UiO : Department of Mathematics
University of Oslo

A Pipeline for Extraction of Patient-Specific Geometries with Machine Learning

Per Magne Florvaag
Master's Thesis, Spring 2018

Abstract

Modeling of the blood flow in and around aneurysms with computational fluid dynamics (CFD) is important to better understand why aneurysms form and rupture. CFD modeling requires an accurate representation of the patient-specific arteries for simulations to be reproducible and reflect the reality. State-of-the-art methods use semi-manual tools to extract patient-specific geometries, which result in inconsistent results and a lot of tedious work. This limits the potential clinical impact of CFD-based aneurysm modeling. In this thesis, we develop an automated pipeline for extracting consistent patient-specific geometries. The pipeline consists of two parts: 1) Image restoration based on dictionary learning, and 2) vessel extraction by multiscale segmentation techniques. We show that dictionary learning based methods are able to restore (denoise and inpaint) 3D computed tomography (CT) images, and multiscale segmentation techniques can accurately extract both small and large arteries. Finally, we summarize the proposed pipeline and show its efficiency on a number of 3D CT images from the Aneurisk Project. The suggested pipeline is provided as a ready-to-use python library.

Acknowledgments

First and foremost, I would like to thank my supervisors Dr. Kristian Valen-Sendstad, Dr. Valeriya Naumova, and Aslak W. Bergersen, for your guidance, ideas and time during my work on this thesis. I would also like to thank Simula Research Laboratory for allowing me to work on such an interesting project.

Finally, I would like to express my gratitude to my family and friends for your support along the way.

Per Magne Florvaag
Oslo, April 2018

Contents

Notation

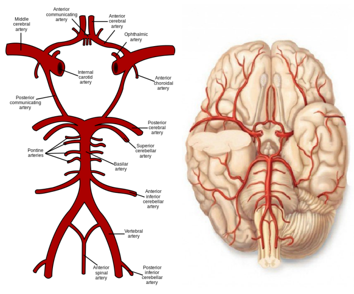
$\mathbf{A}$	A matrix
$\mathbf{a}$	A vector
a_i	Element i of vector $\mathbf{a}$
$\mathbf{a}_i$	Column i of matrix $\mathbf{A}$
$\mathbf{A}_{ij}$ or a_{ij}	Element (i,j) of $n \times m$ matrix $\mathbf{A}$
$\mathbf{A}_I$	Matrix $\mathbf{A}$ restricted to columns $I = [i_1, i_2, .., i_n]$
$\mathbf{A}^\dagger$	Moore-Penrose pseudo inverse of $\mathbf{A}$. $\mathbf{A}^\dagger = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$
$\mathbf{A} \odot \mathbf{B}$	Hadamard product of $\mathbf{A}$ and $\mathbf{B}$
$\|\mathbf{a}\|_0$	ℓ_0-penalty of $\mathbf{a} = [a_1, a_2, ..., a_n]$, $\|\mathbf{a}\|_0 = \#\{i : \ a_i \neq 0\}$
$\|\mathbf{a}\|_p$	ℓ_p-norm of $\mathbf{a}$, $\|\mathbf{a}\|_p = \left(\sum_i \|a_i\|^p\right)^{1/p}$
$\|\mathbf{A}\|_F$	Frobenius norm of $\mathbf{A}$, $\|\mathbf{A}\|_F = \left(\sum_{ij} a_{ij}^2\right)^{1/2}$
$\mathbb{I}_n$	Identity matrix of size $n \times n$

Chapter 1

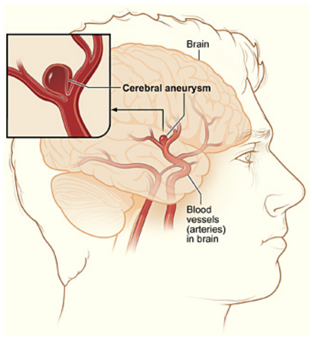
Introduction and Medical Background

In this thesis, we are interested in automatically extracting patient-specific models of arteries from *Computed Tomography* (CT) images of patients with cerebral aneurysms. Currently, the extraction of geometries is performed (semi-)manually, leading to inconsistent results [77], and tedious work. An automated tool for geometry extraction will allow engineers to speed up their preprocessing work and more importantly, reduce interlaboratory differences.

Cerebral aneurysms are balloon-like features on artery walls, most often found in bifurcations around the Circle of Willis, see Figure 1.1a. It is estimated that 5% of the population harbor at least one cerebral aneurysm with an annual rupture risk of about 0.2% [68]. A ruptured aneurysm results in subarachnoid hemorrhage (SAH), a type of stroke caused by bleeding into the subarachnoid space. SAH accounts for approximately 5% of all strokes and carries with it a 50% mortality rate and among survivors a 50% morbidity rate [24, 42]. Because SAH occurs at a young age and has a high mortality rate, the loss of productive years in the general population is as large as that from the most common type of stroke [34].


Anterior communicating artery
Anterior cerebral artery
Middle cerebral artery
Ophthalmic artery
Internal carotid artery
Anterior choroidal artery
Posterior communicating artery
Posterior cerebral artery
Superior cerebellar artery
Pontine arteries
Basilar artery
Anterior inferior cerebellar artery
Vertebral artery
Anterior spinal artery
Posterior inferior cerebellar artery

(a) Circle of Willis



(b) Cerebral Aneurysm

Figure 1.1

Aneurysms are treatable, but its treatment is not without risk. Therefore, identifying factors for aneurysm rupture are important to more confidently decide if a patient should undergo treatment [33]. Morphological markers such as size, irregular shape, size of parent artery, bifurcation angle, and more, are of interest for understanding the risk of aneurysm rupture [68, 25, 44, 69]. These markers will affect the blood flow through factors such as pressure, wall shear stress, flow impingement, etc., which are expected to play a role in the pathogenesis[1] of aneurysms [14, 68]. In the field of *Computational Fluid Dynamics* (CFD), there is great interest in studying aneurysms. Modeling of blood flow is important to better understand and predict the risk of aneurysm rupture. However, one of the bottlenecks of CFD modeling is the need for accurate and efficient extraction of patient-specific geometries [14, 8, 64]. A patient-specific geometry is an accurate model of the underlying anatomy of interest, i.e, the arteries and aneurysm sac. Recent studies indicate that manual extraction of geometries is one of the main causes for inconsistent results [77].

Motivated by this challenge, this thesis aims to efficiently and consistently extract accurate models of the underlying anatomy. The current pipeline starts with taking an image of the patients head using contrast-enhanced CT, called CT angiography (CTA). During CTA imaging contrast fluid is injected into the patient's bloodstream. The contrast fluid will absorb more radiation than other tissue and the arteries will become brighter and more visible.

The next, and challenging, step is to extract the arteries from the CTA images. The challenge lies with the extraction being done consistently and

[1]The manner of development of a disease.

efficiently. The current approach is to manually extract the arteries with the help of tools like *the Vascular Modeling Toolkit* (VMTK) [4], *the Visualization Toolkit* (VTK) [72] and *Insight Segmentation and Registration Toolkit* (ITK) [43]. This approach is slow and prone to inconsistencies [77]. Therefore, current state-of-the-art CFD models provides inconsistent results for the same CT images. This hinders the potential clinical impact of the field, as clinical deployment requires large-scale and consistent studies.

Goals

The goal of this thesis is to develop a pipeline for automatically extracting patient-specific geometries from CTA images, with the help of machine learning techniques. This will enable faster, and reproducible extraction of patient-specific models for use in CFD modeling. Although we focus on aneurysms, the developed pipeline can in future work be extended to any field requiring models of vessel-like structures.

In addition to this thesis, we provide a python library called `dictlearn`[2] that contains algorithms and methods discussed in later chapters. The library also includes the final pipeline described in Chapter 5, which can easily be used and extended with other specialized methods to better fit the needs at hand.

Structure of this work

In Chapter 2 and 3, we introduce theory and methods for image restoration based on sparse image representation. In particular, in Chapter 2, we provide an overview of the development in sparse representation and sparse signal recovery. We define concepts such as, sparse coding, overcomplete signal representation, and how to solve its corresponding optimization problems. In Chapter 3, we introduce methods for adapting and extending the techniques presented in Chapter 2, for reconstruction of 3D CTA images. Chapter 4, explores different ways of consistently extracting arteries from CT images. Finally, in Chapter 5, we summarize the final pipeline together with its results.

[2]Found at `https://github.com/permfl/dictlearn`

1.1 Digital Image Representation

Since we are interested in analyzing 3D CTA image, we first introduce some terminology before we can properly define our tasks.

A 2D grayscale image is represented as a matrix $\mathbf{I} \in \mathbb{R}^{H \times W}$, where the number of rows H is the height of the image and the number of columns W is its width. Further, the element $\mathbf{I}_{ij}$ is a pixel and its value is called the intensity. For 2D color images this is similar, usually we define it as $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$. This corresponds to a *red, green and blue* (RBG) image and each pixel is a triple $\mathbf{I}_{i,j} = (r, g, b)$, which defines how much of the colors red, green, and blue are used to paint this pixel. This representation directly extends to 3D image volumes. A 3D image is defined as $\mathbf{I} \in \mathbb{R}^{H \times W \times D}$, where we have height and width as before and the extra dimension D denotes the depth. The points in a 3D image are given by three coordinates, (i, j, k), and the value at $\mathbf{I}_{ijk}$ is the intensity.

The methods we introduce in the next chapters are designed to process vectorized signals, $\mathbf{x} \in \mathbb{R}^n$. We therefore need to transform the image representation defined above to fit this form. To achieve this, we use *vectorized image patches*. For a 2D image, an image patch is a small rectangle $\mathbf{p} \in \mathbb{R}^{a \times b}$ with $a \cdot b = n$, extracted from some location (i, j) in the image. Additionally, we assume that $a \ll H$ and $b \ll W$ such that an image consists of many patches. If not explicitly stated otherwise the patch $\mathbf{p}$ will have the pixel at (i, j) as its upper left corner. A *vectorized* image patch is a vector $\mathbf{x} \in \mathbb{R}^n$ that is created by stacking the columns of $\mathbf{p}$. The definition of 3D image patches is similar. A 3D image patch is a small volume $\mathbf{p} \in \mathbb{R}^{a \times b \times c}$ with $a \cdot b \cdot c = n$, extracted from position (i, j, k) in a 3D image volume.

If we extract multiple patches from an image and two or more patches holds a copy of the same pixel, then they are called overlapping image patches. An example of overlapping image patches is shown below.

To transform a signal $\mathbf{x} \in \mathbb{R}^n$ back into an image the same steps are followed. First, $\mathbf{x}$ is reshaped into its original matrix form. Then, it is inserted into the image at the same location from where it was extracted. If the patches are overlapping, all pixels contained in multiple patches are averaged.

Overlapping Image Patches

Let $\mathbf{I}$ be a 2D grayscale image.

$$\mathbf{I} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

2×2 overlapping image patches are created by sliding a 2×2 window along the rows, starting at pixel $(0, 0)$. When the last column is reached, the window is moved down to the next row, and back to the first column. The first five image patches are

$$\mathbf{p}_{0,0} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}, \mathbf{p}_{0,1} = \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix}, \mathbf{p}_{0,2} = \begin{bmatrix} 3 & 4 \\ 7 & 8 \end{bmatrix}, \mathbf{p}_{1,0} = \begin{bmatrix} 5 & 6 \\ 9 & 10 \end{bmatrix}$$

And their vectorized representation is

$$\mathbf{p}_{0,0} = \begin{bmatrix} 1 \\ 5 \\ 2 \\ 6 \end{bmatrix}, \mathbf{p}_{0,1} = \begin{bmatrix} 2 \\ 6 \\ 3 \\ 7 \end{bmatrix}, \mathbf{p}_{0,2} = \begin{bmatrix} 3 \\ 7 \\ 4 \\ 8 \end{bmatrix}, \mathbf{p}_{1,0} = \begin{bmatrix} 5 \\ 9 \\ 6 \\ 10 \end{bmatrix}$$

Chapter 2

Sparse Coding

In this Chapter, we provide an overview of how sparsity is used in signal processing. In recent years, sparsity and overcomplete signal representations have been found useful in a wide range of signal and image processing tasks. Among these are compression, upscaling and demosaicing [53], and tasks like denoising and inpainting, which are discussed in Chapter 3.

The main focus of this chapter is to introduce concepts and terminology. We will also explore the main problems associated with sparse coding together with their solutions, and compare different sparse recovery algorithms.

The first thing we need to define is a signal. A signal is a measurement of some phenomenon, for example, it could be an image, a video, speech or audio. In the rest of this work, a signal is represented as a vector $\mathbf{x} \in \mathbb{R}^n$.

A vector or signal is called sparse if it has only a few nonzero elements, and we define sparsity in terms of its *support.* Let $\mathbf{x} = [x_1, x_2, ..., x_n] \in \mathbb{R}^n$ be a signal, its support is then defined as

$$\operatorname{supp}(\mathbf{x}) = \{i : \; x_i \neq 0\} \tag{2.1}$$

If $|\operatorname{supp}(\mathbf{x})| \leq m$, then $\mathbf{x}$ is called *m-sparse.* Given a set of elementary signals, a sparse signal $\mathbf{x}$ can be represented as a linear combination of a few elementary signals. In mathematical terms, let $\mathbf{d}_i \in \mathbb{R}^n$ for $i = 1, ...K$ be the elementary signals, then we define the representation of $\mathbf{x}$ as

$$\mathbf{x} = \sum_{i=1}^{K} \mathbf{d}_i a_i \tag{2.2}$$

If most of the coefficients $\mathbf{a} = [a_1, a_2, ..., a_K] \in \mathbb{R}^K$ are zero, expression (2.2) is called the sparse representation of the signal $\mathbf{x}$. The elementary signals $\mathbf{d}_i$ are called *atoms*, and each atom describes a basic feature of the signal. There are multiple ways to choose the atoms. Possible choices are using basis functions from well-known transformations such as Fourier Transform, Wavelet [66], or the *Discrete Cosine Transform* (Sections 3.6 and 3.7). These functions, or transformations, are chosen depending on the signal, for example, the Discrete Cosine Transform (DCT) is good for representing an image.

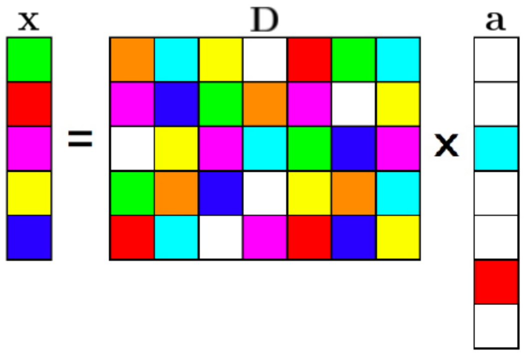


Figure 2.1: Sparse representation of a signal $\mathbf{x} = \mathbf{Da}$ with an overcomplete dictionary and a sparse vector with two nonzero coefficients

We can rewrite formulation (2.2) into matrix form

$$\mathbf{x} = \mathbf{Da} \tag{2.3}$$

The formulation (2.3) holds the same information as formulation (2.2), but the atoms are now ordered as columns of the matrix $\mathbf{D} \in \mathbb{R}^{n \times K}$. The matrix $\mathbf{D}$ is called a *dictionary*, and we say the signal $\mathbf{x}$ is sparse in the dictionary $\mathbf{D}$. It is usually considered the case where $K \gg n$. In this case the dictionary is *overcomplete*. An overcomplete dictionary holds more atoms which allows to better describe the signals compared to a square dictionary with $n = K$.

As there are more columns than rows in $\mathbf{D}$, we need to solve the underdetermined linear system $\mathbf{x} = \mathbf{Da}$ which has infinitely many solutions, Figure 2.1. In order to more efficiently find a solution we impose some additional constraints on the dictionary. In particular, we require the atoms to have unit ℓ_2 norm, and we have $\mathbf{D} \in \mathcal{C}^{n \times K}$, where

$$\mathcal{C}^{n \times K} = \{\mathbf{D} \in \mathbb{R}^{n \times K} : \ \|\mathbf{d}_i\|_2 = 1, \ i = 1, ..., K\}$$

The representation (2.3) assumes that the signal is sparse. Most natural signals are not directly sparse, but they are *compressible-sparse* [74]. This means that the sorted magnitudes $|a_i|$ of the coefficients from (2.3) decay according to the power law

$$|a_i| \leq C i^{-1/2}, \quad i = 1, ..., K \tag{2.4}$$

It is known that we can accurately represent a compressible-sparse signal using only its k largest coefficients [74].

2.1 Problem Formulation

The sparse coding problem is defined as following. Given a signal $\mathbf{x} \in \mathbb{R}^n$ and a dictionary $\mathbf{D} \in \mathcal{C}^{n \times K}$, find the vector $\mathbf{a} \in \mathbb{R}^K$ with the minimum number of nonzero coefficients such that $\mathbf{x} = \mathbf{Da}$. Formally, it is defined as

$$\min_{\mathbf{a} \in \mathbb{R}^K} \quad \|\mathbf{a}\|_0 \text{ such that } \mathbf{x} = \mathbf{Da} \tag{2.5}$$

$\|\cdot\|_0$ is the ℓ_0-norm[1] which counts all nonzero elements in a vector, and is often denoted as $\|\mathbf{x}\|_0 = \#\{i : x_i \neq 0, \ i = 1, ..., n\}$. Formulation (2.5) assumes the signal is noiseless. A more realistic case is when a signal is corrupted by noise. In this case, we want to solve

$$\min_{\mathbf{a} \in \mathbb{R}^K} \quad \|\mathbf{a}\|_0 \text{ such that } \|\mathbf{x} - \mathbf{Da}\|_2^2 < \epsilon \tag{2.6}$$

This formulation is more robust. In formulation (2.6) we are only interested in recovering the support up to a tolerance ϵ, and noise or other artifacts can be accounted for by setting the appropriate tolerance. This formulation also works for compressible-sparse signals.

We can reformulate (2.6) using the Lagrangian formulation

$$\operatorname*{argmin}_{\mathbf{a} \in \mathbb{R}^K} \quad \frac{1}{2} \|\mathbf{x} - \mathbf{Da}\|_2^2 + \lambda \|\mathbf{a}\|_0 \tag{2.7}$$

We will consider this form, rather than its constrained optimization version throughout the rest of the thesis. The term $\|\mathbf{x} - \mathbf{Da}\|_2^2$ is the reconstruction error which measures the accuracy of the representation, and the last term $\lambda \|\mathbf{a}\|_0$ is the regularization. The parameter $\lambda \in \mathbb{R}$ is called the regularization parameter and provides a way to control the trade-off between data fitting accuracy and sparsity of $\mathbf{a}$.

[1]The ℓ_0-penalty does not define a norm, but we use the $\|\cdot\|_0$ notation for consistency

Solving the sparse coding problem with ℓ_0-regularization is a non-convex optimization problem, which is known to be NP-Hard [23]. There are two main approaches to solving (2.7), one of them is using efficient greedy algorithms, such as *Orthogonal Matching Pursuit* (Section 2.2). The other approach is to replace the non-convex ℓ_0-penalty with the convex ℓ_1-norm. The ℓ_1-regularized problem is a convex-relaxation of the ℓ_0 problem, its formulation is known as the *Lasso*, and is defined as

$$\underset{\mathbf{a}\in\mathbb{R}^k}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{x}-\mathbf{D}\mathbf{a}\|_2^2 + \lambda\|\mathbf{a}\|_1 \tag{2.8}$$

This formulation is convex, and the global minimum can be found [49]. Convex optimization problems are well-studied, and software for solving such problems are readily available [22].

Much work has been done to establish the relationship between, and prove uniqueness of the ℓ_0- and ℓ_1-regularized problems. The theoretical results uses *maximum coherence*, $\mu = \max_{i\neq j}|\langle \mathbf{d}_i, \mathbf{d}_j\rangle|$, of a dictionary as a measure of how well the dictionary can describe the signals. A large maximum coherence, $\mu \approx 1$, means that the atoms contains a lot of redundant information. The work of Donoho, Hou, Fuchs, Tropp, and others, in [27, 38, 26, 76] provide the following result.

Theorem 2.1.1 (Unique Sparse Recovery [38]) *Let $\mu = \max_{i\neq j}|\langle \mathbf{d}_i, \mathbf{d}_j\rangle|$ be the maximum coherence of a dictionary $\mathbf{D} \in \mathcal{C}^{n\times K}$, if*

$$\|\mathbf{a}\|_0 \leq \frac{1}{2}\Big(1+\frac{1}{\mu}\Big)$$

then $\mathbf{a}$ is a unique solution of (2.7), and also a unique solution of (2.8).

Theorem 2.1.1 was first proved by Donoho and Hou in [27], but with the restriction that the dictionary is a concatenation of two square orthogonal matrices, $\mathbf{D} = [\mathbf{A}_1\ \mathbf{A}_2]$. Their proof was later expanded to include arbitrary dictionaries with ℓ_2-normalized columns [38, 26, 76].

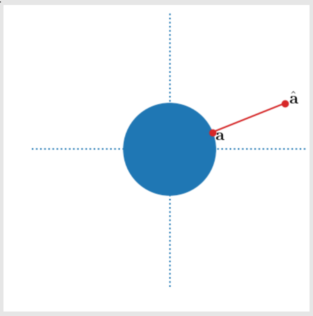
Why do ℓ_0- and ℓ_1-regularization lead to a sparse solution, and not ℓ_2-regularization?
The solutions of the ℓ_0-, ℓ_1- and ℓ_2-regularized problems can in some sense be viewed as first solving the ordinary least squares problem

$$\hat{\mathbf{a}} = \underset{\mathbf{a}\in\mathbb{R}^K}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{x} - \mathbf{D}\mathbf{a}\|_2^2$$

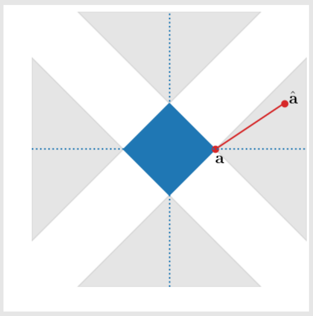
then, projecting the solution $\hat{\mathbf{a}}$ onto the appropriate ℓ_i-ball, $i = 0, 1, 2$. In Figure 2.2, an example where $K = 2$ is given, $\hat{\mathbf{a}}$ is the least squares solution plotted together with each of the ℓ_0, ℓ_1 and ℓ_2 balls. In Figure 2.2c, all points on the ℓ_0 ball are sparse, thus projecting $\hat{\mathbf{a}}$ onto this ball will clearly yield a sparse solution.

The ℓ_1-ball has corners where the solution is sparse. In this example there are only four points where the solution is sparse, but when the dimension of $\hat{\mathbf{a}}$ increases the number of corners do to. Then, the probability of projecting onto any of the corners increase. The dark areas in Figure 2.2b are all points from which $\hat{\mathbf{a}}$ will be projected onto one of the corners on the ℓ_1 ball.

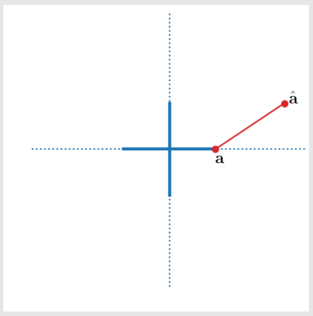
For the ℓ_2-ball, Figure 2.2a, the points giving a sparse solution represent a very small part of all the possible solutions. The projection of $\hat{\mathbf{a}}$ onto the ℓ_2-ball will give a solution with small coefficients. However, it is only sparse if $\hat{\mathbf{a}}$ already is sparse.



(a) ℓ_2 ball



(b) ℓ_1 ball



(c) ℓ_0 ball

Figure 2.2: Projection of least-squares solution $\hat{\mathbf{a}}$ onto the ℓ_i unit balls

2.2 Reconstruction with the ℓ_0-penalty

Finding the exact solution to the ℓ_0 problem requires checking $\binom{K}{s}$ combinations of nonzero coefficients, which is impractical. In this section, we will give an overview of the most well-known and recognized greedy algorithm and its variations for solving the ℓ_0-problem. Greedy algorithms provide an efficient mechanism for solving problems of this sort. A greedy algorithm is an iterative approach, which at each iteration will choose the best possible solution.

