UNIVERSITÄT PASSAU
Fakultät für Informatik und Mathematik

AI-Supported Interactive Segmentation of 3D Volumes

Thomas Lang

Dissertation zur Erlangung des Doktorgrades
der Naturwissenschaften (Dr. rer. nat.)
eingereicht an der Fakultät für Informatik und Mathematik
der Universität Passau

Dissertation submitted to
the Faculty of Computer Science and Mathematics
of the University of Passau
in Partial Fulfillment of Obtaining
the Degree of a Doctor of Natural Sciences

Betreuer/Supervisor:	Prof. Dr. Tomas Sauer
Externer Gutachter/External Examiner:	Prof. Dr. Thomas Pock

Passau, 14th April 2021

Abstract

The segmentation of volumetric datasets, i.e., the partitioning of the data into disjoint subvolumes with the goal to extract information about these regions, is a difficult problem and has been discussed in medical imaging for decades. Due to the ever-increasing imaging capabilities, in particular in X-ray computed tomography (CT) or magnetic resonance imaging, segmentation in industrial applications also gains interest. Especially in industrial applications the generated datasets increase in size. Hence, most applications apply well-known techniques in a 2+1-dimensional manner, i.e., they apply image segmentation procedures on each slice separately and track the progress along the axis of the volume in which the slices are stacked on. This discards the information on preceding or subsequent slices, which is often assumed to be nearly identical. However, in the industrial context this might prove wrong since industrial parts might change their appearance significantly over the course of even a few slices. Moreover, artifacts can further distort the content of the slices. Therefore, three-dimensional processing of voxel volumes has to be preferred, which induces constraints upon the segmentation procedures. For example, they must not consider global information as it is usually not feasible in big scans to compute them efficiently. Yet another frequent problem is that applications focus on individual parts only and algorithms are tailored to that case. Most prominent medical segmentation procedures do so by applying methods to specifically find the liver and only the liver of a patient, for example. The implication is that the same method then cannot be applied to find other parts of the scan and such methods have to be designed individually for any object to be segmented. Flexible segmentation methods are needed too specifically when partitioning *unique* scans. We define a unique scan to be a voxel dataset for which no comparable volume exists. Classical examples include the use case of cultural heritage where not only the objects themselves are unique but also scan parameters are optimized to obtain the best image quality possible for that specific scan.

This thesis aims at introducing novel methods for voxelwise classifications based on local geometric features. The latter are computed from local environments around each voxel and extract information in similar ways as humans do, namely by observing their similarity to geometric or textural primitives. These features serve as the foundation to learning the proposed voxelwise classifiers and to discriminate between segmented and unsegmented voxels. On the one hand, they perform fully automated clustering of volumes for which a representative random sample is extracted first. On the other hand, a set of segmenting classifiers can be trained from few *seed voxels*, i.e., volume elements for which a domain expert marked if they belong to the components that shall be segmented. The interactive selection offers the advantage that no completely labeled voxel volumes are necessary and hence that unique scans of objects can be segmented for which no comparable scans exist. Overall, it will be shown that all proposed segmentation methods are effectively of linear runtime with respect to the number of voxels in the volume. Thus, voxel volumes without size restrictions can be segmented in an efficient linear pass through the volume. Finally, the segmentation performance is evaluated on selected datasets which shows that the introduced methods can achieve good results on scans from a broad variety of domains for both small and big voxel volumes.

Zusammenfassung

Die Segmentierung von Volumendaten, also die Partitionierung der Daten in disjunkte Teilvolumen zur weiteren Informationsextraktion, ist ein Problem, welches in der medizinischen Bildverarbeitung seit Jahrzehnten behandelt wird. Bedingt durch die sich ständig verbessernden Bilderfassungsmethoden, speziell im Bereich der Röntgen-Computertomographie (CT) oder der Magnetresonanztomographie, gewinnt die Segmentierung von industriellen Volumendaten auch an Wichtigkeit. Insbesondere im industriellen Kontext steigt die Größe der zu segmentierenden Daten jedoch rasant an, so dass sich die meisten Segmentierungsapplikationen auf den 2+1-dimensionalen Fall beschränken, also Bilder verarbeiten und die Ergebnisse über mehrere Bilder hinweg verfolgen. Jedoch werden somit beispielsweise geometrische Informationen über benachbarte Schichten ignoriert. Diese können sich aber gerade im industriellen Bereich signifikant ändern. Aus diesem Grund ist hier die dreidimensionale Bildverarbeitung vorzuziehen. Dadurch ergeben sich neue Einschränkungen, beispielsweise können keine globalen Informationen zur Segmentierung herangezogen werden, da diese typischerweise nicht effizient berechenbar sind. Ferner fokussieren sich dreidimensionale Methoden aus medizinischen Bereichen zumeist auf bestimmte Bestandteile der Daten, wie einzelne Organe. Dies schränkt die Generalität dieser Methoden signifikant ein und somit sind separate Verfahren für jedes zu segmentierende Objekt notwendig. Flexible Methoden sind darüber hinaus bei Anwendung auf *einzigartige* Scans erforderlich. Ein einzigartiger Scan ist ein Voxelvolumen, für welches kein vergleichbares Datum existiert. Klassische Beispiele sind Kulturgutdigitalisate, da dort nicht nur die Objekte einzigartig sind, sondern auch die Aufnahmeparameter spezifisch für diesen einen Scan optimiert wurden.

Die vorliegende Dissertation führt neuartige Methoden zur voxelweisen dreidimensionalen Segmentierung von Volumendaten auf Basis lokaler geometrischer Informationen ein. Die Bewertung dieser Informationen imitiert die menschliche Objektwahrnehmung, indem lokale Regionen mit geometrischen oder strukturellen Primitiven verglichen werden. Mit Hilfe dieser Bewertungen werden voxelweise anzuwendende Klassifikatoren trainiert, welche zwischen erwünschten und unerwünschten Voxeln unterscheiden sollen. Ein Teil dieser Klassifikatoren führt eine vollautomatische Clustering-Analyse durch, nachdem eine repräsentative und zufällig ausgewählte Teilmenge fester Größe an Voxeln selektiert wurde. Die verbliebenen Segmentierungsalgorithmen erhalten Trainingsdaten in Form von *Seed-Voxeln*, also wenige Volumenelemente, die von einem Domänenexperten markiert wurden. Diese interaktive Herangehensweise ermöglicht das Einbringen von Expertenwissen ohne die Notwendigkeit vollständig annotierter Trainingsvolumen, wodurch auch einzigartige Scans segmentiert werden können. Für alle Verfahren wird dargelegt, dass die eingeführten Algorithmen von asymptotisch linearer Laufzeit in der Anzahl der Voxel im Volumen sind. Somit können Voxeldaten ohne Größenbeschränkungen in einem effizienten linearen Durchgang verarbeitet werden. Abschließend wird die Performanz der vorgestellten Verfahren auf ausgewählten Daten evaluiert und aufgezeigt, dass mit denselben wenigen Verfahren gute Ergebnisse auf vielen unterschiedlichen Domänen und gleichfalls auf kleinen und großen Volumen erzielt werden können.

Dedicated to my mother Silvia who sacrificed a lot to enable me to study and consequently write this thesis, to her partner Walter and to my father Thomas.

Acknowledgements

This thesis has only been made possible through the support of many people.
First of all, I want to thank Prof. Dr. Tomas Sauer, who not only gave me the opportunity to work on the interesting field of segmentation of industrial computed tomography data but also introduced me to the world of machine learning and its mathematical foundations. I am further grateful that he formed the first connection between me and the people working at the Fraunhofer EZRT in Fürth which enabled an easy and fun collaboration. In particular, I want to thank him for proofreading this thesis and intermediate reports as well as improving my English language skills in general.
Next, I would further like to express my gratitude to all of my colleagues at the institute FORWISS at the University of Passau. Of them, I especially thank my office co-workers Dr. Benedikt Diederichs and Michael Stock. Benedikt specifically supported me by improving my mathematical background and demonstrating that an abstract mathematical viewpoint can often lead to elegant yet efficient solutions, specifically when we collaborated on the mathematical details of the Wasserstein histogram embedding and when he defined a new spherical histogram type used here for the HOG feature descriptor. Michael instead always took time to answer my questions regarding algorithmic and implementation details which enabled us to quickly develop a prototypical application. This in turn provided means to demonstrate the proposed interactive workflows to audiences and helped to explain them as well as to pitch new ideas and projects.
I am also thankful for the help of my colleagues at the Fraunhofer EZRT in Fürth, who provided the datasets used in this thesis and further also supported me with technical details and ideas to improve the methods presented in this work. Specifically, I want to mention the group of co-workers from both the FORWISS and the EZRT who together with myself met virtually on each day in the "Früher Kaffee" meeting which helped me not only socializing with all of them but also gave me the opportunity for quickly discussing issues.
Finally, I especially want to thank my friends of the "Stammtisch" group, my family and especially my mother and grandparents for bearing with me over the last three years and supporting me emotionally, especially during the isolation due to the Covid-19 pandemic.

Contents

List of Figures

List of Algorithms

List of Symbols and Abbreviations

General

$\boldsymbol{x}_{\boldsymbol{\alpha}}$	Voxel of a volume at position $\boldsymbol{\alpha}$
$\boldsymbol{x} = (x_i)_{i=1}^{d}$	Vector of dimension d
$\boldsymbol{e}_i$	i-th standard basis unit vector
$\boldsymbol{M} = (\boldsymbol{M}_{i,j})_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}}$	Matrices in $\mathbb{R}^{m \times n}$
$\mathrm{p}(\boldsymbol{x}_{\boldsymbol{\alpha}}), \mathrm{v}(\boldsymbol{x}_{\boldsymbol{\alpha}})$	Position and value of voxel $\boldsymbol{x}_{\boldsymbol{\alpha}}$, respectively
$R_K(\boldsymbol{x}_{\boldsymbol{\alpha}})$	Region of size $K \times K \times K$ centered around a voxel $\boldsymbol{x}_{\boldsymbol{\alpha}}$
d	Dimensionality of feature vectors
$\mathcal{O}(\cdot)$	Landau notation used in complexity analysis
N_T, N, M	Number of threads, voxels in a volume and a selection, respectively
T	Thresholds of all kinds

Clustering

$\mathbb{E}[X]$	Mean/Expected value of X
F_ν	Cumulative distribution function of a probability measure ν
Ω	Random samples
π	GMM mixture coefficient
$\mathcal{N}(\cdot \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate normal distribution having mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$\log \mathcal{L}(X \mid \boldsymbol{\Theta})$	Logarithmized likelihood of X given parameters $\boldsymbol{\Theta}$

FAITH

λ, μ	FAITH regularization parameters
$\partial g(\cdot)$	Subgradient of a function g at some point
prox_f	Proximal operator of a function f
S_T	Soft-thresholding operator with threshold T
W	Maximum admissible grayscale value
L	Lipschitz constant
$\mathrm{Eig}_{\max}(\boldsymbol{A})$	The largest eigenvalue of $\boldsymbol{A}$

SVM-based Segmentation

$\mathcal{S}$	SVM classification operator
C, ν	SVM regularization parameters
ξ	SVM slack variables
k	Kernel function

Multiresolution Segmentation

$\ell_{\max}$	Number of resolution levels for a volume
ℓ	Resolution level, $1 \leq \ell \leq \ell_{\max}$
$\Upsilon_{\ell'}^{\ell}$	Sampling policy from resolution level ℓ' to ℓ
$\mathfrak{S}_{\ell'}^{\ell}(\boldsymbol{x})$	Set of voxels on level ℓ' covered by $\boldsymbol{x}$ on level ℓ
ρ^{ℓ}	Confidence threshold for level ℓ
C	Candidate Regions of Interest

Part I.

Introduction

Lasciate ogni speranza, voi ch'entrate.

Dante Alighieri, Divine Comedy

Table of Contents

This first part of this thesis starts by motivating why segmentation of voxel volumes is an essential tool in many image processing applications. After listing recent advances in segmentation techniques, we briefly examine how volumetric data is generated and how we process it. Additionally, we discuss several challenges encountered in practice whose issues we aim to solve or at least reduce by our introduced techniques.
The next chapter deals with simple yet efficient pre- and postprocessing methods used for improving segmentation results. They include methods for noise or artifact reduction as well as for simplifying and updating the results.

1. Background

Abstract This introductory chapter first discusses why segmentation is still an important part of many volume processing applications. Next, the acquisition and processing of volumetric data is explained in our framework which forms the basis of all local segmentation procedures introduced in subsequent parts. The chapter is concluded by highlighting some challenges occuring in volumetric segmentation.

1.1. Motivation

Segmentation of images and volumes is a crucial requirement in many applications ever since images and volumes could be generated. We focus specifically on volumetric segmentation, which plays a fundamental role in several fields:

- Volume inspection
- Surgery planning
- Computer Aided Diagnosis
- Industrial Quality Control

In medical context, the first aspects of this list are of high interest since decades. The first steps were done by rendering and inspecting medical data, e.g., computed tomography or magnetic resonance images, so experts could manually diagnose patients, e.g., by finding tumors. Another use case is the planning of surgeries such that they are minimally invasive. The latter profits from such methods as they highlight segmented parts and especially their location, allowing to plan the simplest and least dangerous way of accessing them.
Improvements over manual inspection include (semi-)automated Computer Aided Diagnosis. Typically, such applications provide the described visual features while also automatically classifying objects that have been found. As an example, consider a diagnosis tool which inspects computed tomography scans of human colons, detects polyps and additionally informs doctors whether the detections are of malignant nature.
In recent years, tomography gained interest in industrial applications too. There, one often uses this technique to inspect products and classify if they fulfill certain quality standards. Advances in segmentation aim to inform producers if their manufacturings can still be used even if they contain small faults, thus reducing rejections and saving money as well as resources.
Especially regarding industrial tomography, the resulting size of the datasets are ever-increasing. In this thesis, we introduce algorithms applicable to volumetric data without size restrictions. Therefore, we will consider segmentation techniques which classify voxelwise while still being of approximate linear runtime in the number of voxels. At this point, we especially mention the Fraunhofer Development Center for X-ray Technology (EZRT) who developed a computed tomography scanner in which big objects like complete cars or small airplanes can be scanned at once, allowing for inspecting them in high detail[1]. An implied use case is the inspection of

[1]Technical details can be viewed under `https://www.iis.fraunhofer.de/en/ff/zfp/tech/hochenergie-computertomographie.html`.

crashed cars, in which individual parts can be checked if they conform to simulations without the need to disassemble them. The latter would be unfavorable as tensions in parts could result in further damage during the disassembly. Hence, visual inspections supported by segmentation procedures can help engineers interpreting crash test results as is.

1.2. Advances in Segmentation

Since segmentation forms an important step in most image or volume processing pipelines, a huge variety of different methods has been created over the past decades, and ever-increasing data generation capabilities require future work to be done in this field.
Here, we give a brief overview over existing techniques used for volumetric segmentation, but explicitly exclude algorithms involving adaptive thresholding, Support Vector Machines or multiresolution approaches, as dedicated literature surveys are provided in the chapters dealing with these.

We start with deformable contour models which are initialized with a prior shape close to that of the desired object and adapt (*deform*) it to fit the part under consideration by minimizing a shape error functional that penalizes mis-segmentations as well as the boundary surface area. Note that active contours are mostly 2+1-dimensional methods which determine the delineating boundary and track it along the depth of the object. A related approach is the so called *level set* method. These are improvements over the active contour method in the sense that during the segmentation process the topology of the object can change. To achieve this, they switch from minimizing an energy-based loss function to tracking an evolving contour numerically based on gradient features of some surface function φ. The models thus assume that the searched surface is given in implicit form as a level set $\Gamma(t) = \{(x, y) \mid \varphi(x, y, t) = 0\}$. Note that level set methods are a special case of active contour models and often both approaches are used symbiotically when gradient-based tracking is combined with the minimization of an error functional. Examples stem mainly from medical image processing like [1] in which the authors refine the basic method by adding a probabilistic shape prior as well as a probabilistic speed function which further incorporates the estimated mean curvature. In [2], level sets for both registration and segmentation methods are defined to allow proper shape initialization and improving the results. A similar procedure is found in [3], in which the authors present a variational framework including the combination of segmentation and registration techniques based on active contours. [4] further compares several active contour approaches with respect to their applicability in automatic bone segmentation in computed tomography scans. A comprehensive literature survey dealing with level set methods can be found in [5], whereas a list of publications concerning active contours in medical imaging is given in [6].

A completely different approach to segmentation of voxel data is by using graph algorithms. Conceptually, these methods interpret pixels/voxels or regions in an image/volume as a graph, on which techniques comprised of finding shortest paths, minimum spanning trees or cutting the graph can be employed for segmentation. Popular methods used in two-dimensional image segmentation which in principle can be lifted to the three-dimensional case are listed in [7], while [8] explains applications in 3D medical imaging. In particular, we mention the work by Bauer et al. [9] in which the authors presented a method for segmenting tubular structures from tomography data using prior shape information as well as graph cuts. For this, a graph is constructed using voxels as vertices and a cost is assigned to each edge connecting the nodes. Graph cuts then search for the optimal solution of a cost functional discretized and applied on said graph. Recent advances also combine graphical models with probabilistic approaches, e.g., Hidden Markov Models or random fields. Using that method, graphical models consider observable

input given in form of voxel values and unobserved values representing the assigned class labels. After training, class labels for each voxel can be assigned in a single pass through that network. Examples include [10] where a graph-based model was trained using the popular Expectation Maximization algorithm to cluster magnetic resonance images.

Especially in the context of clinical tomography, also atlas-based techniques are used. They use manually annotated datasets, the so called *atlasses*, in which commonly certain Regions of Interest are annotated with a class label. During segmentation, the ground truth image is registered to the currently processed data and the annotation labels are propagated to the registration result. Depending on the application, also multiple atlasses are used, as explained in [11], where the multitude of labels are propagated to the scan after registration and are combined using a spatially varying decision function. Similarly, [12] used several atlas images which are registered to magnetic resonance scans in order to obtain a bunch of segmentations which later are combined into a final segmentation. Further methods are summarized in [13].

Naturally, we also like to give a very brief overview over the most recent advances in segmenting images and volumes achieved by *neural networks*. As that name hints, these are graphical models in which each individual node is a neuron which mimics biological ones found in brains. The major breakthrough brought by neural networks is that they learn features and invariants from the data by themselves and can represent nonlinearities. Over the years, a lot of different types and techniques were developed to improve and accelerate segmentations, of which we specifically name convolutional neural networks and their applications in instance and semantic segmentation. The latter describes segmentation results where not only all voxels have a label assigned but also that these labels are identified with each other if their objects are trained to have a semantic connection. Examples for semantic segmentation are numerous, in the two-dimensional case extensive literature surveys are found in [14, 15] while for volumetric segmentation relatively few methods were published yet, from which we highlight [16–18]. First, Çiçek et al. [16] construct a dense volumetric segmentation from sparse annotations based on a modified U-Net network architecture. Next, [17] also base their segmentation dealing with full head and neck regions on the U-Net architecture but extend it by allowing local segmentations and using a modified loss function. Finally, [18] also uses a modified U-Net model for semantic segmentation of kidney vessels by generating multiple segmentation results and producing a final one by averaging them. On top of these approaches, the field of instance segmentation does not only segment its input semantically in the just described sense, but also enumerates them per class, i.e., assigns unique labels to each instance of a semantically grouped region. A survey of instance segmentation approaches for images is given in [19]. A quite recent survey encompassing both techniques can be found in [20]. Concludingly, we want to highlight that neural networks, being famous for constructing features on their own, have been successfully combined with other methods like a variational approach presented in the work of Ranftl et al. [21].

While one could discuss many more details regarding recent advances in segmentation, this thesis specifically focusses on interactive segmentation. Until now, only few interactive approaches were proposed, of which both two-dimensional and three-dimensional methods exist. In the 2D case, we want to mention [22] which uses interactive annotations to train an active contour model used for delineating the boundaries of the desired object. In a similar fashion, Santner et al. [23] used seed pixel information to train a random forest classifier combined with regularization. Regarding three-dimensional segmentation, an increasing number of techniques were proposed in recent years, of which we like to cite [24], where the authors proposed an early interactive approach in which a contour segmentation is combined with probabilistic region growing. A much more modern path is taken by [25] by combining user interaction with

probabilistic maps, neural networks and geodesic distances for volumetric segmentation. A brief survey over additional interactive segmentation techniques is given in [26].

As the above paragraphs demonstrated, many if not most existing segmentation procedures stem from medical imaging and are often specifically tuned towards that domain. However, in such applications the size of the datasets are typically small, ranging up to a few hundred Megabytes, for the reason that a patient shall receive as few radiation as possible. In industrial tomography, the reconstructed volumes can be substantially bigger, ranging up to Terabytes in extreme cases. Another caveat specific to industrial tomography is that typically few scans are of the same kind, hence making it difficult to train a robust segmentation model. Then, many algorithms described thus far are not feasible anymore, e.g., common neural network models.
To cope with these big volume sizes, additional methods were proposed which mostly exploit locality or add locality to existing methods. Popular methods include Bayesian methods, which together with additional techniques are compared in [27] for their usage in analysis of pore structures. In the same field, [28] detect pores using neural networks both slicewise and by 3D nets. Instead, in [29] the authors proposed a markov random field model for segmentation while simultaneously dealing with metal artifacts. On the other hand, [30] uses a gradient-based technique to detect edges in blurry regions trained from Computer Aided Design input.

We conclude that a humongous number of different approaches exist with an increasing trend towards three-dimensional segmentation. Moreover, the following chapters will describe even more methods for segmenting volumes.
However, inspection on most techniques mentioned above need special requirements to be fulfilled. These often consist of *a priori* labeled training data, i.e., volumes where each individual voxel is assigned a class label, e.g, in methods using atlasses or neural networks. Naturally, the creation of such voxel data is cumbersome and since experts are required to do so it is also expensive. Other prerequisites include nonlocal information in case the methods explicitly aim for globally optimal segmentation or when large neighborhoods are necessary for graph cut algorithms.
During the course of this thesis, we will propose extensions and novel algorithms for volumetric segmentation which are suited to handle big volumes. We will intuitively prove this by arguing that they are effectively of linear runtime complexity with respect to the number of voxels. Furthermore, we demonstrate that the presented methods work locally enough, i.e., consume few enough memory, such that they are applicable to volumes without size restrictions.

1.3. Data Acquisition

Since the algorithms introduced later process three-dimensional data, we like to mention a few basic details of how that data is acquired. There are plenty of methods producing 3D images, e.g., X-ray computed tomography, magnetic resonance imaging, ultrasound imaging, positron electron tomography and many more. We focus on X-ray computed tomography (CT) and magnetic resonance imaging (MRI) as we primarily work with such data. However, we emphasize that all methods proposed only assume three-dimensional voxel grids and thus are applicable to many different domains.
First, we discuss X-ray computed tomography which employs X-rays first found by Wilhelm Conrad Röntgen who was awarded the first Nobel prize in 1901 [31]. He famously called them X-rays where X stands for *unknown*, while at least the German community started to call them Röntgen rays after his talk given at the third sitting of the *Physics and Medical Society* in Würzburg. His discoveries described in that talk can be read up in [32]. In said presentation Röntgen described how the newly found radiation is able to “penetrate materials beyond optical inspection”

Figure 1.1.: The first X-ray image showing the hand of Röntgen's wife [33].

