientation with $\mathbf{v}_1$.

Eigendecomposition of a 3-by-3 real symmetric matrix can be computed efficiently using an analytic approach by Smith [2] which uses an affine transformation and a trigonometric solution of a third order polynomial.

If there is no strong orientation in the volume, all eigenvalues will be similar, and dominant direction will be influenced by small local variations or the noise in the data. For this reason it is customary to analyze the ratio between eigenvalues to determine the degree of anisotropy in the neighborhood, and how (locally) line-like or plane-like the imaged structure is, see illustration 17.1. Inspired by diffusion tensor processing [3], we define values, so-called shape measures,

$$c_l = \frac{\lambda_2 - \lambda_1}{\lambda_3} , \quad c_p = \frac{\lambda_3 - \lambda_2}{\lambda_3} , \quad c_s = \frac{\lambda_1}{\lambda_3}$$

where c_l gives a measure of linearity, while c_p and c_s measure planarity and sphericity. Shape values are positive and sum to 1.

In summary, structure tensor is a 3-by-3 matrix that can be computed in each volume voxel. The computation of structure tensor requires

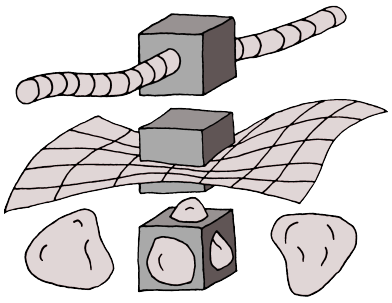


Figure 17.1: Neighborhoods and structures corresponding to linear, planar and spherical shape. On a linear structure *(top)* cubical neighborhood can move along the predominant direction leading to small change in intensities, while other two orthogonal directions lead to significantly larger and approximately equal change. On a planar structure *(middle)* two directions lead to small and approximately equal change in intensities, while third direction leads to significantly larger change. For a case with no predominant direction *(bottom)* the three orthogonal directions lead to roughly equal changes in intensities.

[2] Oliver K Smith. Eigenvalues of a symmetric 3× 3 matrix. *Communications of the ACM*, 4(4):168, 1961. URL https://dl.acm.org/citation.cfm?id=366316

[3] C-F Westin, Stephan E Maier, Hatsuho Mamata, Arya Nabavi, Ferenc A Jolesz, and Ron Kikinis. Processing and visualization for diffusion tensor MRI. *Medical image analysis*, 6(2):93–108, 2002

two parameters: noise scale σ and integration scale ρ. Being symmetric, structure tensor can be represented with 6 scalar values. The most important information extracted from structure tensor is a dominant orientation. Dominant orientation is a unit vector with equivalence relation $-\mathbf{v} \equiv \mathbf{v}$. Shape measures, also extracted from structure tensor, may also be of interest. Shape measures are three scalar ,c_l, c_p and c_s, summing to 1.

17.3 *Visualization*

Having extracted structure tensor and performed its eigendecomposition, following values are available for every volume voxel.

- Voxel intensity V, a scalar value in a certain range.
- Shape measures c_l, c_p and c_s, three scalar values summing to 1.
- Dominant orientation $\mathbf{v_1}$, an orientation vector (unit vector with equivalence relation $-\mathbf{v} \equiv \mathbf{v}$).
- Other information, such as $\mathbf{v_2}$, $\mathbf{v_3}$, and the values λ_1, λ_2 and λ_3 is also available, but usually of no special interest.

Visualizing this information often requires some care.

Shape measures, being three scalar values can be conveniently represented using three RGB color channels. Voxels with large c_l will appear red, large c_p will be green, and large c_s blue. This is the approach used in Figure 17.2.