Orthogonal Matching Pursuit

To reiterate, given a signal $\mathbf{x} \in \mathbb{R}^n$ and a dictionary $\mathbf{D} \in \mathbb{R}^{n \times K}$, the ℓ_0 sparse coding problem is as follows.

$$\operatorname*{argmin}_{\mathbf{a} \in \mathbb{R}^K} \quad \frac{1}{2}\|\mathbf{x} - \mathbf{D}\mathbf{a}\|_2^2 + \lambda \|\mathbf{a}\|_0$$

Orthogonal Matching Pursuit (OMP) is an iterative algorithm for recovering the support of $\mathbf{a}$ [17, 61, 67]. An outline of the algorithm is given below.

Algorithm 1: Orthogonal Matching Pursuit

```
   Data: Dictionary $\mathbf{D}$, signal $\mathbf{x}$, and sparsity target $s$ or tolerance $\epsilon$
   Result: Sparse approximation $\mathbf{a}$ such that $\mathbf{x} \approx \mathbf{D}\mathbf{a}$
1  $I = []$, $\mathbf{r} = \mathbf{x}$, $\mathbf{a} = \mathbf{0}$;
2  while length($I$) $< s$ or $\|\mathbf{r}\|_2 > \epsilon$ do
3      Find element with largest correlation;
4          $k = \operatorname*{argmax}_k \ |\mathbf{d}_k^T \cdot \mathbf{r}|$;
5      Update index set: $I = [I, k]$;
6      Solve for $\mathbf{a}_I$: $\mathbf{D}_I \mathbf{a}_I = \mathbf{x}$;
7      $\mathbf{r} = \mathbf{x} - \mathbf{D}_I \mathbf{a}_I$;
8  end
```

Algorithm 1, is the original implementation of OMP [23]. This algorithm iteratively builds the sparse coefficients $\mathbf{a}$ for the signal $\mathbf{x}$. It is initialized with the residual $\mathbf{r}$, being equal to the input signal, and the trivial initial solution $\mathbf{a} = \mathbf{0}$. Then, at each iteration the atom from $\mathbf{D}$ with the highest correlation with the current residual $\mathbf{r}$ is selected (line 3). This atom is added to an index set I, called the *active-set*, to keep track of previously activated coefficients. The second, and final step, is to orthogonally project the signal onto the span of the selected dictionary atoms (line 5). This process repeats until a stopping criterion is met. The stopping criteria used in OMP are the maximum number of nonzero coefficients, or sparsity target s, or a given reconstruction tolerance ϵ.

Analysis

If the condition in Theorem 2.1.1 holds, then OMP will recover the exact support of $\mathbf{a}$. A signal does not always have an exact sparse representation over the given dictionary, and in that case Theorem 2.1.1 is not applicable. In this case we have the following result.

Theorem 2.2.1 (Compressible Signal Recovery [76]) *Let μ be the maximum coherence of a dictionary $\mathbf{D} \in \mathcal{C}^{n\times K}$, and $\mathbf{x} \in \mathbb{R}^n$ is a compressible sparse signal, then there exists a best m-sparse approximation $\mathbf{x}_{opt} = \mathbf{D}\mathbf{a}_{opt}$, with $\operatorname{supp}(\mathbf{a}_{opt}) = m$. If*

$$\mu \leq \frac{1}{3m}$$

then, the m-sparse approximation $\mathbf{x}_m$, obtained by OMP is bounded by

$$\|\mathbf{x} - \mathbf{x}_m\|_2 \leq \sqrt{1+6m}\|\mathbf{x} - \mathbf{x}_{opt}\|_2$$

Improvements

The OMP algorithm is computationally expensive. Below, we will give an overview of two efficient, alternative implementations of OMP using a Cholesky factorization to speed up the calculations.

Algorithm 1 is computationally expensive, mainly because of the need to solve the linear system $\mathbf{D}_I\mathbf{a}_I = \mathbf{x}$. Typically this is solved as $\mathbf{a}_I = \mathbf{D}_I^\dagger\mathbf{x} = (\mathbf{D}_I^T\mathbf{D}_I)^{-1}\mathbf{D}_I^T\mathbf{x}$. Since OMP selects only linearly independent atoms, the matrix $\mathbf{D}_I^T\mathbf{D}_I$ will always be non-singular and its inverse exists. In addition, $\mathbf{D}_I^T\mathbf{D}_I$ is symmetric positive definite with one row and one column appended to it at every iteration. For this reason, an efficient incremental Cholesky factorization can be used to reduce the computational cost of solving the linear system. This new variation of OMP using a progressive Cholesky factorization is presented under the name OMP-Cholesky [10, 19, 67].

Algorithm 2: OMP-Cholesky [67]

```
   Result: Sparse approximation $\mathbf{a}$ of $\mathbf{x}$ such that $\mathbf{x} \approx \mathbf{D}\mathbf{a}$
   Data: Signal $\mathbf{x}$, dictionary $\mathbf{D}$, sparsity target $s$, tolerance $\epsilon$
1  $I = []$, $\mathbf{L} = [1]$, $\mathbf{r} = \mathbf{x}$, $\mathbf{a} = 0$, $\boldsymbol{\alpha} = \mathbf{D}^T\mathbf{x}$, $n = 0$;
2  while length$(I) < s$ or $\|\mathbf{r}\|_2 > \epsilon$ do
3      $\hat{k} = \underset{k \in [1,K]}{\text{argmax}} \; |\mathbf{d}_k^T \mathbf{r}|$;
4      if $n > 0$ then
5          Solve for $\mathbf{w}$: $\mathbf{L}\mathbf{w} = \mathbf{D}_I^T \mathbf{d}_{\hat{k}}$;
6          $\mathbf{L} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{w}^T & \sqrt{1 - \mathbf{w}^T\mathbf{w}} \end{bmatrix}$;
7      end
8      $I = [I, \hat{k}]$;
9      Solve for $\mathbf{a}_I$: $\mathbf{L}\mathbf{L}^T\mathbf{a}_I = \boldsymbol{\alpha}_I$;
10     $\mathbf{r} = \mathbf{x} - \mathbf{D}_I\mathbf{a}_I$;
11     $n = n + 1$;
12 end
```

Using OMP-Cholesky, Algorithm 2, $\mathbf{a}_I = (\mathbf{D}_I^T\mathbf{D}_I)^{-1}\mathbf{D}_I^T\mathbf{x}$ is reduced to computing the Cholesky factorization $\mathbf{L}\mathbf{L}^T$ (lines 5-6) and solving $\mathbf{L}\mathbf{L}^T\mathbf{a}_I = \boldsymbol{\alpha}_I$ (line 9).

If $\hat{\mathbf{A}} = \hat{\mathbf{L}}\hat{\mathbf{L}}^T \in \mathbb{R}^{n \times n}$ is a Cholesky factorization of $\hat{\mathbf{A}}$, then the matrix with one row and one column appended to it,

$$\mathbf{A} = \begin{pmatrix} \hat{\mathbf{A}} & \mathbf{g} \\ \mathbf{g} & c \end{pmatrix}$$

has the Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^T \in \mathbb{R}^{(n+1)\times(n+1)}$ [67], where

$$\mathbf{L} = \begin{pmatrix} \hat{\mathbf{L}} & \mathbf{0} \\ \mathbf{w} & (c - \mathbf{w}^T\mathbf{w})^{1/2} \end{pmatrix}, \quad \text{and } \mathbf{w} = \hat{\mathbf{L}}^{-1}\mathbf{g}$$

While the dominating operation in the original algorithm $(\mathbf{D}_I^T\mathbf{D}_I)^{-1}$ requires $\mathcal{O}(n^3)$ operations, the new system (line 9) requires $\mathcal{O}(n^2)$ operations. This system can be solved as $\mathbf{L}\mathbf{z} = \boldsymbol{\alpha}_I$ with forward substitution, and $\mathbf{L}^T\mathbf{a}_I = \mathbf{z}$ using backwards substitution. Both forward- and backward substitution have complexity $\mathcal{O}(n^2)$ and solving $\mathbf{L}\mathbf{L}^T\mathbf{a}_I = \boldsymbol{\alpha}_I$ requires both operations, thus it also has complexity $\mathcal{O}(n^2)$. $\mathbf{L}\mathbf{L}^T$ is the Cholesky factorization of $\mathbf{D}_I^T\mathbf{D}_I$, so the linear system changes from $\mathbf{D}_I\mathbf{a}_I = \mathbf{x}_I$ to $\mathbf{D}_I^T\mathbf{D}_I\mathbf{a}_I = \boldsymbol{\alpha}$ which is why the right-hand side changes to $\boldsymbol{\alpha}_I = (\mathbf{D}_I)^T\mathbf{x}$.

Encoding Multiple Signals

In practice, one is usually interested in approximating more than one signal, in this case further optimization can be introduced [67]. One example of approximating multiple signals is finding the sparse representation of an image over a dictionary $\mathbf{D} \in \mathcal{C}^{n \times K}$. The problem of encoding multiple signals can be formulated as a matrix factorization problem

$$\underset{\mathbf{A} \in \mathbb{R}^{K \times N}}{\operatorname{argmin}} \quad \frac{1}{2} \|\mathbf{X} - \mathbf{DA}\|_F^2 + \lambda \|\mathbf{A}\|_0 \tag{2.9}$$

Now $\mathbf{X} \in \mathbb{R}^{n \times N}$ is a matrix of N signals represented as its columns, and all N signals are to be sparsely encoded over the same dictionary. *OMP-Batch* is an efficient algorithm for approximating multiple signals over the same dictionary, its details are given in the algorithm below.

Algorithm 3: OMP-Batch [67]

```
Result: Sparse approximation $\mathbf{a}$ of $\mathbf{x}$ such that $\mathbf{x} \approx \mathbf{Da}$
Data: $\boldsymbol{\alpha}_0 = \mathbf{D}^T\mathbf{x}$, $\mathbf{G} = \mathbf{D}^T\mathbf{D}$, $\epsilon_0 = \mathbf{x}^T\mathbf{x}$, sparsity target $s$ or tolerance $\epsilon$
1  $I = []$, $\mathbf{L} = [1]$, $\mathbf{r} = \mathbf{x}$, $\mathbf{a} = 0$, $\boldsymbol{\alpha} := \boldsymbol{\alpha}_0$, $n := 0$;
2  while length$(I) < s$ or $\epsilon_n > \epsilon$ do
3      $\hat{k} = \underset{k \in [1,K]}{\operatorname{argmax}} \ |\boldsymbol{\alpha}_k|$;
4      if $n > 0$ then
5          Solve for $\mathbf{w}$: $\mathbf{Lw} = \mathbf{G}_{\hat{k},\mathbf{I}}$;
6          $\mathbf{L} = \begin{bmatrix} \mathbf{L} & \mathbf{0} \\ \mathbf{w}^T & \sqrt{1 - \mathbf{w}^T\mathbf{w}} \end{bmatrix}$;
7      end
8      $\mathbf{I} = [\mathbf{I}, k]$;
9      Solve for $\mathbf{a}_I$: $\mathbf{LL}^T\mathbf{a}_I = \boldsymbol{\alpha}_I^0$;
10     $\mathbf{b} = \mathbf{G}_\mathbf{I}\mathbf{x}_I$;
11     $\boldsymbol{\alpha} = \boldsymbol{\alpha}^0 - \mathbf{b}$;
12     $\delta^{n+1} = \mathbf{x}_I^T\mathbf{b}_I$;
13     $\epsilon^{n+1} = \epsilon^n - \delta^{n+1} + \delta^n$;
14     $n = n + 1$;
15 end
```

By pre-computing the Gram matrix of the dictionary, $\mathbf{G} = \mathbf{D}^T\mathbf{D}$ and $\boldsymbol{\alpha}_0 = \mathbf{D}^T\mathbf{x}$, the residual $\mathbf{r}$ no longer needs to be computed explicitly. With these pre-computations we save two matrix-vector multiplications per iteration for the cost of computing the Gram matrix and $\boldsymbol{\alpha}_0 = \mathbf{D}^T\mathbf{x}$. In practice, computing $\boldsymbol{\alpha}_0$ can be quite expensive since it requires the matrix multiplication $\mathbf{D}^T\mathbf{X}$, where $\mathbf{X}$ can be very large. For example, if $\mathbf{X}$

is generated from all overlapping 8×8 image patches from a 256×256 image, we have $N = 62001$, and $\mathbf{X} \in \mathbb{R}^{64 \times 62\,001}$. The computations saved by not explicitly calculating the residual makes up for the extra upfront costs of the pre-computations already when the number of signals exceeds approximately $\sqrt{n}$.

Computational Complexity

The upper bound on the number of iterations is always known. If a sparsity target s is supplied, OMP will use at most s iterations. If tolerance is given as stopping criterion, then the maximum number of iterations is equal to the number of atoms, K. Let T denote the number of iterations, then the complexities for sparse coding one signal with Algorithms 2 and 3, are as given by Rubinstein, Zibulevsky and Elad [67]

$$\mathrm{C}_{\text{omp-cholesky}} = 2TKn + 2T^2n + 2T(K+n) + T^3 \tag{2.10}$$
$$\mathrm{C}_{\text{omp-batch}} = 2Kn + T^2K + 3TK + T^3 \tag{2.11}$$

where K is the number of dictionary atoms and n is the signal size. The last term is the complexity for solving the Cholesky system (line 9). The first term in $\mathrm{C}_{\text{omp-c}}$ corresponds to finding the most correlated atom, and in $\mathrm{C}_{\text{omp-b}}$ calculating $\boldsymbol{\alpha}_0$. The two middle terms are the complexities of updating the residual in *OMP-Cholesky*, α in *OMP-Batch*, and updating the Cholesky factorization.

When setting[2] $K = 2n$ and $T = \sqrt{n}$ the complexities for encoding one signal reduces to $\mathcal{O}(n^{5/2})$ for OMP-Cholesky and $\mathcal{O}(n^2)$ for OMP-Batch. One can easily see the increased efficiency of OMP-Batch. The computational complexity of encoding one signal with the original implementation of OMP, Algorithm 1, is $\mathcal{O}(n^3)$ [75], which is larger than its two accelerated versions.

Implementation Details

The memory required to compute the sparse coefficients can be pre-allocated. Since the maximum number of iterations is known, we also know how much memory is needed for the computations. Therefore, arrays such as $\mathbf{w}$ for computing the Cholesky factorization can be allocated once and reused for later iterations to speed up the calculations.

[2]Setting $K = 2n$ is a commonly used size for overcomplete dictionaries. $\sqrt{n}$ is a reasonable choice for the sparsity level in a sparse representation.

The OMP versions above encodes each signal independently, therefore, it can easily be parallelized. Algorithms 1, 2 and 3, are implemented in the provided software[3]. These implementations take advantage of the mentioned techniques for speeding up the computations. Algorithms 2 and 3 are implemented in the programming language C using `Cython` [7] to interface with python. Parallelization is achieved using OpenMP [21].

It is worth-while to mention that Algorithms 2 and 3, are also available via the popular machine learning library *scikit-learn* [62]. Those implementations however, do not apply pre-allocation or parallelization. Therefore, it is worth-while to perform a comparison of our implementations with those of *scikit-learn*. For the comparison we run two experiments. Firstly, we compare the single core performance of both `dictlearn` and *scikit-learn*. Then, we compare the multi-threaded performance of our two OMP implementations.

Figure 2.3 contains the result of comparing the computation time of the provided implementations versus the implementations provided by *scikit-learn*. For this test, a variable number of signals are encoded over a dictionary $\mathbf{D} \in \mathbb{R}^{64\times256}$, created using the *Discrete Cosine Transform* bases as atoms. For OMP-Cholesky, we calculated the sparse codes approximately *75 times* faster than the equivalent implementation in *scikit-learn*. For OMP-Batch, the speedup factor is approximately 50.

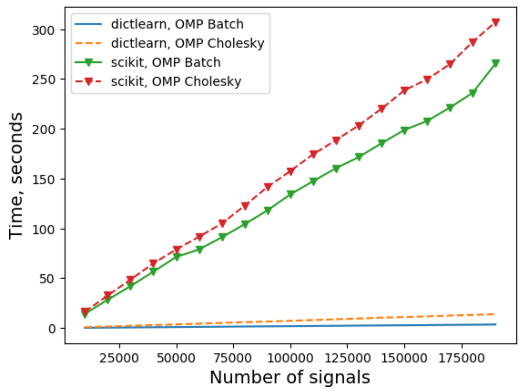


Figure 2.3: Single core performance of `scikit-learn` and `dictlearn`

Other reasons for these single core speedups are that the provided implementations are implemented in C, while the `scikit-learn` implementations

[3]OMP-Batch and OMP-Cholesky are found in `dictlearn/sparse.py`

calls LAPACK [2] directly from *python* which introduces quite a lot of overhead. Therefore, the exact factor of speedup is not of interest, but rather how the implementations scale when the number of signals increase.

The implementations provided with this work will achieve further speedups by running more threads, Figure 2.4. In Figure 2.4, 70000 signals are sparse coded over the same dictionary as in Figure 2.3. We can see that OMP-Cholesky has a larger speedup compared to OMP-Batch. This is because the entire OMP-Cholesky algorithm is parallelized, and OMP-Batch is not. The pre-computations in OMP-Batch are not run in parallel.

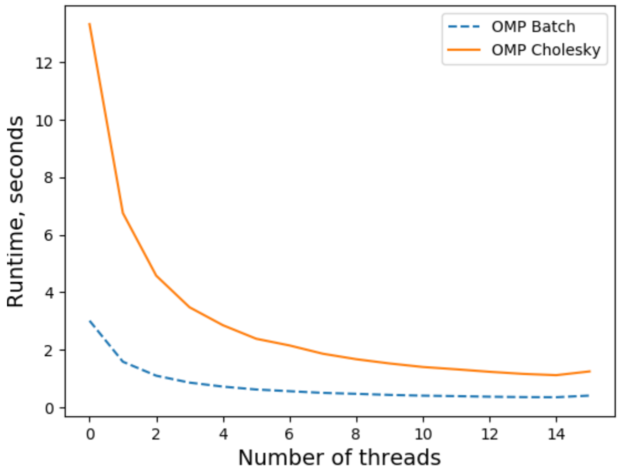


Figure 2.4: Multi-core performance of `dictlearn OMP`

2.3 Reconstruction with the ℓ_1-norm

Another approach to sparse coding is where we replace the ℓ_0-penalty with the ℓ_1-norm. Replacing the non-convex ℓ_0-penalty with the convex ℓ_1-norm is known as *convex-relaxation*. The problem is then defined as

$$\underset{\mathbf{a}\in\mathbb{R}^K}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{x}-\mathbf{D}\mathbf{a}\|_2^2 + \lambda\|\mathbf{a}\|_1 \tag{2.12}$$

As before, we have the signal $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{D} \in \mathcal{C}^{n\times K}$ the dictionary, and we are interested in recovering the coefficients $\mathbf{a} \in \mathbb{R}^K$. Some authors [54, 76, 16] advocate the use of the ℓ_1-regularization compared to its non-convex alternative. By Theorem 2.1.1, we know the optimal solution to this problem is unique and identical to the ℓ_0-regularized problem.

Even though formulation (2.12) is convex, it is not continuously differentiable and applying gradient based methods is difficult [49]. There exist a number of approaches for solving (2.12), among the most popular methods are *Least Angle Regression* [29], *feature-sign search* [49] and *thresholding* [53]. The first two methods will recover the support one element at the time, similar to OMP. Thresholding-based methods require an initial guess for the solution (can be the least squares solution) which then are shrunk using the soft-thresholding operator.

In the rest of this section, we will briefly introduce the Least Angle Regression (LARS) algorithm for solving the ℓ_1-problem. The *feature-sign search* algorithm [49] and *soft-thresholding* based algorithm [53] are also available in our software, but their details are left out of this thesis for the sake of concise presentation. OMP and ℓ_0-regularization are the main sparse coding techniques we use in this thesis and the provided software. Therefore, we provide only a short summary of LARS to prepare its comparison with OMP for solving the ℓ_0 and ℓ_1 sparse coding problems. A comparison of computation time and reconstruction accuracy of sparse coding with both OMP and LARS is found at the end of this Chapter.

Least Angle Regression

Least Angle Regression (LARS) [29] is an iterative method, similar to OMP, for solving the ℓ_1-regularized problem (2.12). LARS starts by initializing the solution $\mathbf{a} = \mathbf{0}$, and the residual $\mathbf{r} = \mathbf{x}$. In the first step, LARS selects the atom most correlated with the residual

$$h = \operatorname*{argmax}_{i=0,\dots,K-1} |\mathbf{d}_i^T \mathbf{r}| \tag{2.13}$$

Rather than directly project the signal onto the span of selected dictionary atoms, as done by OMP, the selected coefficient a_h is moved in the direction $\mathbf{d}_h$ starting at $a_h = 0$ until a new coefficient enter the set of active coefficients. Initially the set of active coefficients $\hat{\mathcal{A}}$ is defined

$$\hat{C} = \max_{i=0,\dots,K-1} |\mathbf{d}_i^T \mathbf{r}|, \tag{2.14}$$

$$\hat{\mathcal{A}} = \{j : \ j = 0, \dots, K-1 \text{ and } \mathbf{d}_j^T \mathbf{r} \geq \hat{C}\} \tag{2.15}$$

At the first iteration $\hat{\mathcal{A}}$ contains only one index h from equation (2.13). Thus, a_h is moved in the direction of its least-square coefficient, $\mathbf{d}_h^T \mathbf{r}$. At some point on this path, a new coefficient will reach the same level of correlation with the residual and enter the active set. The active coefficients $\mathbf{a}_{\hat{\mathcal{A}}}$ are now moved in the direction of their joint least-squares coefficients, which is given by the normal equations

$$\mathbf{b} = (\mathbf{D}_{\hat{\mathcal{A}}}^T \mathbf{D}_{\hat{\mathcal{A}}})^{-1} \mathbf{D}_{\hat{\mathcal{A}}}^T \mathbf{r} \in \mathbb{R}^{\text{size}(\hat{\mathcal{A}})}$$

The coefficient change is done according to

$$\hat{\mathbf{a}}_{\hat{\mathcal{A}}} = \mathbf{a}_{\hat{\mathcal{A}}} + \gamma \cdot \mathbf{b}$$

where γ is the step size. This process repeats until the requested number of nonzero coefficients are achieved or the reconstruction error is within some tolerance.

2.4 Sparse Coding CTA Volumes

In this thesis, we are interested in analyzing 3D CTA images. Overlapping image patches are generated as explained in Chapter 1, such that the patches fit the required form $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{n \times N}$. One of the challenges of working with this class of images is the memory requirements. In this section, we will see how this issue is solved in the provided software such that the sparse coding algorithms can be applied on images where the set of image patches is too large to fit in memory.

For a 2D image of size $H \times W$, we can extract all overlapping image patches into the matrix $\mathbf{X}$. For a 256×256 image and patch size $p \times p$ we have $S^2 = (256 - p + 1)^2$ patches. The amount of memory needed to store these patches is $S^2 \cdot p^2 \cdot \texttt{type\_size}$ bytes. A common choice for patch size is 8×8, storing all overlapping patches of this size from a 256×256 image needs 31.7 megabytes (MB), assuming the image is stored as `double`. When adding the memory needed for storing the sparse coefficients[4], the dictionary and other temporary arrays, the total amount of memory needed will usually not exceed 100MB.

The memory increases substantially when adding the extra dimension. The amount of memory needed to store all overlapping $8 \times 8 \times 8$ patches from a $256 \times 256 \times 256$ image volume is $S^3 \cdot p^3 \cdot \texttt{type\_size} = 63.2$ GB. On top of this, we have to add the memory for the sparse coefficients, dictionary, etc., and the total consumption will easily reach 100s of gigabytes. One of

[4]The size of the sparse coefficients is $S^2 \cdot K \cdot \texttt{type\_size}$, which usually is larger than the signals. Overcomplete dictionaries have $n = p^i \ll K$, $i = 2, 3$

the intentions with the software developed for this thesis is that is should be usable on a standard computer, then storing more than a few gigabytes is unreasonable.

The first step in reducing the memory consumption is noticing that the sparse coding methods only depends on a single signal. Therefore, the sparse coding process can be reduced from a batch algorithm requiring all signals, to an online algorithm encoding one signal at the time (or in smaller batches). The lower bound on the memory needed is now $p^3 \cdot$ `type_size`, or one signal. The implementation of image patches[5] included in our software will handle these issues automatically, by only creating the matrix of image patches if it can fit in memory. If the set of image patches is too large to be kept in memory, the patches are only accessible in smaller batches through an iterator. Iterative reconstruction is also supported. A more detailed example showing how this implementation is used with both large and small sets of image patches is provided in Example 1.