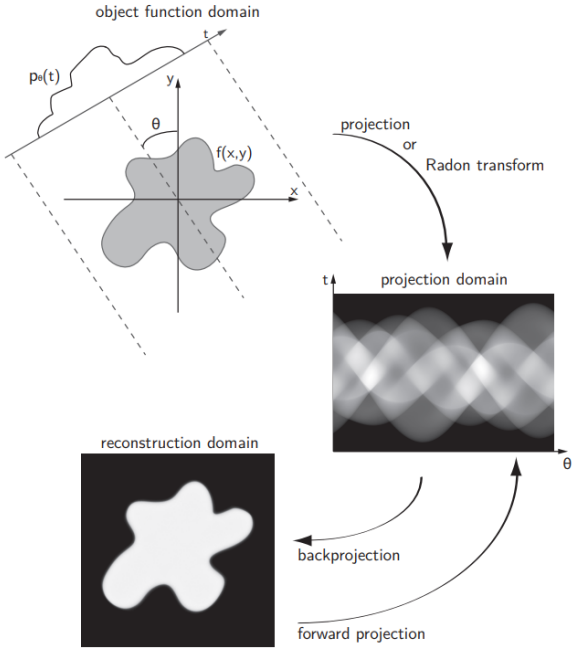


Figure 1.2.: Schematic CT overview [34, Fig. 1.6].

and that the produced images depend on the densities of the objects scanned. Furthermore, he also proposed the usage of his findings for medical inspection, one of the very first such images is depicted in Figure 1.1 which shows the hand of Röntgen's wife Anna Bertha Ludwig with clearly visible bones and her ring.
As the image quality of X-ray images improved, his invention became increasingly popular for medical inspection. However, X-ray imaging lacks information about the three-dimensional shape of the objects considered. Thus, the idea of *X-ray computed tomography* was born: How can we reconstruct a 3D volume from several X-ray images?
The theoretical background to the answer of this question was developed by Johann Radon who defined an integral transform named after him as well as a corresponding inverse transform. That transform maps a two-dimensional image function into its set of integrals along all straight lines parametrized by a projection angle and an offset. In the following paragraphs, we call this mapping the *projection*, cf. Figure 1.2. Under some assumptions regarding the image function, the integral expressions converge absolutely and the image function is uniquely defined by its transformation result [35]. In context of X-ray imaging, the Radon transform of an object along some direction corresponds to the X-ray image recorded in that direction. Regarding the reconstruction of the object from the individual images, one now exploits that the one-dimensional Fourier transform of the projections are slices of the two-dimensional Fourier transform of the original image. This fact is ensured by the Fourier slice theorem [36] which can be generalized to higher dimensions. Consequently, one computes a 2D Fourier transform on the projection images, aggregates these slices and finally applies an inverse 3D Fourier transform on the result to obtain a reconstructed three-dimensional object. The workflow just described is depicted schematically in Figure 1.2 for a two-dimensional reconstruction problem. These connections can be lifted to higher dimensions.
Ever-ongoing improvements in terms of reconstruction quality, accuracy and speed made CT imaging a highly popular method of inspecting objects. As a historical remark, first steps towards

practical tomography were taken by Korenblum et al. [37] in 1958. In their work, the authors describe the two-dimensional reconstruction problem specifically solved for the fan-beam scan geometry as well as providing a scheme to build a simple tomography scanner for the stated slice reconstruction. However, their work got largely unrecognized due to the paper being in Russian and we suppose also due to the political climate at that time. A few years later, further progress for medical applications was made as advanced scanners became available in the 1970s when Sir Godfrey Hounsfield created a prototype CT scanner based on the works of Allan MacLeod Cormack. For their work, both were awarded the Nobel prize in Physiology in 1979 *for the development of computer assisted tomography* [38]. Considering industrial tomography, the probably first recorded experiment was conducted by William H. Oldendorf in 1961 which already shows the typical setup commonly found in commercial industrial CT scanners [39]. Ever since that experiment, the industrial use case developed in parallel to advances in medical tomography imaging.
Although X-ray computed tomography is arguably the most prominent noninvasive inspection technique used for medical and industrial applications, other methods exist for special use cases, e.g., magnetic resonance imaging (MRI) or positron emission tomography (PET). Such methods try to depict certain parts of the desired images as good as possible. In case of MRI, applications include the visualization and processing of images of internal organs which are often not properly recognized by X-ray CT. On the contrary, PET makes uses of a radioactive substance which enhances the visibility of, e.g., metabolic flows in organs. Considering industrial imaging, possible uses for MRI are automatic inspection of nonmedical soft-tissue material like fruit or baby food in order to detect harmful content like glass fragments [40]. A short survey of applications of other tomography methods in process engineering can be found in [41].

Whatever way is chosen to acquire data, we assume that the generated voxel datasets are three-dimensional volumes containing nonnegative values. Hence, by abstracting from the actually used technology we provide algorithms operating on general volumetric datasets. For an in-depth view into the mathematical and physical background of computed tomography we forward the reader to [36, 42, 43].

1.4. Processing 3D Data

Regardless of the technology used for data acquisition, the result is a digital three-dimensional reconstruction of the scanned object. We call such a reconstruction a volume consisting of voxels.

Definition 1.4.1. A set $\mathbf{\Gamma_d} \subset \mathbb{N}^3$ of the form $\mathbf{\Gamma_d} := \{1, \dots, d_1\} \times \{1, \dots, d_2\} \times \{1, \dots, d_3\}$ is called a regular grid of dimensions $\boldsymbol{d} = (d_1, d_2, d_3) \in \mathbb{N}^3$. Moreover, let $f_v \colon \mathbf{\Gamma_d} \to \mathbb{R}_{\geq 0}$ be a function assigning a nonnegative real value to each position in the grid.
A *volume* is a tuple $(\mathbf{\Gamma_d}, f_v, V)$ where $V = \{\boldsymbol{x_\alpha} \mid \boldsymbol{\alpha} \in \mathbf{\Gamma_d}\}$ is the set of all volume elements $\boldsymbol{x}_\alpha$ on the grid $\mathbf{\Gamma_d}$.
Additionally, define

$$\begin{aligned} \mathrm{p} \colon V &\to \mathbf{\Gamma_d} & \qquad \mathrm{v} \colon V &\to \mathbb{R}_{\geq 0} \\ \boldsymbol{x_\alpha} &\mapsto \boldsymbol{\alpha}, & \qquad \boldsymbol{x_\alpha} &\mapsto f_v(\boldsymbol{\alpha}), \end{aligned}$$

as functions retrieving the position and value associated to a *voxel* $\boldsymbol{x_\alpha} \in V$.

Here, we use a voxel as an abstraction of a cuboid of which the volume consists, which can possess additional attributes like the actual voxel size in micrometers. Contrary, the grid points are only the positions of the voxels in the discrete grid.
Since the voxel set V contains one volume element for each point in the grid $\mathbf{\Gamma_d}$, the mapping p assigns a unique position to each voxel and further p is surjective since V enumerates every

point in the grid. Thus, p forms a bijection between the set of voxels enumerated by multi-indices and the grid points.
With respect to Definition 1.4.1, applying any segmentation operator introduced in this thesis on a volume $(\mathbf{\Gamma}_{\boldsymbol{d}}, f_v, V)$ creates a new physical volume $(\mathbf{\Gamma}_{\boldsymbol{d}}, \widetilde{f}_v, V)$ of same dimensions, and hence also the same set of enumerated voxels, but with a new voxel value function $\widetilde{f}_v$. Syntactically, we write $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ when the value of the voxel at position $\boldsymbol{\alpha}$ in the *target volume* is set.

Remark 1.4.2. In medical imaging and most industrial applications the voxel values actually are integral due to datatype conversion to save memory. We generalize this to real values, but nevertheless we require nonnegative values. If in certain applications also negative values are produced, we scale them accordingly.

Remark 1.4.3. Definition 1.4.1 explains that a discrete voxel volume as gained from computed tomography forms a three-dimensional rectangular parallelepiped consisting of voxels. Regardless of the physical orientation of the object scanned, its digital reconstruction is oriented as a continuous data stream. In this stream, the voxels are enumerated in lexicographic order of their associated multi-indices in the ordering $z \leq y \leq x$. We interpret the third multi-index component as the z axis, all combinations of multi-indices for a fixed z value form an individual slice image.

To each volume we can further define Regions of Interest which define sub-volumes.

Definition 1.4.4. A *Region of Interest* (abbreviated as RoI) is a tuple $(\boldsymbol{o}, \boldsymbol{d})$ consisting of an *origin* $\boldsymbol{o} \in \mathbb{N}_0^3$ and a *dimension* $\boldsymbol{d} \in \mathbb{N}^3$.

Another useful definition necessary to define the further processing is the *iterable* RoI describing the largest region such that each voxel can be the center point of a cube having a side length of K voxels along each axis, which we will refer to as *mask*.

Definition 1.4.5. Given a volume of dimensions $\boldsymbol{d} \in \mathbb{N}^3$ and some padding $\boldsymbol{p} \in \mathbb{N}_0^3$, the *iterable RoI* of this volume is given by the Region of Interest

$$(\boldsymbol{p}, \boldsymbol{d} - 2\boldsymbol{p})\,. \tag{1.1}$$

Note that while $\boldsymbol{p} = \mathbf{0}$ is permissible, we require that the volumes contain at least one voxel, i.e., $2\boldsymbol{p} + \mathbf{1} \leq \boldsymbol{d}$.

We further want to describe how volumes are processed in a space-efficient way. In order to exploit parallelism, we split the volume along its logical z axis and process the sub-volumes in parallel. In detail, consider a system having $N_T \in \mathbb{N}$ threads available. Then, the volume is split into N_T RoIs along its logical z axis and all regions are processed in parallel, where in each thread a function is applied sequentially. In subsequent pseudocode listings we will denote this by the keyword parfor. Hence, Algorithm 1 describes that two physical volumes of same dimensions are split up in the same way along their z axes. In parallel, the value $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ of the voxel at position $\boldsymbol{\alpha}$ in the target volume is assigned a new value as computed by some function f depending on the voxel $\boldsymbol{x}_{\boldsymbol{\alpha}}$ at the same position. Processing every iterable voxel in the volume V further ensures that every voxel is taken into account.

Algorithm 1: Parallel volume processing.

parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V$ **do**
  $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow f(\boldsymbol{x}_{\boldsymbol{\alpha}})$

Additionally, we distinguish between voxelwise and mask-based processing. The first method simply uses the current voxel and disregards any surroudings. For certain applications like

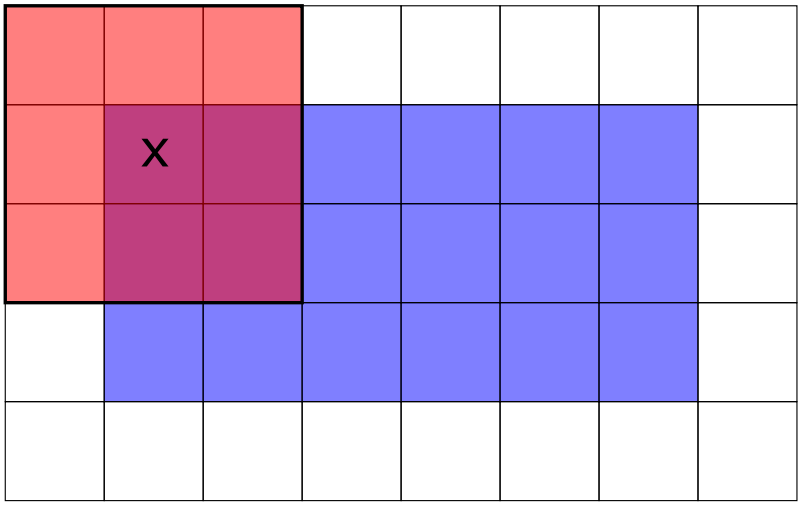


Figure 1.3.: Mask-based processing for $K = 3$.

thresholding, which we will explain later, this suffices. However, we specifically want to exploit local environments around each voxel according to Definition 1.4.6.

Definition 1.4.6. Let $K \in 2\mathbb{N} + 1$ be the environment size which we require to be always odd since our anchor point is always in the middle of the mask. This induces a padding of $K_2 := \lfloor K/2 \rfloor$ voxels. Let V be the set of voxels according to Definition 1.4.1.
Define the *local environment* of size K centered on a voxel $\boldsymbol{x_\alpha} \in V$ by

$$R_K(\boldsymbol{x_\alpha}) = \{\boldsymbol{x_\beta} \in V \mid \|\mathrm{p}(\boldsymbol{x_\alpha}) - \mathrm{p}(\boldsymbol{x_\beta})\|_\infty \leq K_2\}.$$

Naturally, the induced padding also shrinks the iterable RoI according to Definition 1.4.5.

Remark 1.4.7. Be aware that in many image processing systems it is common to process each individual voxel instead of just considering the ones included in the iterable RoI. The missing information at the boundary is then often assumed to have values of zero, or that the missing information extends the boundary of the image. We choose to only process the inner voxels as specified in the framework we implemented our methods in. The values of the target volume voxels which are not processed are explicitly set to zero.

A visual interpretation of local environments and the iterable Region of Interest is given in Figure 1.3 in which we fix $K = 3$. As inferred from Definition 1.4.6, choosing $K = 3$ implies a padding of one voxel. Hence, only for the inner voxels (depicted blue) a local environment of size $K \times K \times K$ can be fully constructed. Such a region for the first iterable voxel is drawn as red square in the image.
Overall, for parallel processing the volume is split along its z axis with overlap as big as the padding such that the demonstrated iteration can be applied on each sub-volume.
Since we deal with big voxel datasets, we want to highlight one implementation detail. If the source as well the target volume fit into main memory as a whole, we apply the simple technique described above. Otherwise, the voxels of the volumes are *streamed* from/to disk, i.e., only the currently processed voxels are loaded in a buffered manner. Both ensure that volumes without size restrictions can be handled in a space-efficient way.

1.5. Challenges

We conclude the current chapter by describing the challenges one faces when segmenting industrial computed tomography data.
First of all, we deal with big voxel volumes in principle. This fact implies two restrictions on segmentation algorithms, namely *locality* and *efficiency*. Locality describes that any algorithm needs to work on local environments only (cf. Definition 1.4.6) as global consideration is not feasible for big volumes. By efficiency we mean that the asymptotic runtime of segmentation procedures should depend only linearly on the number of voxels in the volume. That runtime might depend on other factors too, but since the voxel count is typically the dominating factor during processing, only algorithms linear in it are admissible. For each subsequently introduced technique we will provide approximate analyses of their asymptotic runtimes as well as the memory requirements to show their suitability for our purposes.
Additionally, we face the challenge that we want to avoid hand-tuned models as well as keeping the algorithms as general as possible. Thus, our methods are useful for many different applications and allow for a highly flexible segmentation. The versatility is further enhanced by introducing an interactive component in our algorithms which allows for using domain expert knowledge ad hoc. Our solutions further involve several machine learning techniques which automatically infer properties of local environments and enable a quick training procedure as well as easy generalization of algorithms.
In Part V we will demonstrate the flexibility of our algorithms by providing qualitative segmentation results which were generated with the same few algorithms proposed in this thesis. A quantitative evaluation will also be performed on selected volumes which further demonstrates that a good segmentation performance can be achieved.

2. Basic Pre- and Postprocessing

Abstract In any segmentation and also machine learning system, preprocessing and postprocessing is omnipresent and crucial for dealing with noise or for accelerating algorithms.
This chapter introduces basic yet efficient pre- and postprocessing techniques. It starts by defining simple thresholding used to reduce measurement noise. Next, a brief overview over advanced artifact reduction methods is given. Lastly, two efficient postprocessing steps are introduced which allow for further improvement of segmentation results.

2.1. Thresholding

The first technique we like to present is thresholding, which is perhaps the most widely used segmentation method in all of image processing.

Definition 2.1.1. Let $b, f \in \mathbb{R}_{\geq 0}$ be a background and foreground value, respectively. Given a *threshold* $\theta \in \mathbb{R}_{\geq 0}$, we define the *parametrized thresholding operator*

$$T_\theta^{b,f} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$$
$$x \mapsto \begin{cases} f, & x \geq \theta, \\ b, & x < \theta. \end{cases}$$

We see that given some threshold $\theta \in \mathbb{R}_{\geq 0}$, we replace the voxel value by either a foreground value f or a background value b, depending on the voxel value.
In our applications, we use thresholding two-fold: to reduce measurement noise and to refine segmentation results. The first case exploits that in both computed tomography and magnetic resonance imaging, measurement noise is typically located in a grayscale value range far below material voxel values. Thresholding sets noise voxel values to zero, allowing for skipping them automatically and thus significantly reducing runtime. The second case is useful if a segmentation procedure does not simply produce a binary decision but rather yields values in a discrete or continuous range. Then, thresholding these generated values yields a binary decision by setting *unlikely* voxels to zero.
We also want to note that thresholding is easily incorporated in other segmentation methods. Basically, whenever some method aims to decide if a voxel shall be kept in the result volume it can first check if the voxel value is below a given threshold. If so, the value of the voxel in the target volume is set to zero and the algorithm skips to the next voxel immediately, thus omitting probably costly calculations.

For these two common cases, we specialize the above operator further.

Definition 2.1.2. Given a *threshold* $\theta \in \mathbb{R}_{\geq 0}$, we define the *thresholding operator*

$$T_\theta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$$
$$T_\theta = T_\theta^{0,x}$$

The operator specified in Definition 2.1.2 is commonly called *hard-thresholding* since there is a noncontinuous jump at the threshold value. There exists a continuous thresholding operator

where additionally all values above the threshold are shrinked by the same value to eliminate the jump. A formal definition of such a *soft-thresholding* operator will be provided in Section 8.3.2.

2.2. Advanced Artifact and Noise Reduction

In addition to thresholding, it may be beneficial to apply additional procedures for reducing noise or artifacts from scans.
Regarding denoising, [44] gives a good overview over existing noise reduction techniques. These include simple filters, wavelet-based methods, statistical and also PDE-based approaches. A promising method was introduced in [45] which computes a denoised image by a maximum a posteriori procedure whose parametrization is given by a solution of the Lyapunov equation used in stability analysis of differential equations.
Similar yet different methods focus on artifact reduction. There is a broad variety of artifacts occuring in tomography due to a lot of physical effects taking place, e.g., rings, streaks or beam hardening, to just name a few. A comprehensive introduction to artifacts in computed tomography can be found in [46]. Recent advances to reduce these effects include [47] which aims to solve the mentioned problem using deep learning with convolutional neural networks. Comprehensive surveys over related methods can be found in [48] which focusses on sinogram and special reconstruction methods reducing these unwanted effects before or during reconstruction of the digital model, while another survey in [49] compares several reduction methods to be applied after reconstruction.
As a small final remark we note that although many of the cited technologies look promising, our applications do not use them due to framework restrictions. Hence, all results presented later were produced without them. But in general, we would suggest to apply them too, as segmentation methods certainly benefit from better input data quality.

We now move on to present postprocessing methods used to improve segmentation results.

2.3. Removing Isolated Voxels

An often occuring phenomenon in our voxelwise segmentation is that individual volume elements are selected to be in the result, which however do not belong to the desired part at all. That might happen due to the local environment around such voxels being structurally similar to the ones of the object to be segmented. Artifacts can cause this too.

Definition 2.3.1. Let $K \in 2\mathbb{N}+1$ be an odd environment size and $n \in \{0, \dots, K^3-1\}$ be some voxel count threshold. Consider a local environment $R_K(\boldsymbol{x_\alpha})$ around a voxel $\boldsymbol{x_\alpha}$ in the iterable region of a volume according to Definition 1.4.6.
We call $\boldsymbol{x_\alpha}$ n-isolated if

$$\#\left\{\boldsymbol{x_\beta} \in R_K(\boldsymbol{x_\alpha}) \setminus \{\boldsymbol{x_\alpha}\} \mid \mathrm{v}(\boldsymbol{x_\beta}) > 0\right\} < n.$$

We see that a voxel being isolated depends on a threshold defining the minimum number of neighboring voxels required to have a positive value. This implies a simple postprocessing step which thresholds isolated voxels away, as given in the following definition.

Definition 2.3.2. Consider the setting of Definition 2.3.1 and let $n \in \{0, \dots, K^3 - 1\}$ be a voxel count threshold. Then define the *isolated voxel removal operator* $\mathcal{R}_n$ by

$$\mathcal{R}_n \colon V \to V$$
$$\widetilde{\mathrm{v}}(\boldsymbol{x_\alpha}) = \begin{cases} 0, & \boldsymbol{x_\alpha} \text{ is } n\text{-isolated}, \\ \mathrm{v}(\boldsymbol{x_\alpha}), & \text{otherwise}. \end{cases}$$

Using the introduced removal operator, one can implement a removal technique which lets an expert choose the threshold n interactively as well as the environment size used.

2.4. Connected Components Analysis

Any voxelwise classification procedure proposed in this thesis yields a voxel volume where either a binary decision or a likelihood value in a discrete range is assigned to each voxel. If we additionally want to identify certain components, we mostly aggregate them by a connectedness criterion, i.e., we apply a *connected component labeling* algorithm. Multiple different techniques to do so were proposed in literature, we choose a simple two-pass algorithm similar to the one introduced in [50] but lifted to the three-dimensional case.
The result is again a volume in which to each voxel there is assigned either zero, the background value, or a unique index denoting the component the voxel belongs to. Individual components then can be extracted by simple thresholding on these indices.

2.5. Applying Further Segmentation Algorithms

Lastly, we want to emphasize that when applying interactive segmentation algorithms like we do, it is almost always beneficial to combine multiple methods to achieve a good result. Typically, one first applies simple thresholding to reduce measurement noise. Next, we suggest applying one of our proposed segmentation algorithms followed by one or more postprocessing steps mentioned before. All steps then can be iterated until the result is satisfactory.
A typical workflow in our applications is given in Algorithm 2.

Algorithm 2: Typical iterative segmentation workflow.

$V^0 \leftarrow$ preprocess volume V
$k \leftarrow 0$
repeat
  $k \leftarrow k + 1$
  $V_s^k \leftarrow$ segment V^{k-1}
  $V^k \leftarrow$ remove isolated voxels from V_s^k
until V^k *is satisfactory*
$V^{\mathrm{final}} \leftarrow$ select connected components from V^k

References

[1] Fahmi Khalifa et al. "3D Kidney Segmentation from CT Images Using a Level Set Approach Guided by a Novel Stochastic Speed Function". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 587–594.

[2] Piotr Swierczynski et al. "A level-set approach to joint image segmentation and registration with application to CT lung imaging". In: *Computerized Medical Imaging and Graphics* 65 (2018). Advances in Biomedical Image Processing, pp. 58–68.