Predominant orientation, is a 3D unit vector with equivalence relation $-\mathbf{v} \equiv \mathbf{v}$. A common way of visualizing orientation is to use absolute values of vector coordinates, i.e. $|v_x|$, $|v_y|$, and $|v_z|$, as three RGB color channels. Orientations roughly aligned with x direction will be red, those aligned with y green, and z blue. This has a desirable property that$-\mathbf{v}$ and $\mathbf{v}$ map to the same color. Furthermore, when the imaged object has a certain geometry aligned with the coordinate system (for example, elongated object aligned with z axis), this coloring scheme may be favorable for the interpretation of orientations. Undesirable property of the RBG color scheme is that different orientations map to the same color, for example four orientations corresponding to main diagonals in unit cube all map to gray.

Shape measure and predominant orientation are computed for every voxel in the volume – regardless of whether the voxel is within the object or material which we investigate. When using volume rendering to visualize the extracted measures, if the value is shown in every voxel, the valuable information might be occluded. For this reason it might be beneficiary to combine visualization of extracted measures

with the intensity values, which carry information on voxels containing, or not containing, material. A visually pleasing result is obtained if voxels containing no material are shown transparent. Furthermore, predominant orientation is only relevant for voxels exhibiting high linearity, so shape measure may be combined with visualization of predominant orientation. This is the approach used in Figure 17.2.

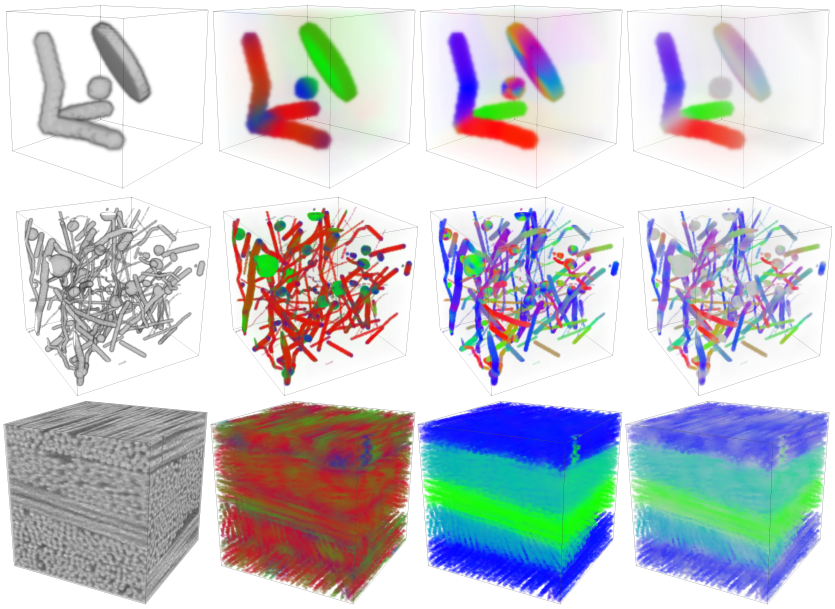


Figure 17.2: Orientation information. In each row, from left to right: 3D rendering of the volumetric data, shape measures visualized in colors, predominant orientation in the material phase, visualized in colors, predominant orientation weighed by the measure of linearity. Top two rows show the result on synthetic data, while in the bottom row we see the result of orientation analysis for composite material.

17.4 *Applications*

Some uses of orientation information are:

- Volumetric visualization of the shape measures and/or the predominant orientation.
- Producing histograms of orientations by binning orientation vectors. These distributions live on a half sphere, which can complicate the visualization, comparison, and fitting of distributions.
- Comparing orientation information extracted form volumetric data with, for example, the orientations obtained via modeling and/or simulation.
- Using local orientation to segment the volume into regions of constant orientation.
- Using local orientation to tune smoothness constraint in MRF segmentation.
- Using local orientation to guide fibre tracking.

18 CNN for segmentation

Image segmentation is often needed as an intermediate step when quantifying structures in images, and we have previously been working with patch-based segmentation. In this exercise you should train a convolutional neural network to segment electron microscopy images of neuronal structures. This data was part of the ISBI Challenge: *Segmentation of neuronal structures in EM stacks*[1], and can be found in the file `EM_ISBI_Challenge.zip`.