2.5 Comparison of sparse coding algorithms

In this section, we will conduct a comparison of OMP and LARS for solving the sparse coding problems. This comparison is often left out of the literature. Therefore, it is of interest to run it in order to better understand the practical differences of the algorithms. The first part of this section will give an overview of how reconstruction accuracy or error is measured in the image domain, and then, we present the comparison itself.

Accuracy Measures

The reconstructed image patches are defined as $\hat{\mathbf{X}} = \mathbf{DA}$ with $\mathbf{D} \in \mathcal{C}^{n \times K}$ and $\mathbf{A} \in \mathbb{R}^{K \times N}$ the dictionary and sparse codes respectively. We define $I_r \in \mathbb{R}^{H \times W}$ to be the image created from the reconstructed image patches $\hat{\mathbf{X}}$, and $I \in \mathbb{R}^{H \times W}$ is the original image. The accuracy, or reconstruction error is $\epsilon = d(I, I_r)$, d is a function $d : (I, I) \to \mathbb{R}$ that measures the distance between the original image and its reconstruction. There is no single way to chose d for measuring the image reconstruction quality, two common choices are the *mean squared error* (MSE) and *peak signal to noise ratio* (PSNR). MSE is defined as

[5]Found in `dictlearn/preprocess.py`

```
>>> image_patches = Patches(small_volume, size=(16, 16, 16))
>>> patches = image_patches.patches  # Get matrix of signals
>>> reconstructed = image_patches.reconstruct(2*patches)
>>> # The reconstructed volume will have all points doubled
>>> numpy.allclose(reconstructed, 2*small_volume)
True

>>> image_patches = Patches(large_volume, size=(16, 16, 16))
>>> image_patches.patches # Too large to access matrix
MemoryError: Not enough memory for patches, need 143.18 GB.
Use Patches.generator(batch_size)
>>> iterator = image_patches.generator(batch_size=1000)
>>> next(iterator) # Get 1000 first patches
array([[16480., ..., 20662.],
       ...,
       [21212., ..., 19563.]],
      dtype=float32)
>>> # To double all patches do:
>>> iterator = image_patches.generator(1000, callback=True)
>>> for patches, reconstruct in iterator:
>>>     reconstruct(2*patches)

>>> numpy.allclose(image_patches.reconstructed,
>>>                      2*large_volume)
True
```

Example 1: Handle large image patches with `dictlearn`

$$\mathrm{MSE}(I, I_r) = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}(I[i,j] - I_r[i,j])^2 \tag{2.16}$$

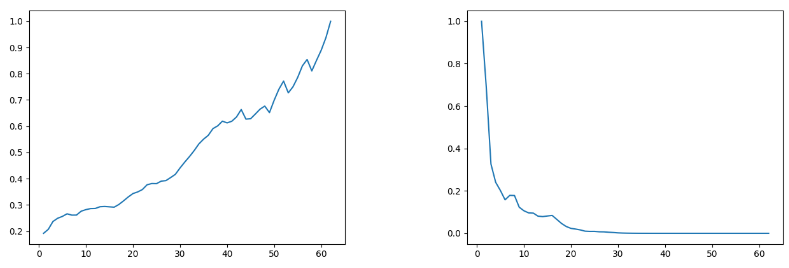
And PSNR is defined

$$\mathrm{PSNR}(I, I_r) = 20\log\ \mathrm{MAX}_I - 10\log\mathrm{MSE}(I, I_r) \tag{2.17}$$

where MAX_I is the maximum possible intensity in the image. MSE is an *error-measure*, it measures the error between its two inputs, and we want MSE close to zero for a high quality reconstruction. PSNR measures the quality between the original image I and its reconstruction I_r, in decibel (dB). The range of the PSNR is $(0, \infty)$, where ∞ corresponds to the case where $I_r = I$. To give some context to PSNR values, it is often said a PSNR in the range of $30dB$ to $50dB$ is acceptable for lossy image compression.

Starting at around $30dB$, it can also be hard to tell the difference between the original image and its reconstruction without comparing the images side by side.

Below is the result from a simple experiment where the image *lena* (Figure 2.6a) is reconstructed using an increasing number of nonzero coefficients. The image is reconstructed using an overcomplete DCT dictionary $\mathbf{D} \in \mathbb{R}^{64\times256}$, Figure 2.7, and the number of active coefficients are increased from $s = 1, ..., 65$. MSE and PSNR of reconstruction for a given sparsity level are presented in Figure 2.5. In Figure 2.6, we show some of the reconstructed images. The purpose for including this experiment is to show the accuracy of representing an image via a sparse representation. We see that by using only ten nonzero coefficients per image patch, the reconstruction, Figure 2.6c, looks almost identical to the original image, Figure 2.6a.



(a) PSNR (b) MSE

Figure 2.5: Accuracy of reconstruction with a variable number of nonzero coefficients

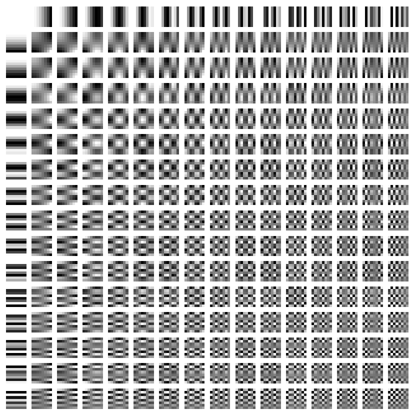
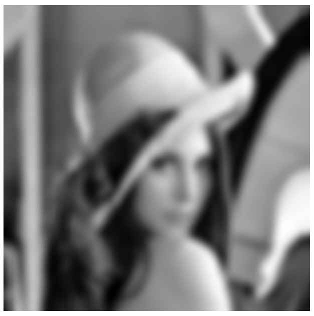
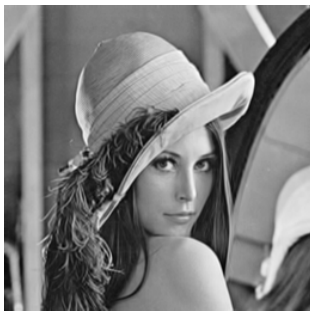


Figure 2.7: *Overcomplete Discrete Cosine Transform* dictionary, size `64, 256`

(a) Original. PSNR $= \infty$, MSE $= 0$


(b) One nonzero coefficient. PSNR $= 20.7dB$, MSE $= 8 \cdot 10^{-3}$


(c) Ten nonzeros coefficients. PSNR $= 30.4dB$, MSE $= 9 \cdot 10^{-4}$

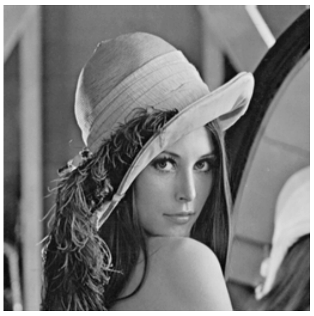

(d) 65 nonzeros coefficient. PSNR $= 81.7dB$, MSE $= 6 \cdot 10^{-9}$

Figure 2.6: Reconstructions of the image 2.6a with increasing number of nonzero coefficients, using the ODCT dictionary

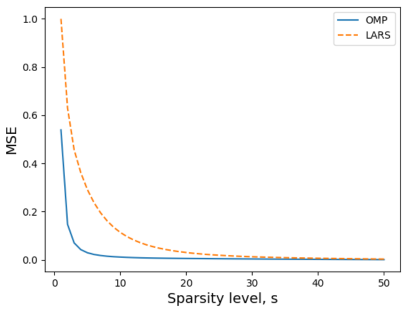
Comparison of OMP and LARS

In this section, we compare the solutions of the ℓ_0 and ℓ_1 sparse coding problem, as this comparison is often not included in the literature. Bach et al. [6] did an extensive review of ℓ_1 methods for solving problem (2.12). The review also contains a thorough benchmark[6] for different problem sizes and dictionary structures. They concluded that LARS is the preferred method for solving (2.12). Among the methods they reviewed, LARS recovered the most accurate solution as well as being the most efficient method. However, they did not compare LARS with OMP. Therefore, we focus on this comparison.

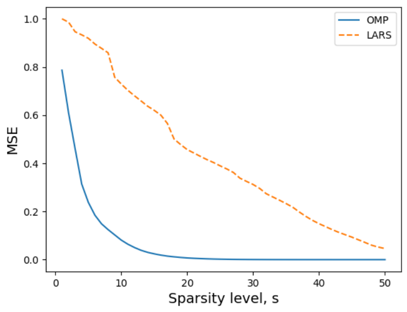
Below are the results from an experiment designed similar to that of Bach el at. We compare the reconstruction accuracy and computation time for

[6]Section 8.1 in [6]

the OMP-Batch implementation included with this thesis, and the LARS implementation[7] used for the benchmark in [6].



(a) MSE of reconstruction with LARS and OMP as sparsity increase using a dictionary trained from image patches



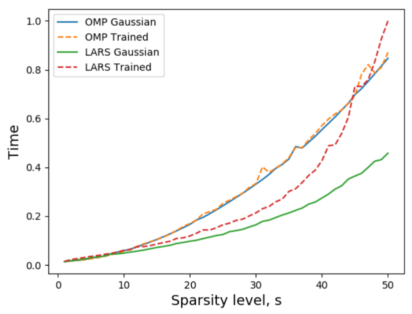
(b) MSE of reconstruction with LARS, OMP and as sparsity increase using a Gaussian dictionary

Figure 2.8: Accuracy of reconstruction with LARS and OMP for two different dictionaries.

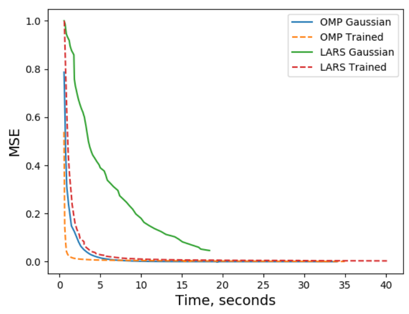
These tests were run with signals from a 3D medical image volume using $(8 \times 8 \times 8)$ image patches. The dictionary size was in both cases 512×1024. The dictionary for the second test was created by drawing each row from a Gaussian distribution, similar to that of Bach et al. and the dictionary for the first experiment was learned using 100,000 image patches from CTA volumes. In both experiments, OMP is the superior in terms of reconstruction error, see Figure 2.8.

In terms of computational time, the methods have a similar performance, but OMP is slightly slower when the same number of coefficients are used, Figure 2.9a. From this comparison, we can conclude that LARS is faster to achieve a given sparsity level. At the same time, OMP will produce a more accurate solution than LARS given the same number of nonzero coefficients. Figure 3.1b shows MSE of the reconstruction and computation time. For reconstruction with trained dictionaries, LARS is slightly slower than OMP to achieve a given accuracy. For Gaussian dictionaries, OMP is faster by a large margin. From these tests, we can conclude that OMP performs better than LARS for image reconstruction applications.

[7]The implementation is found here: http://spams-devel.gforge.inria.fr/



(a) Computation time for each level of sparsity for OMP and LARS relative to slowest run



(b) Relative MSE as a function of time for OMP and LARS

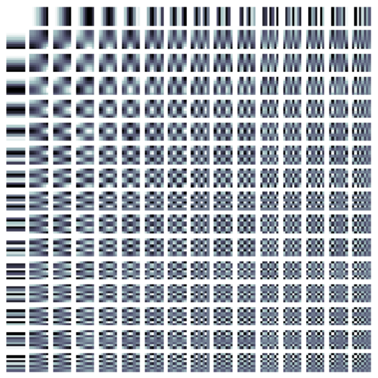
Figure 2.9: Comparison of computation time for LARS and OMP.

Chapter 3

Dictionary Learning

In Chapter 2, we introduced algorithms for computing a sparse representation of some signals $\mathbf{X} \in \mathbb{R}^{n \times N}$, over a fixed dictionary $\mathbf{D} \in \mathcal{C}^{n \times K}$. In Chapter, we introduce methods for learning the dictionary from the input signals to create a highly specific dictionary that is very good at representing the signals. We will also see how the introduced methods are applied to problems of image restoration.

Dictionary learning is a form of representation learning which learns a dictionary $\mathbf{D} \in \mathcal{C}^{n \times K}$ of elementary signal features, from a set of input signals $\mathbf{X} \in \mathbb{R}^{n \times N}$, allowing to represent the signals as a linear combination of just a few atoms. The best dictionary is the one providing the sparsest representation.


(a) Overcomplete DCT dictionary

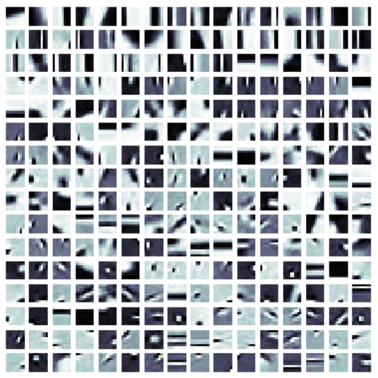

(b) Dictionary trained with K-SVD

Figure 3.1: Static ODCT dictionary (left) and a dictionary learned from image patches (right)

In practice, we would like to have a dictionary with a fast, implicit transformation, so that we can obtain the coefficients and reconstruct the signals in near linear time. This so-called analytical approach to dictionary learning has been widely popular in signal processing in the 80's and 90's. However, choosing the best analytical dictionary require prior knowledge of the signals. Fourier Transform, Wavelets [56], Curvelets [73], Ridgelets [12] and others, are all designed to provide a very good representation for signals with a specific structure [74]. For example, a harmonic signal will have an accurate sparse representation with a Fourier dictionary, but it may not have a sparse representation in any of the other dictionaries mentioned above. With dictionary learning, we learn a highly specific dictionary from the input signals. Thus, we can always efficiently represent the input signals sparsely, also more complex signals where an analytic dictionary cannot guarantee a sparse representation. Examples of analytic and learned dictionaries are presented in Figure 3.1.

The dictionary learning problem is a joint minimization problem and is as following. Find a dictionary $\mathbf{D} \in \mathcal{C}^{n\times K}$, and a matrix of sparse coefficients $\mathbf{A} \in \mathbb{R}^{K\times N}$ such that $\mathbf{X} \approx \mathbf{DA}$, that is

$$\underset{\mathbf{D}\in\mathcal{C}^{n\times K},\mathbf{A}\in\mathbb{R}^{K\times N}}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{X}-\mathbf{DA}\|_F^2 + \lambda\|\mathbf{A}\|_0 \tag{3.1}$$

$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N] \in \mathbb{R}^{n\times N}$ is the matrix of all training signals as its columns and $\mathbf{A} \in \mathbb{R}^{K\times N}$ is the sparse matrix where each column i is the sparse coefficients of the signal $\mathbf{x}_i$. Here, we use ℓ_0-regularization for finding $\mathbf{A}$, but ℓ_1-regularization can also be used.

The formulation (3.1) is a non-convex minimization problem and is usually solved by an alternate minimization scheme. An alternate minimization algorithm will first fix the dictionary, then minimize (3.1) with respect to the sparse coefficients. Then, fix the sparse codes and update the dictionary. In the next sections, we give an overview of two batch algorithms and two online algorithms for training a dictionary. Although they all follow the alternate minimization scheme, they differ in the form the sparse coefficients and the dictionary are updated. Batch algorithms take all the training signals into account when updating the solution, while online algorithms perform the updates after looking at a single (or few) training signals. Therefore, an online algorithm may be preferred when the number of training signals is very large.

3.1 Method of Optimal Directions

Method of Optimal Directions (MOD) [31] was the first popular algorithm for dictionary learning. It provides an efficient and flexible way for training a dictionary, as it typically converges in just a few iterations. The goal of MOD is to find a dictionary $\mathbf{D}$ and a matrix of sparse coefficients $\mathbf{A}$ that minimize the representation error (3.2), given a set of training signals $\mathbf{X} \in \mathbb{R}^{n \times N}$. MOD minimizes

$$\underset{\mathbf{D}\in\mathcal{C}^{n\times K},\mathbf{A}\in\mathbb{R}^{K\times N}}{\operatorname{argmin}} \quad \|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2 \quad \text{such that} \quad \|\mathbf{a}_i\|_0 \leq s \quad i=1,...N, \tag{3.2}$$

Like other training methods, MOD alternates the sparse coding and dictionary update steps. The sparse coding step can be done by any sparse coding technique.

Algorithm 4: Method of Optimal Directions

```
Input: Training data $\mathbf{X} \in \mathbb{R}^{n\times N}$, initial dictionary $\mathbf{D}_0$, stopping
       criterion for sparse coding $\mathbf{s}$, number of iterations $T$
Result: Trained dictionary $\mathbf{D} \in \mathbb{R}^{n\times k}$
1 for $t = 1, ..., T$ do
2     Compute sparse codes;
3         $\mathbf{A}_t \approx \underset{\mathbf{A}\in\mathbb{R}^{K\times N}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{X}-\mathbf{D}_{t-1}\mathbf{A}\|_2^2 + \lambda\|\mathbf{A}\|_0$;
4     Update the dictionary;
5         $\mathbf{D}_t = \text{Proj}_{\mathcal{C}}(\mathbf{X}\mathbf{A}_t^T(\mathbf{A}_t\mathbf{A}_t^T)^{-1})$;
6 end
```

The dictionary update (line 5) is done by computing the analytical solution of the problem, given by $\mathbf{D} = \mathbf{X}\mathbf{A}^{\dagger} = \mathbf{X}\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}$, using the Moore-Penrose inverse of $\mathbf{A}$. Then, $\mathbf{D}$ is renormalized to fit the constraint. Typically, MOD only needs a few iterations to converge and is overall an efficient method. On the other hand, the relatively large complexity of the matrix inversion is a large weakness [31, 32, 66].

Complexity

The sparse coding (line 3) can be done with the efficient *OMP-Batch* algorithm. The dictionary update (line 5) requires the computation of $\mathbf{A}\mathbf{A}^T$ which has complexity $\mathcal{O}(K^2N)$, inverting $\mathbf{A}\mathbf{A}^T$ is $\mathcal{O}(K^3)$, multiplying this with $\mathbf{A}^T$ is $\mathcal{O}(NK^2)$. Applying $\mathbf{X}$ to the pseudo-inverse is $\mathcal{O}(nKN)$. Projecting the dictionary onto the space $\mathcal{C}$ is insignificant in this context, as it only requires the normalization of the columns of a $n \times K$ matrix, where $n < K \ll N$. The number of signals, N, is typically very large which make the dictionary update computationally demanding, and for this reason is MOD not an appropriate choice for training dictionaries from large training sets. This shortcoming has inspired the development of other dictionary learning methods.

3.2 K-SVD and its Modifications

The K-SVD algorithm was introduced because of the wish to efficiently train a generic dictionary for sparse signal representation [1]. Like MOD, K-SVD uses the alternate minimization scheme by first solving the sparse coding problem before iteratively updating the dictionary atom by atom using the singular value decomposition (SVD).

Algorithm 5: K-SVD

```
   Input: Training data $\mathbf{X} \in \mathbb{R}^{n \times N}$, initial dictionary $\mathbf{D}_0$, sparsity target
          $\mathbf{s}$, number of iterations $\mathbf{T}$ or tolerance $\epsilon$
   Result: Trained dictionary $\mathbf{D} \in \mathbb{R}^{n \times K}$
 1 t = 0;
 2 while $t < T$ or until $\|\mathbf{X} - \mathbf{DA}\|_F < \epsilon$ do
 3     Compute the sparse codes;
 4         $\mathbf{A} \approx \underset{\mathbf{A} \in \mathbb{R}^{K \times N}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{X} - \mathbf{DA}\|_2^2 + \lambda\|\mathbf{A}\|_0$;
 5     Update the dictionary;
 6     for $i$ = 1,...,$K$ do
 7         $\mathbf{w}$ = {Index of samples in $\mathbf{X}$ using atom $i$};
 8         Compute the error without using atom $i$;
 9             $\mathbf{E}_i = \mathbf{X} - \sum_{j \neq i} \mathbf{d}_j \mathbf{a}_j^T$;
10         Restrict $\mathbf{E}_i$ to only columns given by $\mathbf{w}$, $\mathbf{E}_i^{\mathbf{w}}$;
11         $\mathbf{E}_i^{\mathbf{w}} = \mathbf{U\Sigma V}^T$;
12         Set dictionary atom $\mathbf{d}_i = \mathbf{u}_1$;
13         Let $\mathbf{a}_i^{\mathbf{w}}$ be row $i$ of $\mathbf{A}$ with columns corresponding to $\mathbf{w}$, then set;
14             $\mathbf{a}_i^{\mathbf{w}} = \mathbf{\Sigma}_{1,1}\mathbf{v}_1$;
15     end
16     t = t + 1;
17 end
```

Looking at the reconstruction error in iteration i of the dictionary update step, we have

$$\begin{aligned}\|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2 &= \left\|\mathbf{X}-\sum_{j=1}^{K}\mathbf{d}_j\mathbf{a}_T^j\right\|_F^2 \\ &= \left\|\mathbf{X}-\sum_{j\neq i}\mathbf{d}_j\mathbf{a}_T^j-\mathbf{d}_i\mathbf{a}_T^i\right\|_F^2 \\ &= \left\|\mathbf{E}_i-\mathbf{d}_i\mathbf{a}_T^i\right\|_F^2\end{aligned}$$

Where $\mathbf{a}_T^i$ is row i in matrix $\mathbf{A}$. For every atom update the goal is to reduce the error $\mathbf{E} = \|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2$ by finding new vectors $\mathbf{d}_i$ and $\mathbf{a}_T^i$. $\mathbf{E}_i$ is the current reconstruction error without using atom i. Since the goal is to minimize $\mathbf{E}$ by replacing $\mathbf{d}_i$ and $\mathbf{a}_T^i$, we can replace $\mathbf{d}_i\mathbf{a}_T^i$ with the best rank-1 approximation of $\mathbf{E}_i$. By taking the left and right singular vectors corresponding to the largest singular value of the SVD of $\mathbf{E}_i$, its best rank-1 approximation is found. Using the SVD of $\mathbf{E}_i$ directly will minimize the error, but there is no guarantee that the sparsity constraint is still satisfied. To account for this Aharon et al. [1] restricts the error matrix and the sparse coefficient vector to indices for the signals whose representation uses atom i. That is the index set $\mathbf{w} = \{j : \mathbf{a}_T^i(j) \neq 0\}$. Denoting this restricted error matrix as $\mathbf{E}_i^{\mathbf{w}}$ and the sparse coefficients $\mathbf{a}_i^{\mathbf{w}}$, the SVD can now be applied without loosing the sparsity in $\mathbf{a}_i^{\mathbf{w}}$. Write $\mathbf{E}_i^{\mathbf{w}} = \mathbf{U\Sigma V}^T$, then the new atom becomes $\mathbf{d}_i = \mathbf{u}_1$. The update for the sparse coefficients will be $\mathbf{a}_i^{\mathbf{w}} = \mathbf{\Sigma}_{1,1}\mathbf{v}_1$.

Computing the error matrix $\mathbf{E}_i$ can be computationally demanding, and the cost of this computation grows with the number of training samples. The sum of outer products $\sum_{j\neq i}\mathbf{d}_j\mathbf{a}_j^T$ requires $\mathcal{O}((k-1)kN)$ operations for every atom update, with the number of training samples N typically very large. To increase the efficiency of the dictionary update, a new method was proposed by Rubinstein et al. in [67] under the name Approximate K-SVD, Algorithm 6. In K-SVD, Algorithm 5, the error matrix is computed explicitly and a full singular value decomposition is done, but none of these are strictly needed. We only need the singular vectors corresponding to the largest singular value for the restricted error matrix. In [67], the error matrix and SVD computation are replaced with one iteration of an alternate-optimization scheme. The new atom update is given by

$$\begin{aligned}\mathbf{d} &= \mathbf{E}_i\mathbf{g}/\|\mathbf{E}_i\mathbf{g}\|_2 \\ \mathbf{g} &= \mathbf{E}_i^T\mathbf{d}_i\end{aligned} \tag{3.3}$$

The update (3.3), is one iteration of the power method for computing the Singular Value Decomposition. It will eventually converge to the optimum, but it was shown that a single iteration is sufficient to achieve results comparable to the original algorithm [67]. The full details of the method are given in Algorithm 6.