[3] A Yezzi, Lilla Zollei and T Kapur. "A variational framework for integrating segmentation and registration through active contours". In: *Medical image analysis* 7 (July 2003), pp. 171–185.

[4] Phan T. H. Truc et al. "A Study on the Feasibility of Active Contours on Automatic CT Bone Segmentation". In: *Journal of Digital Imaging* 23.6 (Dec. 2010), pp. 793–805.

[5] Daniel Cremers, Mikael Rousson and Rachid Deriche. "A Review of Statistical Approaches to Level Set Segmentation: Integrating Color, Texture, Motion and Shape". In: *International Journal of Computer Vision* 72.2 (Apr. 2007), pp. 195–215.

[6] Lei He et al. "A comparative study of deformable contour methods on medical image segmentation". In: *Image and Vision Computing* 26.2 (2008), pp. 141–163.

[7] Bo Peng, Lei Zhang and David Zhang. "A Survey of Graph Theoretical Approaches to Image Segmentation". In: *Pattern Recognition* 46.3 (2013), pp. 1020–1038.

[8] Xinjian Chen and Lingjiao Pan. "A Survey of Graph Cuts/Graph Search Based Medical Image Segmentation". In: *IEEE Reviews in Biomedical Engineering* 11 (2018), pp. 112–124.

[9] Christian Bauer et al. "Segmentation of interwoven 3d tubular tree structures utilizing shape priors and graph cuts". In: *Medical Image Analysis* 14.2 (2010), pp. 172–184.

[10] Yongyue Zhang, Michael Brady and Stephen Smith. "Segmentation of Brain MR Images Through a Hidden Markov Random Field Model and the Expectation-Maximization Algorithm". In: *IEEE Transactions on Medical Imaging* 20.1 (Jan. 2001), pp. 45–57.

[11] Ivana Išgum et al. "Multi-Atlas-Based Segmentation With Local Decision Fusion - Application to Cardiac and Aortic Segmentation in CT Scans". In: *IEEE Transactions on Medical Imaging* 28.7 (July 2009), pp. 1000–1010.

[12] Angel Torrado-Carvajal et al. "Multi-Atlas and Label Fusion Approach for Patient-Specific MRI Based Skull Estimation". In: *Magnetic Resonance in Medicine* 75.4 (2016), pp. 1797–1807.

[13] Torsten Rohlfing et al. "Quo Vadis, Atlas-Based Segmentation?" In: *Handbook of Biomedical Image Analysis: Volume III: Registration Models*. Springer US, 2005, pp. 435–486.

[14] Martin Thoma. *A Survey of Semantic Segmentation*. 2016. arXiv: 1602.06541.

[15] Bo Zhao et al. "A Survey on Deep Learning-based Fine-grained Object Classification and Semantic Segmentation". In: *International Journal of Automation and Computing* 14.2 (Apr. 2017), pp. 119–135.

[16] Özgün Çiçek et al. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation". In: *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016*. Cham: Springer International Publishing, 2016, pp. 424–432.

[17] Wentao Zhu et al. "AnatomyNet: Deep learning for fast and fully automated whole-volume segmentation of head and neck anatomy". In: *Medical Physics* 46.2 (Feb. 2019), pp. 576–589.

[18] Ahmed Taha et al. "Kid-Net: Convolution Networks for Kidney Vessels Segmentation from CT-Volumes". In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018*. Cham: Springer International Publishing, 2018, pp. 463–471.

[19] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. "A Survey on Instance Segmentation: State of the art". In: *International Journal of Multimedia Information Retrieval* 9.3 (July 2020), pp. 171–189.

[20] Farhana Sultana, Abu Sufian and Paramartha Dutta. "Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey". In: *Knowledge-Based Systems* 201-202 (June 2020), p. 106062.

[21] René Ranftl and Thomas Pock. "A Deep Variational Model for Image Segmentation". In: *Pattern Recognition*. Springer International Publishing, 2014, pp. 107–118.

[22] Markus Unger et al. "TVSeg - Interactive Total Variation Based Image Segmentation". In: *Proceedings of British Machine Vision Conference (BMVC 2008)*. Jan. 2008.

[23] Jakob Santner, Thomas Pock and Horst Bischof. "Interactive Multi-Label Segmentation". In: *Proceedings of the 10th Asian Conference on Computer Vision - Volume Part I*. ACCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 397–410.

[24] Peter Hastreiter and Thomas Ertl. "Fast and Interactive 3D-Segmentation of Medical Volume Data". In: *Proc. Image and Multidimensional Digital Signal Processing (IMDSP'98)*. 1998, pp. 41–44.

[25] X. Liao et al. "Iteratively-Refined Interactive 3D Medical Image Segmentation With Multi-Agent Reinforcement Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 9391–9399.

[26] Feng Zhao and Xianghua Xie. "Interactive Segmentation of Medical Images: A Survey". In: *Computer Science Swansea University, Singleton Park Swansea SA2 8PP, UK* (2012), pp. 263–270.

[27] Pavel Iassonov, Thomas Gebrenegus and Markus Tuller. "Segmentation of X-ray computed tomography images of porous materials: A crucial step for characterization and quantitative analysis of pore structures". In: *Water Resources Research* 45 (Sept. 2009).

[28] Patrick Fuchs, Thorben Kröger and Christoph S. Garbe. "Self-supervised Learning for Pore Detection in CT-Scans of Cast Aluminum Parts". In: *The e-Journal of Nondestructive Testing* 24.11 (July 2019).

[29] Avinash Jaiswal et al. "Markov random field segmentation for industrial computed tomography with metal artefacts". In: *Journal of X-ray science and technology* 26.4 (Apr. 2018), pp. 573–591.

[30] Yuki Doi et al. "3D Segmentation of CT Volumetric Image for Mechanical Assemblies with Forcible Edge Enhancement". In: *The e-Journal of Nondestructive Testing* 25.2 (Feb. 2020).

[31] Nobelprize.org. *MLA style: The Nobel Prize in Physics 1901*. Online. Accessed on September 10th, 2020.

[32] Wilhelm Conrad Röntgen. "Über eine neue Art von Strahlen". In: *Sitzungsberichte der Physikalisch-Medizinischen Gesellschaft zu Würzburg* 3 (Jan. 1896), pp. 1–22.

[33] Wilhelm Conrad Röntgen. *Hand mit Ringen*. Online. Courtesy of the *Physikalisches Institut der Universität Würzburg*. Dec. 1895. URL: https://commons.wikimedia.org/w/index.php?title=File:Roentgen_first_medical_xray.jpg&redirect=yes.

[34] Wim van Aarle. "Tomographic segmentation and discrete tomography for quantitative analysis of transmission tomography data". PhD thesis. University of Antwerp, 2012.

[35] Johann Radon. "Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten". In: *Berichte über die Verhandlungen der Königlich-Sächsischen Gesellschaft der Wissenschaften zu Leipzig. Mathematisch-Physische Klasse* 69 (Apr. 1917), pp. 262–277.

[36] Avinash C. Kak and Malcolm Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988.

[37] Alex Gustschin. *Translation: About one scheme of tomography*. 2020. arXiv: 2004.03750.

[38] Nobelprize.org. *MLA style: The Nobel Prize in Physiology or Medicine 1979*. Online. Accessed on September 11th, 2020.

[39] W. H. Oldendorf. "Isolated Flying Spot Detection of Radiodensity Discontinuities - Displaying the Internal Structural Pattern of a Complex Object". In: *IRE Transactions on Bio-Medical Electronics* 8.1 (1961), pp. 68–72.

[40] Laurance D. Hall and Thomas A. Carpenter. "Magnetic resonance imaging: A new window into industrial processing". In: *Magnetic Resonance Imaging* 10.5 (1992), pp. 713–721.

[41] F. J. Dickin et al. "Tomographic imaging of industrial process equipment: techniques and applications". In: *IEE Proceedings G - Circuits, Devices and Systems* 139.1 (1992), pp. 72–82.

[42] Thorsten Buzug. *Computed Tomography : From Photon Statistics to Modern Cone-Beam CT*. Jan. 2008, 522 pp.

[43] Frank Natterer. *The Mathematics of Computerized Tomography*. USA: Society for Industrial and Applied Mathematics, 2001.

[44] J. Mohan, V. Krishnaveni and Yanhui Guo. "A survey on the magnetic resonance image denoising methods". In: *Biomedical Signal Processing and Control* 9 (2014), pp. 56–69.

[45] João M. Sanches, Jacinto C. Nascimento and Marques Jorge S. "Medical Image Noise Reduction Using the Sylvester–Lyapunov Equation". In: *IEEE Transactions on Image Processing* 17.9 (2008), pp. 1522–1539.

[46] Julia F. Barret and Nicholas Keat. "Artifacts in CT: Recognition and Avoidance". In: *RadioGraphics* 24.6 (Nov. 2004), pp. 1679–1691.

[47] A. Trbalić et al. "CT Metal Artefacts Reduction Using Convolutional Neural Networks". In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. May 2019, pp. 251–255.

[48] Andre Mouton et al. "An experimental survey of metal artefact reduction in computed tomography". In: *Journal of X-ray science and technology* 21.2 (2013), pp. 193–226.

[49] Shrinivas D. Desai and Linganagouda Kulkarni. "Comprehensive Survey on Metal Artifact Reduction Methods in Computed Tomography Images". In: *International Journal of Rough Sets and Data Analysis* 2 (June 2015), pp. 92–114.

[50] Luigi Di Stefano and Andrea Bulgarelli. "A Simple and Efficient Connected Components Labeling Algorithm". In: *Proceedings 10th International Conference on Image Analysis and Processing*. 1999, pp. 322–327.

Part II.

Describing Local Geometry

Geometry, in particular, is a Greek invention, without which modern science would have been impossible.

Bertrand Russel, A History of Western Philosophy

Table of Contents

In this part, we describe how local voxel environments can be assessed based on geometric features. Descriptions of how much a region resembles the introduced geometries will be aggregated in feature vectors which play a fundamental role in later segmentation techniques.
We start by first describing a custom histogram embedding which enables us to make use of the popular Wasserstein distance when comparing feature vectors. The following chapter then defines the features used throughout this thesis which also incorporate our embedding.

3. A Wasserstein Histogram Embedding

Abstract Often, the feature vectors computed from local regions are histograms which can be interpreted as discrete probability distributions. However, comparing histograms by vector norms is not preferable since they do not consider the positions of the coefficients, but the latter plays a fundamental role in probability distributions. The Wasserstein distances, however, allow for a better comparison. This chapter begins by providing an example where a simple vector norm comparison fails to distinguish histograms while the Wasserstein distance can. Next, the family of Wasserstein distances is formally introduced. After that, an embedding used to approximate the better metric is proposed. Finally, certain approximation qualities as well as efficient implementation algorithms are listed.

3.1. Introduction

It is common to compare histograms not by their Euclidean distance but rather using a so called *Wasserstein distance*. The family of Wasserstein distances models optimal transport with respect to some ground distance metric.

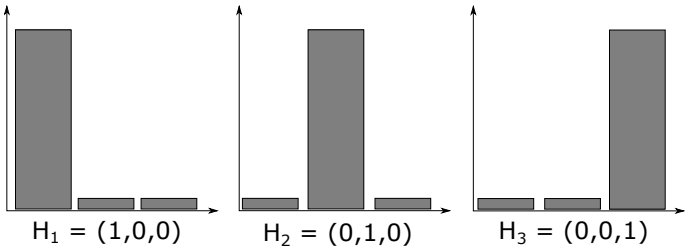


Figure 3.1.: The idea behind Wasserstein histogram comparison.

The basic idea behind them is visualized in Figure 3.1 showing three normalized histograms. As one can see, a simple comparison of the depicted vectors using the Euclidean distance yields the same value, i.e., all three histograms have the same distance to each other when considered as vectors, making them indistinguishable. However, intuitively we would say that the histogram $\mathcal{H}_1$ seems closer to $\mathcal{H}_2$ than to $\mathcal{H}_3$. That is, because it takes more *energy* to *move* the leftmost column of $\mathcal{H}_1$ to the right so it matches $\mathcal{H}_3$ than it does to do the same to match $\mathcal{H}_2$. The p-Wasserstein distance $W_p, p \geq 1$ helps disambiguating them as it produces $W_p(\mathcal{H}_1, \mathcal{H}_2) < W_p(\mathcal{H}_1, \mathcal{H}_3)$. In the following, we will introduce these metrics and provide an embedding for the case $p = 2$.

The *Wasserstein* (or *Monge-Kantorovich*) distance is a distance function between probability measures which often gives a natural way of comparison. As motivated, it describes the least work necessary to transform one measure into the other without losing any mass.

In general, the p-Wasserstein distance $W_p(\mu, \nu)$ for two probability measures $\mu, \nu \in \mathcal{P}(\mathcal{X})$ over some complete and separable metric space $(\mathcal{X}, c)$ endowed with its Borel σ-algebra is defined in [1] by

$$W_p(\mu, \nu) = \left(\inf_{\pi \in \Gamma(\mu,\nu)} \int_{\mathcal{X}\times\mathcal{X}} (c(x, y))^p \, \mathrm{d}\pi(x, y) \right)^{1/p},$$

where $\Gamma(\mu, \nu)$ is the set of all measures on $\mathcal{X} \times \mathcal{X}$ with marginals μ and ν on its first and second components, respectively. That is, we have the additional constraints that $\pi(A \times \mathcal{X}) = \mu(A)$ and $\pi(\mathcal{X} \times B) = \nu(B)$ for all Borel sets $A, B \subseteq \mathcal{X}$ [1].
Many algorithms in machine learning, most notably Support Vector Machines, act in Euclidean space and consider histogram features which represent d-dimensional discretizations of probability measures. In this context, we recapitulate the definitions of a cumulative distribution function associated to a probability measure [2].

Definition 3.1.1. Let ν be a probability measure on $\mathbb{R}$. The mapping

$$F_\nu \colon \mathbb{R} \to [0, 1], \quad x \mapsto \nu\left((-\infty, x]\right)$$

is called the *cumulative distribution function* (cdf) for ν.
For a probability mass function ν interpreted as a vector $\nu \in \mathbb{R}^d$, the *discrete cdf* is defined as $F_\nu(i) = \sum_{j=1}^{i} \nu_j,\ i = 1, \ldots, d$.

Even in this relatively simple setting, the 1-Wasserstein distance (aka *Earth Mover's distance*) is used primarily because it can be computed efficiently using

$$W_1(\mu, \nu) = \inf_{\pi \in \Gamma(\mu,\nu)} \int_{\mathbb{R}\times\mathbb{R}} |x - y| \, \mathrm{d}\pi(x, y) = \int_{\mathbb{R}} |F_\mu(x) - F_\nu(x)| \, \mathrm{d}x$$

as shown by Vallander [3].
However, this definition uses the ℓ_1 ground distance metric, whereas in Euclidean space the natural distance metric would be the ℓ_2 metric which induces the 2-Wasserstein distance. Unfortunately, there does not exist such an easy isometric relation in the latter case.
In the subsequent sections, we introduce an embedding for probability measures over $\mathbb{R}$ such that their regular Euclidean distance converges against the 2-Wasserstein distance of the original measures.

3.2. Related Work

In many scientific fields, efforts have been made to use the Wasserstein distances, and most often the Earth Mover's distance is used as mentioned before and new algorithms or kernels for existing kernelized methods are provided. Publications involve [4–9], to just cite a few.
Regarding the 2-Wasserstein distance, current scientific progress mostly focusses on yet another machine learning approach to learn such embeddings discussed in [10, 11].
Other publications focus on modified loss functions that involve the 2-Wasserstein distance by either solving a linear program each time (cf. [12], [13]) or by deriving estimates of it proportional to the Kullback-Leibler divergence (cf. [14]).
Recently, [15] used a *Sliced Wasserstein distance* which, similar to this chapter, aims to construct an embedding and also proves certain properties about derived kernels. However, their embedding uses the Radon transform which seems intractable for large scale applications, and use of their derived kernels also required a modification of the used frameworks.

3.3. Embedding the 2-Wasserstein Distance

As we focus on embedding the 2-Wasserstein distance W_2, we briefly recapitulate the definition of this distance metric in our setting.

Definition 3.3.1. Let $\mu, \nu \in \mathcal{P}(\mathbb{R})$ be two probability measures and let $\Gamma(\mu, \nu)$ denote the set of measures on $\mathbb{R} \times \mathbb{R}$ having marginals μ and ν on its first and second components, respectively. Then, the squared 2-Wasserstein distance W_2 is defined by

$$W_2^2(\mu, \nu) = \inf_{\pi \in \Gamma(\mu,\nu)} \int_{\mathbb{R}\times\mathbb{R}} |x - y|^2 \, \mathrm{d}\pi(x, y).$$

To define our embedding, we further reintroduce the quantile function (or generalized inverse distribution function) associated to a distribution function [2].

Definition 3.3.2 (Quantile function)**.** Let μ be a probability measure with its associated cumulative distribution function F_μ. Then the mapping

$$F_\mu^{-1} \colon (0, 1) \to \mathbb{R}$$
$$p \mapsto \inf \{x \in \mathbb{R} \mid F_\mu(x) \geq p\}$$

is called the *quantile function* for μ.

Using these definitions, we now define our embedding which also consists of a function approximation by $M \in \mathbb{N}$ terms. We approximate them using their first Fourier coefficients.

Definition 3.3.3 (Wasserstein embedding)**.** Let $\mu \in \mathcal{P}(\mathbb{R})$ be a probability measure on $\mathbb{R}$. Define the *Wasserstein embedding* of dimension $M \in \mathbb{N}$ as

$$\mathcal{E}_M(\mu) = \left(\langle F_\mu^{-1}, \xi_n \rangle\right)_{n=1}^M$$

where F_μ^{-1} is the corresponding quantile function and the system

$$\begin{aligned} \xi_1 &= 1, \\ \xi_{2n} &= \sqrt{2}\cos(2\pi n \cdot), \quad n > 0, \\ \xi_{2n+1} &= \sqrt{2}\sin(2\pi n \cdot), \quad n > 0, \end{aligned}$$

is the classical sine/cosine basis of $\mathrm{L}^2(0, 1)$.

In other words, the Wasserstein embedding is the Fourier transform of the quantile function.
Our goal is to construct an embedding such that the 2-Wasserstein metric between two distributions is approached by the ℓ_2 distance between their embeddings.
We first prove the pointwise convergence.

Theorem 3.3.4. Let μ, ν be probability measures with associated quantile functions F_μ^{-1} and F_ν^{-1}, respectively. Assume that $g := F_\mu^{-1} - F_\nu^{-1} \in \mathrm{L}^2(0, 1)$. Let $W_2(\mu, \nu)$ denote the 2-Wasserstein distance between them. Then

$$\|\mathcal{E}_M(\mu) - \mathcal{E}_M(\nu)\|_2 \xrightarrow{M\to\infty} W_2(\mu, \nu).$$

Proof. Panaretos and Zemel [1] showed that an isometry into $\mathrm{L}^2(0, 1)$ using the quantile functions is given by $\mu \mapsto F_\mu^{-1}$ and that the Wasserstein distance can be expressed as

$$W_2^2(\mu, \nu) = \int_0^1 |F_\mu^{-1}(x) - F_\nu^{-1}(x)|^2 \, \mathrm{d}x. \tag{3.1}$$

Let $\{\xi_n\}_{n\in\mathbb{N}}$ be any orthonormal basis of $\mathrm{L}^2(0, 1)$. Then we get

$$\|\mathcal{E}_M(\mu) - \mathcal{E}_M(\nu)\|_2^2 = \sum_{n=1}^M |\langle F_\mu^{-1}, \xi_n\rangle - \langle F_\nu^{-1}, \xi_n\rangle|^2 = \sum_{n=1}^M |\langle g, \xi_n\rangle|^2.$$

As M approaches infinity, the series converges against the 2-Wasserstein distance since

$$\lim_{M\to\infty}\sum_{n=1}^{M}|\langle g,\xi_n\rangle|^2=\sum_{n=1}^{\infty}|\langle g,\xi_n\rangle|^2=\|g\|^2\overset{(3.1)}{=}W_2^2(\mu,\nu).$$

Therefore, pointwise convergence is immediate. □

In this very general case, we cannot formulate any statements about the rate of convergence. Let us now improve this by restricting our assumptions.
To this end, we note that the general approximation error satisfies

$$E_M(g):=\|g\|^2-\sum_{n=1}^{M}|\langle g,\xi_n\rangle|^2=\sum_{n=1}^{\infty}|\langle g,\xi_n\rangle|^2-\sum_{n=1}^{M}|\langle g,\xi_n\rangle|^2=\sum_{n=M+1}^{\infty}|\langle g,\xi_n\rangle|^2.$$

In the following, let again $g=F_\mu^{-1}-F_\nu^{-1}$ denote the difference between some quantile functions specified in the later lemmas. Furthermore, let $\|g\|_V$ be the total variation of such a function g.

Lemma 3.3.5. Let μ,ν be probability measures whose quantile functions are bounded. Then

$$\left|\|\mathcal{E}_M(\mu)-\mathcal{E}_M(\nu)\|_2^2-W_2^2(\mu,\nu)\right|=\mathcal{O}(\|g\|_V^2M^{-1}).$$

Proof. Again, we use quantile functions F_μ^{-1} and F_ν^{-1} which are bounded by assumption and nondecreasing by definition and therefore of bounded variation. Thus, their difference function $g=F_\mu^{-1}-F_\nu^{-1}$ is of bounded variation as well, denoted by $\|g\|_V=\int_0^1|g'(t)|\mathrm{d}t<\infty$ where g' is the derivative of g in the distributional sense.
It is well-known (see e.g. [16, Theorem 4.12]) that the Fourier coefficients of a function of bounded variation satisfy

$$|\langle g,\sqrt{2}\cos(2\pi n\cdot)\rangle|,\ |\langle g,\sqrt{2}\sin(2\pi n\cdot)\rangle|\le\frac{\sqrt{2}\|g\|_V}{\pi n}.$$

Thus, for the approximation error we have that

$$E_M(g)=\sum_{n=M+1}^{\infty}|\langle g,\xi_n\rangle|^2\le\sum_{n=M+1}^{\infty}\frac{2\|g\|_V^2}{\pi^2n^2}=\frac{2\|g\|_V^2}{\pi^2}\sum_{n=M+1}^{\infty}\frac{1}{n^2}\le\frac{2\|g\|_V^2}{\pi^2M}.$$

□

We see that for bounded quantile functions the approximation converges linearly. Over any family of measures with uniformly bounded variation, that convergence is clearly uniform. One particularly interesting case is the following.

Corollary 3.3.6. Given a family $\mathcal{G}$ of probability measures with uniformly bounded quantile functions, i.e., $-\infty<C_{\min}\le F_\mu^{-1}(x)\le C_{\max}<+\infty$ for any $\mu\in\mathcal{G}$ and all $x\in(0,1)$.
Then, for all $\mu,\nu\in\mathcal{G}$

$$\|\mathcal{E}_M(\mu)-\mathcal{E}_M(\nu)\|_2\xrightarrow{\text{unif.}}W_2(\mu,\nu),\quad(M\to\infty).$$

Proof. It suffices to show that for any $\mu,\nu\in\mathcal{G}$ the variation of g is uniformly bounded. But

$$\|g\|_V\le\|F_\mu^{-1}\|_V+\|F_\nu^{-1}\|_V\le2(C_{\max}-C_{\min}),$$

where $C_{\max}$ and $C_{\min}$ is the uniform upper and lower bound, respectively. Here we used the monotonicity of the quantile functions. □

Remark 3.3.7. As Fourier approximations are particularly efficient for smooth functions, we get even better rates of convergence over families of measures with smooth quantile functions.