The data is from a serial section Transmission Electron Microscope and depicts the ventral nerve of a Drosophila larva. An example image is shown in Figure 18.1. The task is to segment the membranes between cells. These appear mostly darker than the rest of the image, but they are often thin and smeared out, and there are other dark regions that are not membranes, but structures such as mitochondria, that should be part of the cell class. Therefore, this segmentation problem is difficult and requires either a biologist that knows the anatomy or an advanced automated image segmentation method. You should aim at building the automated segmentation method.

The data consists of 30 images with associated labels and additional 30 test images without labels. All images are 512×512 pixels. The task is to build and train a neural network that can segment this type of images. Normally, you would split your data into a training, validation and test set. You would use the training set for learning the model parameters and the validation set to ensure generalization of the model, e.g. that it does not overfit to the training set. In an ideal performance assessment, you will only use the test set once to measure the actual performance of your method. When the same test set is used multiple times, you would optimize for precisely that test set, which will bias your performance assessment.

The problem here is that you only have 30 images with labels to train, validate, and test your algorithm, if it should be done using quantitative measures. The test set does not come with the ground truth labels, since it was kept for evaluating performance of algorithms at the ISBI Challenge, and therefore the images in the test set only allows you to

[1] Ignacio Arganda-Carreras, Srinivas C Turaga, Daniel R Berger, Dan Cireşan, Alessandro Giusti, Luca M Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M Buhmann, et al. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in neuroanatomy*, 9:142, 2015

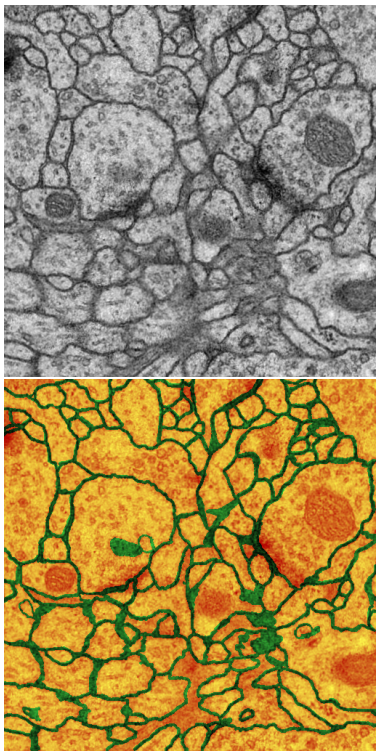


Figure 18.1: Example of EM data for segmentation. Top image is the original EM slice and the bottom image is the labels overlaied.

do a qualitative evaluation. It is your task to choose the images for training, validation, and test, and you should argue for your choice.

You can find inspiration for building your segmentation method in other peoples work. The ISBI challenge data set has e.g. been used in the U-net paper[2], which is a very successful model for segmentation based on deep learning. This network uses four down-sampling steps followed by four up-sampling steps, and at each resolution there more convolutions and activation steps. In addition there are so-called skip connections that connects layers at different scales. This architecture can be drawn in an u-shape, which has given the name to the method. Furthermore, the paper describes data augmentation suited for this type of data, so you might use the paper to get inspiration for your solution.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015

18.1 Suggestions

You are suggested to implement your model using either the Deep Learning Toolbox from MATLAB or Pytorch or similar for Python. Furthermore, to avoid spending too much time in the initial training, it can be a good idea to start with smaller images than the 512×512 pixels. You can e.g. split the images into smaller patches and work from these. Then you can at a later stage, when you are sure that your model is working as expected, increase the size of your images.

It is also a good idea to start with a simple model, where it is easy to ensure that all steps are working as they should. Then you can gradually increase the complexity.

Besides training the deep learning model and setting it up such that it works as expected, there are many choices to be made for building your model and for optimizing it. You are welcome to try out different approaches and investigate the effect of e.g. data augmentation. It is a very good idea to be systematic and show the effect of different parameter choices – either using quantitative and qualitative evaluation measures.