Algorithm 6: Approximate K-SVD (AK-SVD)

```
Input: Training data $\mathbf{X} \in \mathbb{R}^{n \times N}$, initial dictionary $\mathbf{D}_0$, sparsity target
       $\mathbf{s}$, number of iterations $\mathbf{T}$ or tolerance $\epsilon$
Result: Trained dictionary $\mathbf{D} \in \mathbb{R}^{n \times K}$
1  t = 0;
2  while $t < T$ or until $\|\mathbf{X} - \mathbf{DA}\|_F < \epsilon$ do
3      Compute sparse codes using;
4          $\mathbf{A} \approx \underset{\mathbf{A}}{\operatorname{argmin}} \ \frac{1}{2}\|\mathbf{X} - \mathbf{DA}\|_2^2 + \lambda\|\mathbf{A}\|_0$;
5      Update the dictionary;
6      for $i$ = 1,...,$K$ do
7          $\mathbf{d}_i = 0$;
8          $\mathbf{w} = \{$Index to samples in $\mathbf{X}$ using atom $i\}$;
9          $\mathbf{g} = \mathbf{A}_{i,\mathbf{w}}$;
10         $\mathbf{d} = \mathbf{X}_\mathbf{w}\mathbf{g} - \mathbf{DA}_\mathbf{w}\mathbf{g}$;
11         $\mathbf{d} = \mathbf{d}/\|\mathbf{d}\|_2$;
12         $\mathbf{g} = \mathbf{X}_\mathbf{w}^T\mathbf{d} - (\mathbf{DA}_\mathbf{w})^T\mathbf{d}$;
13         $\mathbf{d}_i = \mathbf{d}$;
14         $\mathbf{A}_{i,\mathbf{w}} = \mathbf{g}^T$;
15     end
16     t = t + 1;
17 end
```

Complexity

The complexity of the sparse coding step is described in Chapter 2. The asymptotic complexity of Approximate K-SVD is

$$\mathrm{C}_{\text{AK-SVD}} = N \cdot \mathrm{C}_{\text{omp-b}} \tag{3.4}$$

The complexity of one iteration with AK-SVD is equivalent to sparse coding of N signals with *OMP-Batch*. The dictionary update step becomes insignificant when compared to the sparse coding [67].

The dominating operations for the dictionary update in the original K-SVD is sparse coding, with complexity equal to (3.4) and creating the index set $\mathbf{w}$ which is $\mathcal{O}(N)$. The computations of the error matrix and its singular value decomposition are dependent on the size of the index set. Calculating $\mathbf{E}^{\mathbf{w}} \in \mathbb{R}^{n \times C_{\mathbf{w}}}$ is approximately $\mathcal{O}(nKC_{\mathbf{w}})$ where $C_{\mathbf{w}} \in [0, N]$ is the size of the index set. The complexity of the SVD of $\mathbf{E}^{\mathbf{w}}$ is given by $\mathcal{O}(\min\{n^2 C_{\mathbf{w}}, nC_{\mathbf{w}}^2\})$ [36]. The size $C_{\mathbf{w}}$ cannot be determined for each iteration, but the total size of $C_{\mathbf{w}}$ when summing over all iterations of the dictionary update is known. This is sN, the number of nonzero coefficients in each signals multiplied with the total number of signals.

3.3 Online Dictionary Learning

So far we have seen two batch algorithms for training dictionaries. A batch learning algorithm will look at the entire training set before doing the dictionary update, while an online algorithm only looks at a single (or few) training sample before doing the update.

Online Dictionary Learning for Sparse Coding (ODL) algorithm from Mairal et al. [54] is an efficient online algorithm for learning a dictionary from large sets of input signals. The computational cost and memory requirements per iteration is lower compared to K-SVD and MOD. In [54], ODL minimizes

$$\underset{\mathbf{D} \in \mathcal{C}, \mathbf{a}_i \in \mathbb{R}^K}{\operatorname{argmin}} \quad \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{2} \|\mathbf{x}_i - \mathbf{D}\mathbf{a}_i\| + \lambda \|\mathbf{a}_i\|_1 \right) \tag{3.5}$$

Similar to MOD and K-SVD, ODL alternates between finding the sparse coefficients $\mathbf{a}_i$ and updating the dictionary. The authors uses the LARS algorithm [29] to solve the sparse coding step, but any sparse coding method can be used. For the dictionary update block-coordinate descent with warm restarts is used.

At every iteration, of Algorithm 7, a signal $\mathbf{x}_t$ is drawn randomly from the matrix of input signals $\mathbf{X} \in \mathbb{R}^{n \times N}$. Information about previous signals and its sparse representation is stored in the matrices $\mathbf{A} \in \mathbb{R}^{K \times K}$ and $\mathbf{B} \in \mathbb{R}^{n \times K}$. The dictionary update uses block-coordinate descent with warm restarts. A block coordinate-descent (BCD) optimizes the objective for one *block* of variables at the time. Due to the structure of the matrix $\mathbf{A}$, Mairal et al. found a BCD based dictionary update step, Algorithm 8, to be a more efficient choice for the optimization method [54].

Algorithm 7: Online Dictionary Learning

```
Input: Signals $\mathbf{X} \in \mathbb{R}^{n\times N}$, initial dictionary $\mathbf{D}_0 \in \mathbb{R}^{n\times K}$, regularization
       parameter $\lambda$, number of iterations $T$
Result: Trained dictionary $\mathbf{D} \in \mathbb{R}^{n\times k}$
1  $\mathbf{A}_0 = 0$, $\mathbf{B}_0 = 0$;
2  for $t = 1, ..., T$ do
3      Draw signal $\mathbf{x}_t$ randomly from $\mathbf{X}$;
4      Sparse Coding;
5          $\mathbf{a}_t = \underset{\mathbf{a}\in\mathbb{R}^K}{\text{argmin}} \quad \frac{1}{2}\|\mathbf{x}_t - \mathbf{D}_{t-1}\mathbf{a}\|_2^2 + \lambda\|\mathbf{a}\|_1$;
6      $\mathbf{A}_t = \mathbf{A}_{t-1} + \mathbf{a}_t\mathbf{a}_t^T$;
7      $\mathbf{B}_t = \mathbf{B}_{t-1} + \mathbf{x}_t\mathbf{a}_t^T$;
8      Update the dictionary using algorithm 8 such that;
9          $\mathbf{D}_t = \underset{\mathbf{D}\in\mathcal{C}^{n\times K}}{\text{argmin}} \quad \frac{1}{t}\sum_{i=1}^{t}\left(\frac{1}{2}\|\mathbf{x}_i - \mathbf{D}_{t-1}\mathbf{a}_i\| + \lambda\|\mathbf{a}_i\|_1\right)$;
10 end
```

In Algorithm 8, one atom is updated at the time. Since the dictionary from the previous iteration is used as a warm restart, and we keep previous information by summing over all seen signals (line 9 Algorithm 7), one iteration with BCD is sufficient for each atom update ($T = 1$ in Algorithm 8). It is shown that this method converges to the optimum [54, 9].

Algorithm 8: Online Dictionary Update

```
Input: Dictionary $\mathbf{D} \in \mathcal{C}^{n\times K}$, matrices with information from previous
       iterations $\mathbf{A} \in \mathbb{R}^{K\times K}$ and $\mathbf{B} \in \mathbb{R}^{n\times K}$, number of iterations $T$
Result: Trained dictionary $\mathbf{D} \in \mathbb{R}^{n\times k}$
1  $t = 0$;
2  while $t < T$ do
3      for $i = 1, ..., K$ do
4          $\mathbf{u}_i = \frac{1}{\mathbf{A}_{ii}}(\mathbf{b}_j - \mathbf{D}\mathbf{a}_i) + \mathbf{d}_j$;
5          $\mathbf{d}_i = \frac{1}{\max(\|\mathbf{u}_i\|_2, 1)}\mathbf{u}_i$;
6      end
7      $t = t + 1$;
8  end
```

Being an online algorithm, ODL works better with large sets of signals compared to K-SVD or MOD. The training signals can be split into smaller batches, and then the dictionary is trained on one batch at the time. Let $\mathbf{D}_i$ be the dictionary trained on the first i batches. To train the dictionary on batch $i + 1$, $\mathbf{D}_i$ is used as the initial dictionary such that the structures learned from the signals in previous batches are kept. The usage of ODL for large sets of signals fits very well with the provided implementation of image patches, such that the training can be split over multiple batches of training signals. An example of how ODL is used is given in example 2.

```
>>> image_patches = Patches(large_volume, size=(16, 16, 16))
>>> iterator = image_patches.generator(1000)
>>> # Create an initial dictionary
>>> dictionary = random_dictionary(image_patches.size,
>>>                                  2*image_patches.size)
>>> for patches in iterator:
>>>     # Train the dictionary on one batch with ODL
>>>     dictionary = odl(patches, dictionary, iters=1000)
```

Example 2: Train a dictionary using a large training set

3.4 Iterative Thresholding and K-residual Means

Iterative thresholding and K-residual Means (ITKrM) [71] is an online learning algorithm that belongs to the class of alternate optimization algorithms. ITKrM uses thresholding for the sparse coding step and residual means for the dictionary update. Due to these simple operations, the algorithm is computationally light and can be easily parallelized. There are theoretical results concerning its local convergence and experimental results concerning its global convergence [71, 59].

Algorithm 9: ITKrM (one iteration)

```
  Input: Signals $\mathbf{X} \in \mathbb{R}^{n\times N}$, initial dictionary $\mathbf{D}_0 \in \mathcal{C}^{n\times K}$, sparsity target
         $s$
  Result: Trained dictionary $\mathbf{D} \in \mathbb{C}^{n\times K}$
1 for $i = 1, ..., N$ do
2     Recover support;
3         $I_i = \underset{I:\,|I|=s}{\text{argmax}} \quad \|\mathbf{D}_I^T \mathbf{x}_i\|_1$;
4     for $k = 1, ..., K$ do
5         $\bar{\mathbf{d}}_k = \sum_{i:k\in I_i} [\mathbb{I}_n - P(\mathbf{D}_{I_i}) + P(\mathbf{d}_k)]\mathbf{x}_i \cdot \text{sign}(\langle \mathbf{d}_k, \mathbf{x}_i \rangle)$
6     end
7 end
8 Output $\mathbf{D} = (\bar{\mathbf{d}}_1/\|\bar{\mathbf{d}}_1\|_2, ..., \bar{\mathbf{d}}_K/\|\bar{\mathbf{d}}_K\|_2)$;
```

In Algorithm 9, $P(\mathbf{A})$ denotes the orthogonal projection onto the column span of $\mathbf{A}$, defined $P(\mathbf{A}) = \mathbf{A}\mathbf{A}^{\dagger}$. In ITKrM the generating dictionary is a fixed point, and the initial dictionary $\mathbf{D}_0$, will converge towards the generating dictionary.

ITKrM does not search for the sparse representation exactly, but rather finds a combination of atoms I_i which is used for signal representation, via the thresholding operation (line 3). Then, ITKrM updates the recovered dictionary atoms using K-residual means (line 6).

Complexity

The dominating operations in each iteration of ITKrM are the matrix-vector multiplication in the sparse coding step, which is $\mathcal{O}(nKN)$, and the projection $P(\mathbf{D}_{I_i})\mathbf{x}_i$ is $\mathcal{O}(nNs^2)$. This projection can be optimized by computing the Gram matrix $\mathbf{D}^T\mathbf{D}$, then the complexity of the projections are $\mathcal{O}(s^3N)$ [71]. Thus, for low sparsity target, s, ITKrM is computationally light compared to K-SVD.

3.5 Comparison of Training Algorithms

In the previous sections we introduced multiple algorithms for training dictionaries, in the next sections, we will apply these algorithms for image restoration. Before we can apply them to the specific applications it needs to be verified that our implementations of the training algorithms can indeed learn a dictionary. The tests are done by training a dictionary with each algorithm using the same set of signals and the same initialization. The dictionary size is 64×128, and 16 nonzero coefficients are used for both training and signal reconstruction. After each algorithm has been running for a given amount of time, the signals are reconstructed using OMP and the trained dictionary. The figure below shows the relative error of the reconstructed signals and the original signals.

The results in Figure 3.2 show that all algorithms can learn a dictionary that produce a sparse representation that is very close to the exact solution. These results are not meant to be used for ranking the methods, only verify that our implementation can learn a high quality dictionary from image signals.

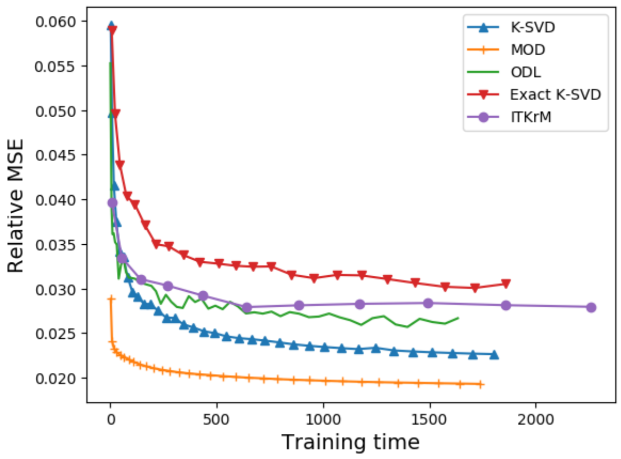


Figure 3.2: Relative reconstruction over training time, averaged over five runs. The dictionary size is $(64, 128)$, and we used 16 coefficients for the sparse coding for both training and reconstruction.

3.6 Image Denoising

The first image restoration task we will discuss is the removal of unwanted noise. Denoising is an important part of our segmentation pipeline discussed in Chapter 5. If an image is noisy, it can be hard to distinguish a small, or low intensity artery from the noise, thus the image has to be denoised before we can accurately extract the arteries.

There are multiple reasons for why an image contains noise. One reason is that an image is a discretization of an analog signal, the analog signal is converted to a discrete digital signal by quantization which can introduce some noise. Other reasons are lighting, camera quality or movement of either the subject or the camera. A special case for CT imaging, is noise introduced during *tomographic reconstruction* [41]. A CT image is created by taking multiple x-ray images from different angles, these images are reconstructed using a process called tomographic reconstruction to create a 3D image volume.

Image denoising is one of the fundamental problems in image processing. Mathematically the problem is formulated as following. Given a noisy image $\mathbf{y}$, defined by

$$\mathbf{y} = \mathbf{x} + \mathbf{v}, \tag{3.6}$$

recover the noise free, ground truth image $\mathbf{x}$. $\mathbf{v}$ is the noise, here it is assumed to be additive zero-mean white and homogeneous Gaussian noise, with standard deviation σ.

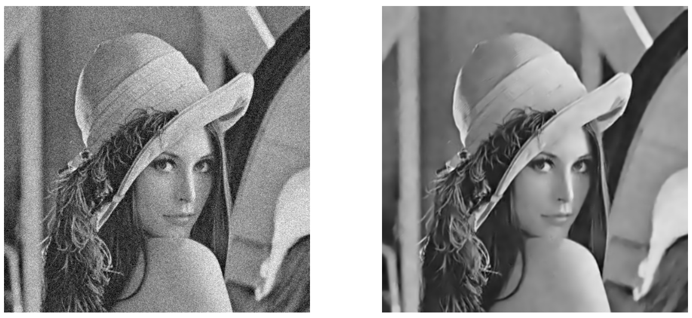


Figure 3.3: Image corrupted with white Gaussian noise with $\sigma = 20$ and PSNR $=$ 17.9 (left) and its denoised version with PSNR $= 32.7dB$

Denoising fits well with the ideas of sparsity and sparse representations. If the ground truth image $\mathbf{x}$ is sparse, we can assume the largest coefficients in the sparse representation of the measured image describes the ground truth image, while the smaller coefficients correspond to the noise. Therefore, we can recover its support by the following minimization procedure

$$\min_{\mathbf{A}\in\mathbb{R}^{K\times N}} \quad \|\mathbf{A}\|_0 \text{ such that } \|\mathbf{Y}-\mathbf{DA}\|_F^2 < \epsilon, \tag{3.7}$$

where $\mathbf{Y} \in \mathbb{R}^{n\times N}$ is the matrix of image patches from the measured image $\mathbf{y}$. The noisy image $\mathbf{y}$ may not be sparse in the dictionary because of the noise, but its m-sparse approximation will be a good estimate of the underlying ground truth image, see *compressible-sparse signals* (2.4).

Extending the sparse coding and learning methods to handle denoising require minimal changes. The first denoising method based on dictionary learning was introduced by Elad and Aharon in [30], and still provides very good results. In [30], K-SVD is used to train a high quality dictionary, but any method for obtaining a dictionary can be used. For training the dictionary one can either use the noisy image itself, or any set of high-quality images. However, the denoising procedure using a dictionary trained on patches from the corrupted image was found to give the best results. In particular, the dictionary is learned using

$$\operatorname*{argmin}_{\mathbf{D}\in\mathcal{C}^{n\times K},\ \mathbf{A}\in\mathbb{R}^{K\times N}} \quad \|\mathbf{A}\|_0 \text{ such that } \|\mathbf{Y}-\mathbf{DA}\|_F^2 < L\sigma \tag{3.8}$$

The tolerance constraint $L\sigma$ depends on both the standard deviation of the noise σ, and some constant L. Empirically it was shown $L = 1.15$ gave the best results [30]. The resulting sparse representation $\hat{\mathbf{X}} = \mathbf{DA}$ from equation (3.8) is accurate up to the constant $L\sigma$, proportional to the noise. The final estimate of the ground truth image is obtained by reconstructing the image from the image patches $\hat{\mathbf{X}}$. The averaging of overlapping pixels when reconstructing the image patches also acts as additional noise removal.

This denoising procedure requires the noise level σ to be known, which is not typically the case. The noise level can be estimated from the noisy image to various degree of success [28, 65]. In the provided software, *Threshold Selection by SURE* [28] is the chosen method for estimating the level of noise.

Denoising 3D CTA Volumes

The algorithm presented by Elad and Aharon in [30] trains the dictionary from the noisy image with a tolerance-based sparse coding step in K-SVD. The implementation provided in `dictlearn`[1] is more general, and any dictionary learning algorithm can be used with either a tolerance- or sparsity-based stopping criterion for the sparse coding stages. A simple usage of the denoising implementation is shown in Example 3.

To verify the implementation, the first step is to test the quality of denoised 2D images. The results presented in Figure 3.5 use the same images and noise levels as some tests described in the original paper [30].

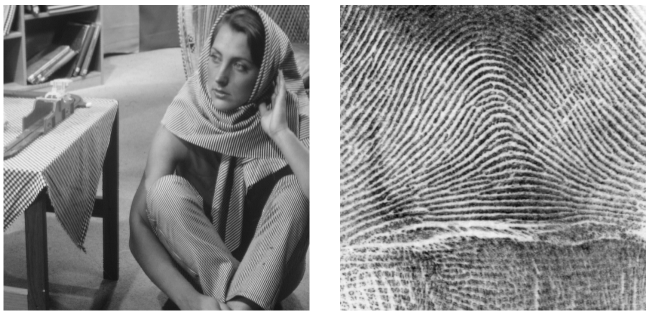

(a) Barbara (b) Fingerprint

Figure 3.4: Original images used for the denoising experiment presented below

Each of the images *Barbara* and *Fingerprint*, Figure 3.4, are denoised separately, and the tests are done according to the following steps.

[1]Found in `dictlearn/algorithms.py`

Data acquisition: The images are first corrupted with Gaussian noise with $\sigma = 20, 50$. Then, all overlapping $p = (10 \times 10)$ image patches from the noisy image are put as columns in the matrix of training signals $\mathbf{X} \in \mathbb{R}^{100 \times 253\,009}$.
Training: Dictionary size is chosen as 100×256. We train one dictionary per image for both training algorithms. We use K-SVD and ODL such that we have one batch and one online algorithm. Training with K-SVD uses $T = 80$ iterations, and 30 000 iterations for ODL. The number of iterations for ODL is the same as the number of signals used for training. Thus, ODL uses only 12% of the signals for training, where K-SVD uses all available signals.
Reconstruction: The image is reconstructed using OMP-Batch with tolerance stopping as described in Section 3.6. The tolerance is defined by $\epsilon = p \cdot (1.15\sigma)^2$

	Barbara		Fingerprint	
σ (PSNR)	20 (22.11)	50 (14.15)	20 (22.11)	50 (14.15)
ODCT	30.07	25.09	28.15	22.98
K-SVD	**30.86**	**26.13**	**28.62**	**23.83**
ODL	29.89	25.91	28.47	23.66
Elad [30]	30.83	25.47	28.47	23.25

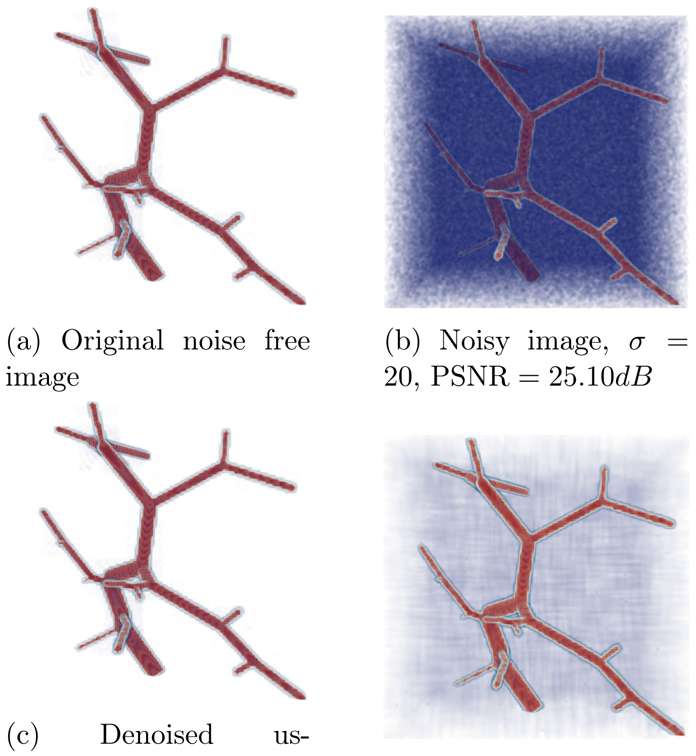
Figure 3.5: PSNR of denoised images. The last row are results from the method presented in [30].

Denoising with the fixed ODCT dictionary follows the first and last step outlined above. The DCT dictionary produce results comparable to the trained dictionaries. Since denoising with a static dictionary require no training, it will be more efficient than trained dictionaries. K-SVD (Algorithm 6) and ODL (Algorithm 7), both used a stopping criteria for training, and the results in the last row, are generated using tolerance stopping, and the implementation of the exact K-SVD published with [30].

Verification on 3D images is done using synthetic image volumes created with the software VascuSynth [40, 45]. This software creates very simple images (Figure 3.6a), where only the vessel-like structures take a value larger than zero. But, for verifying whether the implementation can remove noise from 3D image volumes it is sufficient. For these experiments, we use K-SVD to train the dictionaries, since it provides the best result for 2D image denoising and the images are small enough for a batch algorithm to be used.

There are two main ways denoising can be applied to image volumes. The first is to process the volume using 3D image patches. Then, all overlapping

3D image patches are extracted into the matrix of signals $\mathbf{X} \in \mathbb{R}^{n\times N}$, and denoising is done as described above. The second approach is to process the volume as separate 2D slices. Using this approach, each slice in a given direction (height, width or depth) is denoised as a 2D image. The image patches for the first approach will contain more spatial information that may be helpful for producing the best results. The latter method will save computation time, but discard all spatial information in the dimension where the slices are extracted.


(a) Original noise free image

(b) Noisy image, $\sigma = 20$, PSNR $= 25.10dB$

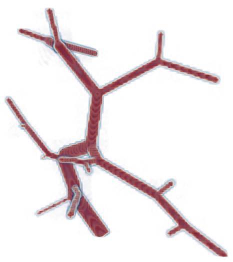
(c) Denoised using 3D patches, PSNR $= 30.15dB$

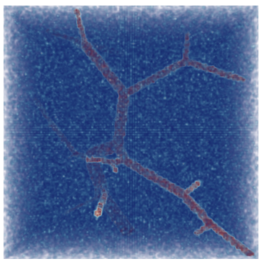
(d) Denoised using 2D slices, PSNR $= 26.66dB$

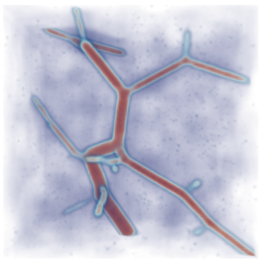
Figure 3.6: $(100 \times 100 \times 100)$ image volumes denoised with a trained dictionary using 3D and 2D image patches

In Figures 3.6 and 3.7, we see the denoising procedure can remove noise from 3D images using both 3D and 2D image patches. Denoising with $10 \times 10 \times 10$ 3D image patches starts with randomly extracting 100 000 overlapping patches into the matrix of training signals, $\mathbf{X} \in \mathbb{R}^{1000\times 100\,000}$. Then, a 1000×2000 dictionary is trained using 10 iterations with K-SVD, and $s = 60$ nonzero coefficients as the stopping criterion. Approximate time for denoising the $100 \times 100 \times 100$ images is one hour, training the dictionary takes around 20 minutes, the rest is for the final sparse coding.