3.4. Implementing the Embedding

Since the embedding acts on probability distributions, an implementation of it must first normalize the input histogram such that its entries sum to one. Furthermore, the approximation quality depends on the number of approximation terms $M \in \mathbb{N}$ that are used which we choose always odd since our chosen basis of $\mathrm{L}^2(0,1)$ consists of the constant function 1 and pairs of sine and cosine functions. If $M \leq d$, we compute $M' = \max\{M, \lfloor 1.25\, d \rfloor\}$ as determined experimentally and keep that number odd, i.e., $M = M' + 1 - (M' \mod 2)$. This M is then fed into Algorithm 3. Another important parameter is the number of quantiles Q used to approximate the quantile function. Generally, the higher the number of quantiles, the better the approximation gets, but also the longer the computation of the embedding takes.
Algorithm 3 shows the computation of a histogram embedding given approximation quantities M and Q for the embedding and quantile function, respectively. In it, the symbol $\odot$ denotes elementwise multiplication of two arrays of same length or of each array element with a scalar value. Furthermore, $\mathbb{E}[\boldsymbol{x}]$ denotes the arithmetic mean of all elements in $\boldsymbol{x}$.

Algorithm 3: Computation of a histogram embedding.

Function `EmbedHist`$(\mathcal{H}, M, Q)$
- **Input :** A histogram $\mathcal{H} \in \mathbb{R}^d_{\geq 0}$
- **Input :** Number of terms $M \in 2\mathbb{N}+1$
- **Input :** Number of quantiles $Q \in \mathbb{N}$
- **Output:** An embedded histogram $\mathcal{E}_M(\mathcal{H}) \in \mathbb{R}^M$
- $\mathcal{H}_n \leftarrow$ normalize($\mathcal{H}$)
- $\boldsymbol{y} \leftarrow (k/Q)_{k=0}^{Q-1}$
- $\boldsymbol{q} \leftarrow$ `Quantiles` $(\mathcal{H}_n, \boldsymbol{y})$
- $\boldsymbol{e} \leftarrow (\mathbb{E}[\boldsymbol{q}])$
- **for** $n \leftarrow 1$ **to** $\lfloor M/2 \rfloor$ **do**
  - $\boldsymbol{a} \leftarrow \boldsymbol{y} \odot 2\pi n$
  - $\boldsymbol{e} \leftarrow (\boldsymbol{e}, \sqrt{2}\, \mathbb{E}[\boldsymbol{q} \odot \cos(\boldsymbol{a})])$
  - $\boldsymbol{e} \leftarrow (\boldsymbol{e}, \sqrt{2}\, \mathbb{E}[\boldsymbol{q} \odot \sin(\boldsymbol{a})])$
- $\mathcal{E}_M(\mathcal{H}) \leftarrow \boldsymbol{e}$

Function `Quantiles`$(\mathcal{H}, S)$
- **Input :** A normalized histogram $\mathcal{H} \in \mathbb{R}^d_{\geq 0}$
- **Input :** A sequence S of N evenly spaced numbers over the interval $[0, 1)$
- **Output:** The vector $\boldsymbol{q}$ of estimated quantile values
- $\boldsymbol{q} \leftarrow ()$
- $i \leftarrow 1$
- partialSum $\leftarrow \mathcal{H}_1$
- **for** $k \leftarrow 1$ **to** N **do**
  - **while true do**
    - **if** $S_k \leq$ partialSum **or** $i = d$ **then**
      - $\boldsymbol{q} \leftarrow (\boldsymbol{q}, i)$
      - **break**
    - **else**
      - $i \leftarrow i + 1$
      - partialSum $\leftarrow$ partialSum $+\mathcal{H}_i$
- **return** $\boldsymbol{q}$

3.5. Applications and Consequences

While the modeling and the experiments show that we can indeed mimic and utilize the 2-Wasserstein distance in practical applications, we are aware that other applications, for example, use specialized kernels to do so [15, 17].
However, [15] and [17] use histogram features *exclusively*, i.e., without any additional features. In contrast, using this embedding allows to use both unrelated numerical features as well as histogram features in the same application. So, given a feature vector composed of unrelated as well as histogram feature values, replacing the histograms by their embeddings inside the feature vector compares unrelated feature values using the Euclidean distance while histograms are compared using the 2-Wasserstein distance.
As an example, consider a feature vector of the form

$$f = (f_1, f_2, \underbrace{f_3, \ldots, f_d}_{=:f_h}),$$

where f_1, f_2 are two unrelated features, while f_h is some histogram feature. Conceptually, we would like to compare two vectors f, g of this form using

$$(d(f,g))^2 = (f_1 - g_1)^2 + (f_2 - g_2)^2 + W_2^2(f_h, g_h) \tag{3.2}$$

Using our embedding, this is possible by embedding the histogram feature to obtain new feature vectors

$$\begin{aligned} f' &= (f_1, f_2, \overbrace{f'_3, \ldots, f'_{M+2}}^{=\mathcal{E}_M(f_h)}), \\ g' &= (g_1, g_2, \underbrace{g'_3, \ldots, g'_{M+2}}_{=\mathcal{E}_M(g_h)}). \end{aligned}$$

The desired comparison in (3.2) is now achieved by $\|f' - g'\|_2^2$. This again symbolizes that the resulting distance function is just the regular Euclidean norm which is natively used by many classifiers.
Especially with regard to the fact that the presented embedding is intended to be used in machine learning applications, we want to stress that often a sufficient number of training data instances might render the benefits introduced void. In such a case, the results will not be improved but the runtime increases. However, numerical experiments hint that the performance is only affected slightly. But, we conjecture that the better comparability induced by using the Wasserstein distance embedding enables us to achieve a good segmentation with relatively few seed voxels.

4. Describing Local Structure

Abstract All techniques introduced in subsequent chapters have the same foundation, namely that they use features based on *local* voxel environments. Currently, the majority of related applications rely entirely on voxel region descriptors derived from the distribution of grayscale values in it, with few approaches additionally considering geometry (cf. [18]) or texture (cf. [19]). The latter methods are limited in the sense that they either fix the environment size or involve costly operations slowing the overall segmentation down.
This chapter will improve upon this by first introducing means to make descriptors more robust against measurement noise, especially if histograms are produced. This is followed by a set of features which either enhance existing or introduce new ones, both explicitly considering nongeometric and geometric indicators. For each presented feature it will be discussed if its result shall be embedded using the Wasserstein embedding introduced in the preceding chapter. Finally, a feature selection method is given for keeping relevant features while simultaneously reducing the computational effort.

We further make use of the following notation.

Definition 4.0.1. Let $R_K(\boldsymbol{x_\alpha})$ be a local environment of dimensions $K \times K \times K$ around a voxel $\boldsymbol{x_\alpha}$ in the iterable region of a volume according to Definition 1.4.6.
We denote by $V_K(\boldsymbol{x})$ the collection of all voxel values in the environment around the voxel, i.e., $V_K(\boldsymbol{x_\alpha}) = \{\mathrm{v}(\boldsymbol{x_\beta}) \mid \boldsymbol{x_\beta} \in R_K(\boldsymbol{x_\alpha})\}$, where multiple values are admissible too.

4.1. Local Thresholding

In order to improve the noise robustness of the features introduced subsequently, we apply a simple and fast thresholding on the local environment. For the threshold, we choose the mean of the grayscale values in the region.

Definition 4.1.1. Let $R_K(\boldsymbol{x_\alpha})$ be any local environment around some voxel $\boldsymbol{x_\alpha}$ in a volume's iterable region. Let $V_K(\boldsymbol{x_\alpha})$ be the set of grayscale values in that region.
Then *local thresholding* yields a new environment $R'_K(\boldsymbol{x_\alpha})$ around the same voxel $\boldsymbol{x_\alpha}$ with the new voxel values

$$V'_K(\boldsymbol{x_\alpha}) = \{T_\mu(v) \mid v \in V_K(\boldsymbol{x_\alpha})\}\,,$$

where $\mu = \mathbb{E}[V_K(\boldsymbol{x_\alpha})]$ denotes the mean over all voxel values in the region and T_μ is the hard-thresholding operator according to Definition 2.1.2. Again, multiple values are admissible here.

The reason we use the mean of the voxel values instead of their median is that if the majority of the voxel values belong to material voxels the median would indicate the same. Hence, the local thresholding would clear the entire region without recognizing material. Although this is a very simple heuristic, experiments showed that the robustness of features against noise is considerably improved, especially in combination with occasional thresholding later on. Thus, before the computation of any of the following features, we perform such a local thresholding and obtain a new local region. For efficiency, we also record for a given region if it is *completely homogenous* for which we can in many cases compute default values for features and save costly operations.

4.2. Histogram Thresholding

A phenomenon occuring often during the generation of histograms as feature vectors is that noisy input contributes to it in form of individual bins with a small number of elements. We compensate this by additionally thresholding the histogram values but still redistributing the thresholded "mass" equally among the other bins, thus the sum over all histogram entries remains the same.

Definition 4.2.1. Let $\mathcal{H} \in \mathbb{R}^d_{\geq 0}$ be some d-dimensional histogram and let $\theta \geq 0$ be some threshold. First, we compute the thresholding $\mathcal{H}^t = (T_\theta(\mathcal{H}_k))_{k=1}^d$. The *lost mass* is given by $m = \|\mathcal{H} - \mathcal{H}^t\|_1$ and define $n^t := \sum_{k=1}^d \mathbb{1}(\mathcal{H}_k^t > 0)$.
The histogram thresholding then computes a new histogram

$$\mathcal{H}'_k = \begin{cases} 0, & \mathcal{H}_k < \theta, \\ \mathcal{H}_k + m/n^t, & \text{otherwise}, \end{cases} \qquad k = 1, \ldots, d.$$

Empirically, we determined the median of the histogram values to be a good threshold, although other thresholds like the 25 percent quantile may be valid application-dependent choices too. Thus, all bins whose values are strictly lower than the median are emptied, while the mass lost due to the thresholding is redistributed equally across all remaining bins. Consequently, bins which are only slightly filled as is likely in noisy context are cleared while the overall mass is kept.
The following sections will introduce and generalize features both describing geometric and nongeometric features of local three-dimensional regions. In practical applications we turn individual features on and off, depending on the dataset. For instance, in most computed tomography segmentation problems we apply the subsequently described grayscale and orientation features. On the other hand, in especially difficult cases additionally the position of the center voxel might become important.

4.3. Nongeometric Features

The grayscale values in a region are certainly the most important features. Hence, we employ a set of nongeometric features based on them. They do not consider geometry, which we will complete later.

4.3.1. Position

The first feature we consider is the position of the center voxel.

Definition 4.3.1. For a local environment $R_K(\boldsymbol{x_\alpha})$ around a voxel $\boldsymbol{x_\alpha}$, the *position* feature is defined as $\mathrm{p}(\boldsymbol{x_\alpha})$.

Alternatively, the barycenter might be a fitting choice for the position too. Naturally, the position of the center voxel is useful for segmenting an individual object. In general, however, note that when segmenting multiple components one either has to provide enough seed voxels in all components or refrain from that feature. We also want to mention that the position does not generalize well, i.e., a model trained with the position feature can be applied on unseen voxel data only in rare cases. A previous registration might help here, but we do not cover that.

4.3.2. Statistical moments

The probably most used features in current segmentation systems are descriptors calculated from the grayscale value distribution, especially its first standardized statistical moments.
We choose the first four such moments as each of them has a clear interpretation regarding the shape of the distribution.

Definition 4.3.2. Consider the setting of Definition 4.1.1 and let $W := V_K(\boldsymbol{x}_{\boldsymbol{\alpha}})$ be the voxel values in a region which has a cardinality of $N := K^3$.
The first four (standardized) statistical moments μ, σ, s and k are given by

$$\begin{aligned}
\mu &= N^{-1} \sum_{v \in W} v, \\
\sigma^2 &= N^{-1} \sum_{v \in W} (v - \mu)^2, \\
s &= N^{-1} \sum_{v \in W} \left(\frac{v - \mu}{\sigma} \right)^3, \\
k &= -3 + N^{-1} \sum_{v \in W} \left(\frac{v - \mu}{\sigma} \right)^4.
\end{aligned}$$

In the given definition, μ is the mean value of the voxel values and σ is the contained standard deviation. The other metrics are the skewness s and the (excess) kurtosis k. For maximum efficiency, all four values are estimated in a single pass using the algorithm presented in [20]. This algorithm is composed of certain addition formulas known to provide a numerically stable estimation of the above four moments.

4.3.3. Local Binary Pattern

The final nongeometric feature we propose is an extension of the established *uniform local binary pattern* (LBP), which aims to capture local structure by encoding it in a bit string. To handle different environment sizes, which implies a huge growth in the number of voxels, we decided to implement a simple variant of the LBP feature, the LBP/TOP (local binary pattern for three orthogonal planes). But first, we reintroduce the classical LBP descriptor on two-dimensional images as described in [21].

Definition 4.3.3. Consider a square two-dimensional image of odd side length and some pixel $\boldsymbol{c}$ in it having a grayscale value of $c \in \mathbb{R}_{\geq 0}$. Further let $P \in \mathbb{N}$ and $R > 0$ be the parameters of the LBP descriptors denoting a number of points and a radius, respectively.
Then, we first calculate P coordinates of points sampling a circle of radius R around $\boldsymbol{c}$ equidistantly. Let $t \in \mathbb{R}^P_{\geq 0}$ be the bilinearly interpolated pixel values at the computed points.
Next, we derive a *texture vector* $\mathfrak{T} = (\mathbb{1}(t_k \geq c))_{k=1}^P$ and count the number of texture changes

$$n_c = \sum_{k=1}^{P-1} |\mathfrak{T}_k - \mathfrak{T}_{k+1}|.$$

Finally, the local binary pattern around the pixel $\boldsymbol{c}$ is given by

$$LBP = \begin{cases} \|\mathfrak{T}\|_1, & n_c \leq 2, \\ P + 1, & \text{otherwise.} \end{cases}$$

A graphical representation of how texture vectors are obtained is given in Figure 4.1. The interpolation of values on points on a circle is shown on the bottom half of the picture, while the

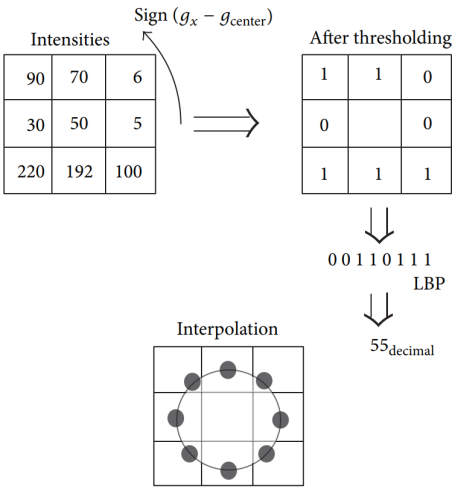


Figure 4.1.: Obtaining texture vectors in LBP [22].

top half shows conceptually how a texture vector is obtained. The final reduction to a decimal value completes the general LBP feature, that last step is replaced by the last summation step in Definition 4.3.3. In general, the local binary pattern captures the texture of a local environment in a bit string or texture vector and collapses it into a real number just as above. In our industrial use case, we notice that very often components are homogenous, e.g., is of the form 11111111, 00000000, or else consists of at most two 0-1 changes, e.g., 00111000. Special treatment for these often occuring cases is done by using the *uniform* LBP described in Definition 4.3.3 which improves the performance and obtains some kind of rotation invariance [22].
As hinted, for three-dimensional local regions we compute the LBP/TOP descriptor according to the following definition.

Definition 4.3.4. Let $R_K(\boldsymbol{x_\alpha})$ be any local region of size K according to Definition 1.4.6 centered around some voxel $\boldsymbol{x_\alpha}$. First, we compute the three orthogonal planes as *projection images* $S^{(a)}$, i.e., for all $i, j = 1, \dots, K$ we have

$$S^{(a)}_{i,j} = \sum_{k=1}^{K} \mathrm{v}\left(\boldsymbol{x}_{\boldsymbol{\beta}(k)}\right), \qquad \boldsymbol{\beta}(k) = \begin{bmatrix} k & i & i \\ i & k & j \\ j & j & k \end{bmatrix} \boldsymbol{e}_a, \quad k = 1, \dots, K,$$

for each axis $a \in \{1, 2, 3\}$. Then, for every axis we proceed as follows:

1. Compute the projection $S^{(a)}$ along the current axis.
2. Choose P points on a circle of radius R around a pixel on the projection. Compute the classical LBP descriptor value for each pixel in $S^{(a)}$ except the boundary of width R. Experimentally, we set $R = 1$ and $P = 4$.
3. Generate a histogram $\mathcal{H}_a$ of generated values having $P + 1$ bins.

The final descriptor is given by the concatenation $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$.

Although the constructed feature vector consists of three histograms, we do *not* embed them for Wasserstein comparability, since there is no known meaningful measure of similarity between texture vectors.

4.4. Geometric Features

Up to now, we described simple descriptors based solely on voxel values. In the current section, we will instead introduce new features representing geometry, i.e., compute vectors showing how similar a local environment is to some geometric primitive.
The geometries considered here will cover the following questions:

- How linear or planar is the current region?
- How curved is the locally considered surface?
- What overall shape has the environment?
- How is the region oriented?

The following sections will list the definitions of our constructed features and also explain default feature values in ambiguous or trivial cases. Additionally, we discuss whether computed histograms shall be embedded for Wasserstein comparability according to Definition 3.3.3.

4.4.1. SVD-Based Features

To answer the question about linearity or planarity of local regions, a set of descriptors fits primitives like lines and planes to the nonzero voxels in the current region (after local thresholding). We first calculate the local orientation which is implicitly encoded in a Singular Value Decomposition (SVD) over the position vectors of the voxels having a nonzero grayscale value in the current region. This orientation serves as a feature descriptor on its own.

Definition 4.4.1. Consider a local environment $R_K(\boldsymbol{x_\alpha})$ around some voxel $\boldsymbol{x_\alpha}$ in the volume's iterable region, consisting of $M \in \mathbb{N}$ nonzero voxels after local thresholding.
Construct a matrix $\boldsymbol{C} \in \mathbb{N}_0^{3\times M}$ containing the positions of these voxels relative to the environment's origin columnwise. Next, compute a centered coordinate matrix

$$\boldsymbol{C'} = \boldsymbol{C} - \boldsymbol{\mu}\mathbf{1}^T,$$

where $\boldsymbol{\mu} = M^{-1}\boldsymbol{C}\mathbf{1} \in \mathbb{R}^3$ is the vector containing the rowwise means.
The *SVD feature* consist of the three singular values of $\boldsymbol{C'}$.

Default values. If the region is completely homogenous or less than three voxels are nonzero after local thresholding, there is no detectable dominant direction in that environment. Thus, the local orientation descriptor will become the vector $\mathbf{1}$ then, indicating a fully isotropic region.

For the mentioned primitives we fit them through the center of the current region and derive descriptors based on distances to them. In both cases, the maximum possible distance of any voxel in the cubical region to either a line or a plane through the center voxel is $d_{\max} = \sqrt{3}\lfloor K/2 \rfloor$.
Next, we use the matrix $\boldsymbol{U} \in \mathbb{R}^{3\times 3}$ from the Singular Value Decomposition $\boldsymbol{C'} = \boldsymbol{U\Sigma V}^T$ computed as in Definition 4.4.1. Denote by $\boldsymbol{x_\alpha}$ the center voxel of the current region.
We know that the least squares error fit line is given by the center voxel point and the vector $\boldsymbol{u}_1$ pointing in the direction of the fit line, while the least squares error fit plane is modeled using the center voxel and the column vector $\boldsymbol{u}_3$ which is orthonormal to the fit plane. Additionally, we need an offset point lying on the line or plane, respectively, for which we choose explicitly the center voxel $\boldsymbol{x_\alpha}$. Typically, one would choose the center of gravity, however, this implies that computed characteristics cannot distinguish between planes or lines passing through the center voxel or the border voxels of the local region. Therefore, we choose a fixed location.

Overall, let $L = L(\boldsymbol{u}_1, \mathrm{p}(\boldsymbol{x}_{\boldsymbol{\alpha}}))$ be the fitted line and let $P = P(\boldsymbol{u}_3, \mathrm{p}(\boldsymbol{x}_{\boldsymbol{\alpha}}))$ be the fitted plane. Then, denote by D_L, D_P the collections of distances to the fit primitives [23] computed by

$$d_L \colon \mathbb{N}_0^3 \to \mathbb{R}, \quad \boldsymbol{v} \mapsto \|(\boldsymbol{v} - \mathrm{p}(\boldsymbol{x}_{\boldsymbol{\alpha}})) \times \boldsymbol{u}_1\|_2,$$
$$d_P \colon \mathbb{N}_0^3 \to \mathbb{R}, \quad \boldsymbol{v} \mapsto |\langle \boldsymbol{v} - \mathrm{p}(\boldsymbol{x}_{\boldsymbol{\alpha}}), \boldsymbol{u}_3 \rangle|.$$

From these we compute the descriptors describing linearity and planarity.