19 Superresolution from line scans

Scanning along a set of parallel lines is a common setting in medical imaging. For example, consider optical coherence tomography (OCT), well established in ophthalmology for obtaining images of the retina. Using OCT, the retina is scanned in along a line with a high transversal resolution. Collecting a number of scans along parallel lines, a larger area of the retina may be covered. Since scanning speed of the OCT systems employed in clinic is limited, and prolonged scanning is unpleasant for the patient, the distance between the parallel lines is often large compared to the transverse resolution of the scans. Therefore, the resolution of the scan is much coarser in one direction. In other words, each pixel covers a non-square area. Elongated pixels appear as stripes and influence the visual appearance of the image. The stripes can disturb the interpretation of the image and make it difficult to distinguish the anatomical structures, especially evident with blood vessels running parallel to scan lines.

To reveal additional anatomical structures, another OCT scan may be performed, along the lines orthogonal to the first scan, as a pari of images shown in Figure 19.1. Several problems emerge in connection to this. The eye might move during scanning, and the intensity might vary significantly between the scans. And most importantly, how to combine two scans covering the same area, one with high resolution in x direction, and the other in y, such that the resulting image has a satisfactory quality? To simplify the problem, we consider a set up as in Figure 19.2, where pixels of unknown values are to be estimated from the pixels of known values.

This problem is very similar to single-image superresolution, which can be obtained with great quality using neural networks. In this mini-project, you can try using neural network for upsampling line scans. The project involves setting up a framework based on the publicly available frameworks. We would expect the performance to improve significantly if prior knowledge about the appearance of the images is incorporated in the method. The training should therefore be performed on images having similar appearance as OCT image, but

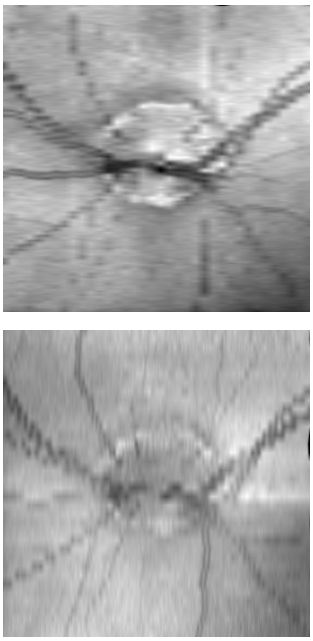


Figure 19.1: Example of images obtained using line scans in orthogonal directions. Notice that vertical lines are less clear in the first image, while horizontal lines are unclear in the second.

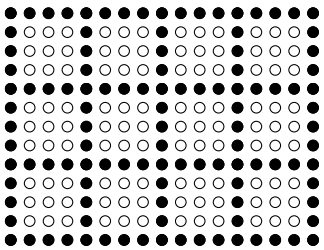


Figure 19.2: An image with pixels divided in pixels with known intensity (black), and pixels with unknown intensity (white), here shown with an upsampling rate of 4.

that can be images of vasculature obtained using other modalities. We will provide you such images.

To evaluate the performance of the upsampling, the conventional approach is to use peak signal-to-noise ratio (PSNR) metric. Furthermore, a comparison with the simple base-line upsampling scheme would be nice. For example, consider a schme where each direction is linearly upsampled, and the two contributions are combined as an average. Such an approach is show in Figure 19.3.