For 10×10 2D image patches we use a 100×256 dictionary, all overlapping patches in each slice, $s = 10$ nonzero coefficients, and 10 iterations per slice. Denoising 2D slices only needs 10 minutes for the whole volume. The accuracy lost for choosing 2D patches over 3D is the same for both noise levels.


(a) Original noise free image


(b) Noisy version, $\sigma = 50$, PSNR $= 17.18dB$


(c) Denoised using 3D patches, PSNR $= 25.10dB$

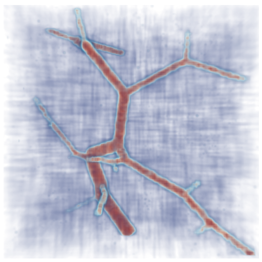

(d) Denoised using 2D slices, PSNR $= 21.37dB$

Figure 3.7: $(100 \times 100 \times 100)$ image volumes denoised with a trained dictionary 3D and 2D image patches

```
>>> denoise = Denoise('noisy_image.png', patch_size=10)
>>> denoise.train(n_atoms=256, n_nonzero=8)
>>> denoised_img = denoise.denoise(sigma=20)
```

Example 3: Denoise a 2D image with a learned dictionary

3.7 Inpainting

Image inpainting is the next image restoration problem we consider. Inpainting is the process of removing unwanted structures or filling in missing points in an image. Dictionary-based inpainting delivers good results as long as the corruptions are of reasonable size, see left image in Figure 3.8. If the corrupted areas are too large, methods based on *Texture Synthesis* has to be applied. The method *Object removal by exemplar-based inpainting* by Criminisi et al. [20] is one of such methods, which is implemented in the provided software[2]. In this section, we focus on dictionary-based inpainting as it fits nicely with the established framework. The next reason is that vessels in a 3D CTA image are usually small enough to be inpainted with dictionary-based methods.

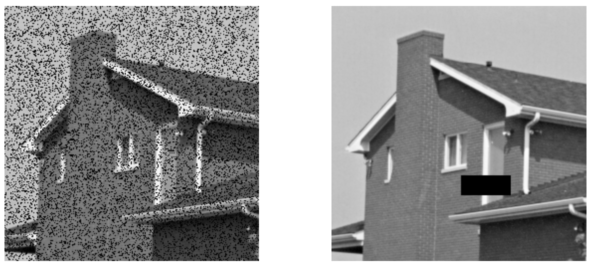


Figure 3.8: Example of small (left) and large (right) corruption patterns

Dictionary-based Inpainting

We assume the corruption to be distributed independently of the signals, and each pixel, (i, j), in the image is observed with a probability p_{ij}. A requirement for inpainting with dictionary learning is that no image patch is covered entirely by the corruption. The corrupted image is assumed to be on the form

$$\mathbf{y} = \mathbf{x} \odot \mathbf{m} + \mathbf{v} \tag{3.9}$$

$\mathbf{y} \in \mathbb{R}^{H \times W}$ is the measured image, $\mathbf{x} \in \mathbb{R}^{H \times W}$ is the ground truth image we want to recover, and $\mathbf{m} \in \{0, 1\}^{H \times W}$ is the corruption. $\mathbf{v}$ is noise as described in Section 3.6, which can be zero. The idea is to include a binary mask, where the value 1 denotes an intact pixel, and 0 denotes a corrupted pixels, and by using the data available in the intact pixels we can fill in values for the corrupted.

[2]Multiple inpainting methods are implemented in `dictlearn`, their details can be viewed in *dictlearn/inpaint.py*

Let $\mathbf{X} \in \mathbb{R}^{n\times N}$ be the image patches from the ground truth image, $\mathbf{M} \in \{0,1\}^{n\times N}$ is the masks, where each $\mathbf{m}_i$ is the mask for $\mathbf{x}_i$, such that $\mathbf{y}_i = \mathbf{m}_i \odot \mathbf{x}_i$ defines the measured, corrupted image patches. Dictionary based inpainting relies on the fact that if every image patch $\mathbf{x}_i$ is sparse in the dictionary $\mathbf{D}$, then every corrupted patch $\mathbf{y}_i = \mathbf{m}_i \odot \mathbf{x}_i$ is sparse in the corrupted dictionary $\mathbf{Dm}_i$. We denote the corrupted dictionary by $\mathbf{Dm}_i = [\mathbf{d}_1 \odot \mathbf{m}_i, ..., \mathbf{d}_K \odot \mathbf{m}_i]$, that is, each atom in the corrupted dictionary corresponding to signal i, is multiplied with the mask for signal i. From this we have that if $\mathbf{a}_i$ are the coefficients for the sparse representation of the corrupted patches $\mathbf{y}_i = (\mathbf{Dm}_i)\mathbf{a}_i$, then $\hat{\mathbf{x}}_i = \mathbf{Da}_i$ is an estimate of the ground truth image patches $\mathbf{x}_i$ [59].

The learning problem is defined

$$\underset{\mathbf{D}\in\mathcal{C}^{n\times K},\ \mathbf{A}\in\mathbb{R}^{K\times N}}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{M}\odot(\mathbf{Y}-\mathbf{DA})\|_F^2 + \lambda\|\mathbf{A}\|_0 \tag{3.10}$$

The idea is similar to that of denoising; learn a dictionary $\mathbf{D}$, and find a matrix of sparse coefficients $\mathbf{A}$, then the representation $\hat{\mathbf{X}} = \mathbf{DA}$ is an estimate of the image patches for the ground truth image $\mathbf{x}$.

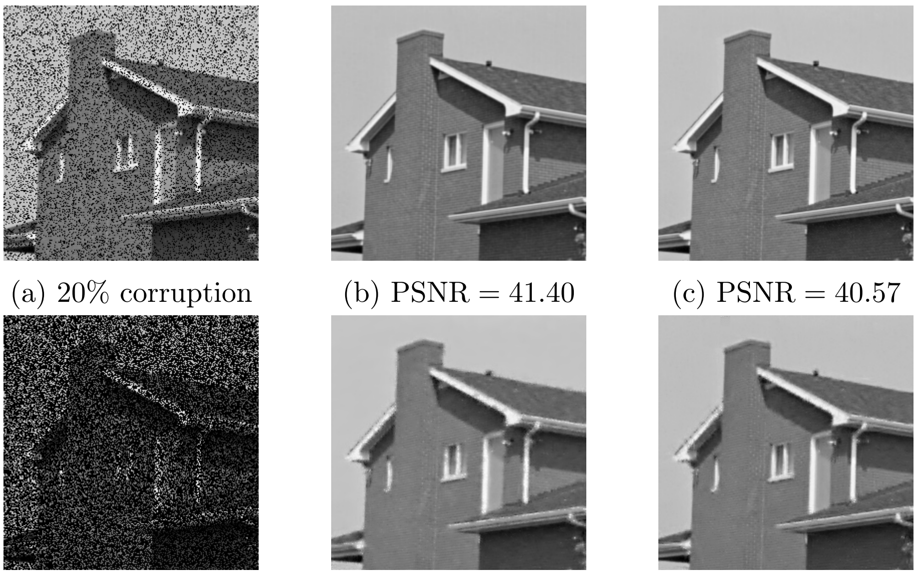
Where denoising was achieved by only setting an appropriate stopping criterion on the final sparse coding step, inpainting needs a few more changes. Since the points in $\mathbf{Y}$ marked by zero in the mask are corrupted, these cannot be taken into account while training the dictionary and sparse coding the signals. First, assume the dictionary is fixed, then we can obtain the sparse codes for the corrupted signals over the corrupted dictionary with the original implementation of OMP (Algorithm 1), by changing only its inputs. The new inputs are the corrupted signal $\mathbf{y}_i \in \mathbb{R}^n$, mask $\mathbf{m}_i \in \{0,1\}^n$, and the renormalized masked dictionary $\mathbf{Dm}_i \in \mathcal{C}^{n\times K}$, then OMP can progress as given by Algorithm 1. The dictionary has to be renormalized since $\|\mathbf{d}_j \odot \mathbf{m}_i\|_2 \leq \|\mathbf{d}_j\|_2 = 1$ for $j = 1, ..., K$ with equality only if $m_i(k) = 1$ for all $k = 1, ..., n$. The corruption does not necessarily affect the norm of the atoms in the same way, thus less corrupted atoms will take precedence in the atom selection step [59].

K-SVD and ODL can both solve the inpainting problem [55, 54], but they use the modified OMP to compute the sparse codes, which is quite slow. Thus, methods relying on OMP for sparse coding during training are impractical. Sparse coding a 256×256 image using a sparsity target of 8, (8×8) image patches, and a dictionary of 256 atoms is almost 100 times slower with the original OMP, compared to OMP-Batch. A better approach is to do sparse coding with a thresholding technique. *Iterative Thresholding and K-residual Means for corrupt data* (ITKrMM) [59] is an extension of the efficient ITKrM, Algorithm 9, for learning a dictionary from corrupted data. It belongs to the class alternate optimization algorithms that alternates between sparse coding the signals with hard-thresholding and updating the dictionary using residual averages.

ITKrMM require only one sparse coding with OMP for recovering the support with the final, learned dictionary, which makes it more efficient than K-SVD or ODL. Inpainting with ITKrMM was shown to be between $8-12$ times faster than inpainting with K-SVD [59].

Inpaint CTA Volumes

Inpainting is verified in the same way as denoising. First using 2D images, then with 3D CTA images. Figure 3.9, shows the results of inpainting a 2D image with random corruption. The dictionaries are trained with ITKrMM, using 20 iterations, 10 nonzero coefficients, one low-rank component, and (10×10) image patches. The final OMP step is done with tolerance stopping at $\epsilon = 10^{-3}$. The reconstructions using a DCT dictionary only require the sparse approximation of the image patches, thus we do sparse approximation with OMP the same way as for ITKrMM.



(a) 20% corruption (b) PSNR = 41.40 (c) PSNR = 40.57

(d) 70% corruption (e) PSNR = 29.60 (f) PSNR = 29.99

Figure 3.9: Random corruption restored with ITKrMM (middle) and DCT (right), with $p = 0.8$ and $p = 0.3$. PSNR is given in dB.

In the case of 20% corruption, both ITKrMM and DCT dictionaries provide very good results. Training the dictionary with ITKrMM on all overlapping image patches from the 256×256 image in Figure 3.9 takes approximately five minutes, and ten minutes for the final sparse coding with OMP, for both ITKrMM and DCT dictionaries. The dictionaries from ITKrMM are trained with only 20 iterations, which is quite low, if trained for more iterations ITKrMM will outperform DCT [59].

Figures 3.11 and 3.10 are the results of inpainting a 3D CTA volume with $(8 \times 8 \times 8)$ image patches. The dictionary is in both cases (512×1024). For random corruption, the dictionary is trained on all overlapping image patches. When only a part of a vessel is missing, such as Figure 3.10, inpainting can be very efficient, since the final sparse coding step only has to be done on the corrupted patches. Figure 3.10 is inpainted by training the dictionary on 60 000 intact image patches, and reconstruction is done by sparse coding only the corrupted patches. For this corruption, sparse coding with OMP is faster than the 2D images in Figure 3.9, taking approximately two minutes.

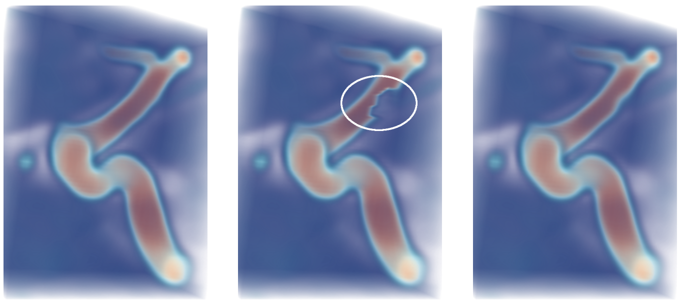


Figure 3.10: Inpainting of larger corrupted area, PSNR(corrupted) = $30.68dB$, PSNR(restored) = $43.32dB$

Inpainting random corruption is slower as all image patches are required to be sparsely approximated with OMP. In Figure 3.10 only 2000 patches is affected by the corruption and have to be approximated, but for Figure 3.11 all 65 000 image patches has to be approximated, thus computation time will increase with a factor of 32.5.

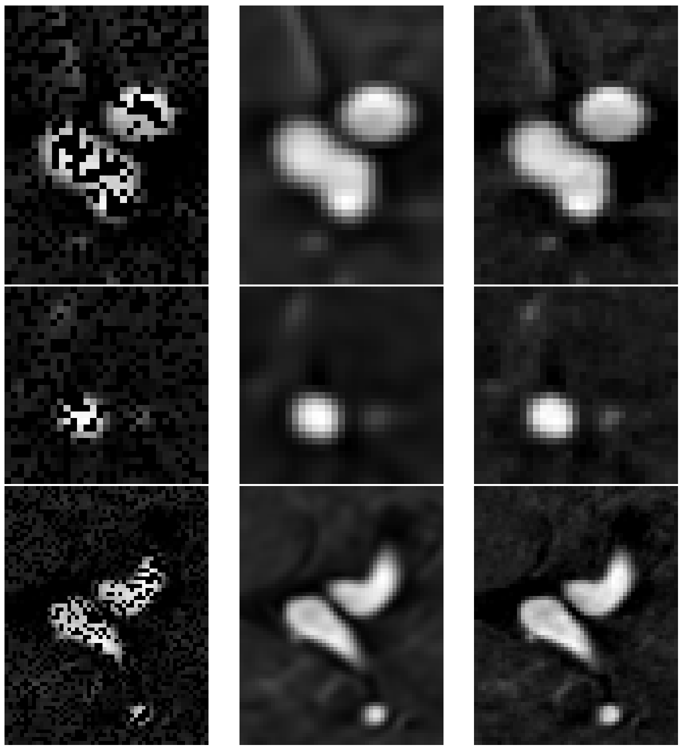
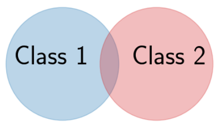


Figure 3.11: Sliced from 3D image. *Left:* Corruption, *middle:* Inpainted, *right:* Original. PSNR(corrupted) = $18.83dB$, PSNR(restored) = $34.46dB$

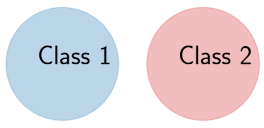
Chapter 4

Segmentation and Feature Enhancement

In Chapter 2 and 3, we saw how sparse coding and dictionary learning based image restoration can be applied to 3D images. Image restoration is used as the initial step in our segmentation pipeline, presented in Chapter 5, to increase the quality of the extracted vessels. This chapter will explore various techniques for extracting arteries from the restored images. Roughly speaking, all points in an image needs to be marked as either an *artery* point or a *not-artery* point. This process of dividing an image into two or more regions by labeling every point, is known as image segmentation.



(a) Two overlapping classes



(b) Classes separated with feature enhancement

Figure 4.1: Two overlapping classes that are separated with feature enhancement

In some cases, the boundary between two classes can overlap, Figure 4.1a, then structures of different classes will be similar and hard to segment into the correct classes. In images where this occur, an appropriate feature enhancement technique should be applied first. The goal of feature enhancement is to make the feature of the different classes more distinct and, thus, make the image easier to segment, Figure 4.1b.

In this Chapter, we look at two techniques for enhancing vessel features, and two methods for extracting the vessels into a binary image, where all points that belong to a vessel take the value 1 and everything else 0. The main issue we need to keep in mind is that a CTA image is not an exact representation of a patient's anatomy. There is no ground truth image, and thus, any inaccuracy introduced by the vessel extraction technique will be added on top of the errors already existing in the CTA. Hence, we should keep the extracted structures as close to the CTA as possible. For example, if there is a narrowing of a vessel, similar to Figure 3.10, then we cannot determine if this is due to an imaging artifact or a stenosis found on the patient's vessel. Therefore, we do no attempt at altering the structure of the vessels, but rather leave this as an optional step in the final pipeline, Chapter 5.

4.1 Thresholding

Thresholding is the simplest way of segmenting an image. By setting all pixels less than a threshold τ to zero, a binary image is created. Often the selection of the threshold is done manually by visual inspection, but methods for automated threshold selection do exist to various degree of success. Global histogram-based threshold selection algorithms are the most commonly used. *Minimum*, *entropy*, *median*, and *mean* are some of the threshold selectors proposed in [39].

The histogram for an image is denoted by q_i, where q_i is the number of pixels with intensity contained in the bin $b_i = [t_i, t_{i+1}, ..., t_{i+s-1}]$. The possible intensity values for a point in the image, t_i, are divided into N bins. For example, if I is a grayscale image with intensities in $[0, 255]$, and its histogram is chosen to have $N = 16$ bins, then each bin contains the intensities $b_i = [16 \cdot (i-1), ..., 16i - 1]$ for $i = 1, ..., 16$.

The *minimum* threshold assumes the image has a bimodal histogram, Figure 4.2. The threshold τ is chosen $\tau = i$, such that q_i takes the minimum value in the valley between the two modes. This is done by finding two local maxima, then finding the minimum point between them. Computing the thresholds mean, median and entropy requires the partial sums

$$A_j = \sum_{i=1}^{j} q_i \text{ and } B_j = \sum_{i=1}^{j} i q_i$$

A_j is the number of pixels with intensity contained in the first j bins, and B_j is approximately the sum of intensities of the pixels in the first j bins. The *median* threshold divides the histogram into two classes, each containing approximately 50% of the pixels, and can be defined as

$$\tau = \operatorname*{argmin}_{t=1,\ldots,N} \left| \sum_{i=1}^{t} q_i - \sum_{i=t+1}^{N} q_i \right|$$

The *mean* threshold is the mean value of the image, $\tau = E(B_N/A_N)$, where $E(x)$ denotes the integer part of x. The next threshold of interest is the *entropy* threshold [39, 46]. This method requires an additional sum; $E_j = \sum_{i=1}^{j} q_i \cdot \log(q_i)$, and the optimal threshold is found by

$$\tau = \operatorname*{argmin}_{i=1,\ldots,N} \frac{E_i}{A_i} - \log(A_i) + \frac{E_N - E_i}{A_N - A_i} - \log(A_N - A_i)$$

Otsu's-method [60], is another commonly used technique for automatically estimating the threshold. This method assume that the image consists of two classes, which correspond to a bimodal histogram as in Figure 4.2. The threshold is chosen such that the intra-class variance is minimized. First, define class weights $\omega_0(t) = \sum_{i=1}^{t-1} q_i$ and $\omega_1(t) = \sum_{i=t}^{N} q_i$, and its corresponding means

$$\mu_0(t) = \sum_{i=1}^{t-1} \frac{iq_i}{\omega_0(t)} \text{ and } \mu_1(t) = \sum_{i=t}^{N} \frac{iq_i}{\omega_1(t)}$$

Finding the optimal threshold τ, which minimizes the intra-class variance, is equivalent to finding the threshold which maximizes the between-class variance $\sigma_B^2(t) = \omega_0(t)\omega_1(t)(\mu_1(t) - \mu_0(t))^2$, then

$$\tau = \operatorname*{argmax}_{t=1,\ldots,N} \quad \sigma_B^2(t)$$

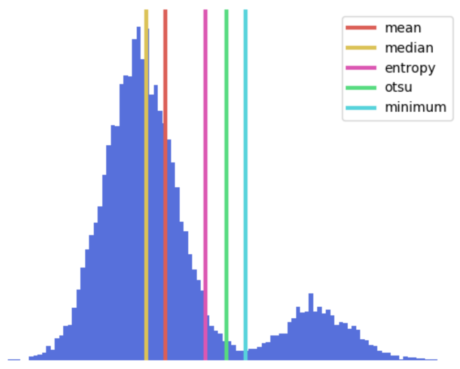


Figure 4.2: Various automatic thresholds on a bimodal histogram created from drawing from two Gaussian distributions

Application to CTA volumes

Histograms constructed from CTA-volumes will roughly resemble the bimodal histogram in Figure 4.2. A histogram from a CTA volume is shown in Figure 4.3. Arteries containing contrast fluid will have higher intensity than the rest of the volume, and will make up the smaller peak at the end of the intensity spectrum. For some volumes, the arteries can be extracted by setting the threshold somewhere in-between the two peaks, this can be achieved by either *Otsu*-, *entropy*- or *minimum*-threshold, see Figures 4.4 and 4.5.

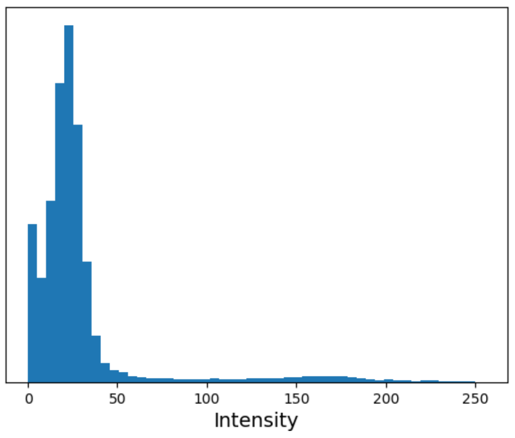


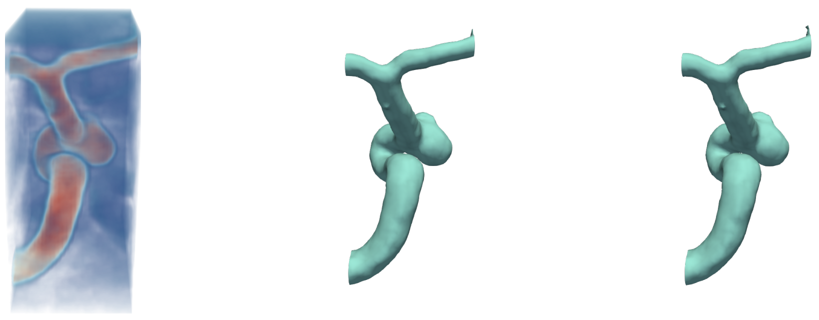
Figure 4.3: Histogram of a CTA image volume. Vessels are found in the bins with intensity 50 and up

The challenge with this approach, is that the intensity of the points in an artery is directly dependent on its size. Larger arteries contain more blood and contrast fluid, thus they will be brighter than smaller arteries. This problem arises around smaller arteries, and in areas where there are other structures that absorb radiation. These structures, see the light blue patterns in Figure 4.5a, have multiple sources. Some are due to noise not removed during image restoration, but some are also due to other anatomical structures. For example, the structure in the lower right corner of Figure 4.5a is the skull. It is the region where the *Internal Carotid Artery* passes through the skull, and the artery will be darker since it is partially hidden by bone. This is also the reason for the large hole in the segmented vessels in Figure 4.5b.

In the above mentioned areas, artery voxels and non-artery voxels will have overlapping intensity, and there is no single point where the classes can be separated exactly. For this reason, the resulting geometry may include non-artery voxels labeled as arteries or artery voxels are missed. In smaller volumes where the artery size and intensity are fairly similar through the whole volume, thresholding methods work very well, see Figure 4.4. One the other hand, Figure 4.5, is an example where thresholding does not work, the geometry extracted with *minimum*-threshold has large holes where there should be an artery. Whereas with *Otsu's* threshold the opposite occur.