Definition 4.4.2. First, we construct histograms $\mathcal{H}_L, \mathcal{H}_P$ from D_L and D_P. Both histograms have K bins as chosen experimentally and are thresholded as in Definition 4.2.1. Then, we embed these histograms as specified in Definition 3.3.3, we denote them by $\mathcal{H}'_L$ and $\mathcal{H}'_P$.
Moreover, we compute characteristics

$$f_1^p = \exp\left(-\mathbb{E}[D_p]\right) \quad \text{and} \quad f_2^p = \frac{\max_{d \in D_p} d - \min_{d \in D_p} d + 1}{2d_{\max}}$$

for both features $p \in \{L, P\}$.
In case of $p = L$, these two values represent the linearity, i.e., how much the current region resembles a straight line, and also the diameter, i.e., the maximum diameter a fit cylinder has.
In case of $p = P$, they represent the planarity and width of the environment instead.
The overall descriptor vector is given through

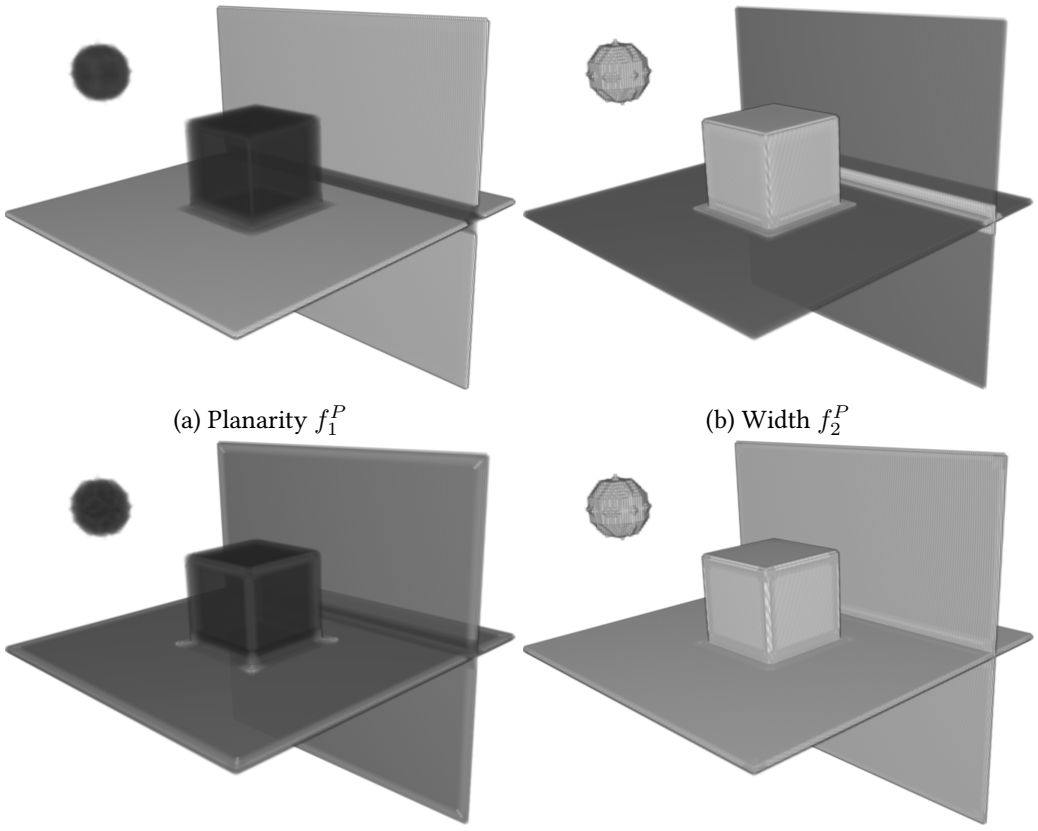
$$\left(f_1^p, f_2^p, \mathcal{H}'_p\right) \in \mathbb{R}^{2+K}$$

for $p \in \{L, P\}$.

Again, we have to define default values covering ambiguous cases like completely homogenous regions or when less than three voxels have remain nonzero after local thresholding.

Default values. Considering the default characteristics, we choose $f_1^p = f_2^p = 0, p \in \{L, P\}$. Setting them zero indicates completely homogenous regions, i.e., maximally nonlinear and nonplanar. Regarding the histogram features, on default uniform histograms are returned, i.e., histograms where all bins are equally filled. They are further normalized and embedded.

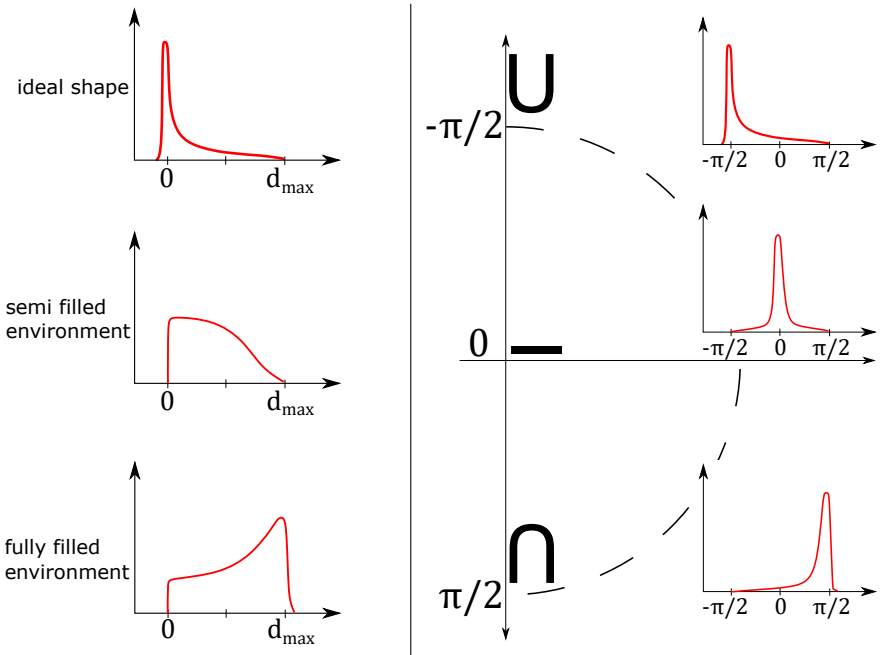
Concerning the computed characteristics representing the planarity/linearity and width as stated in Definition 4.4.2, we used these feature values to colorize each voxel of an idealized test dataset consisting of a few shapes, including a filled sphere, a filled cube and two intersecting planes, of the same intensity value. Concretely, for each iterable voxel in that volume we computed the mentioned characteristics in a local environment and assigned the result to the voxel, where we scaled the values to the grayscale value range for visualization purposes. The results are depicted in Figure 4.2, where in all images brighter values represent higher feature values after scaling. Regarding a local plane fit, the planes are well identified by the planarity where the feature values are close to one, while on nonplanar regions they are significantly smaller. The width also represents the planes as thin objects locally while isotropic regions are identified as being thick. A similar yet less prominent effect can be observed considering a local line fit, where edges are marked stronger than flat or isotropic regions.
By careful inspection of different histograms generated described above we notice that the possible histogram shapes transform continuously from a peak at the first bin, if the voxels form a perfect plane, to an exponential-like curve as the voxel region gets filled up. Thus, the current features follow the trend depicted in Figure 4.3a. Hence, a better comparison is achieved if the histograms are compared by a Wasserstein distance than with the usual Euclidean metric. Thus, we embed the histograms according to Definition 3.3.3.



(a) Planarity f_1^P (b) Width f_2^P

(c) Linearity f_1^L (d) Width f_2^L

Figure 4.2.: Ideal data colorized with plane and line fit characteristics.



(a) Typical distance histograms.

(b) Typical curvature histograms.

Figure 4.3.: Typical geometrical feature histograms.

4.4.2. Curvature Features

As mentioned before, one major question is how curvature can be estimated in local environments. Typically, one would aim for extracting a surface in the local region and estimate the curvature on it. However, this would require another optimization algorithm or at least a threshold. The first approach is typically not tractable for large-scale applications while such thresholds would have to be adapted for each local region. To estimate the local curvature fast and without the need for additional algorithms, we propose the use of a variant of Yoshida's estimation [18, Part IV.A].
In their approach, they consider an iso surface specified by a function over the coordinates and some surface extraction threshold in a small neighborhood around each voxel. Next, they formally derive explicit expressions for the first and second fundamental forms which do not contain this threshold anymore, but rather rely on discrete gradient information. From this, the principal curvatures can be estimated, from which the curvature feature is extracted in turn.
As a small addition to Yoshida's approach, we added some additional error handling:
If the squared mean curvature H^2 is smaller than the Gaussian curvature K, the proposed computations would issue a domain error since the term $\sqrt{H^2 - K}$ would not be well-defined. Furthermore, if the difference of the principal curvatures is close to zero, the curvature estimation would not be well-defined too. In both cases, this is due to an ambiguity where the current voxel either lies on a plane or on a saddle point surface. In these cases, we define them to have zero curvature, and hence the scaling in Yoshida's paper produces a default value of $\pi/2$.
This consideration is for very small environments around each voxel only. To make this feature applicable for bigger environment sizes too, we basically record the curvatures on each voxel inside our region.

Definition 4.4.3. Let $R_K(\boldsymbol{x_\alpha})$ be a local environment around some voxel $\boldsymbol{x_\alpha}$ in the iterable region of a volume. For each voxel in the iterable region of $R_K(\boldsymbol{x_\alpha})$ (considering a padding of one voxel on each side), the curvature is estimated on each $3 \times 3 \times 3$ environment around it and recorded in a histogram $\mathcal{H}_c$ whose values range from 0 to π. The feature consists of the histogram after normalization, thresholding and embedding according to Definition 3.3.3.

For the number of bins during histogram creation we choose five bins experimentally.
As stated, we also embed the histogram of curvatures for Wasserstein comparability. A justification is given as a shape in form of $\bigcup$ creates a histogram resembling a descending exponential curve while the opposite shape $\bigcap$ describes its ascending counterpart. In ambiguous cases, e.g., considering planes, the histogram is a peak at the middle curvature $\pi/2$. Intermediate shapes form a continuous transformation of histograms. That trend is visualized in Figure 4.3b which shows the histograms associated to the border cases and in between them depending on the calculated curvatures before shifting the values by $\pi/2$. Hence, we suppose that the Wasserstein distance is the preferred comparison metric here which was supported by numerical experiments.

Default values. In case of a completely homogenous region, we return a histogram containing exactly one entry at the $\pi/2$ bin, assimilating all ambiguous cases.

4.4.3. Distance Histogram Feature

Having discussed linearity and curvedness of local regions, we now aim to describe the overall shape. To accomplish that, we consider the distances of all nonzero voxels to the center point of the region after local thresholding. All distances then are accumulated in a histogram.

Definition 4.4.4. Let $R_K(\boldsymbol{x_\alpha})$ be some local environment centered around some voxel $\boldsymbol{x_\alpha}$ in the iterable Region of Interest of a volume.
Then, we compute the distances of each nonzero voxel to the center voxel, i.e., $\|\mathrm{p}(\boldsymbol{x_\beta}) - \mathrm{p}(\boldsymbol{x_\alpha})\|_2$ for all $\boldsymbol{x_\beta} \in R_K(\boldsymbol{x_\alpha}) \setminus \{\boldsymbol{x_\alpha}\}$ for which $\mathrm{v}(\boldsymbol{x_\beta}) > 0$. All these distances are arranged in a histogram, which is further normalized, thresholded and embedded for Wasserstein comparability according to Definition 3.3.3.

The number of bins was empirically determined as round$(2.4K - 2.2)$ where K is the size of the local environment. This rule was found by simulating environments and estimating the optimal bin counts by a Freedman-Diaconis estimator [24], after which linear regression over the predicted counts was done. Furthermore, the distance values range from zero to $\sqrt{3}\lfloor K/2 \rfloor$, defining the value range of the histogram.

Default values. In case of a completely homogenous region, all voxels have a value of zero after local thresholding. Then, we compute the descriptor as if each voxel has a nonzero value instead.

Another popular realization of a shape describing feature is a histogram of pairwise distances. However, we explicitly choose the distances to the center voxel instead as both descriptors share the same expressiveness but only the latter is of linear runtime with regard to the environment size. Considering pairwise distances would introduce a quadratic runtime factor instead.

4.4.4. Texture and Orientation Features

After discussing the similarity of local environments to geometric primitives like lines, planes or curved surfaces, we are now going to deal with the texture of local environments.
Such a texture or orientation is inherently encoded by the grayscale values. However, contrary to the plain statistical moments discussed earlier, texture usually possesses more discriminative power. The presented features will cover both textures as found in the spatial domain as well as in the Fourier domain. Both features make use of an *inertia tensor* which is a concise representation of mass orientation in a matrix. In physical considerations, it typically uses a mass distribution to encode inertia, where the eigenvectors of this matrix represent the axis of inertia, i.e., the dominant inertia direction.
In the Fourier domain, we want to characterize local texture by the Fourier domain shape. Hence, we compute a Fourier transform of the local region, followed by a construction of an inertia tensor $I^F \in \mathbb{R}^{3\times3}$. In the following, let $\mathcal{J}(i) := \{1,2,3\}\setminus\{i\}$.

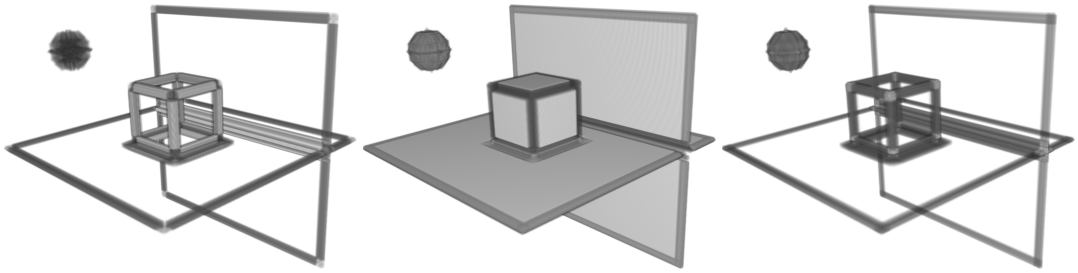
Definition 4.4.5. Let $R := R_K(\boldsymbol{x_\alpha})$ be a local environment around some voxel $\boldsymbol{x_\alpha}$. Additionally, denote by $\mathcal{F}$ the three-dimensional Fourier transform of that environment, from which we compute the *spectral energy* $E = |\mathcal{F}|^2$ componentwise, i.e., $E \subset \mathbb{R}^{K\times K\times K}$.
The Fourier inertia tensor as proposed by Bhalerao and Wilson [19] is given by

$$I^F_{i,i} = \sum_{\boldsymbol{x_\beta}\in R} \left(\sum_{j\in\mathcal{J}(i)} \boldsymbol{\beta}_j^2 \right) E_{\boldsymbol{\beta}}, \qquad i = 1,2,3,$$

$$I^F_{i,j} = I^F_{j,i} = -\sum_{\boldsymbol{x_\beta}\in R} \boldsymbol{\beta}_i\, \boldsymbol{\beta}_j\, E_{\boldsymbol{\beta}}, \qquad i,j = 1,2,3,\ i \neq j,$$

where $E_{\boldsymbol{\beta}}$ is the value of the spectral energy at position $\boldsymbol{\beta}$.

To avoid the very costly Fourier transform to be evaluated at each voxel region, we aim to compute equivalent features in the spatial domain, as proposed by Jähne.



(a) Linearity f_l (b) Planarity f_p (c) Isotropy f_s

Figure 4.4.: Ideal data colorized with orientation features.

Definition 4.4.6. Given a local three-dimensional region of size $N := K \times K \times K$, compute the three partial derivatives $G_x, G_y, G_z \in \mathbb{R}^N$ along each axis, yielding

$$G_1 = G_x, \quad G_2 = G_y, \quad G_3 = G_z.$$

The inertia tensor is set up as proposed by Jähne [25, Eq. 8.11]:

$$I^O_{i,i} = \sum_{j \in \mathcal{J}(i)} \|G_j\|_2^2, \qquad i = 1, 2, 3,$$

$$I^O_{i,j} = I^O_{j,i} = -\langle G_i, G_j \rangle, \qquad i, j = 1, 2, 3,\ i \neq j.$$

For both kinds of features, we use the constructed inertia tensor to calculate the feature values.

Definition 4.4.7. Let I^F and $I^O \in \mathbb{R}^{3\times3}$ be as defined in Definition 4.4.5 and 4.4.6, respectively. All these tensors are symmetric real matrices, thus they have three real eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0.$$

The resulting feature values are given by

$$f_l = \frac{\lambda_1 - \lambda_2}{\lambda_1}, \qquad f_p = \frac{\lambda_2 - \lambda_3}{\lambda_1}, \qquad f_s = \frac{\lambda_3}{\lambda_1},$$

which together form the descriptor vector (f_l, f_p, f_s).

Geometrically, f_l represents the linearity which is high if one direction dominates the other two. f_p denotes the planarity, which increases as two directions dominate the third one and finally f_s can be interpreted as isotropy which assumes a high value if no direction is dominant. Note that these values form probabilities and must sum to one, i.e., $f_l + f_p + f_s = 1$.

Default values. For completely homogeneous or empty regions, we return $f_l = f_p = f_s = 0$. In the Fourier domain, this setting identifies a single infinitesimal point, disambiguating this from almost completely isotropic regions.

Similar to the description of local plane or line fit features we also colorized the same idealized dataset with the descriptor values introduced in Definition 4.4.7. The results are depicted in Figure 4.4. We see that the features keep their respective structure well. Especially interesting is the isotropy feature which identifies very homogenous regions, e.g., the inside of the sphere but also the corners of the filled cube.

4.4.5. Spherical HOG Features

In literature, the *Histogram of Oriented Gradients* (HOG) feature captures local orientation by computing gradients and estimating three-dimensional orientations. These are added to a two-dimensional histogram binning the azimuthal and elevation angles. The histogram entries are

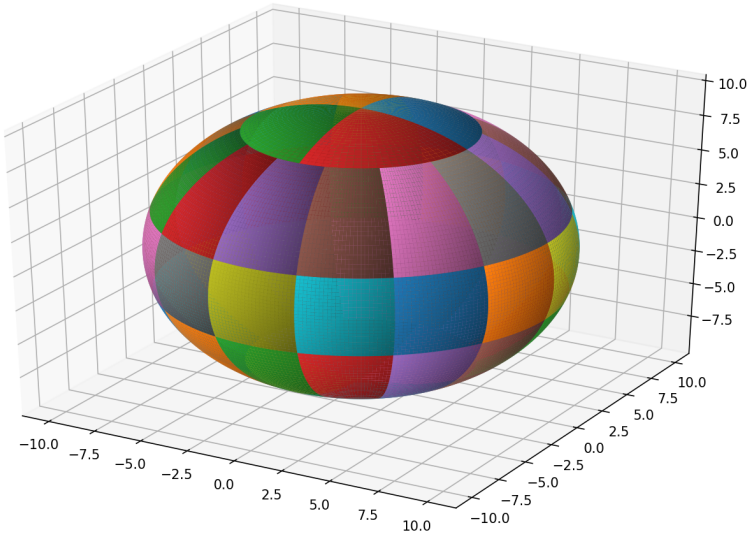


Figure 4.5.: Spherical histogram in our setting.

incremented by the intensity, i.e., the Euclidean norm, of the gradient. Conceptually, all possible orientations form a sphere in 3D space. But a naive equidistant binning of that space would result in "small" bins, i.e., that some bins correspond to very few angles and some encompass a large variety of orientations. To avoid that, we instead partition the sphere formed by the orientations such that each surface patch has the same area. Hence, we consider an equidistant partitioning over the elevation angles into n_φ sectors, and further each sector is divided into n_ψ parts such that the area of each surface patch on the sphere covered by a bin is equal to any other one. Specifically, we suppose that a partitioning into two pole regions, an equator and two intermediate zones suffices, hence we consider five elevation sectors only. We apply the same scheme to both the western and eastern hemisphere. However, for both pole regions we use only four bins on each as these regions are smaller. The remaining 42 bins are distributed evenly over the middle three horizontal sectors, thus each one is split into 14 bins. This partitioning is depicted in Figure 4.5. Using the concept of spherical histograms, we can define a three-dimensional Histogram of Oriented Gradients descriptor as follows.

Definition 4.4.8. Given a local three-dimensional region of size $N := K \times K \times K$, compute the three partial derivatives $G_x, G_y, G_z \in \mathbb{R}^N$ along each axis. Next, we compute the orientation from each element $g_j = (g_{j,x}, g_{j,y}, g_{j,z}) = (G_{x,j}, G_{y,j}, G_{z,j}) \in \mathbb{R}^3, j = 1, \dots, N$, by

$$\varphi_j = \tan^{-1}\left(\frac{g_{j,z}}{g_{j,x}}\right),$$
$$\psi_j = \sin^{-1}\left(\frac{g_{j,y}}{\|g_j\|_2}\right).$$

A two-dimensional histogram is formed where the bins are identified by the angles computed above and incremented by the gradient magnitude if they are relevant.
The HOG descriptor for the local region then is given by the normalized and linearized histogram.

Regarding the bin increment, given some element $g_j, j = 1, \dots, N$, the value of the bin at position (φ_j, ψ_j) is increased by $\|g_j\|_2$. The increment is done only if the element is relevant, i.e., if $\|g_j\|_2 > \varepsilon$ where we choose $\varepsilon = 10^{-5}$.

We want to note that we do *not* embed a HOG histogram to be Wasserstein comparable as it is unclear if the Wasserstein distance actually improves comparability in this two-dimensional case.

4.5. Scaling Feature Vectors

For many classification systems which compute distances in an Euclidean space, feature scaling is very important. If some feature value has a wide range of values, i.e., a high variance in its values, while others are concentrated around some value, the first one will dominate the distance calculated. Consequently, the other features does not participate much to the distance and thus not for the classification.
Therefore, feature scaling aims to transform feature values such that the generated values do not dominate other features just because of a bigger variance. In many applications, feature vectors are standardized elementwise, i.e., each component is transformed to have zero mean and unit variance which is commonly called *z-score* [26].
Given a set of training data feature vectors $\{t^1, t^2, \dots, t^M\} \subset \mathbb{R}^d$, one first estimates the mean and standard deviation from them via

$$\mu_k = \frac{1}{M}\sum_{j=1}^{M} t_k^j, \quad \sigma_k^2 = \frac{1}{M}\sum_{j=1}^{M}(t_k^j - \mu_k)^2, \qquad k = 1, \dots, d.$$

Typically, a feature vector $f \in \mathbb{R}^d$ is transformed into a vector $g \in \mathbb{R}^d$ by

$$g_k = \frac{f_k - \mu_k}{\sigma_k}, \qquad k = 1, \dots, d.$$

The mentioned dominating behavior occurs frequently if the computed feature values are of different scales, like for different statistical moments combined with grayscale values. But in other cases, e.g., when using histogram features, the feature values actually have the same value range and elementwise scaling might even decrease the discriminative power of histograms. In order to use both unrelated feature values and histograms in a classifier, we propose the following feature scaling scheme.

Definition 4.5.1. Let $\{t^1, t^2, \dots, t^M\} \subset \mathbb{R}^d$ be a set of training feature vectors.
Consider an index function which maps a vector component index $k = 1, \dots, d$ to a tuple of component indices (s_k, s'_k) that shall be scaled together, where $s_k \leq s'_k$. Hence, in total $N := M(s'_k - s_k + 1)$ elements are scaled at once.
Then, compute the means and standard deviations via

$$\mu_k = N^{-1}\sum_{\ell=1}^{M}\sum_{j=s_k}^{s'_k} t_j^\ell, \qquad \sigma_k^2 = N^{-1}\sum_{\ell=1}^{M}\sum_{j=s_k}^{s'_k}(t_j^\ell - \mu_k)^2,$$

for $k \in \{1, \dots, d\}$. Then, a feature vector $f \in \mathbb{R}^d$ is scaled by $f_k \mapsto (f_k - \mu_k)/\sigma_k,\ k = 1, \dots, d.$

To clarify the above definition and the role of the index function, we give a small example.