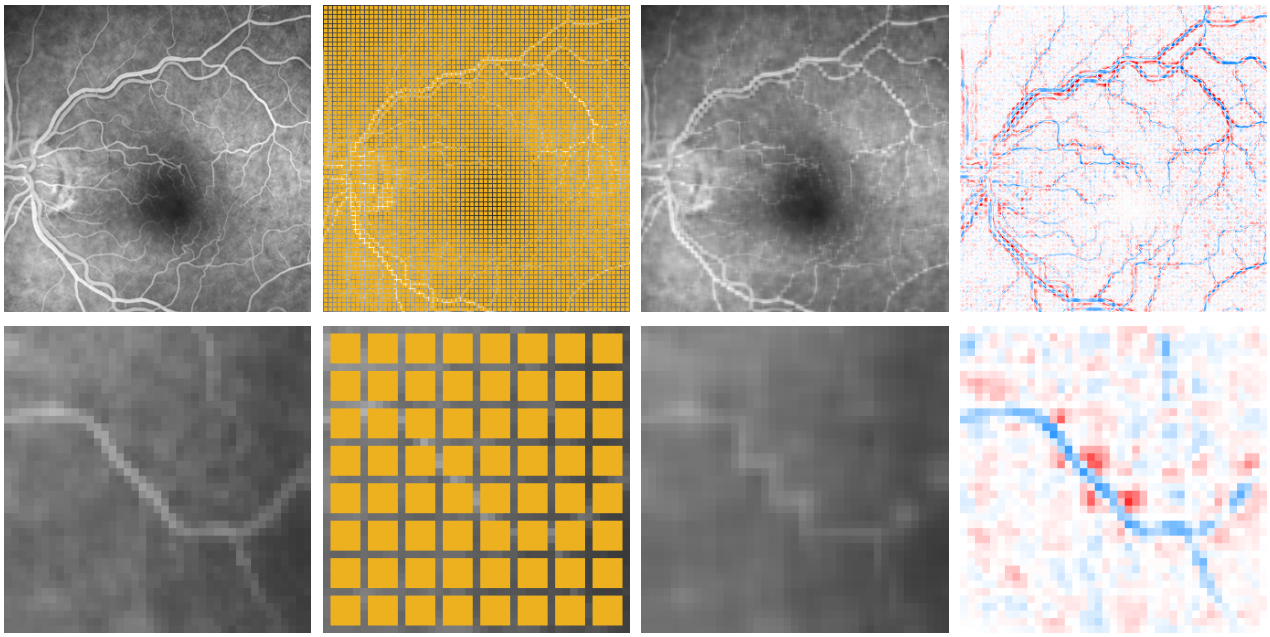
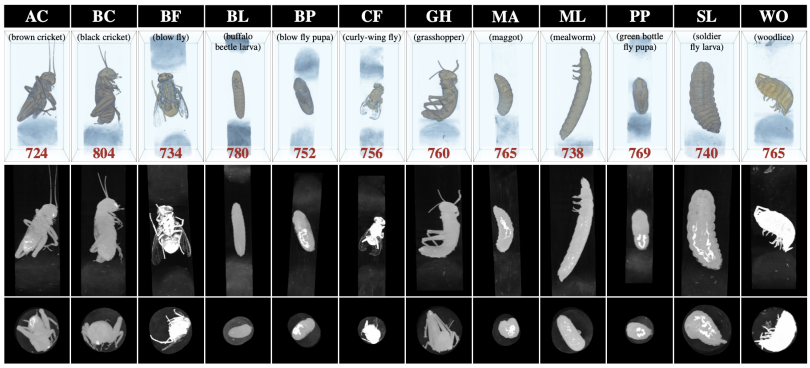


Figure 19.3: A simple linear upsampling scheme. First column: ground truth; second column: unknown pixels masked; third column: upsampling result; fourth column: the error, with color indicating the sign. Top row: image of vasculature; bottom row: zoom in on a small part of the image.

20 BugNIST analysis

BugNIST is a dataset that we have made to promote the development of new deep learning algorithms for volumetric images[1]. The dataset contains 9437 volumetric micro-CT scans of 12 types of bugs. 9087 of these volumes are individual scans and and 350 are mixtures, where several bugs are in the same volume. You can see examples of the data in Figure 20.1.