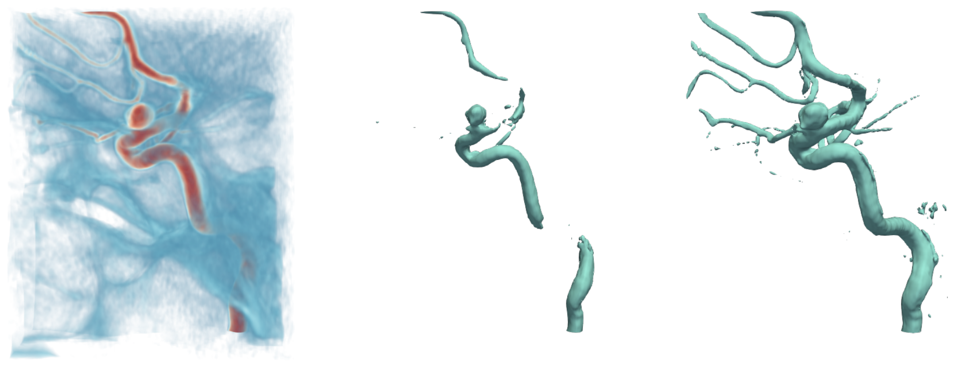
Based on this, it is clear that a vessel extraction method based solely on voxel intensity is not the best approach for a general method. In order to improve the results, one can apply an appropriate feature enhancement technique to increase the difference between artery and non-artery classes. However, for volumes with similar vessel size and intensity, there is little to no overlap between arteries and non-artery points, and thresholding should be used. The advantage of using thresholding compared to the more complex techniques, is its efficiency and simplicity. When the threshold is known, one pass through the image is sufficient to divide all voxels into the two classes.

Thresholding is not the default segmentation method in the provided pipeline, Chapter 5, but support for thresholding is included, and its use require minimal changes to the default configuration.



(a) Denoised volume of size (30 × 52 × 66) (b) Minimum threshold (c) Otsu's threshold

Figure 4.4: Arteries extracted with thresholding. The vessels have similar size and intensity through the whole image.



(a) Denoised volume of size (256 × 256 × 256) (b) Minimum threshold (c) Otsu's threshold

Figure 4.5: Arteries extracted with thresholding. The image has large variations in both vessel size and intensity.

4.2 Feature Enhancement with K-Means

In the previous section, we saw that thresholding yields good segmentation results for images with a simple vessel structure, but for more complex images we need to perform feature enhancement first in order to be able to segment the image accurately. In this section, we show how clustering with K-Means can be used to enhance vessel features, to create a good and efficient vessel extraction method.

K-Means is a clustering algorithm, which groups together similar data samples. By analyzing the training data, a clustering algorithm will split the data into classes with similar features and learn the decision boundaries between the different classes.

K-Means

K-Means is a simple heuristically-driven clustering algorithm. Given some data $\{\mathbf{x}_i\}_{i=1}^N$ and the number of clusters k, K-Means divides the input data into k clusters by minimizing the variance within each cluster.

The algorithm can be initialized in many ways, the simplest is by placing k points called centroids randomly in the data. A data point is said to be in cluster j, if it is closer to centroid μ_j, than any other centroid μ_i, $i \neq j$. Every centroid μ_j is the mean of cluster j. After each data point has been assigned to a cluster, the centroids are moved to the new mean of its cluster. These operations repeats until some stopping criterion is met. Two commonly used stopping criteria are setting the maximum number of iterations, or keep updating the centroids until they reach a stable configuration. That is

$$\sum_{i=0}^{n} \|\mu_i^{\text{old}} - \mu_i^{\text{new}}\|_2 < \epsilon$$

K-Means is sensitive to the initial placement of the centroids, and multiple runs with different initializations should be done before obtaining consistent results. A simple, but efficient implementation is provided in Algorithm 10.

Complexity

The dominating operations in Algorithm 10, are assigning a training sample to a class and updating the cluster means, which is $\mathcal{O}(kn)$ for each training sample, $\mathcal{O}(Nkn)$ for all N training samples. Total complexity is $\mathcal{O}(NknT)$, where T is the number of iterations until the stopping criterion is met.

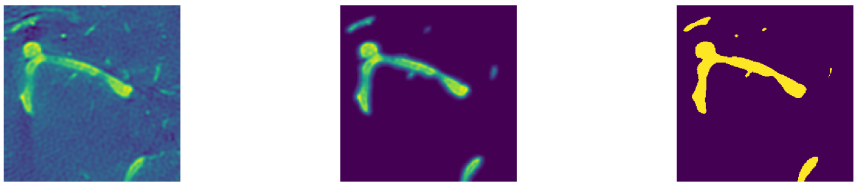
Algorithm 10: K-Means Clustering

```
Input: Training data $\mathbf{X} \in \mathbb{R}^{n \times N}$, number of clusters $k$, a stopping
       criterion
Result: Class labels $\mathbf{L} \in \mathbb{Z}_+^N$
1  Initialize centroid, $\mathbf{M} = [\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k] \in \mathbb{R}^{n \times k}$;
2  while stopping criterion not reached do
3      Reset labels;
4          $\mathbf{Y} = \{0\}^N$;
5          sum $= \{0\}^{n \times k}$;
6          $\mathbf{s} = \{0\}^k$;
7      Assigns new labels to all samples;
8          for $i = 0, ..., N$ do
9              $j = \underset{j=0,...,k}{\text{argmin}} \quad \|\mathbf{x}_i - \boldsymbol{\mu}_j\|_2$;
10             $\text{sum}_j = \text{sum}_j + \mathbf{x}_i$;
11             $\mathbf{s}_j = \mathbf{s}_j + 1$;
12             $\mathbf{L}_i = j$;
13         end
14     Update and move centroids;
15         for $j = 0, ..., k$ do
16             $\boldsymbol{\mu}_j = \frac{1}{\mathbf{s}_j} \cdot \text{sum}_j$;
17         end
18 end
```

Applications to CTA

K-Means is not able to work as a standalone vessel extraction method, mainly due to image brightness still being an important factor. Thus, we use it as a way to enhance vessel features in CTA images. The idea is that a sufficiently clean image can be clustered into two classes. One small cluster containing mostly arteries and a second, larger cluster, containing everything else. After running K-Means on the image volume we can adjust the resulting labels, $\mathbf{L}$, such that the label corresponding to the smaller cluster have the value one and the larger are zero. We then create a new version of the input data containing only the smallest cluster, $\mathbf{Y} = \mathbf{X} \odot \mathbf{L}$. This multiplication will set all image patches not in the smaller cluster to zero. To generate the final enhanced image, a convex combination is used. Choose $0 \leq \alpha \leq 1$, then $\mathbf{X}_e = \alpha\mathbf{X} + (1 - \alpha)\mathbf{Y}$ defines the vessel enhanced image. Choosing $\alpha = 1$, we obtain the original image, choosing α closer to zero, we enhance arteries.

Figure 4.6 shows the results of vessel enhancement with K-Means. The algorithm is trained on all overlapping 2D image patches of size 6×6 using 2D 256×256 slices from a $256 \times 256 \times 256$ volume. 3D patches can also be used. The choice for using 2D or 3D image patches is the same as for image denoising or inpainting (Sections 3.6 and 3.7), using 2D patches is computationally more efficient, but 3D patches give better results. The image in Figure 4.6b, is constructed by setting $\alpha = 0.3$ in the expression for $\mathbf{X}_e$.

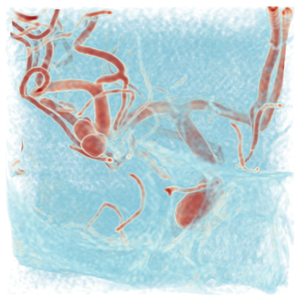


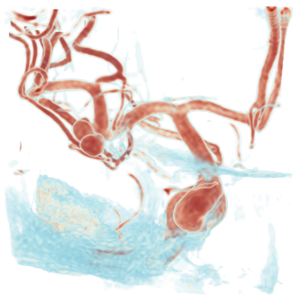
(a) 256×256 volume slice (b) Enhanced vessel image (c) K-Means and minimum threshold

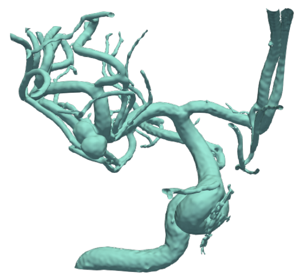
Figure 4.6: Clustering with K-Means

K-Means can be applied directly on the sparse codes. If a sparse representation is available, it is more efficient to cluster the sparse codes rather than full image patches. The sparse codes can be compressed using *Compressed Sparse Column*-matrix format, which only stores nonzero elements in each column. This approach reduces both running time and memory consumption. The size of the image patches should be chosen such that the smallest side in the patches have approximately the same size as the diameter of the smallest vessels.

Figure 4.7 shows the result of K-Means with 3D image patches and entropy-threshold. Figure 4.7d, is the resulting segmentation when using feature enhancement, which we see provide a better result compared to Figure 4.7c, where no feature enhancement is used.


(a) $256 \times 256 \times 256$ CTA image


(b) K-Means enhanced


(c) Entropy threshold on CTA image

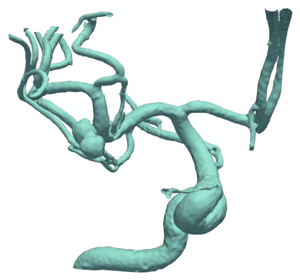

(d) Entropy threshold on enhanced image

Figure 4.7: Successful vessel extraction with K-Means and thresholding

4.3 Hessian-Based Methods

In this section, we present two methods for image enhancement and segmentation respectively, based on the structure of the Hessian matrix of the image. These methods are expected to show a better general performance compared to the previous methods, as both structure and brightness is examined when labeling the points. Hessian-based methods combine multiscale analysis, and analysis of the eigenvalues and eigenvectors of the Hessian matrix to determine what kind of structure (plate-, tubular-, or blob-like) each point belongs to. In a 3D image, a plate-like structure will have large variations in its intensity when moving in only one direction (height, width or depth). A tubular (vessel) structure will show large intensity variations in two directions, and a blob-like structure shows large variations in all directions, and can in many cases be considered as noise. In summary, we are interested in extracting points belonging to tubular structures.

Before we can define the details of the Hessian-based segmentation techniques, we need to define how an image is represented at multiple scales. This is done using a *Scale-Space representation.*

Scale-Spaces

A scale-space representation, is a way to describe an image at multiple resolutions [51, 63]. The idea is that different structures, or elements, is best described at different scales, see Figure 4.8. To obtain the scale-space representation, the initial image $I_0(\mathbf{x})$ is embedded into a family of derived images $I(\mathbf{x};t)$, where t denotes the scale. Lindeberg [51], terms this operation as *scale-space smoothing.*

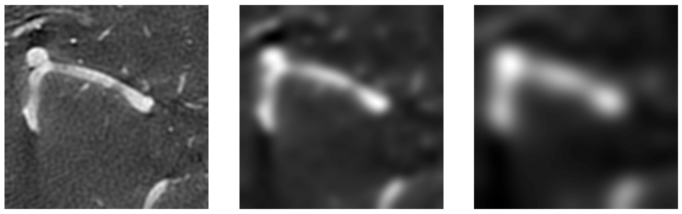


Figure 4.8: (256×256) CTA slice at three different scales. *Left:* Original image with $t = 0$, *middle:* $t = 6$, and *right:* Coarse scale with $t = 12$

Scale-space smoothing can be done in multiple ways, but the most useful is to filter the image with a Gaussian kernel, $G(\mathbf{x};t)$. The n-dimensional kernel, with $\mathbf{x} \in \mathbb{R}^n$ is defined as

$$G(\mathbf{x};t) = \frac{1}{\sqrt{2\pi t^2}^n} \cdot e^{-\frac{\|\mathbf{x}\|_2^2}{2t^2}} \tag{4.1}$$

Where t defines the scale. Given a set of scales

$$\mathcal{T} \subseteq \mathbb{R}^+ = \{t \in \mathbb{R} : \ t > 0\},$$

the scale space of an image $I_0(\mathbf{x})$ is defined

$$S(I_0(\mathbf{x}); \mathcal{T}) = \{I_0(\mathbf{x}) \star G(\mathbf{x}, t) : \ t \in \mathcal{T}\}$$

$f \star g$ denotes the convolution of f with g. Additionally, the image at scale $t = 0$ is defined as $I_0(\mathbf{x})$

Images created with larger t correspond to images at coarser resolutions, while smaller t corresponds to a finer resolution. The details in the image decrease when the scale t increase.

Normalized Derivatives

The Hessian-based segmentation techniques uses scale-spaces to represent images and their derivatives. One challenge associated with scale spaces, is that the Gaussian kernel and its derivatives are decreasing functions of scale, and we have

$$\lim_{t\to\infty} \frac{\partial^n}{\partial x^n} G(x;t) = 0 \quad \forall n \geq 0$$

Therefore, by comparing an image at multiple scales, the high resolution images (small t) will take precedence over the low resolution (large t) versions. Normalized derivatives were introduced by T. Lindeberg [50] to make comparisons between scales fair. The derivative of an image is defined in terms of convolutions with the derivative of the Gaussian

$$\frac{\partial}{\partial x_i} I(\mathbf{x};t) = I_0(\mathbf{x}) \star \frac{\partial}{\partial x_i} G(\mathbf{x};t) \tag{4.2}$$

To normalize, such that $\frac{\partial}{\partial x_i} I(\mathbf{x};t)$ can be compared across scales, it is scaled by a factor t^γ

$$\frac{\partial}{\partial x_i} I(\mathbf{x};t) = t^\gamma I_0(\mathbf{x}) \star \frac{\partial}{\partial x_i} G(\mathbf{x};t) \tag{4.3}$$

When $\gamma = 1$, equation (4.3) is called the *normalized*-derivative of $I_0(\mathbf{x})$. Otherwise, it is called γ-*parameterized* derivative, and the value of γ is important when equation (4.3) is used to detect shapes in the image, Section 4.3.

Analysis of the Hessian Matrix

The Hessian-based methods presented in this thesis are based on the same idea of examining the eigenvalues of the Hessian matrix. By examining the eigenvalues corresponding to each point $\mathbf{x}_0 = (i, j, k)$, we can determine the type of structure the point $\mathbf{x}_0$ belongs to [35, 52, 70]. The Hessian matrix describes the second-order structure of local intensity variations around each point $\mathbf{x}_0$. These structures are reflected in the eigenvalues of the Hessian matrix. Let $I(\mathbf{x};t)$ be a 3D image at scale t, then consider the 2nd degree Taylor expansion in the neighborhood of a point $\mathbf{x}_0$

$$I(\mathbf{x}_0 + h\mathbf{x};t) \approx I(\mathbf{x}_0;t) + h\mathbf{x}^T \nabla_t I(\mathbf{x}_0) + h^2 \mathbf{x}^T \mathcal{H}_{\mathbf{x}_0,t}\mathbf{x} \tag{4.4}$$

Where $\nabla_t I \in \mathbb{R}^{H \times W \times D \times 3}$ is the normalized gradients at scale t. $\mathcal{H}_{\mathbf{x}_0,t} \in \mathbb{R}^{3\times 3}$ is the Hessian matrix for point $\mathbf{x}_0$ at scale t, defined as

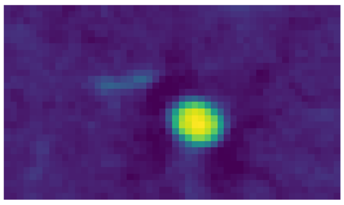
$$\mathcal{H}_{\mathbf{x}_0,t}(i,j) = \frac{\partial^2}{\partial x_i x_j} I(\mathbf{x}_0;t)$$

Let $|\lambda_1| \leq |\lambda_2| \leq |\lambda_3|$ be the eigenvalues of $\mathcal{H}_{\mathbf{x},t}$, then for an ideal bright tubular (vessel) structure the following holds [35].

$$\lambda_1 \approx 0 \tag{4.5}$$
$$\lambda_2 \ll \lambda_1 \tag{4.6}$$
$$\lambda_2 \approx \lambda_3 \tag{4.7}$$

Let $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ be the eigenvectors corresponding the eigenvalues $\lambda_1, \lambda_2, \lambda_3$. Further, assume the eigenvalues of the Hessian matrix of a point $\mathbf{x}$ follows the structure (4.5)-(4.7), where λ_1 is close to zero, λ_2 and λ_3 are both negative and of large magnitude. Then, any change in the direction of $\mathbf{v}_1$, $I(\mathbf{x} + r\mathbf{v}_1)$, results in a small change in intensity. Since λ_2 and λ_3 are negative and of large magnitude, any change in the directions of $\mathbf{v}_2$ or $\mathbf{v}_3$ will result in large changes towards a lower intensity. All relationships between size and magnitude of the eigenvalues and shape can be seen in Figure 4.10.

An example of an ideal vessel is shown in Figure 4.9. Let $\mathbf{x}$ be the point on the center of the vessel in Figure 4.9a, which corresponds to the top of the peak in Figure 4.9b. The eigenvalues of the Hessian matrix to the point $\mathbf{x}$ then follows a structure similar to (4.5)-(4.7). We have $\lambda_1 \approx 0$, and any change in the direction of $\mathbf{v}_1$ results in very small changes in intensity. In Figure 4.9a, $\mathbf{v}_1$ is the direction normal to the cross section, and is also the direction of the vessel. The other eigenvalues, λ_2 and λ_3, are negative with large magnitude, thus, changes in the directions of $\mathbf{v}_2$ or $\mathbf{v}_3$ results in large changes towards a lower intensity. This can be visualized in Figure 4.9b, $\mathbf{v}_2$ and $\mathbf{v}_3$ points away from the top of the peak, and we see that the intensity decrease quickly when moving away from the peak. Moving away from the peak is equivalent to moving away from the center of a vessel and towards its boundary.


(a) Cross section of a vessel

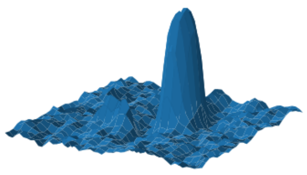

(b) Intensity profile of the image. A higher peak gives higher intensity

Figure 4.9: Vessel cross section and its intensity profile

2D		3D			Structure
λ_1	λ_2	λ_1	λ_2	λ_3	
N	N	N	N	N	Noise, no structure
		L	L	H-	Plate-like (bright)
		L	L	H+	Plate-like (dark)
L	H-	L	H-	H-	Tubular (bright)
L	H+	L	H+	H+	Tubular (dark)
H-	H-	H-	H-	H-	Blob-like (bright)
H+	H+	H+	H+	H+	Blob-like (dark)

Figure 4.10: Relationships between local image structures and Hessian eigenvalues. L is a low value, H for high and N is noisy, +/- denotes the sign.

Frangi Vessel Enhancement Filter

Frangi et al. [35], have developed a multiscale vessel enhancement filter that measures to which extent a point resembles a vessel, called the vesselness of a point. Based on the relationships in Figure 4.10, each point is assigned a vesselness value between zero and one to create a feature image, similar to K-Means in Section 4.2. Points with a value close to one are assumed to belong to a vessel. Due to its multiscale approach, *Frangi Vessel Enhancement Filter* is better equipped to handle images with small vessels compared to K-Means, for instance. Used together with thresholding, the Frangi Vessel Enhancement filter creates a good and efficient method for extracting vessels.

The Frangi vessel filter works by assigning each point in the image a vesselness values according to the *vesselness function*, defined as

$$\mathcal{V}(t) = \begin{cases} 0 & \text{if } \lambda_2 > 0 \text{ or } \lambda_3 > 0 \\ \Big[1 - \exp\Big(-\frac{\mathcal{R}_\mathcal{A}^2}{2\alpha^2}\Big)\Big] \exp\Big(-\frac{\mathcal{R}_\mathcal{B}^2}{2\beta^2}\Big)\Big[1 - \exp\Big(-\frac{\mathcal{S}^2}{2c^2}\Big)\Big] & \text{otherwise} \end{cases} \tag{4.8}$$

Here, α, β and c are parameters that controls the sensitivity of the filter, and $\mathcal{R}_\mathcal{A}$, $\mathcal{R}_\mathcal{B}$ and $\mathcal{S}$ are geometric ratios that are used to distinguish between tubular, plate- and blob-like structures. $\mathcal{V}(t)$ is designed to take large values if the point corresponding to the three ratios is vessel-like. Details on how to compute the vesselness response is given in Algorithm 11.

Algorithm 11: Frangi Vesselness Filter

```
   Input: Image $I_0 \in \mathbb{R}^{H\times W\times D}$, set of scales $\mathcal{T}$
   Result: Feature image $\mathcal{V}^* \in \mathbb{R}^{H\times W\times D}$
 1 for $t \in \mathcal{T}$ do
 2     Calculates partial derivatives, equation (4.3);
 3         $$I_{ij} = t^\gamma I_0(\mathbf{x}) \star \frac{\partial^2}{\partial x_i x_j} G(\mathbf{x}; t)$$
 4     For each $\mathbf{x} = (x, y, z) \in [0, H] \times [0, W] \times [0, D]$;
 5         $\mathcal{H}_{\mathbf{x},t}(i, j) = I_{i,j}[x, y, z]$;
 6         $\lambda_1, \lambda_2, \lambda_3 = \text{eigvals}(\mathcal{H}_{\mathbf{x},t})$;
 7         Calculate $\mathcal{V}(t)$ (4.8) for point $\mathbf{x}$ with $\lambda_1, \lambda_2, \lambda_3$;
 8 end
 9 Get maximum response for each point over all scales;
10     $\mathcal{V}^*[x, y, z] = \max_{t\in\mathcal{T}} \mathcal{V}(t)[x, y, z]$;
```

The first ratio, $\mathcal{R}_\mathcal{B}$, accounts for the deviation from a blob-like structure

$$\mathcal{R}_\mathcal{B} = \frac{|\lambda_1|}{\sqrt{|\lambda_2 \lambda_3|}}, \tag{4.9}$$

and is close to zero for both tubular and plate-like structures. $\mathcal{R}_\mathcal{B}$ cannot distinguish between tubular and plate-like structures, therefore, another ratio is needed

$$\mathcal{R}_\mathcal{A} = \frac{|\lambda_2|}{|\lambda_3|} \tag{4.10}$$

$\mathcal{R}_\mathcal{A}$ is close to zero for plate-like structures and close to one for tubular structures. For a tubular structure, we have $\mathcal{R}_\mathcal{B} \approx 0$ and $\mathcal{R}_\mathcal{A} \approx 1$. These two ratios are intensity invariant, and background noise can give a high response for vesselness. To account for this we have $\mathcal{S}$, which Frangi et al. defined as the Frobenius norm of the Hessian matrix [35]

$$\mathcal{S} = \|\mathcal{H}_{\mathbf{x},t}\|_F = \sqrt{\lambda_1^2 + \lambda_2^2 + \lambda_3^2}$$

$\mathcal{S}$ is low for background points and high for vessels. Thus, with these three ratios, $\mathcal{R}_\mathcal{A}$, $\mathcal{R}_\mathcal{B}$ and $\mathcal{S}$, the vesselness function is able to distinguish between vessel-like structures and everything else.

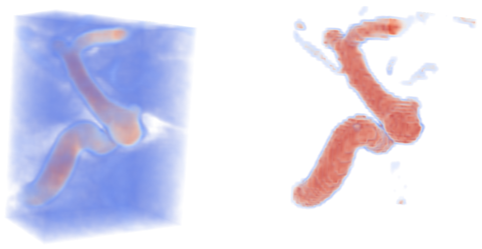


Figure 4.11: Original image volume and its vesselness response computed at scales $\mathcal{T} = \{1, 2, 3.5, 5\}$

The vesselness response, equation (4.8), is calculated at multiple scales $\mathcal{T} = \{t_{\text{min}}, t_1, ..., t_{\text{max}}\}$, and the final feature image is obtained by taking the maximum value of a point $\mathbf{x}$ over all scales,

$$\mathcal{V}^*[\mathbf{x}] = \max_{t \in \mathcal{T}} \quad \mathcal{V}(t)[\mathbf{x}]$$

The scales are chosen such that t_{min} and t_{max} corresponds to the width of the smallest and largest vessels.

Complexity

The dominating operations in Algorithm 11 are computation of the partial derivatives and iterating through all points in the image. Finding the eigenvalues of a 3×3 matrix is very fast, and computing the response (4.8) is also very fast, and can be considered as constant operations.

The partial derivatives are computed efficiently in the Fourier domain. The *Convolution Theorem* [11], states that $f \star g = \mathcal{F}^{-1}(\mathcal{F}(f) \odot \mathcal{F}(g))$, where $\mathcal{F}, \mathcal{F}^{-1}$ are the Fourier transform and its inverse. The complexity of convolution in the Fourier domain is $\mathcal{O}(HWD \log(HWD))$. To iterate through all points and compute $\mathcal{V}$, is $\mathcal{O}(HWD)$. Thus, the complexity of the Frangi filter is equivalent to convolutions in the Fourier domain.