Example 4.5.2. Consider the above setting and assume that the feature vectors contain three independent features and one histogram, e.g., they are of shape

$$f = (f_1, f_2, f_3^h, \ldots, f_{d-1}^h, f_d),$$

where the superscript h denotes that these marked values form a histogram and shall be scaled together. The corresponding index function then is defined as

$$i(k) = \begin{cases} (1,1), & k = 1, \\ (2,2), & k = 2, \\ (3, d-1), & k = 3, \ldots, d-1, \\ (d,d), & k = d. \end{cases}$$

Given this, let $N := M(d-3)$ and then we estimate the means and standard deviations by

$$\mu_j = M^{-1} \sum_{\ell=1}^{M} t_j^\ell, \qquad j = 1, 2, d, \qquad \mu_3 = \ldots = \mu_{d-1} = N^{-1} \sum_{\ell=1}^{M} \sum_{j=3}^{d-1} t_j^\ell,$$

$$\sigma_j^2 = M^{-1} \sum_{\ell=1}^{M} (t_j^\ell - \mu_j)^2, \quad j = 1, 2, d, \qquad \sigma_3^2 = \ldots = \sigma_{d-1}^2 = N^{-1} \sum_{\ell=1}^{M} \sum_{j=3}^{d-1} (t_j^\ell - \mu_3)^2.$$

We see that all feature vector components belonging to the histogram are scaled with the same mean and standard deviation instead of one for each component.

4.6. Automatic Feature Selection

When we combine multiple geometric features that produce histograms, the resulting feature vectors potentially get very large. But high-dimensional vectors increase the computational burden and they might also impede the segmentation performance of classifiers if they are likely to suffer from the *curse of dimensionality*. Another important point is that if local regions do not resemble certain geometric primitives at all, the constructed histograms sometimes are the same for different classes too, thus there is redundancy which elongates feature vectors but has no discriminative power. Conventionally, one uses dimensionality reduction to cope with these problems, e.g., using a *Principal Component Analysis* (PCA), which takes feature vectors and creates new ones from them containing only the relevant parts. Of such techniques, PCA is arguably the most-used technique in many mathematical fields with its basic idea dating back to 1901 [27] and its formal derivation given in [28]. To put it briefly, PCA determines a best-fit ellipsoid to the dataset where the orientation of it correlates to the variance of the individual feature vector components. Hence, the component having the biggest variance will become the first principal component. Thus, the least important feature values will reside in the last principal components. For dimensionality reduction, one typically computes a PCA in form of a transformation matrix and further truncates that matrix, i.e., produces a matrix which projects feature vectors onto a PCA subspace of lower dimension. However, a disadvantage using dimensionality reduction with PCA is that we still have to compute all feature values which are reduced afterwards. Instead, we perform *feature selection* to estimate the most important features and during segmentation we only compute the necessary features, thus reducing the computational effort.
The computation of the relevant features follows the approach introduced in [29]. Their method starts similar to one popular variant of computing a PCA by first calculating the sample covariance matrix of the feature matrix. Let $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{M \times d}$ be the matrix containing one d-dimensional feature vector rowwise for each of the M training voxels and let $\boldsymbol{\mu} = M^{-1} \boldsymbol{\mathcal{F}}^T \mathbf{1} \in \mathbb{R}^d$ be the columnwise means of $\boldsymbol{\mathcal{F}}$. Each feature vector is comprised of all geometric feature vectors. Next,

we compute the covariance matrix by

$$\mathbf{\Sigma} = (M-1)^{-1}(\boldsymbol{\mathcal{F}} - \mathbf{1}\,\boldsymbol{\mu}^T)^T(\boldsymbol{\mathcal{F}} - \mathbf{1}\,\boldsymbol{\mu}^T),$$

which is of dimensionality $d \times d$. In practice, the dimensionality d is far smaller than the number M of seed voxels, and hence $\mathbf{\Sigma}$ is relatively small in size which is further independent of M. That matrix has d nonnegative real eigenvalues λ_k and associated eigenvectors $\boldsymbol{w}_k$, $k = 1, \ldots, d$. In [29], the authors state that the influence of the feature value at index $j = 1, \ldots, d$ on the dimensionality reduction result corresponds to the j-th coefficient of the eigenvectors. Hence, let $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_s$ be the eigenvectors associated to the s largest eigenvalues of $\mathbf{\Sigma}$, where s defines how many features we want to select. Next, we calculate the *contribution* of the j-th feature component by $c_j = \sum_{k=1}^{s} |\boldsymbol{w}_{k,j}|$ where $\boldsymbol{w}_{k,j}$ denotes the j-th coefficient of the vector $\boldsymbol{w}_k$. That process is repeated for all $j = 1, \ldots, d$. The computed contributions are sorted in descending order and the ordering is recorded in another ordering $\boldsymbol{o}$, e.g., if the l-th feature component has the biggest influence on the selection result, then we obtain $\boldsymbol{o}_1 = l$. The selection then consists of the s feature components highest ranked on the computed order.
As in our application the order of the features computed is fixed, we are able to calculate the actual feature selection. On a technical note, we note that if a feature produces multiple values (cf. histogram features) and at least one component is to be selected, we select that feature. Consequently, although the procedure selects only few features, the remaining feature vector length can and often will be bigger than the number of requested features s.

4.6.1. Runtime and memory complexity

Let M denote the number of voxels used for feature selection as above. Furthermore, let d be the maximum possible feature vector size, i.e., its size when all geometric features are selected.
First, the presented procedure computes the eigenvalues and eigenvectors of the covariance matrix computed from the feature matrix, which is a symmetric real matrix of dimensions $d \times d$. The needed runtime for computing the covariance matrix is of order $\mathcal{O}(d^2 M)$. On the other hand, the solver used for the computation of the eigenvalues and eigenvectors implements Algorithm 8.3.3 described in [30, p.463] and hence has a runtime complexity of $\mathcal{O}(d^3)$. Note that since d is constant a priori and independent of M, the runtime effectively reduces to $\mathcal{O}(M)$. Secondly, the computed eigenvalues and associated feature indices are sorted which has a negligible runtime as d is constant. All following steps solely operate on the computed eigenvalues and their ordering. Thus, they are of constant runtime too. Hence, the overall runtime complexity is linear in the number of voxels considered for feature selection.
For the space complexity, we first have to keep the feature matrix of dimensions $M \times d$ in main memory. After computation of the covariance matrix, the resulting operations just need constant memory as d is constant.

References

[1] Panaretos, Victor and Zemel, Yoav. *An Invitation to Statistics in Wasserstein Space*. Springer, Jan. 2020.

[2] Patrick Billingsley. *Probability and Measure*. 3rd ed. Series in Probability and Mathematical Statistics. Wiley, 1995.

[3] S. S. Vallander. "Calculation of the Wasserstein Distance Between Probability Distributions on the Line". In: *Theory of Probability & Its Applications* 18.4 (1974), pp. 784–786.

[4] Kangyu Ni et al. "Local Histogram Based Segmentation Using the Wasserstein Distance". In: *Int. J. Comput. Vision* 84.1 (Aug. 2009), pp. 97–111.

[5] Piotr Indyk and Nitin Thaper. "Fast Image Retrieval via Embeddings". In: *3rd Workshop on Statistical and computational Theories of Vision*. Nice, France, 2003.

[6] Yulia Dodonova and Mikhail Belyaev and Anna Tkachev and Dmitry Petrov and Leonid Zhukov. *Kernel classification of connectomes based on earth mover's distance between graph spectra*. Aug. 2016. arXiv: 1611.08812.

[7] Le Hou and Chen-Ping Yu and Dimitris Samaras. *Squared Earth Mover's Distance-based Loss for Training Deep Neural Networks*. 2017. arXiv: 1611.05916.

[8] Z. Yu and G. Herman. "On the Earth Mover's Distance as a histogram similarity metric for image retrieval". In: *2005 IEEE International Conference on Multimedia and Expo*. 2005, pp. 686–689.

[9] Jialin Liu and Wotao Yin and Wuchen Li and Yat Tin Chow. *Multilevel Optimal Transport: a Fast Approximation of Wasserstein-1 distances*. Sept. 2018. arXiv: 1810.00118.

[10] Charlie Frogner and Farzaneh Mirzazadeh and Justin Solomon. *Learning Embeddings into Entropic Wasserstein Spaces*. 2019. arXiv: 1905.03329.

[11] Nicolas Courty, Rémi Flamary and Mélanie Ducoffe. "Learning Wasserstein Embeddings". In: *ICLR 2018 - 6th International Conference on Learning Representations*. Vancouver, Canada, Apr. 2018, pp. 1–13.

[12] Charlie Frogner and Chiyuan Zhang and Hossein Mobahi and Mauricio Araya-Polo and Tomaso A. Poggio. "Learning with a Wasserstein Loss". In: *Advances in Neural Information Processing Systems (NIPS) 28*. 2015.

[13] Taghvaei, Amirhossein and Jalali, Amin. *2-Wasserstein Approximation via Restricted Convex Potentials with Application to Improved Training for GANs*. Feb. 2019. arXiv: 1902.07197.

[14] Daizong Ding and Mi Zhang and Xudong Pan and Min Yang and Xiangnan He. "Improving the Robustness of Wasserstein Embedding by Adversarial PAC-Bayesian Learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. AAAI Press, Apr. 2020, pp. 3791–3800.

[15] S. Kolouri, Y. Zou and G. K. Rohde. "Sliced Wasserstein Kernels for Probability Distributions". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 5258–5267.

[16] Antoni Zygmund. *Trigonometric Series*. Vol. 1. Cambridge university press, 2002.

[17] Henri De Plaen, Michael Fanuel and Johan A. K. Suykens. *Wasserstein Exponential Kernels*. 2020. arXiv: 2002.01878.

[18] Hiroyuki Yoshida and Janne Näppi. "Three-Dimensional Computer-Aided Diagnosis Scheme for Detection of Colonic Polyps". In: *IEEE Trans. Med. Imaging* 20.12 (2001), pp. 1261–1274.

[19] Abhir Bhalerao and Roland Wilson. “A Fourier Approach to 3D Local Feature Estimation from Volume Data”. In: *Proceedings of BMVC 2001* (Apr. 2002).

[20] Philippe Pebay and Philippe Pierre. *Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments*. Tech. rep. U.S. Department of Energy, Office of Scientific and Technical Information, Sept. 2008.

[21] Guoying Zhao and Matti Pietikäinen. “Dynamic Texture Recognition Using Local Binary Patterns with an Application to Facial Expressions”. In: *IEEE transactions on pattern analysis and machine intelligence* 29 (July 2007), pp. 915–28.

[22] Olli Lahdenoja, Jonne Poikonen and Mika Laiho. “Towards Understanding the Formation of Uniform Local Binary Patterns”. In: *ISRN Machine Vision* 2013 (July 2013).

[23] Sung Joon Ahn. *Least Squares Orthogonal Distance Fitting of Curves and Surfaces in Space*. Vol. 3151. Lecture Notes in Computer Science. Springer, 2004.

[24] David Freedman and Persi Diaconis. “On the Histogram as a Density Estimator: L_2 Theory”. In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 57.4 (Dec. 1981), pp. 453–476.

[25] Bernd Jähne. *Spatial-Temporal Image Processing, Theory and Scientific Applications*. Springer-Verlag Berlin Heidelberg, 1993.

[26] Erwin Kreyszig. *Advanced Engineering Mathematics*. 10th ed. Wiley, Feb. 2011.

[27] Karl Pearson. “LIII. On Lines and Planes of Closest Fit to Systems of Points in Space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (Nov. 1901), pp. 559–572.

[28] Harold Hotelling. “Analysis of a complex of statistical variables into principal components”. In: *Journal of Educational Psychology* 24.6 (1933), pp. 417–441.

[29] F. Song, Z. Guo and D. Mei. “Feature Selection Using Principal Component Analysis”. In: *2010 International Conference on System Science, Engineering Design and Manufacturing Informatization*. Vol. 1. Nov. 2010, pp. 27–30.

[30] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)* USA: Johns Hopkins University Press, 1996.

Part III.

Unsupervised Segmentation

O this learning, what a thing it is!

William Shakespeare, The Taming of the Shrew

Table of Contents

The first segmentation techniques we cover are *unsupervised*, i.e., they segment a volume without using any labeled data or further user interaction. They all belong to the group of clustering algorithms where characteristics like our features describe the similarity between voxel regions so they can be grouped together.

We start by describing a random sampling technique used to extract a representative subset of a voxel volume. This subset is of constant size and thus allows tackling big volumes. We proceed by explainig clustering using K-Means and Gaussian Mixture Models. Both algorithms will be trained on an extracted random sample, after which the voxel volume is segmented in a space-efficient way.

5. Random Sampling

Abstract Typically, clustering algorithms compute characteristics for each data element and perform grouping afterwards. However, the runtime complexity of this first step is often of order $\mathcal{O}(N^\alpha), \alpha > 1$, i.e., polynomial in the dataset size. Such algorithms are typically not feasible for processing large volumes.
In order to tackle this problem, this chapter introduces a strategy for extracting a representative random sample of a *fixed* size $M \in \mathbb{N}$. First, the stratification is explained which partitions the grayscale value distribution value range. Next, the actual random sampling procedure on such a stratification which is linear in the number of input voxels is given.

5.1. Stratification

The goal is to extract a small representative subset from a big voxel volume. A possible consequence is that, e.g., all voxel values which belong to a certain material might get omitted.
To avoid this loss of information, we compute a stratification before sampling. That is, we divide the voxel value range into disjoint parts (the strati), as indicated in Figure 5.1 in which the S_1, S_2, S_3, S_4 denote the strati and the solid line shows a typical grayscale value distribution.

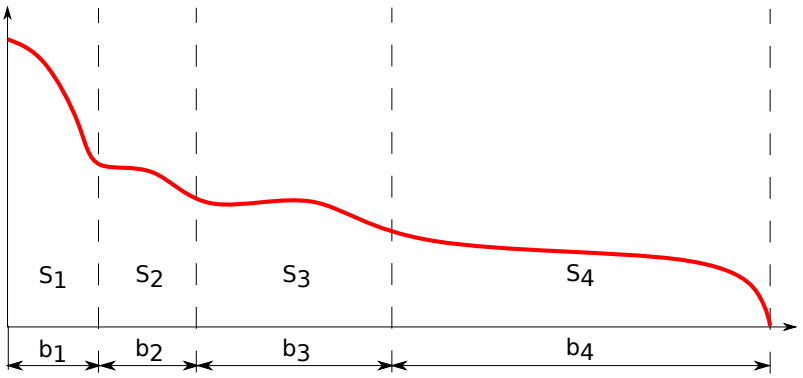


Figure 5.1.: Stratification of a typical grayscale value distribution.

Distributions of this shape occur frequently in industrial tomography scans, which form the main application of our algorithms. Therefore, the computed stratification assumes this "exponential" shape. But we emphasize that the sampling technique which will be presented in the next section can use any stratification and thus the sampling can exploit further application-specific knowledge.
Formally, we express our (multi-threaded) stratification as follows.

Definition 5.1.1. Let W be the overall width of the grayscale value distribution and let $N_T \in \mathbb{N}$ be the number of threads used. Further, let $M \in \mathbb{N}$ be the requested sample size and K be the number of strati. Then the grayscale value interval is divided into strati S_j^t of widths $b_j W$, $j = 1, \ldots, K$, where

$$b_j = \begin{cases} 2^{1-K}, & j = 1, \\ 2^{j-1-K}, & j = 2, \ldots, K. \end{cases}$$

This stratification is done by each thread $t = 1, \ldots, N_T$ in the same way.
The obtained *stratification* is a tuple

$$\left(K,\ (b_j)_{1 \le j \le K},\ \{S_j^t\}_{\substack{1 \le j \le K \\ 1 \le t \le N_T}} \right).$$

We encode the exponential shape depicted in Figure 5.1 in the stratification by partitioning the grayscale value range into disjoint intervals of width $b_j W$ where the fractions $b_j, j = 1, \ldots, K$, decrease exponentially as the index j increases. Thus, we obtain that $b_1 = b_2 \le \ldots \le b_K$, i.e., the width of a stratum increases towards higher grayscale values. As we will see, this will lead to more voxels having higher voxel values getting selected than voxels with lower values, which compensates the typically overwhelming number of voxels close to zero under the exponential shape assumption.

5.2. Reservoir Sampling

Next, for each stratum we perform reservoir sampling in it, which is a common technique to obtain a uniformly distributed random sample from some population. Algorithm 4 shows this sampling method schematically. The presented algorithm was discovered independently by both Alan Waterman (as stated by Knuth [1, p.144] and Vitter [2]) and by McLeod and Bellhouse [3].

Algorithm 4: Reservoir sampling.

Function `ReservoirSampling(`Q, M`)`
- **Input :** A population $Q = \{q_1, \ldots, q_N\}$
- **Input :** A sample size $M \le N$
- **Output:** A tuple Ω of size M containing the sample

```
Ω ← ()
for i ← 1 to N do
    if i ≤ M then
        Ω ← (Ω, q_i)                                  // fill up reservoir
    else
        r ← randInt(1, i)                  // unif. random integer in {1,...,i}
        if r < M then
            Ω_r ← q_i
return Ω
```

As hinted by the algorithm, first the reservoir which will contain the sample at the end is filled with the first M elements of the population. After the reservoir is filled, each following population element q_i is accepted with probability M/i and replaces a random element Ω_r in the reservoir. The latter step is continued until all items from the population are processed.

A speciality of reservoir sampling is that the population size need not be known a priori, which is however known in our case, and that a uniformly distributed sample can be obtained in one linear pass over the population, making it suitable for sampling large data sets like industrial computed tomography voxel volumes.

Furthermore, it is guaranteed that the sample generated by Algorithm 4 is drawn uniformly distributed from the given population.

Lemma 5.2.1. Given some population $Q = \{q_i\}_{i=1}^{N}$, Algorithm 4 produces a uniformly distributed random sample Ω of size M, i.e., each population item $q_i, i \in \{1, \dots, N\}$, has the probability

$$P(q_i \in \Omega) = \frac{M}{N}$$

of being chosen to be in the reservoir.

Several short proofs can be found in literature, e.g. in [1–3]. We give a detailed explanation in Appendix A for the sake of completeness.
Considering our setting, we combine a stratification and reservoir sampling to extract a stratified sample efficiently from a volume. Algorithm 5 shows the sampling procedure.

Algorithm 5: Stratified reservoir sampling on a volume.

Function `StratifiedSampling(`V, M, S`)`
- **Input :** A volume V of N voxels
- **Input :** A sample size $M \leq N$
- **Input :** A stratification $S = (K, b_j, S_j^t)$ acc. to Definition 5.1.1
- **Output:** A sample Ω of size M
- **parfor** $t \leftarrow 1$ **to** N_T **do**
  - **for** $j \leftarrow 1$ **to** K **do**
    - $V_j^t \leftarrow$ sub-volume belonging to stratum S_j^t
    - $\Omega_j^t \leftarrow$ `ReservoirSampling(`V_j^t, $b_j M$`)`
- $\Omega' = \bigcup_{t=1}^{N_T} \bigcup_{j=1}^{K} \Omega_j^t$
- $\Omega \leftarrow$ `ReservoirSampling(`Ω', M`)`
- **return** Ω

We want to remind the reader that in our setting, parallel processing over a volume happens by partitioning the volume into disjoint sub-volumes V^t for $t = 1, \dots, N_T$ along its logical z axis and processing each sub-volume sequentially. Technically, these could have an arbitrary shape, but we follow the parallel processing methodology introduced in Section 1.4.
The first step of Algorithm 5 operates in parallel over the sub-volumes V^t in the mentioned sense and additionally computes a stratification S_j^t for each sub-volume over all threads t and for all $j = 1, \dots, K$. By construction of the parallel processing and the stratification the constructed strati are all pairwise disjoint. From each V_j^t a uniformly distributed random sample Ω_j^t is extracted using reservoir sampling as described in Algorithm 4. We choose the size of each stratum-specific sample proportional to the stratum width as $b_j M$ where $b_j, j = 1, \dots, K$, is given by the stratification computed a priori and M is the sample size. Since all sub-volumes are pairwise disjoint, so are all Ω_j^t. Following this, an intermediate sample Ω' is accumulated as the union over all samples specific to a stratum and a thread. The final sample Ω of size M is obtained by applying reservoir sampling to this intermediate sample.
Naturally, we like to compute the probability of a voxel getting selected in the final sample.

Firstly, the probability that a voxel $\boldsymbol{x}$ gets chosen to be in a stratum-specific sample Ω_j^t is

$$P(\boldsymbol{x} \in \Omega_j^t) = P(\boldsymbol{x} \in \Omega_j^t \cap S_j^t) = P(\boldsymbol{x} \in \Omega_j^t \mid \boldsymbol{x} \in S_j^t)\, P(\boldsymbol{x} \in S_j^t) = \frac{b_j M}{\#S_j^t} P(\boldsymbol{x} \in S_j^t),$$

for $j = 1, \ldots, K$ and $t = 1, \ldots, N_T$, where we used that the sample is always a subset of the stratum, i.e., $\Omega_j^t \cap S_j^t = \Omega_j^t$. This implies that the probability of a voxel ending up in the aggregated intermediate sample Ω' is

$$P(\boldsymbol{x} \in \Omega') = P\left(\boldsymbol{x} \in \bigcup_{t=1}^{N_T} \bigcup_{j=1}^{K} \Omega_j^t\right) = \sum_{t=1}^{N_T} \sum_{j=1}^{K} P(\boldsymbol{x} \in \Omega_j^t) = \sum_{t=1}^{N_T} \sum_{j=1}^{K} \frac{b_j M}{\#S_j^t} P(\boldsymbol{x} \in S_j^t),$$

which makes use of the fact that all Ω_j^t are pairwise disjoint.
The size of Ω' is given by

$$\#\Omega' = \#\bigcup_{t=1}^{N_T} \bigcup_{j=1}^{K} \Omega_j^t = \sum_{t=1}^{N_T} \sum_{j=1}^{K} b_j M = M N_T \sum_{j=1}^{K} b_j = M N_T.$$

This is followed by an additional reservoir sampling step on Ω'.
Therefore, the overall probability of a voxel being in the final sample Ω is given by

$$\begin{aligned} P(\boldsymbol{x} \in \Omega) &= P(\boldsymbol{x} \in \Omega \cap \Omega') \\ &= P(\boldsymbol{x} \in \Omega \mid \boldsymbol{x} \in \Omega')\, P(\boldsymbol{x} \in \Omega') \\ &= \frac{M}{\#\Omega'} P(\boldsymbol{x} \in \Omega') \\ &= \frac{M}{N_T} \sum_{t=1}^{N_T} \sum_{j=1}^{K} \frac{b_j}{\#S_j^t} P(\boldsymbol{x} \in S_j^t). \end{aligned}$$

6. K-Means Clustering

Abstract Having a random sampling technique at hand, we will now discuss a classical clustering algorithm, namely K-Means clustering. It aims to group a given dataset into K different clusters based on the Euclidean distance between individual data points. After introducing that method, we combine it with our random sampling procedure to make the clustering space complexity independent of the volume size. Lastly, we analyze the asymptotical runtime behavior and memory requirements.

6.1. Introduction

The K-Means algorithm is a popular method of clustering data using only the number $K \in \mathbb{N}$ as a hyperparameter. We will briefly recapitulate how it works, further introductions to this technique are given, e.g., in [4, 5].