[1] Anders Bjorholm Dahl, Patrick Møller Jensen, Carsten Gundlach, Rebecca Engberg, Hans Martin Kjer, and Vedrana Andersen Dahl. Bugnist–a new large scale volumetric 3d image dataset for classification and detection. *arXiv preprint arXiv:2304.01838*, 2023

Figure 20.1: Examples of individual scans of the 12 types of bugs in the BugNIST dataset.

While preparing the data, we have evaluated 3D classification algorithms on the scans of individual bugs, and we obtained very high classification rates. The two groups of crickets (black and brown) are, however, difficult to separate. But in general it is an easy classification problem, and we would like to understand what makes the classification so easy.

The scans of mixed bugs are made to test image detection algorithms. Examples are shown in Figure 20.2. The idea is to train a detection algorithm on the scans of individual bugs and test them on the mixtures. This is a very difficult problem, since the individual scans of bugs are in another context than the rest. To evaluate the detection, we have marked the center position of the bug and its type.

The scanning was done by placing the bugs in plastic straws and test tubes, where the bugs are separated using cotton. The cotton is visible in the scans, and it would good to clean it away form the scans of individual bugs such that the images only contain bugs. The challenge

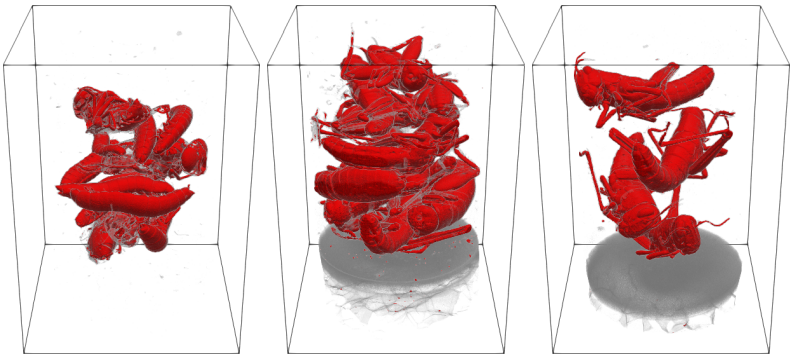


Figure 20.2: Examples of mixed scans in the BugNIST dataset.

is that some parts of the bugs such as wings of the flies have an intensity similar to the cotton. It is also difficult to preserve thin structures such as antennas or legs. So, the task of removing the cotton is not trivial.

20.1 *Suggestions*

There are four suggested tasks for the BugNIST data. These are

1. Volumetric classification
2. Volumetric detection
3. Segmentation of bugs from background and remove cotton
4. Surface generation

Volumetric classification should be done using deep learning classification methods of the volumes. Since our initial investigation shows good results when training on a large part of the bugs, it would be interesting to see how few samples is needed for training a good model and which models will work well with few examples. It is also interesting to investigate the effect of downscaling and find out how small volumes it is possible to classify. Finally, it is interesting to see what leads to the good performance using for example gradient activation maps (Grad-CAM and similar).

Volumetric detection should be done by training a detection algorithm on scans of individual bugs and testing the ability to detect bugs in the mixtures. This includes the ability to identify where a bug is and what species it is. In our initial studies, we did not obtain good results and found the task to be difficult. We expect that the task is difficult because of the change in domain from individual scans, where the context around the bugs are much different form the mixtures. So, there is potentially much to gain, if you can come up with a good solution. We have annotated the data by marking the center position

of the bugs and their species, but it would be up to you to come up with a good measure of detection. One approach would be to train a U-net to segment the bugs in the mixtures and then process the results. Training data for the U-net is easily obtained from the individually scanned bugs.

Segmentation of bugs from background and remove cotton should be done to clean the data and make it better suited for training detection and classification models. Here, it is up to you to come up with an approach.

Surface generation is to create surface meshes of the data. This can be done by segmenting the bugs from the background and and using the marching cubes algorithm to generate a mesh followed by a mesh cleaning step. Here it will be difficult to ensure that thin structures such as antennas and legs are well preserved and that the complex structure of the bugs are preserved. But a 3D surface dataset of all the scanned bugs would be of great value.