Eigenvector-Based Vessel Extraction

The next approach is based on the eigenvectors of the Hessian matrix, and is similar to the *Frangi Vessel Enhancement Filter*. The eigenvectors are included as an additional source of information for describing local intensity variations. The Frangi filter looks only at the eigenvalues of the Hessian matrix to determine if a point belongs to a vessel. Hence, it requires high contrast between the vessels and the background in order to perform well. By using the eigenvectors of the Hessian matrix to a point $\mathbf{x}$, the *eigenvector-based vessel extraction* method will analyze the local structures around $\mathbf{x}$. Therefore, the *eigenvector-based vessel extraction* method is less dependent on high contrast between the vessels and the background to accurately label a point as a vessel.

The general idea is that based on the information stored in the eigenvalues and eigenvectors, we can estimate the medial axis (centerline) and radius of the vessels [48, 5]. Let $I(\mathbf{x}; t_i) \in \mathbb{R}^{H \times W \times D}$ be a 3D image at scale t_i, $\nabla_{t_i}(\mathbf{x})$ its normalized derivatives, and the eigenvalues and eigenvectors are as before, then the point $\mathbf{x}_j$ is on the medial axis of a vessel with radius r if

$$R_{t_i}(\mathbf{x}_j) = \frac{1}{N} \sum_{i=0}^{N-1} -\nabla_{t_i}(\mathbf{x}_j + r\mathbf{v}_\alpha) \cdot \mathbf{v}_\alpha, \tag{4.11}$$

is a local maximum. $(\mathbf{x}_j, t_i)$ is a local maximum if $R_{t_i}(\mathbf{x}_j) \geq R_{t_{i\pm1}}(\mathbf{x}_j)$, $R_{t_i}(\mathbf{x}_j) \geq R_{t_i}(\mathbf{x}_j \pm \mathbf{v}_2)$, and $R_{t_i}(\mathbf{x}_j) \geq R_{t_i}(\mathbf{x}_j \pm \mathbf{v}_3)$. R_t in equation (4.11) is the medialness response of $\mathbf{x}_j$, and is large if $\mathbf{x}_j$ is at the center of a vessel [48]. The points $\mathbf{x}_j + r\mathbf{v}_\alpha$, defines N points on a circle around $\mathbf{x}_j$ with radius r, and $\mathbf{v}_\alpha = \cos(\alpha)\mathbf{v}_2 + \sin(\alpha)\mathbf{v}_3$ and $\alpha = 2\pi i/N$. Further, the set of local maximum points, $\{(\mathbf{x}_j, t_i)\}_j$, is the centerlines of the vessels in the image. To obtain the vessel radius r, $R_{t_i}(\mathbf{x}_j)$ is computed with increasing r until $R_{t_i}(\mathbf{x}_j)$ starts to decrease. The vessels are then reconstructed from the centerlines by drawing disks spanned by $\mathbf{v}_2$ and $\mathbf{v}_3$ normal to $\mathbf{v}_1$, with radius r around each local maximum point $\mathbf{x}_j$.

In the provided segmentation pipeline, the model is not used exactly as proposed by Krissian et al. in [48], due to the assumption that the cross section of a vessel is circular[1]. An aneurysm cannot be assumed to always have a circular cross section. The method used in our pipeline, Algorithm 12, is a relaxed version of the method proposed by Krissian et al.

[1]The implementation of maximum medial response is found in `dictlearn.detection.tube`

Algorithm 12: Hessian Eigenvector Filter

```
   Input: Image $I_0 \in \mathbb{R}^{H\times W\times D}$, set of scales $\mathcal{T} = \{t_{\min}, t_1, ..., t_{\max}\}$
   Result: Vessel image $R^* \in \mathbb{R}^{H\times W\times D}$
1  for $t \in \mathcal{T}$ do
2      Calculates partial derivatives, Equation (4.3);
3          $$I_{ij} = t^{\gamma} I_0(\mathbf{x}) \star \frac{\partial^2}{\partial x_i x_j} G(\mathbf{x}; t)$$
4      For each $\mathbf{x} = (x, y, z) \in [0, H] \times [0, W] \times [0, D]$;
5          $\mathcal{H}_{\mathbf{x},t}(i, j) = I_{i,j}[x, y, z]$;
6          $(\lambda_1, \mathbf{v}_1), (\lambda_2, \mathbf{v}_2), (\lambda_3, \mathbf{v}_3) = \mathrm{eig}(\mathcal{H}_{\mathbf{x},t})$;
7          if $\lambda_2 < 0$ and $\lambda_3 < 0$ then
8              Calculate $R_t(\mathbf{x})$ for point $\mathbf{x}$ with $r = \sqrt{3t}$;
9          end
10 end
11 Get maximum response for each point over all scales;
12     $\hat{R}^*[x, y, z] = \max_{t\in\mathcal{T}} R_t[x, y, z]$;
13 Create output image $R^*$ by setting to zero all points not included in the
     largest connected component in $\hat{R}^*$;
```

Algorithm 12 will compute R_t at all points resembling a vessel (line 7-8), on a given scale t. Then, taking the maximum response over the scales ensures that both small and large vessels are captured. The input image I_0, is scaled to have range $[-1, 1]$, therefore values in $\hat{R}^*$ corresponding to negative values in the input image corresponds to the background or noise, and they can be discarded. The remaining values in $\hat{R}^*$ contains the vessels, but also some points that are part of the background or noise. To extract the vessels from $\hat{R}^*$, we extract the points contained in the largest connected component in $\hat{R}^*$. The parameters r, and γ are fixed. Krissian et al. [48] found $r = \sqrt{3t}$ and $\gamma = 1$ to make the response R_t scale-invariant.

This method is very efficient, its complexity is equivalent to convolutions in the Fourier domain. The scales at which to compute R_t are given as arguments, where $t_{\min}$ corresponds to the width of the smallest vessel to extract, and $t_{\max}$ the largest. Most vessels within this range will be extracted, but some limitations exist. The first, and most important, is that some change in intensity between the vessel boundary and the background is required. If the contrast at boundary is very low, or the boundary is very wide (blurred), then Algorithm 12 cannot determine where the boundary between the background and vessel should be, and such points are not correctly marked as a vessel, see Figure 4.14. The second is when a vessel is split in two due to a hole, then only one of its parts are included in the largest connected component, and the other will be discarded, Figure 4.13.

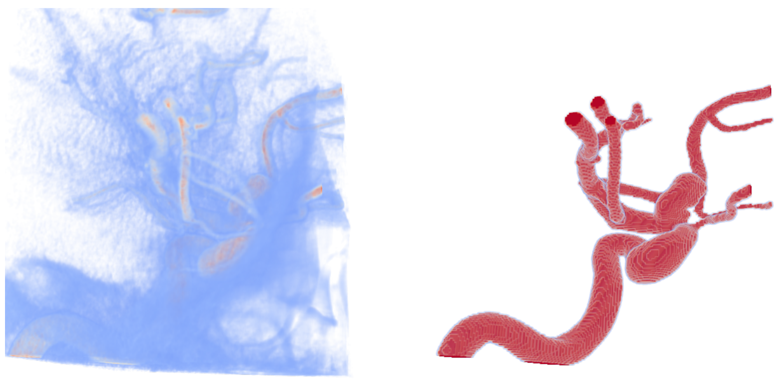
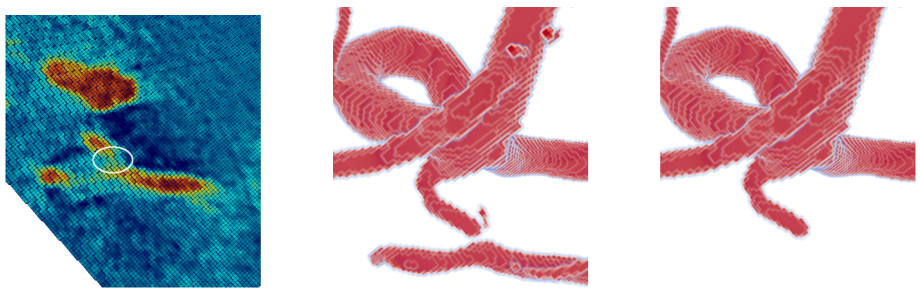


Figure 4.12: Vessels extracted from a denoised image with Algorithm 12



(a) Contours of vessel with low contrast

(b) Vessel split in two due to low contrast edges

(c) Vessel discarded by extraction largest connected component

Figure 4.13: Vessel extraction with Algorithm 12. The low contrast, circled area, results in a hole in the vessels extracted (middle) with Algorithm 12, then the smallest part is discarded in extraction of largest connected component.

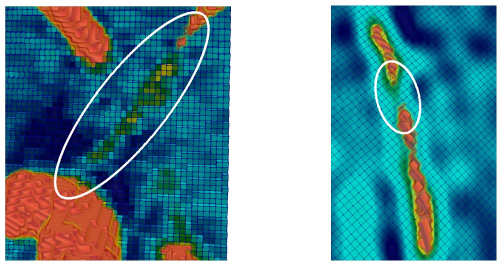


Figure 4.14: Two examples of vessel with poor contrast not being marked correctly. The vessels should pass through the circled areas. The red points are vessels detected by Algorithm 12. Each corner in the grids correspond to one point **x** in the image volume

4.4 Active Contours

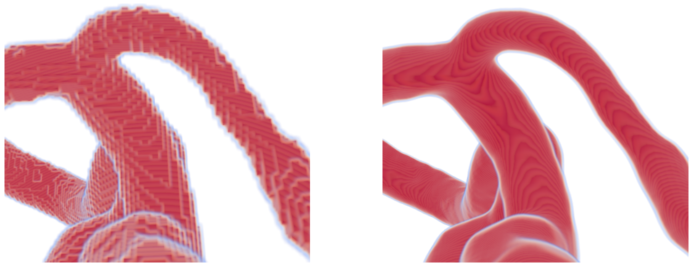
So far we have seen segmentation performed by thresholding and Hessian-based methods. All these methods have their advantages and disadvantages, and can do very well at extracting vessels. However, the resulting geometries can be somewhat rough, since the methods are based on hard thresholds or hard requirements to label a point as either *artery* or *not-artery*, Figure 4.15a. In this section, we introduce the method *Geodesic Active Contours* [13], which is the final method used in our vessel extraction pipeline. *Geodesic Active Contours* will smooth and move the boundaries of an initial, rough segmentation (Figure 4.15a), such that the final surface better fits the true boundaries, see Figure 4.15b.

Active contours, also known as *snakes* are deformable models that grows or shrinks an initial surface towards the boundary of the object to be detected [13, 47, 79, 78]. The deformation of the initial surface is obtained by minimizing a functional that is designed to have its minimum at the object boundary.

Let $\mathcal{C}(q)$: $[0,1] \to \mathbb{R}^2$ be a parametrized planar curve, and $I : [0,H] \times [0,W] \to \mathbb{R}$ the image in which we want to detect boundaries. The goal of geodesic active contours [13] is then to find the curve $\mathcal{C}$, that minimize the following energy

$$E(\mathcal{C}) = \alpha \int_0^1 |\mathcal{C}'(q)|^2 dq + \lambda \int_0^1 |\nabla I(\mathcal{C}(q))|)^2 dq \tag{4.12}$$

The first term controls the smoothness of the contour, also called its internal forces. The second term, is its external force and attracts the contour



(a) Output from Hessian Eigenvector filter

(b) Output from *Geodesic Active Contours*

Figure 4.15: Results from Hessian Filter, Algorithm 12, and *Active Contours* segmentation

towards the object boundary. In equation (4.12) the external force works as an edge detector, and can be generalized. Let $g : [0, H] \times [0, W] \to \mathbb{R}^+$, then

$$E(\mathcal{C}) = \alpha \int_0^1 |\mathcal{C}'(q)|^2 dq + \lambda \int_0^1 g(\mathcal{C}(q)))^2 dq \tag{4.13}$$

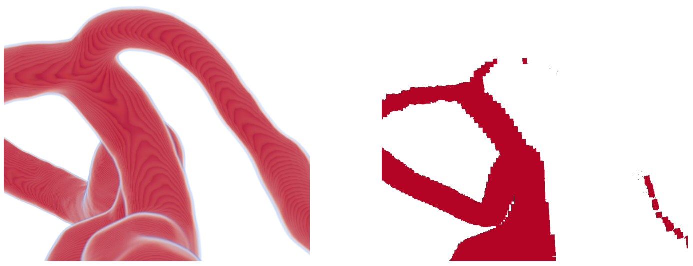
The function g is called the *edge map*. At an ideal edge g is expected to be zero, that is, g is small on the object's boundaries, and large otherwise. If g is created based on a 3D CTA image, it should be close to zero on vessel boundaries and close to one otherwise. The choice of the edge map g is important for the quality of the solution, a better solution is obtained with a better map. In the provided segmentation pipeline, the edge map is defined as

$$g(\mathbf{x}) = \frac{1}{1 + |\nabla I(\mathbf{x})|},$$

where ∇I is the image gradient defined as the maximum normalized derivative (Section 4.3) over multiple scales $\mathcal{T}$. For all points $\mathbf{x} = (i, j, k)$ the maximum gradient is

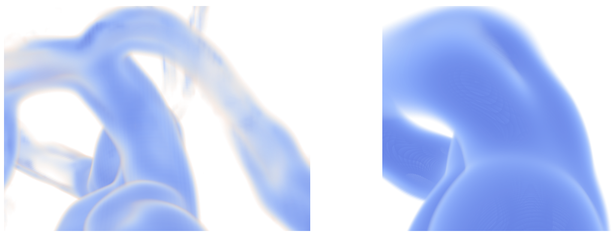
$$\nabla I(\mathbf{x}) = \max_{t \in \mathcal{T}} \quad \left|\frac{\partial}{\partial x} I(\mathbf{x};\, t)\right| + \left|\frac{\partial}{\partial y} I(\mathbf{x};\, t)\right| + \left|\frac{\partial}{\partial z} I(\mathbf{x};\, t)\right|$$

This multiscale approach for computing the gradient is important for capturing the boundaries of both small and large vessels. Figure 4.16 shows the outputs from *Geodesic Active Contours* with both a high and a low quality edge map. By a high quality edge map, we mean a map g with values close to zero, and show high contrast on the vessel boundaries. A low quality edge map is a map that provide an inaccurate representation of the vessel boundaries. For example, by having low contrast or not having the boundaries at the correct points, such as Figure 4.17b. Figure 4.17, shows the edge maps used for generating the results in Figure 4.16. The edge map is also used as a stopping criterion, and segmentation with a high quality edge map will be faster [13]. The figures below show a $(50 \times 50 \times 50)$ subsection of a $(200 \times 200 \times 200)$ image, and segmentation with a good edge maps takes approximately 10 seconds, versus 80 seconds with the bad edge map.



(a) Generated with edge map 4.17a (b) Generated with edge map 4.17b

Figure 4.16: *Geodesic Active Contours* segmentation with the same initial surface, but different edge maps.



(a) High quality edge map (b) Low quality edge map

Figure 4.17: High quality map is computed on scales $\mathcal{T} = \{0.5, 1, 2\}$ and the low quality map $\mathcal{T} = \{7\}$

In the provided segmentation pipeline, we set the output from the *Hessian Eigenvector Filter*, Algorithm 12, to be the initial surface, called the *seed*, for *Geodesic Active Contours*. The main advantage to this, is increased computational efficiency. The vessels extracted from the Hessian filters are close to the true surfaces, and the amount of deforming to be done is low. The next advantage, is that the Hessian filters are better equipped to extract smaller vessels. For *Geodesic Active Contours* to be able to extract a vessel, it needs to have sufficiently high contrast in the image. As we have seen, this is often not the case for smaller vessels. The default configuration (Section 5.1), is set to do small deformations to the initial surface, which results in the topology of the final surface to be the same as the seed. The advantage to this, is again increased computational efficiency, but artifacts, such as holes or fused vessels in the seed, will be found in the final image. Inpainting (Section 3.7), or changing of the default parameters for *Geodesic Active Contours* can be used to fix these issues.

The solution to equation (4.13), is given by a a geodesic curve in Riemannian space induced from the image I, and is obtained using a variational approach [13]. A very efficient solver for equation (4.13), for both 2D and 3D images is implemented in *Insight Segmentation and Registration Toolkit* (ITK) [43].

Chapter 5

Extraction of Patient-Specific Geometries

In this Chapter, we give an overview of the complete pipeline for extracting patient specific geometries. The provided segmentation pipeline[1] combines the techniques from the previous chapters into one large sequential model. The default model is created with the goal of giving good results for many different images, and consist of steps (1)-(3) below.

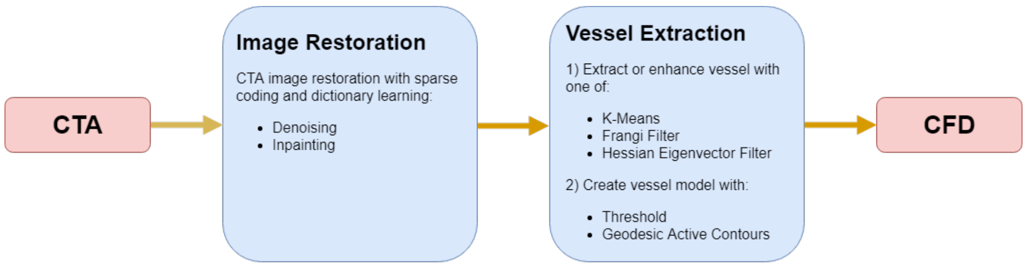

CTA
Image Restoration
CTA image restoration with sparse coding and dictionary learning:
• Denoising
• Inpainting
Vessel Extraction
1) Extract or enhance vessel with one of:
• K-Means
• Frangi Filter
• Hessian Eigenvector Filter
2) Create vessel model with:
• Threshold
• Geodesic Active Contours
CFD

Figure 5.1: Vessel segmentation pipeline. The blue boxes represent the contribution from this work.

1. Denoising

Noise can introduce unwanted artifacts when segmenting an image. Therefore, the first step is to denoise the image. Denoising is done as described in Section 3.6, using 2D image patches. This does not provide as good result as 3D image patches, but the increased quality of the result does not outweigh the extra computational cost. The image is denoised as 2D image

[1]Found at dictlearn/scripts/surfit.py

slices with 8×8 image patches. For each slice the dictionary is trained with K-SVD, 10 iterations and a sparsity target such that approximately 10% of the sparse coefficients are nonzero. The standard deviation of the noise, σ, is estimated with *Threshold Selection by SURE* [28].

2. Seed Creation

The seed for *Active Contours* is created using Algorithm 12. Algorithm 12 extracts an estimate of the vessel network. As mentioned in Section 4.4, holes and other artifacts present in the output from Algorithm 12, will also be found in the output of Geodesic Active Contours. Therefore, inpainting might have to be applied before the output from this step is used as the seed for Geodesic Active Contours. The final operation done before the output from this step can be sent to *Active Contours* is shrinking. Then, most of the vessels are smaller than the true vessel structure, which will reduce the computations needed in the next step.

3. Active Contours

Geodesic Active Contours is the final part of this vessel extraction pipeline. The seed from the previous step is deformed according to its *edge map*. The edge map is an image with values close to zero on the vessel edges, and close to one otherwise. The edge map is created by inverting the normalized derivatives (Section 4.3) of the denoised image at different scales. The normalized derivatives have high values on the vessel edges and low elsewhere, thus inverting it produces the required structure for the edge map. Geodesic Active Contours will smooth the vessel estimate from step two, and increase the accuracy of the segmented vessels.

5.1 Configuration

As stated above, the goal of the default pipeline if to generate sensible results for a wide range of CTA images. To achieve optimal accuracy one might need to change how the pipeline segment the arteries. Therefore, the pipeline is designed to be easy to reconfigure and extend with customized methods[2]. The pipeline is configured with a YAML file that defines the input image, which methods to use, and its parameters. The full set of configuration options are emitted due to its size, but a version that will run the steps (1)-(3) above is included in Example 4. If one would rather segment

[2]Details of how this is done, will be found in the documentation on github

using K-Means feature enhancement (Section 4.2) and thresholding with the *minimum*-threshold (Section 4.1), one would change the configuration accordingly, see Example 5.

```yaml
# configuration.yml

# Global configuration
- config:
    input: image_volume.vti

# Operations to execute
# These operations are executed in the order they appear
- denoise
- create_seed
- active_contours
```

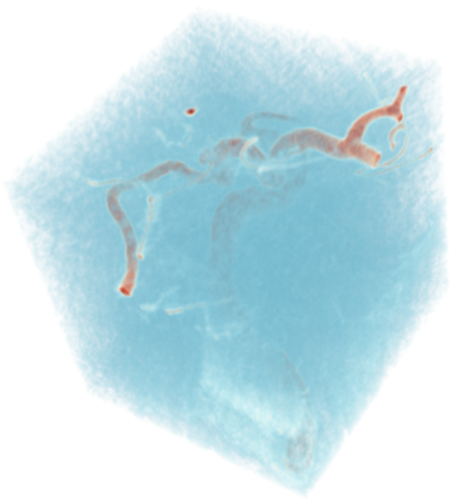
Example 4: Default configuration for vessel extraction pipeline

```yaml
# configuration.yml

# Global configuration
- config:
    input: image_volume.vti

- kmeans_enhance
- surface:
    args:
        level: minimum
```

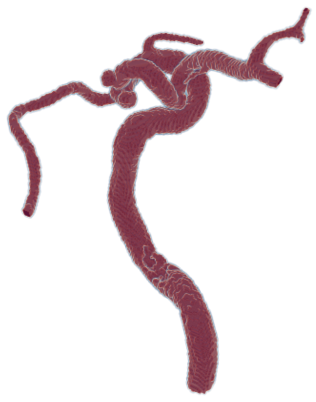
Example 5: Segmentation with K-Means feature enhancement and thresholding

5.2 Results

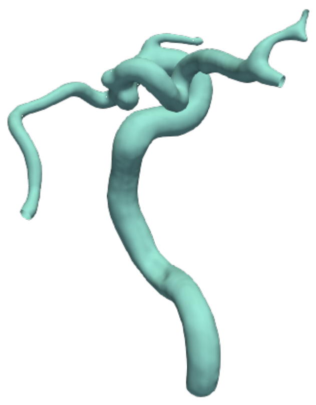
Below are results from running the default pipeline on image volumes from the Aneurisk Project [3].