Definition 6.1.1. Given a dataset X and a number $K \in \mathbb{N}$ of clusters, the *K-Means* algorithm computes a partitioning $P_1, \dots, P_K$ by solving

$$\underset{P_1,\dots,P_K}{\operatorname{argmin}} \sum_{j=1}^{K} \sum_{\boldsymbol{x} \in P_j} \|\boldsymbol{x} - \boldsymbol{\mu}_j\|_2^2 \qquad \text{such that } X = \bigsqcup_{j=1}^{K} P_j,$$

where $\boldsymbol{\mu}_j = \mathbb{E}[P_j]$ is the centroid estimated for partition P_j, $j = 1, \dots, K$.

It is well-known that for one cluster only, i.e., $K = 1$, the optimal solution to this problem is given by the mean $\boldsymbol{\mu} = \mathbb{E}[P_1]$ over the only population P_1. Therefore, we aim to find K population means to minimize the stated expression, which also gives an intuition why this technique is called *K-Means*. However, numerically solving this problem is NP-hard [6] in general. Therefore, practical applications use heuristic algorithms to solve it, of which *Lloyd's algorithm* is without a doubt used the most.
The following sections will introduce Lloyd's algorithm in detail and pay special attention on how to properly initialize it. We continue by providing an algorithm explaining how K-Means is used to cluster potentially big voxel volumes. Finally, the end of the current chapter briefly analyzes both the asymptotic algorithm runtime as well as the memory requirements.

6.2. Lloyd's Algorithm

As mentioned, we use Lloyd's algorithm [7] for solving the K-Means clustering problem, as this procedure is simple, efficient and often yields the optimal result.
This procedure is summarized in Algorithm 6.
The classical two-stage algorithm begins by computing initial means. Lloyd's algorithm depends heavily on a good initialization, for which we use the *K-Means++* strategy that will be explained in the next section.

Next, we continue two-fold:

Algorithm 6: Lloyd's algorithm.

Function `Lloyd`(X, K)
  Input : A population $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$
  Input : Number of clusters K
  Output: Estimated means $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$
  $\boldsymbol{\mu} \leftarrow$ `initialize`(X, K)
  repeat
    $P_j \leftarrow \{\boldsymbol{x} \in X \mid \|\boldsymbol{x} - \boldsymbol{\mu}_j\|_2 \leq \|\boldsymbol{x} - \boldsymbol{\mu}_i\|_2, j \neq i = 1, \ldots, K\},\ j = 1, \ldots, K$
    $\boldsymbol{\mu}_j \leftarrow \mathbb{E}[P_j], \quad j = 1, \ldots, K$
  until *assignment converged*
  return $\boldsymbol{\mu}$

1. We cluster the data according to the current means, i.e., we create sub-populations P_j containing all elements nearest to the mean $\boldsymbol{\mu}_j$ for $j = 1, \ldots, K$. In the ambiguous case of equal distances, we assign the element to the cluster having the smallest cluster index.

2. We re-estimate the means according to this clustering.

This procedure is continued until the algorithm converges. We say that the iterations have converged if less than five percent of cluster assignments change. Given the estimated means, the data set is clustered by assigning each item the index j of the closest mean $\boldsymbol{\mu}_j, j = 1, \ldots, K$.

6.3. Initialization

There exists a broad variety of initialization methods for the K-Means algorithm, an overview of which is given in [8]. Of these, we decided to use the *K-Means++* strategy [9]. That technique starts by choosing the first center uniformly at random from the population. The following centers then are chosen proportionally to the distances to the already selected centers. Intuitively, each following initial center is far away from all already fixed centers.

Algorithm 7: K-Means++ initialization.

Function `initialize`(X, K)
  Input : A population $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$
  Input : Number of clusters K
  Output: Initial means $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K)$
  $\boldsymbol{\mu}_1 \leftarrow$ randomly chosen from X
  $\boldsymbol{\mu} = (\boldsymbol{\mu}_1)$
  for $j \leftarrow 2$ **to** K **do**
    $D \leftarrow \left(\min_{1 \leq i < j} \|\boldsymbol{x}_k - \boldsymbol{\mu}_i\|_2^2\right)_{k=1}^{N}$
    $\nu \leftarrow D / \|D\|_1$
    $r \sim \mathcal{U}([0, 1])$
    $\boldsymbol{\mu}_j \leftarrow \boldsymbol{x}_{i^*}, \quad i^* = \min\{i \in \{1, \ldots, N\} \mid F_\nu(i) > r\}$
    $\boldsymbol{\mu} \leftarrow (\boldsymbol{\mu}, \boldsymbol{\mu}_j)$
  return $\boldsymbol{\mu}$

The K-Means++ initialization is summarized in Algorithm 7. We see, that the first center is chosen uniformly at random from the population. For each subsequent center estimation, we compute the squared distances D of each data item to its closest already known center. Having these distances, the next center is chosen at random from a distribution where probabilities are proportional to the computed distances. Using the well-known inverse tranform sampling method [10], this can be achieved by generating a uniformly distributed number $r \sim \mathcal{U}([0,1])$ and searching for the smallest index $i^* \in \{1,\dots,N\}$ such that the according value of the discrete cumulative distribution function, i.e., $F_\nu(i^*)$, surpasses r. This is equivalent to searching the r-th quantile of the distribution ν. The population item $\boldsymbol{x}_{i^*}$ at that index then is selected as the next initial mean. This estimation procedure is repeated until all K initial centers are chosen.

6.4. Clustering Big Volumes

We conclude this chapter by stating an algorithm for applying K-Means clustering to volumes without restricting their size, which is shown in Algorithm 8.
The grouping of voxels is defined by similarity of feature vectors, i.e., their Euclidean distance. As a special case, the grayscale value $\mathrm{v}(\boldsymbol{x})$ of some voxel $\boldsymbol{x}$ can serve as a feature too.

Algorithm 8: Volume segmentation by K-Means clustering.

Input : A volume V of voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \boldsymbol{\Gamma}_{\boldsymbol{d}}$
Input : A feature computation function f
Input : Number of clusters K
Input : A sample size $M \leq \#\boldsymbol{\Gamma}_{\boldsymbol{d}}$
Output: A segmented volume of voxels

$S \leftarrow$ a stratification acc. to Def. 5.1.1
$\Omega \leftarrow \texttt{StratifiedSampling}(V, M, S)$
$\boldsymbol{\mu} \leftarrow \texttt{Lloyd}(\{f(\boldsymbol{x}_{\boldsymbol{\beta}}) \mid \boldsymbol{x}_{\boldsymbol{\beta}} \in \Omega\}, K)$
parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V$ **do**
  $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow \min\{j \in \{1,\dots,K\} \mid \|f(\boldsymbol{x}_{\boldsymbol{\alpha}}) - \boldsymbol{\mu}_j\|_2 \leq \|f(\boldsymbol{x}_{\boldsymbol{\alpha}}) - \boldsymbol{\mu}_k\|_2,\ j \neq k = 1,\dots,K\}$

First, a stratified random sample is extracted from the volume using Algorithm 5. The size of the sample is chosen such that it fits easily into main memory, e.g., using 4096 elements.
The algorithm proceeds by estimating the centroids $\boldsymbol{\mu}_j, j = 1,\dots,K$, by Lloyd's algorithm from the selected sample. As the sample size is fixed and chosen independently of the volume size, the polynomial runtime is negligible compared to the following loop.
Finally, the input volume V is clustered by assigning to each voxel the index j of the closest cluster center $\boldsymbol{\mu}_j, j = 1,\dots,K$. In the ambiguous case of two or more cluster centers having the same distance to the feature vector, we choose the smallest index.

6.5. Runtime and Memory Analysis

Since the goal of this thesis is to introduce algorithms usable for the segmentation of volumes without size restrictions, we also want to give a brief overview over the asymptotic runtime and memory requirements. In the following, denote $N \in \mathbb{N}$ the number of voxels in the volume and $M \leq N$ the sample size which is typically much smaller than the volume size and in our applications constant with a value of 4096 elements.

6.5.1. Runtime Analysis

Firstly, we perform random sampling on the volume. By definition of the stratified random sampling algorithm given in Algorithm 5, the parallel sampling followed by the sequential random sampling yields a runtime of order $\mathcal{O}(NN_T^{-1} + N_T M)$. Since N_T and the sample size M are constant and independent of N, the sampling runtime is effectively of linear order $\mathcal{O}(N)$. Next, the means are estimated using Lloyd's algorithm, which has a polynomial time complexity [11] in the number of elements. However, in our aforementioned setting the latter is constant. Thus, the mean estimation has an asymptotically constant runtime. Finally, we apply the clustering to each voxel in the algorithm. For this, we iterate over a constant number of clusters K and over all voxels in the volume.
We conclude, that Algorithm 8 has a linear runtime complexity of order $\mathcal{O}(N)$ with a constant additional overhead for the estimation procedures.

6.5.2. Memory Requirements

Let us now discuss the memory requirements of our algorithm.
In the first step, we extract a stratified random sample which must fit into memory. The used reservoir sampling technique guarantees that only the reservoir of constant size M, which will become the sample, needs to fit in memory. Thus, for the random sampling, the memory requried is of order $\mathcal{O}(M)$. Next, we perform Lloyd's algorithm on this sample, during which we keep a cluster assignment in memory which is of the same size as the sample. Overall, the memory requirements do not change. Finally, we iterate over the volume and assign to each voxel the index of its closest cluster mean. For this step, only the estimated means and a local environment centered around the currently processed voxel must be kept in memory at all times, thus the memory requirement drops to a constant of order $\mathcal{O}(K)$.
Overall, the memory required to execute Algorithm 8 is of order $\mathcal{O}(M)$. Since in our applications, the sample size M is constant, the overall memory used is of constant size. That makes Algorithm 8 applicable for volumes regardless of their size, as long as the sample fits into memory.

7. Clustering via Gaussian Mixture Models

Abstract A well-known shortcoming of the K-Means algorithm we discussed in the last section is that it is a *hard* clustering method. That is, a data item is assigned to exactly one cluster, but there is no measure of how likely it is that this item belongs to each cluster. Moreover, the hard assignments are based purely on distance to estimated centers and do not take the variance of a cluster into account. This makes K-Means inflexible, especially when the variance of the data clusters varies. Therefore, the current chapter aims to overcome this inflexibility by considering *soft* assignments based on Gaussian Mixture Models.
First, these types of parametric models are introduced with special focus on how we initialize the trainable parameters. Secondly, the popular Expectation Maximization algorithm is explained and applied on Gaussian Mixture Models, followed by strategies to avoid degeneracies. Next, a model selection technique is presented, after which we give the overall clustering procedure. Finally, a brief complexity analysis is given.

7.1. Gaussian Mixture Models

The middle part of Figure 7.1 depicts the aforementioned shortcoming of simple K-Means clustering when applied on a "mouse" dataset given in the first image. We see that K-Means fails to classify many data items correctly when the cluster variances are not uniform. The right-most image then shows how this can be improved by using Gaussian Mixture Models (GMMs) on that dataset which allows capturing the different cluster variances.
Therefore, we continue to introduce Gaussian Mixture Models which form a special case of general mixture models. Briefly said, mixture models are weighted sums of distributions. With such a model, we can express the likelihood that an item belongs to each individual distribution. The by far most used mixture models are Gaussian Mixture Models, i.e., mixtures where each distribution follows a Gaussian distribution, which offer the possibility to encode the variance in a cluster into the estimation and clustering process.

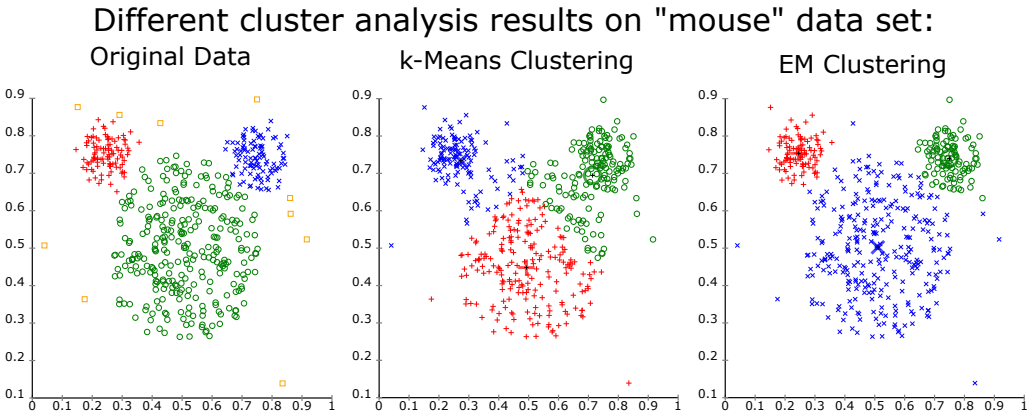


Figure 7.1.: Misclassifications by K-Means [12] due to different cluster variances.

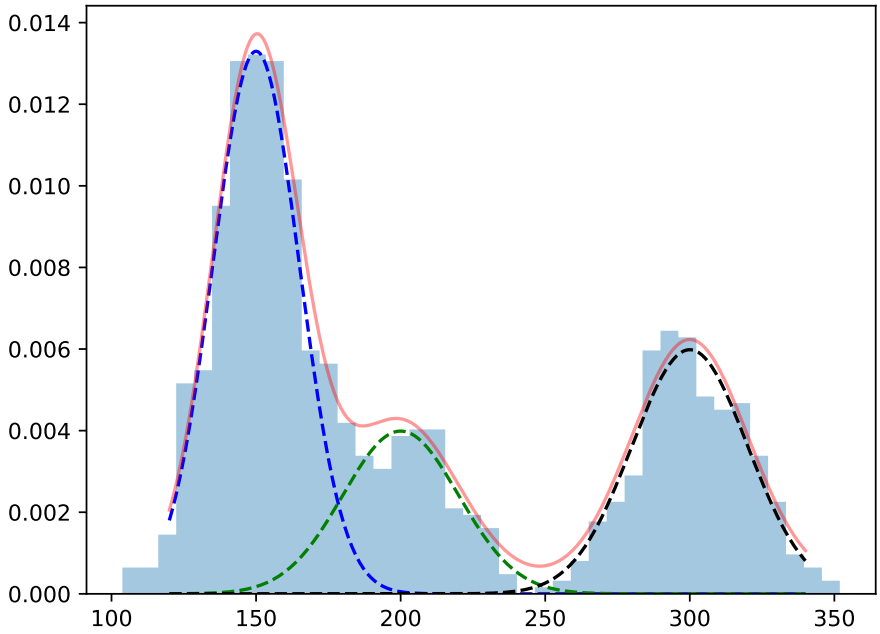


Figure 7.2.: Schematic univariate Gaussian Mixture Model.

Definition 7.1.1. Let $d \in \mathbb{N}$ be the dimensionality of feature vectors and $K \in \mathbb{N}$ be the number of component distributions. The *Gaussian Mixture Model* is defined as

$$\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\cdot \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right), \quad \text{where } 0 \leq \pi_j \leq 1, \sum_{j=1}^{K} \pi_j = 1.$$

Here, π_j are the weights or mixture coefficients and $\boldsymbol{\mu}_j \in \mathbb{R}^d$ and $\boldsymbol{\Sigma}_j \in \mathbb{R}^{d \times d}$ are the mean vector and covariance matrix for the j-th distribution, respectively, for any $j = 1, \ldots, K$.

An univariate GMM is visualized in Figure 7.2.
We see that given some discrete data, we assume that it really consists of a superposition of three individual normal distributions, each depicted using dashed lines. The solid line shows the weighted sum of them, i.e., the mixture distribution, which matches the data pretty well.
The rest of this chapter explains how such a (multivariate) model can be fitted to a given dataset $X = \{x_i\}_{i=1}^{N}$ and discusses difficulties.

Initialization

Before considering the fitting of a mixture distribution to data, we note that a good initialization is the key to fast convergence. Regarding the initialization of Gaussian Mixture Models, a broad variety of initialization procedures exists as surveyed in [13] and [14]. We decided to use a relatively quick and simple, yet well-suited initialization scheme which was inspired by [13, 15]. First, we run our K-Means algorithm on the dataset X which yields initial means denoted by $\boldsymbol{\mu}'_j \in \mathbb{R}^d, j = 1, \ldots, K$. The K-Means algorithm uses the K-Means++ initialization which we discussed in the previous chapter. We further compute an initial partitioning

$$P_j = \{\boldsymbol{x} \in X \mid \|\boldsymbol{x} - \boldsymbol{\mu}'_j\|_2 \leq \|\boldsymbol{x} - \boldsymbol{\mu}'_i\|_2,\ j \neq i = 1, \ldots, K\}, \quad j = 1, \ldots, K.$$

Just as when applying K-Means directly, if an item has equal distance to multiple cluster centers $\boldsymbol{\mu}_I, I \subset \{1, \ldots, K\}$, then it is assigned to the partition P_j endowed with the lowest index

$j = \min I$. Having an initial clustering, we estimate the inital model parameters of the j-th distribution, $j = 1, \ldots, K$, by

$$\begin{aligned}\pi_j &= \frac{\#P_j}{N},\\ \boldsymbol{\mu}_j &= \frac{1}{\#P_j}\sum_{\boldsymbol{x}\in P_j}\boldsymbol{x},\\ \boldsymbol{\Sigma}_j &= \frac{1}{\#P_j}\sum_{\boldsymbol{x}\in P_j}(\boldsymbol{x}-\boldsymbol{\mu}_j)(\boldsymbol{x}-\boldsymbol{\mu}_j)^T.\end{aligned}$$

Furthermore, we estimate the initial *responsibilities* used in the training algorithm. Intuitively, these are values that estimate how much a distribution explains a sample. In our initial hard cluster assignment, exactly one distribution is responsible for a sample, thus for the responsibilities we choose

$$\gamma_{n,j} = \begin{cases}1, & \boldsymbol{x}_n \in P_j,\\ 0, & \boldsymbol{x}_n \notin P_j,\end{cases} \qquad n = 1, \ldots, N,\ j = 1, \ldots, K.$$

7.2. Expectation Maximization

We just defined the initial responsibilities which is a hard cluster assignment due to the K-Means clustering algorithm. The training phase which we discuss in this section will change the model so it can indeed use a soft assignment based on how likely a data item belongs to a cluster.
For this training, we use the popular *Expectation Maximization*[1] (EM) algorithm, which aims at tuning model parameters in order to maximize the likelihood of a model given the training data. Since we concentrate on Gaussian Mixture Models, we want to tune the parameters of the normal distributions in a way such that the logarithmized likelihood given in Equation (7.1) is maximized.

$$\log \mathcal{L}\left(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \sum_{n=1}^{N} \log\left(\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)\right) \tag{7.1}$$

Before we state the overall algorithm, we want to note how the Gaussian density function is evaluated, namely as

$$\mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}\right) = \frac{1}{\sqrt{(2\pi)^d \det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

To avoid the inversion of the covariance matrix, we first compute a Cholesky decomposition of it, yielding a lower triangular matrix $\boldsymbol{L}$ such that $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$. Since the covariance matrix is symmetric and positive semi-definite, such a decomposition exists. Moreover, we will show in the next section that after some minor modifications it is further positive definite in our application, making the computed decomposition unique. The positive definiteness further ensures that $\det \boldsymbol{\Sigma} = (\det \boldsymbol{L})^2 > 0$, so the likelihood expression is well-defined.
Using this decomposition, the exponent in the density reduces to

$$(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) = \|\boldsymbol{L}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\|_2^2,$$

where the expression $\boldsymbol{y} := \boldsymbol{L}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$ can be computed numerically stable by solving the system $\boldsymbol{L}\boldsymbol{y} = (\boldsymbol{x}-\boldsymbol{\mu})$ using forward elimination.

[1] The idea of Expectation Maximization was postulated by many authors, notably Ceppellini et.al. [16], and it was popularized by Dempster et.al. [17].

Overall, we rewrite the density to obtain the form

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \exp\left(\log \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})\right) = \exp\left(-\frac{1}{2}\left(\|\boldsymbol{L}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\|_2^2 + d\log(2\pi) + 2\log\det \boldsymbol{L}\right)\right)$$

which subsequently allows an efficient computation of likelihood values. Finally, we can state the Expectation Maximization algorithm for Gaussian Mixture Models as it was explained in [15] which is summarized in Algorithm 9.

Algorithm 9: Expectation Maximization for Gaussian Mixture Models.

Function EM($\boldsymbol{X}$, K, $\boldsymbol{\theta}^{(0)}$)

Input : A matrix $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ consisting rowwise of feature vectors $\boldsymbol{x}_j, j = 1, \ldots, N$
Input : Number of component distributions K
Input : Initial parameters $\boldsymbol{\theta}^{(0)} = (\boldsymbol{\pi}^{(0)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\Sigma}^{(0)})$
Output: Tuned parameters $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}$

$m \leftarrow 0$
while *not converged* **do**
// E-step

$$\gamma_{n,j} \leftarrow \frac{\pi_j^{(m)} \mathcal{N}\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j^{(m)}, \boldsymbol{\Sigma}_j^{(m)}\right)}{\sum_{i=1}^{K} \pi_i^{(m)} \mathcal{N}\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_i^{(m)}, \boldsymbol{\Sigma}_i^{(m)}\right)}, \quad n = 1, \ldots, N,\ j = 1, \ldots, K$$

$\boldsymbol{\Gamma} = [\boldsymbol{\Gamma}_1 \cdots \boldsymbol{\Gamma}_K], \quad \boldsymbol{\Gamma}_j = (\gamma_{1,j}, \ldots, \gamma_{N,j})^T,\ j = 1, \ldots, K$
$\boldsymbol{H} \leftarrow (\|\boldsymbol{\Gamma}_j\|_1)_{1 \le j \le K}^T$

// M-step
$\boldsymbol{\pi}^{(m+1)} \leftarrow N^{-1}\boldsymbol{H}$
$\boldsymbol{\mu}^{(m+1)} \leftarrow \left(\boldsymbol{\Gamma}^T \boldsymbol{X}\right)^T \oslash \boldsymbol{H}$
$\boldsymbol{\Sigma}_j^{(m+1)} \leftarrow \boldsymbol{H}_j^{-1}\, \boldsymbol{C}^T \operatorname{diag}(\boldsymbol{\Gamma}_j)\, \boldsymbol{C}, \qquad j = 1, \ldots, K$
$m \leftarrow m + 1$

return $\boldsymbol{\pi}^{(m)}, \boldsymbol{\mu}^{(m)}, \boldsymbol{\Sigma}^{(m)}$

The EM algorithm consists of an "expectation" and a "maximization" step. The first step re-estimates the responsibilities $\gamma_{n,j}$ and aggregates them in the matrix $\boldsymbol{\Gamma}$, whose vector of column-wise sums is denoted by $\boldsymbol{H}$. From this new soft cluster assignment, the mixture distribution model parameters are updated using the usual definitions, where the matrix $\boldsymbol{C}$ is the dataset $\boldsymbol{X}$ centered around the currently estimated means, i.e., $\boldsymbol{C}_{n,f} = \boldsymbol{X}_{n,f} - (\boldsymbol{\mu}^{(m+1)})_{n,f}$, $n = 1, \ldots, N, f = 1, \ldots, d$. Furthermore, the $\boldsymbol{A} \oslash \boldsymbol{x}$ operator denotes that every row in $\boldsymbol{A}$ is divided by $\boldsymbol{x}$ componentwise, i.e., $(\boldsymbol{A} \oslash \boldsymbol{x})_{i,j} = \boldsymbol{A}_{i,j}/\boldsymbol{x}_j$ over i rows and j columns.
These two steps are iterated until the algorithm converges or reaches a maximum number of iterations, which was experimentally determined to be 1024 iterations as the algorithm typically converges quickly. To check convergence, we compare the values of the logarithmized likelihood of the mixture given the dataset $\boldsymbol{X}$.