(a) Input Volume

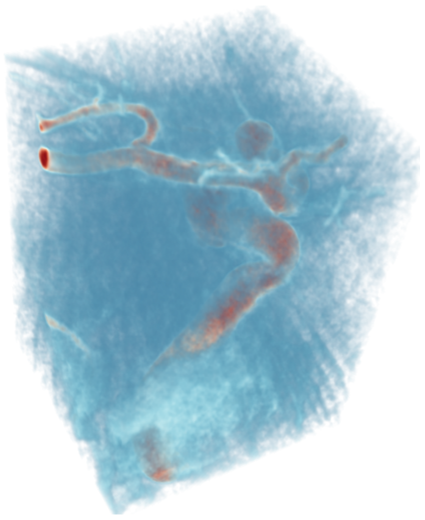


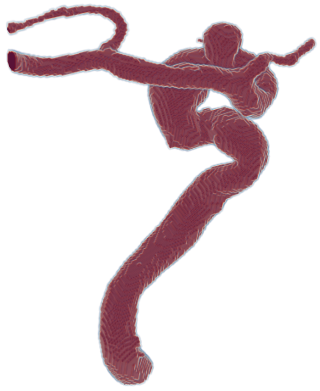
(b) Seed From Hessian Selection



(c) Output Active Contours

Figure 5.2: AneuriskData #4


(a) Input Volume


(b) Seed From Hessian Selection

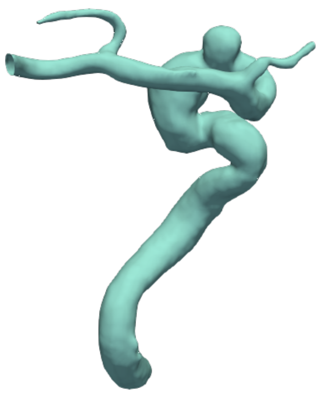
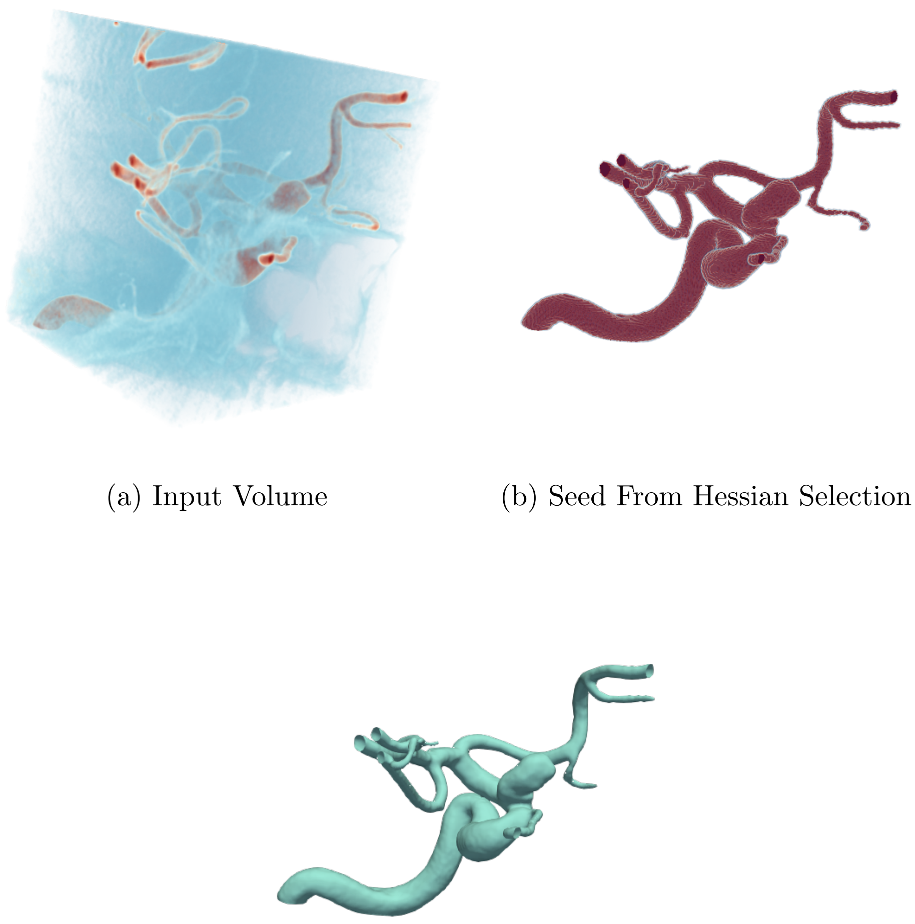

(c) Output Active Contours

Figure 5.3: AneuriskData #41


(a) Input Volume (b) Seed From Hessian Selection

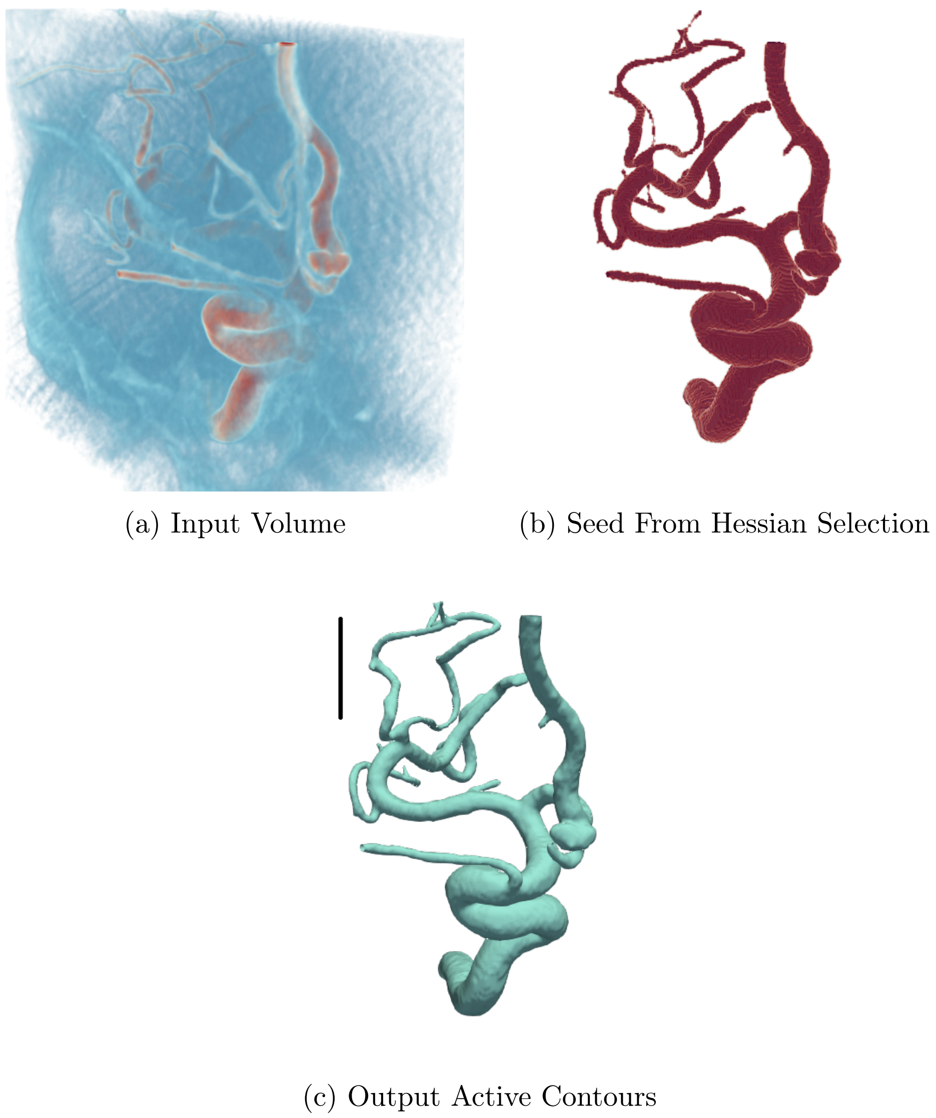
(c) Output Active Contours

Figure 5.4: AneuriskData #2

Challenges

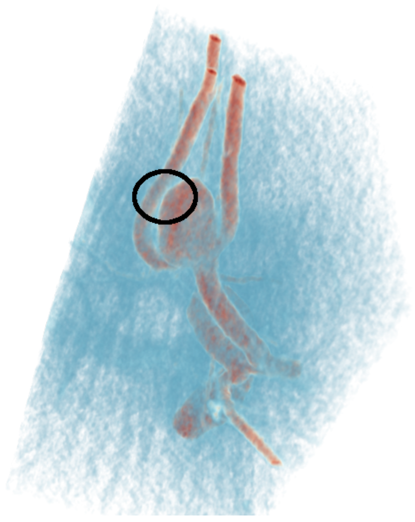
The challenges we saw for Algorithm 12 in Section 4.10, translates to the full pipeline. When the vessels are very small or close together the contrast between the vessels and the background may be weak. Then it will be hard to determine where the boundary should be placed. Figures 5.5 and 5.6 represent two such scenarios. To give some context to the size of the vessels in those figures, a four voxels wide black line is included. In Figure

5.5, there are some very small vessels that are visible in Figure 5.5b, but disappear in the output, Figure 5.5c. In this case, one would need to inpaint the seed for *active contours* to avoid losing small vessels and having uneven vessel-shape. When vessels fuse, Figure 5.6, adding more shrinking to the seed for the *active contours* will often separate the vessels, but the size of the vessels will also change, and smaller vessels may also disappear, Figure 5.6c.

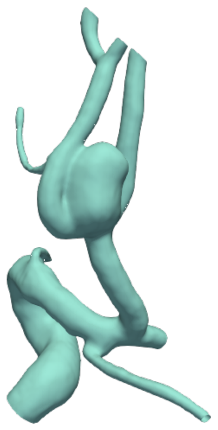

(a) Input Volume (b) Seed From Hessian Selection

(c) Output Active Contours

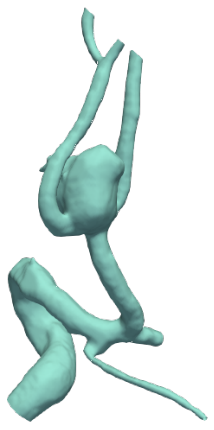
Figure 5.5: AneuriskData #8. Small vessels with low contrast. The black line is four pixels wide in the original data.



(a) Input Volume



(b) Vessels fuse together



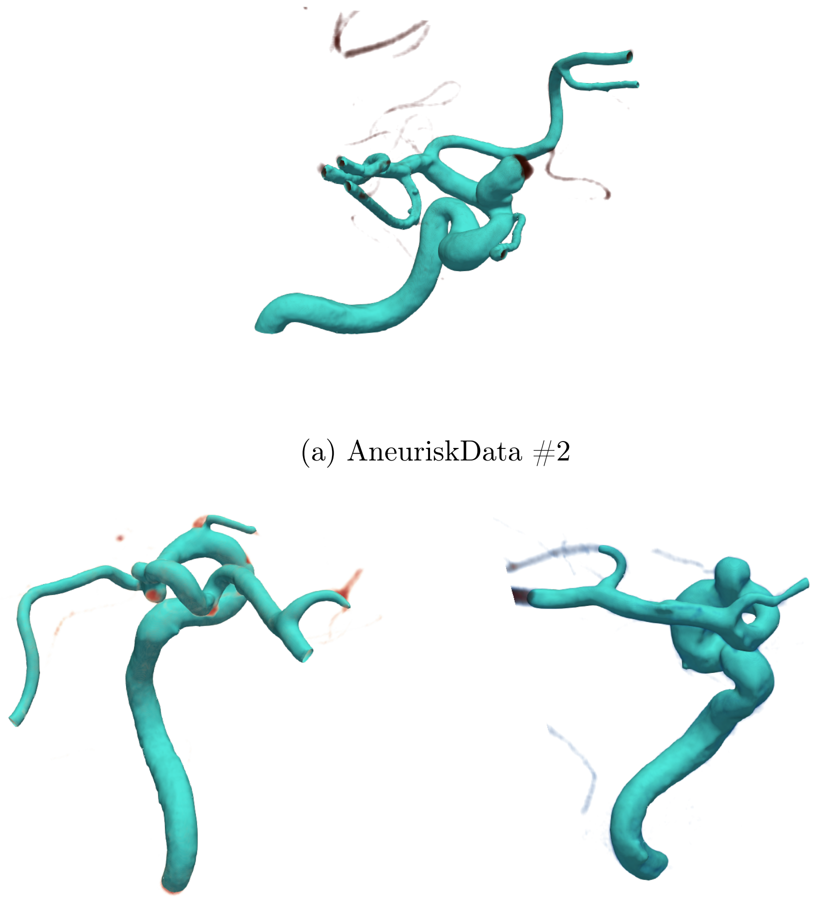
(c) Add more shrinking to avoid

Figure 5.6: AneuriskData #52. The circle in image (a) show where the vessels fuse. The border of the circle is four pixels wide in the original data.

Manual Segmentation

In Figure 5.7 below, results from manual segmentation with VMTK are presented. During manual segmentation, we aim to achieve a result at least as good as the provided pipeline. The segmentation of the CTA images in Figure 5.7 required five hours of continuous work. Segmentation with the pipeline (1)-(3) requires the creation of a configuration file, similar to

Example 4, and running a python script. The whole process of creating the configuration and running the pipeline takes around 30 minutes for all three images.



(a) AneuriskData #2

(b) AneuriskData #4

(c) AneuriskData #41

Figure 5.7: Manual segmentation of CTA volumes with VMTK, overlaid the true vessels.

Chapter 6

Conclusion and Future Work

In this thesis, we present a pipeline for automatic extraction of patient-specific geometries. In particular, we show how to sparsely represent a 3D CTA image using an overcomplete dictionary, and how sparse representation can be used for image restoration. We also found Hessian-based segmentation techniques to be efficient and accurate in extracting vessels from restored CTA volumes.

These image restoration and segmentation methods are implemented to create an automatic pipeline for extracting patient-specific geometries. This pipeline reduces the manual inputs needed compared to state-of-the-art methods. However, for images of poor quality we have to resort to image inpainting to extract an accurate representation of the vessel network, for which the mask has to be created manually. Thus, in future work we are interested in designing techniques to automatically detect and fill holes in the arteries, to remove the need for manual mask creation.

We also aim at overcoming the limits of the Hessian-based segmentation methods in regards to small and low-contrast vessels. One field for where to look for improvements is deep learning. Recently, much work has been done on extracting vessels from MR- and CT images with convolutional neural networks [57, 58, 15, 18]. The Hessian-based vesselness filter, Algorithm 11, has also been combined with a convolutional neural net in [37], which showed improved results. Following up this work is of great interest, as we think these approaches can give major improvements to our segmentation pipeline.

List of examples

Bibliography

[1] Michal Aharon, Michael Elad, and Alfred Bruckstein. **K**-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on signal processing*, 54(11):4311–4322, 2006.

[2] Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK Users' guide*. SIAM, 1999.

[3] Aneurisk-Team. AneuriskWeb project website, http://ecm2.mathcs.emory.edu/aneuriskweb. Web Site, 2012.

[4] Luca Antiga and David A Steinman. Vmtk: vascular modeling toolkit. *VMTK, San Francisco, CA, accessed Apr*, 27:2015, 2006.

[5] Karl KrissianâĂŤGrégoire MalandainâĂŤNicholas Ayache. " model-based multiscale detection and reconstruction of 3d vessels, 1998.

[6] Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.

[7] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[8] Aslak Wigdahl Bergersen. Investigating the link between patient-specific morphology and hemodynamics: Implications for aneurism initiation? Master's thesis, 2016.

[9] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific Belmont, 1999.

[10] Thomas Blumensath and Mike E Davies. Gradient pursuits. *IEEE Transactions on Signal Processing*, 56(6):2370–2382, 2008.

[11] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[12] Emmanuel J Candès and David L Donoho. Ridgelets: A key to higher-dimensional intermittency? *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 357(1760):2495–2509, 1999.

[13] Vicent Caselles, Ron Kimmel, and Guillermo Sapiro. Geodesic active contours. *International journal of computer vision*, 22(1):61–79, 1997.

[14] Juan R Cebral, Marcelo Adrián Castro, Sunil Appanaboyina, Christopher M Putman, Daniel Millan, and Alejandro F Frangi. Efficient pipeline for image-based patient-specific analysis of cerebral aneurysm hemodynamics: technique and sensitivity. *IEEE transactions on medical imaging*, 24(4):457–467, 2005.

[15] Li Chen, Yanjun Xie, Jie Sun, Niranjan Balu, Mahmud Mossa-Basha, Kristi Pimentel, Thomas S Hatsukami, Jenq-Neng Hwang, and Chun Yuan. 3d intracranial artery segmentation using a convolutional autoencoder. In *Bioinformatics and Biomedicine (BIBM), 2017 IEEE International Conference on*, pages 714–717. IEEE, 2017.

[16] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

[17] Sheng Chen, Stephen A Billings, and Wan Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of control*, 50(5):1873–1896, 1989.

[18] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: learning dense volumetric segmentation from sparse annotation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 424–432. Springer, 2016.

[19] Shane F Cotter, BD Rao, K Kreutz-Delgado, and J Adler. Forward sequential algorithms for best basis selection. *IEE Proceedings-Vision, Image and Signal Processing*, 146(5):235–244, 1999.

[20] Antonio Criminisi, Patrick Perez, and Kentaro Toyama. Object removal by exemplar-based inpainting. In *Computer Vision and Pattern*

Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on, volume 2, pages II–II. IEEE, 2003.

[21] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.

[22] Joachin Dahl and Lieven Vandenberghe. Cvxopt: A python package for convex optimization. In *Proc. eur. conf. op. res*, 2006.

[23] Geoff Davis, Stephane Mallat, and Marco Avellaneda. Adaptive greedy approximations. *Constructive approximation*, 13(1):57–98, 1997.

[24] Nicolien K de Rooij, F HH Linn, Jacob A van der Plas, Ale Algra, and G JE Rinkel. Incidence of subarachnoid haemorrhage: a systematic review with emphasis on region, age, gender and time trends. *Journal of Neurology, Neurosurgery & Psychiatry*, 2007.

[25] Sujan Dhar, Markus Tremmel, J Mocco, Minsuok Kim, Junichi Yamamoto, Adnan H Siddiqui, L Nelson Hopkins, and Hui Meng. Morphology parameters for intracranial aneurysm rupture risk assessment. *Neurosurgery*, 63(2):185–197, 2008.

[26] David L Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via âĎŞ1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[27] David L Donoho and Xiaoming Huo. Uncertainty principles and ideal atomic decomposition. *IEEE transactions on information theory*, 47(7):2845–2862, 2001.

[28] David L Donoho and Iain M Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the american statistical association*, 90(432):1200–1224, 1995.

[29] Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[30] Michael Elad and Michal Aharon. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Transactions on Image processing*, 15(12):3736–3745, 2006.

[31] Kjersti Engan, Sven Ole Aase, and J Hakon Husoy. Method of optimal directions for frame design. In *Acoustics, Speech, and Signal*

Processing, 1999. Proceedings., 1999 IEEE International Conference on, volume 5, pages 2443–2446. IEEE, 1999.

[32] Kjersti Engan, Bhaskar D Rao, and Kenneth Kreutz-Delgado. Frame design using focuss with method of optimal directions (mod). In *Proc. NORSIG*, volume 99, pages 65–69, 1999.

[33] Øyvind Evju. Computational hemodynamics in cerebral aneurysms: Robustness of rupture risk indicators under different model assumptions. 2016.

[34] Valery L Feigin, Carlene MM Lawes, Derrick A Bennett, and Craig S Anderson. Stroke epidemiology: a review of population-based studies of incidence, prevalence, and case-fatality in the late 20th century. *The Lancet Neurology*, 2(1):43–53, 2003.

[35] Alejandro F Frangi, Wiro J Niessen, Koen L Vincken, and Max A Viergever. Multiscale vessel enhancement filtering. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 130–137. Springer, 1998.

[36] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[37] Weilin Fu, Katharina Breininger, Roman Schaffert, Nishant Ravikumar, Tobias Würfl, Jim Fujimoto, Eric Moult, and Andreas Maier. Frangi-net. In *Bildverarbeitung für die Medizin 2018*, pages 341–346. Springer, 2018.

[38] Jean Jacques Fuchs. More on sparse representations in arbitrary bases. *IFAC Proceedings Volumes*, 36(16):1315–1320, 2003.

[39] Chris A Glasbey. An analysis of histogram-based thresholding algorithms. *CVGIP: Graphical models and image processing*, 55(6):532–537, 1993.

[40] Ghassan Hamarneh and Preet Jassi. Vascusynth: Simulating vascular trees for generating volumetric image data with ground truth segmentation and tree analysis. *Computerized Medical Imaging and Graphics*, 34(8):605–616, 2010.

[41] Kenneth M Hanson and Douglas P Boyd. The characteristics of computed tomographic reconstruction noise and their effect on detectability. *IEEE Transactions on Nuclear Science*, 25(1):160–163, 1978.

[42] Janice L Hinkle and Mary McKenna Guanci. Acute ischemic stroke review. *Journal of neuroscience nursing*, 39(5):285–293, 2007.

[43] Luis Ibanez, William Schroeder, Lydia Ng, and Josh Cates. The itk software guide. 2005.

[44] Tor Ingebrigtsen, Michael K Morgan, Ken Faulder, Linda Ingebrigtsen, Trygve Sparr, and Henrik Schirmer. Bifurcation geometry and the presence of cerebral artery aneurysms. *Journal of neurosurgery*, 101(1):108–113, 2004.

[45] Preet Jassi and Ghassan Hamarneh. Vascusynth: Vascular tree synthesis software. *Insight Journal*, January-June:1–12, 2011.

[46] Jagat Narain Kapur, Prasanna K Sahoo, and Andrew KC Wong. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer vision, graphics, and image processing*, 29(3):273–285, 1985.

[47] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[48] Karl Krissian, Grégoire Malandain, Nicholas Ayache, Régis Vaillant, and Yves Trousset. Model-based detection of tubular structures in 3d images. *Computer vision and image understanding*, 80(2):130–171, 2000.

[49] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2007.

[50] Tony Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–156, 1998.

[51] Tony Lindeberg. Feature detection with automatic scale selection. *International journal of computer vision*, 30(2):79–116, 1998.

[52] Cristian Lorenz, I-C Carlsen, Thorsten M Buzug, Carola Fassnacht, and Jürgen Weese. Multi-scale line segmentation with automatic estimation of width, contrast and tangential direction in 2d and 3d medical images. In *CVRMed-MRCAS'97*, pages 233–242. Springer, 1997.

[53] Julien Mairal, Francis Bach, Jean Ponce, et al. Sparse modeling for image and vision processing. *Foundations and Trends® in Computer Graphics and Vision*, 8(2-3):85–283, 2014.

[54] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pages 689–696. ACM, 2009.

[55] Julien Mairal, Michael Elad, and Guillermo Sapiro. Sparse representation for color image restoration. *IEEE Transactions on image processing*, 17(1):53–69, 2008.

[56] Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.

[57] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *3D Vision (3DV), 2016 Fourth International Conference on*, pages 565–571. IEEE, 2016.

[58] E Nasr-Esfahani, N Karimi, MH Jafari, SMR Soroushmehr, S Samavi, BK Nallamothu, and K Najarian. Segmentation of vessels in angiograms using convolutional neural networks. *Biomedical Signal Processing and Control*, 40:240–251, 2018.

[59] Valeriya Naumova and Karin Schnass. Fast dictionary learning from incomplete data. *EURASIP journal on advances in signal processing*, 2018(1):12, 2018.

[60] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[61] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 1993.

[62] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[63] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

[64] Marina Piccinelli, Alessandro Veneziani, David A Steinman, Andrea Remuzzi, and Luca Antiga. A framework for geometric analysis of vascular structures: application to cerebral aneurysms. *IEEE transactions on medical imaging*, 28(8):1141–1155, 2009.

[65] Stanislav Pyatykh, Jürgen Hesser, and Lei Zheng. Image noise level estimation by principal component analysis. *IEEE Transactions on Image Processing*, 22(2):687–699, 2013.

[66] Ron Rubinstein, Alfred M Bruckstein, and Michael Elad. Dictionaries for sparse representation modeling. *Proceedings of the IEEE*, 98(6):1045–1057, 2010.

[67] Ron Rubinstein, Michael Zibulevsky, and Michael Elad. Efficient implementation of the k-svd algorithm using batch orthogonal matching pursuit. *Cs Technion*, 40(8):1–15, 2008.

[68] Chander Sadasivan, David J Fiorella, Henry H Woo, and Baruch B Lieber. Physical factors effecting cerebral aneurysm pathophysiology. *Annals of biomedical engineering*, 41(7):1347–1365, 2013.

[69] Laura M Sangalli, Piercesare Secchi, Simone Vantini, and Alessandro Veneziani. A case study in exploratory functional data analysis: geometrical features of the internal carotid artery. *Journal of the American Statistical Association*, 104(485):37–48, 2009.

[70] Yoshinobu Sato, Shin Nakajima, Hideki Atsumi, Thomas Koller, Guido Gerig, Shigeyuki Yoshida, and Ron Kikinis. 3d multi-scale line filter for segmentation and visualization of curvilinear structures in medical images. In *CVRMed-MRCAS'97*, pages 213–222. Springer, 1997.

[71] Karin Schnass. Convergence radius and sample complexity of itkm algorithms for dictionary learning. *Applied and Computational Harmonic Analysis*, 2016.

[72] Will J Schroeder, Bill Lorensen, and Ken Martin. *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware, 2004.

[73] Jean-Luc Starck, Emmanuel J Candès, and David L Donoho. The curvelet transform for image denoising. *IEEE Transactions on image processing*, 11(6):670–684, 2002.

[74] Jean-Luc Starck, Fionn Murtagh, and Jalal Fadili. *Sparse image and signal processing: Wavelets and related geometric multiscale analysis*. Cambridge university press, 2015.

[75] Bob L Sturm and Mads Græsbøll Christensen. Comparison of orthogonal matching pursuit implementations. In *Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European*, pages 220–224. IEEE, 2012.

[76] Joel A Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information theory*, 50(10):2231–2242, 2004.

[77] Kristian Valen-Sendstad. The 2015 aneurysm cfd challenge: Variability of segmentations, hemodynamics, and hemodynamic indices: Qualitative and preliminary results. Summer Biomechanics, Bioengineering & Biotransport Conference, Utah, USA, 2015.

[78] Ross T Whitaker. A level-set approach to 3d reconstruction from range data. *International journal of computer vision*, 29(3):203–231, 1998.

[79] Chenyang Xu and Jerry L Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on image processing*, 7(3):359–369, 1998.