7.3. Avoiding Degeneracies

As can easily be seen from the definition of the density function of both the univariate and multivariate normal distribution, the evaluation of it requires a division by the standard deviation or the multiplication with the inverse of the covariance matrix, respectively.
This clearly causes problems in presence of singularities, i.e., if one distribution is focussed around a single data item, thus having variance zero. This behavior was characterized in [18] in the univariate and in [19] in the multivariate case. Both publications use basically the same approach. They add small positive terms to both the numerator and denominator in the re-estimation of the variance expressions in the maximization step, ensuring that the re-estimations do not get singular. These terms stem from the multiplication of the likelihood functions with the conjugate prior of the distributions with respect to the estimation of unknown variance expressions.

7.3.1. The Univariate Case

In the univariate case, the conjugate prior for the normal distribution is given as inverse gamma distribution prior with parameters $\alpha, \beta > 0$. As shown in [18], this only changes the re-estimation of the variance to the expression

$$(\sigma_j^2)^{(m+1)} = \frac{1}{2\beta + \boldsymbol{H}_j} \left(2\alpha + \sum_{n=1}^{N} \gamma_{n,j} (x_n - \mu_j)^2 \right), \quad j = 1, \ldots, K,$$

and further always yields positive variance terms, effectively avoiding the numerical problems mentioned above. They further state that this modification does not change the convergence behavior. In our setting, these terms were experimentally found to be $\alpha = \beta = 10^{-7}$.

7.3.2. The Multivariate Case

For the multivariate case, the conjugate prior is given by an inverse Wishart prior $\mathcal{W}(\nu, \Lambda)$ as pointed out in [19] where ν denotes the degrees of freedom and Λ shall be a positive definite matrix. We follow this approach as presented in [14, p.9]. Therefore, we choose the degrees of freedom as $\nu = d + 2$ and the prior matrix to be

$$\Lambda = \left(\frac{\sigma_0}{\varepsilon + \det \boldsymbol{S}} \right)^{\frac{1}{d}} \boldsymbol{S}, \qquad \boldsymbol{S} = \frac{1}{N-1} \sum_{n=1}^{N} (\boldsymbol{x}_n - \mathbb{E}[\boldsymbol{X}])(\boldsymbol{x}_n - \mathbb{E}[\boldsymbol{X}])^T,$$

where $\boldsymbol{S}$ is the empirical covariance matrix of the dataset $\boldsymbol{X}$, $\varepsilon = 10^{-5}$ and σ_0 is a small positive number, experimentally determined to be 0.1.
Again, the only change in the EM algorithm is the re-estimation of the covariance matrices, which changes [14] to

$$\boldsymbol{\Sigma}_j^{(m+1)} = \frac{1}{\boldsymbol{H}_j + \nu + d + 2} \left(\boldsymbol{C}^T \operatorname{diag}(\boldsymbol{\Gamma}_j) \boldsymbol{C} + \Lambda \right).$$

By definition, both the denominator is positive and $\boldsymbol{\Lambda}$ is positive definite, thus ensuring that the estimated covariance matrix stays positive definite.

7.4. Model Selection

A key element for mixture-model-based clustering is the choice of the number K of clusters. In general, it is not possible to choose K automatically in a meaningful way and thus it needs to be

provided explicitly. However, we can choose the best number of clusters in a given range using the *Akaike Information Criterion* [20].

Definition 7.4.1. Given some model m with P independently adjustable parameters $\boldsymbol{\theta}$ and a fixed dataset $\boldsymbol{X}$, the *Akaike Information Criterion* (AIC) is defined as [20]

$$\mathrm{AIC}(m) = 2P - 2\log \mathcal{L}(\boldsymbol{X} \mid \boldsymbol{\theta}),$$

where $\log \mathcal{L}(\boldsymbol{X} \mid \boldsymbol{\theta})$ denotes the logarithmized likelihood of $\boldsymbol{X}$ given the model parameters $\boldsymbol{\theta}$ as previously defined in Equation (7.1).

Intuitively, this information criterion estimates how much information is lost or gained when comparing two models using it.

Definition 7.4.2. Let $m_i, i = 1, \ldots, K$, be models of the same kind but with different model parameters $\boldsymbol{\theta}_i$ all trained on a dataset $\boldsymbol{X}$.
Then, the best fitting model m^* according to the Akaike Information Criterion is given by

$$m^* = \underset{m_1,\ldots,m_K}{\operatorname{argmin}} \mathrm{AIC}(m_i).$$

In practice, we fit Gaussian Mixture Models having 2 to K clusters in parallel to the data set. After the training procedures are completed, we select the best model according to Definition 7.4.2.

7.5. Clustering Big Volumes

Having the Expectation Maximization algorithm at hand, we finally give a procedure for clustering big volumes using Gaussian Mixture Models, which is shown in Algorithm 10.

Algorithm 10: Volume clustering based on Gaussian Mixture Models.

Input : A volume V of voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \boldsymbol{\Gamma}_{\boldsymbol{d}}$
Input : A feature computation function f
Input : Number of clusters K
Input : The sample size $M \leq \#\boldsymbol{\Gamma}_{\boldsymbol{d}}$
Output: A segmented volume

$S \leftarrow$ a stratification acc. Def. 5.1.1
$\Omega \leftarrow \texttt{StratifiedSampling}(V, M, S)$
$\boldsymbol{\theta}^{(0)} \leftarrow (\boldsymbol{\pi}^{(0)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\Sigma}^{(0)})$ `// Init. on Ω using f (Section 7.1)`
$\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma} \leftarrow \mathrm{EM}(\{f(\boldsymbol{x}_{\boldsymbol{\beta}}) \mid \boldsymbol{x}_{\boldsymbol{\beta}} \in \Omega\}, K, \boldsymbol{\theta}^{(0)})$
parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V$ **do**

$$\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow \underset{j=1,\ldots,K}{\operatorname{argmax}} \pi_j \mathcal{N}\left(f(\boldsymbol{x}_{\boldsymbol{\alpha}}) \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)$$

The first phases extract a stratified random sample of fixed size from the volume, after which a Gaussian Mixture Model is trained on this sample. Finally, to each output voxel the index of the cluster is assigned which maximizes the likelihood of the input voxel given the trained model.

7.5.1. Accelerating the Univariate Case

In the general case, we need to extract a feature vector from each voxel. These are computed ad hoc and in a concrete implementation this is done using a mask-based processing as explained

in Section 1.4. However, if we want to cluster a volume solely based on its grayscale values, we can accelerate the clustering by switching to a thresholding method and a much faster pointwise processing.
In this univariate case, the model parameters are given as μ_j, σ_j^2, and we assume these parameters to be sorted by $\mu_j \leq \mu_{j+1}$. Now, the goal is to find separating thresholds, i.e., real numbers $T_j, j = 1, \ldots, K-1$, such that $\pi_j \mathcal{N}\left(T_j \mid \mu_j, \sigma_j^2\right) = \pi_{j+1} \mathcal{N}\left(T_j \mid \mu_{j+1}, \sigma_{j+1}^2\right)$, which reduces to finding solutions to the quadratic equation

$$0 = \left(\sigma_{j+1}^2 - \sigma_j^2\right) T_j^2 + 2\left(\sigma_j^2 \mu_{j+1} - \sigma_{j+1}^2 \mu_j\right) T_j + \left(\sigma_{j+1}^2 \mu_j^2 - \sigma_j^2 \mu_{j+1}^2 - 2\sigma_j^2 \sigma_{j+1}^2 \log \frac{\pi_j \sigma_{j+1}}{\pi_{j+1} \sigma_j}\right).$$

We try to solve this equation such that $\mu_j \leq T_j \leq \mu_{j+1}$ holds. If no such solution exists, we use $(\mu_j + \mu_{j+1})/2$ as an approximation.
With these thresholds, Algorithm 11 shows efficient pointwise processing of a volume.

Algorithm 11: Volume clustering based on univariate Gaussian Mixture Models.

Input : A volume V of voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \boldsymbol{\Gamma}_{\boldsymbol{d}}$
Input : Number of clusters K
Input : The sample size $M \leq \#\boldsymbol{\Gamma}_{\boldsymbol{d}}$
Output: A segmented volume

$S \leftarrow$ a stratification acc. Def. 5.1.1
$\Omega \leftarrow$ `StratifiedSampling(`V, M, S`)`
$\boldsymbol{\theta}^{(0)} \leftarrow (\boldsymbol{\pi}^{(0)}, \boldsymbol{\mu}^{(0)}, \boldsymbol{\Sigma}^{(0)})$ `// Init. on` Ω `using` f `(Section 7.1)`
$\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma} \leftarrow$ `EM(`$\{f(\boldsymbol{x}_{\boldsymbol{\beta}}) \mid \boldsymbol{x}_{\boldsymbol{\beta}} \in \Omega\}, K, \boldsymbol{\theta}^{(0)}$`)`
Compute $T_j, j = 1, \ldots, K-1$, let $T_K = +\infty$
parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V$ **do**
  $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow \min\{j \in \{1, \ldots, K\} \mid \mathrm{v}(\boldsymbol{x}_{\boldsymbol{\alpha}}) < T_j\}$

The first phases are identical to Algorithm 10, while the final loop assigns the cluster index to each voxel, where the cluster index is the smallest index such that the voxel value is smaller than the according threshold.

7.6. Runtime and Memory Analysis

Similar to the analysis of the K-Means clustering algorithm given in Section 6.5, this section provides asymptotic estimates for both the runtime and memory requirements of the mixture-model-based segmentation technique presented in Algorithm 10.
Again, let $N \in \mathbb{N}$ be the number of voxels in the volume and denote $M \leq N$ the size of the random sample drawn from the voxel set, which again is fixed in our applications to have 4096 elements.

7.6.1. Runtime Analysis

The first steps of the presented algorithm are similar to the K-Means case: First, a stratified random sample is extracted which uses a runtime of order $\mathcal{O}(NN_T^{-1} + N_T M)$. By the same argumentation as in Section 6.5, the sampling runtime is linear in the number of voxels, i.e., of order $\mathcal{O}(N)$. The Expectation Maximization is guaranteed to converge [21], let $c \in \mathbb{N}$ denote the number of iterations needed for convergence. As mentioned earlier, in our setting we set the maximum number of iterations to 1024 as this often suffices for a good result. Thus, the runtime

necessary is given by $\min\{c, 1024\}$. The final voxelwise classification is clearly linear in the number of voxels in the volume.
Concludingly, the runtime for our clustering algorithm is bounded by two linear passes through the volume and the parameter estimation procedure and hence is linear in N.

7.6.2. Memory Requirements

In the first step of the algorithm, a random sample is extracted, requiring $\mathcal{O}(M)$ space where we emphasize that this is constant as the sample size M does not depend on the volume size in our setting. During the training phase using the EM algorithm, only the extracted sample and the trained parameters must be kept in memory, thus the necessary memory is still constant. Finally, during the voxelwise segmentation we only need constant space for the current voxel, or features extracted from it, and the trained parameters.
Hence, the overall memory consumption for our Gaussian Mixture Model-based clustering algorithm is constant, regardless of the input volume size.

References

[1] Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[2] Jeffrey S. Vitter. "Random Sampling with a Reservoir". In: *ACM Trans. Math. Softw.* 11.1 (Mar. 1985), pp. 37–57.

[3] A. I. McLeod and D. R. Bellhouse. "A Convenient Algorithm for Drawing a Simple Random Sample". In: *Journal of the Royal Statistical Society Series C* 32.2 (June 1983), pp. 182–184.

[4] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[5] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[6] Daniel Aloise et al. "NP-hardness of Euclidean sum-of-squares clustering". In: *Machine Learning* 75 (Jan. 2009), pp. 245–248.

[7] Stuart P. Lloyd. "Least Squares Quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[8] M. Emre Celebi, Hassan A. Kingravi and Patricio A. Vela. "A Comparative Study of Efficient Initialization Methods for the K-Means Clustering Algorithm". In: *Expert Systems with Applications* 40.1 (2013), pp. 200–210.

[9] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[10] John E. Angus. "The Probability Integral Transform and Related Results". In: *SIAM Review* 36.4 (1994), pp. 652–654.

[11] David Arthur, Bodo Manthey and Heiko Röglin. *k-Means has Polynomial Smoothed Complexity*. 2009. arXiv: 0904.1113.

[12] User:Chire. *File:ClusterAnalysis Mouse.svg*. Online under https://commons.wikimedia.org/wiki/File:ClusterAnalysis_Mouse.svg. Oct. 2010.

[13] Emilie Shireman, Douglas Steinley and Michael J. Brusco. "Examining the effect of initialization strategies on the performance of Gaussian mixture modeling". In: *Behavior Research Methods* 49 (Feb. 2017), pp. 282–293.

[14] Jean-Patrick Baudry and Gilles Celeux. "EM for mixtures: Initialization requires special care". In: *Statistics and Computing* 25 (Feb. 2015).

[15] Christopher Bishop. *Pattern Recognition and Machine Learning*. 1st ed. Springer, 2006.

[16] Ruggero Ceppellini, Marcello Siniscalco and C.A.B. Smith. "The estimation of gene frequencies in a random-mating population". In: *Annals of human genetics* 20.2 (Nov. 1955), pp. 97–115.

[17] Arthur P. Dempster, Nan M. Laird and Donald B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 1–38.

[18] Andrea Ridolfi and Jérôme Idier. *Penalized Maximum Likelihood Estimation for Normal Mixture Distributions*. Tech. rep. École Polytechnique Fédérale de Lausanne, School of Computer and Information Sciences, 2002.

[19] Hichem Snoussi and Ali Mohammad-Djafari. "Degeneracy and likelihood penalization in multivariate Gaussian mixture models". In: *Univ. of Technology of Troyes, Troyes, France, Tech. Rep. UTT* (2005).

[20] Hirotugu Akaike. “A New Look at the Statistical Model Identification”. In: *IEEE Transactions on Automatic Control* 19.6 (Dec. 1974), pp. 716–723.

[21] C. F. Jeff Wu. “On the Convergence Properties of the EM Algorithm”. In: *Ann. Statist.* 11.1 (Mar. 1983), pp. 95–103.

Part IV.

Supervised Segmentation

Ab uno disce omnes.

Virgil, Aeneid

Table of Contents

We continue our journey towards scalable voxel volume segmentation techniques by considering *supervised* segmentation techniques. Contrary to the methods discussed in Part III, now explicit

training data comes into play. Typically, an expert provides *mask volumes* in which every single voxel is manually annotated with either the value one if the voxel belongs to the desired object or part, and zero otherwise. However, our general setting does not assume the presence of such labeled data, which is typically cumbersome to generate. With this motivation in mind, this part proposes algorithms which segment volumes using an interactive and iterative approach. Hence, the presented techniques allow for including domain-specific knowledge which stems from an expert interactively labeling voxels which is iterated until the results are good enough. In the following, we will refer to this interactively selected voxels as *seed voxels*.
The first chapter discusses a thresholding algorithm which adapts a given threshold according to locally estimated features in order to counteract voxel intensity deviations occuring on, e.g., thin bone structures. Adaptive thresholding can also improve segmentation results in presence of artifacts occuring frequently in computed tomography scans. Next, we introduce the arguably most versatile algorithm discussed in this dissertation, namely a segmentation procedure based on Support Vector Machines. The memory and runtime analyses will show that both methods are well-suited for processing large voxel volumes. We proceed by defining a general framework for partitioning large data sets by interpreting a volume on different resolution levels with moderate memory overhead. Furthermore, we show how our supervised segmentation algorithms can be embedded in a multiresolution setting using the latter to gain considerable runtime benefits without losing accuracy. Finally, further memory and runtime analyses are performed for general multiresolution algorithms.

8. Feature-Adaptive Interactive Thresholding

Abstract Without a doubt, the most used segmentation technique, especially in the field of computed tomography, is thresholding as described in Chapter 2.1. In its simplest form, this technique uses a global threshold and binarizes the volume pointwise. Since this simple technique is fast and suffices for many applications, a variety of similar methods was developed over time, cf. [1–7], to only cite a few. Here, we propose a refined supervised thresholding technique allowing a modification of the global threshold by local geometric information. The training data is provided interactively by the user, which is why we named the procedure explained in the following *Feature-Adaptive Interactive Thresholding*, or *FAITH* for short.
In subsequent sections, the underlying model is constructed as an optimization problem. Also, an iterative procedure for solving it is proposed. We further prove that that the sequence generated by that procedure converges against the optimal solution. The chapter is concluded by an analysis of the constructed method which shows that it acts local enough such that big volumes can be segmented efficiently.

8.1. Related Work

The usage of additional information in thresholding is no new idea, e.g., [8] proposed an algorithm for local thresholding obtained by minimizing a functional describing the projective distance between the projection data of a CT scan and its segmented reconstruction. Here, we focus on techniques processing volumetric data only. In particular, we cannot assume that projection data is always available, especially in context of processing big volumes where the projection images require a huge amount of memory.
There already exist adaptive thresholding techniques which operate solely on voxel grids. Notable examples are the works by Niblack [9] and Sauvola [10]. Both methods choose a threshold in a local region based on the contained grayscale value distribution and proved to be highly successful in processing text images suffering from illumination problems. But they often perform badly on computed tomography scans having low contrast or a lot of noise. That is, because in local regions filled with noise, i.e., containing significant variance, both methods compute thresholds that explicitly select that noise as material voxels. Hence, in very noisy scans the scanned objects might not be distinguishable from noise. Such misclassifications hint that in the case mentioned simple grayscale features are not sufficient. For this reason, geometric information is expected to improve the segmentation results.
Contrary to the above, we took inspiration from an approach presented in [11] where a global threshold was adapted locally based on the planarity of the local environment. Conceptually, this increases the detection rate of fine planar structures in, e.g., bones in computed tomography scans while also thickening them. But the approach lacks flexibility as it depends on two additional hyperparameters and only allows one feature, the planarity of an environment, to have an influence on the threshold.
Our approach overcomes these shortcomings and generalizes the cited model. It consists of a training and a classification phase. In the training phase the user marks training data interactively. This means that a selection of voxels of the desired object and also of voxels in regions where simple binarization fails are selected by a possibly human operator. Therefore, entirely

noisy regions can be omitted by not selecting them and thus we effectively omit the problems occuring when using Niblack or Sauvola while still operating adaptively and locally. The classification phase segments a volume by local thresholding depending on the trained information.

8.2. Deriving a Regularized Thresholding Problem

We start by deriving an optimization problem which describes how a global threshold should be adapted using local feature information for optimal segmentation. In subsequent sections, a regularization approach is introduced to keep solutions smooth yet sparse.

8.2.1. Derivation of the Initial Problem

Firstly, we note that automatically choosing a threshold locally as demonstrated by Sauvola or Niblack can fail, as pointed out before. Contrary, especially in the context of segmenting computed tomography scans, a global binarizing threshold might suffice for most of the desired object. In order to optimize the segmentation in cases where the global threshold is not optimal, we allow a modification of it in local regions based on geometric features which we discussed in Chapter 4.
We start by considering a global threshold θ_g provided by a user. This value depends on the data and shall be chosen in a way such that most voxels of the desired object are captured by a plain binarization using this threshold. In particular, this threshold lies in the valid grayscale value interval $[0, W]$ where $W \geq 0$ represents the maximum possible representable voxel value given a fixed data type. In the following, consider given training data $\{\boldsymbol{x}_j\}_{j=1}^{M}$ consisting of $M \in \mathbb{N}$ seed voxels and local environments $\mathcal{T} = \{U_j := R_K(\boldsymbol{x}_j)\}_{j=1}^{M}$ around each seed voxel. These data items stem from user interaction and belong to the desired object, especially to regions where simple binarization fails. To each environment U_j we further estimate an optimal binarizing threshold $\theta^*(U_j), j = 1, \ldots, M$, obtained via *Minimum Cross Entropy Thresholding* [12]. That method estimates a threshold minimizing the binarization error in an unsupervised manner similar to Otsu's algorithm, but improves in the sense that it is no longer biased when the underlying distributions are very different. The minimized error describes a comparison of the source value distributions against the segmented distributions. A detailed explanation can be found in [12].
Additionally, denote by $\mathcal{U}$ the set of all local environments around voxels and

$$\begin{aligned} F &: \mathcal{U} \to \mathbb{R}^d, \\ U &\mapsto F(U), \end{aligned} \tag{8.1}$$

as a function which maps an environment U to a d-dimensional real feature vector.
All together, we adapt a threshold locally based on features as follows:

Definition 8.2.1. Let θ_g be the global threshold and let F be the feature function of the form shown in Equation (8.1). For a voxel $\boldsymbol{x}_{\boldsymbol{\alpha}}$ and its respective local environment $U = R_K(\boldsymbol{x}_{\boldsymbol{\alpha}})$ the *local threshold* mapping is defined through

$$\begin{aligned} \theta &: \mathcal{U} \to \mathbb{R} \\ U &\mapsto \theta_g + \boldsymbol{\beta}^T F(U) \end{aligned}$$

for weights $\boldsymbol{\beta} \in \mathbb{R}^d$.

We aim to train the weights $\boldsymbol{\beta}$ in a way such that the local thresholds are as close as possible to the optimal thresholds. Therefore, the goal is to solve the optimization problem

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}\sum_{k=1}^{M} |\theta_g + \boldsymbol{\beta}^T F(U_k) - \theta^*(U_k)|_2^2$$

which can be rewritten compactly in matrix form as

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{\mathcal{F}}\boldsymbol{\beta} - \boldsymbol{\Theta}\|_2^2 \tag{8.2}$$

where

$$\boldsymbol{\mathcal{F}} = \begin{bmatrix} (F(U_1))^T \\ \vdots \\ (F(U_M))^T \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{\Theta} = \begin{bmatrix} \theta^*(U_1) - \theta_g \\ \vdots \\ \theta^*(U_M) - \theta_g \end{bmatrix}.$$

Clearly, this is a classical least squares problem and can be solved as usual. However, for several reasons we further introduce regularization to this problem.

8.2.2. Regularization

Depending on the actual data, some features might yield the same values for at least two environments U_j which in turn renders the resulting problem ill-conditioned. Additionally, redundant information in feature vectors can distort the trained weights. In order to counteract these effects, we introduce an ℓ_2-regularization term. Consequently, the solution gets smoother and omits singularities. Furthermore, we introduce a ℓ_1-regularization term, which steers the solution towards a sparse one, as is commonly known. Then, only *important* feature values, i.e., values significantly different from zero, remain in the solution vector and hence this can be interpreted as a kind of automatic feature selection.
We use both techniques simultaneously which is also known as *elastic net regularization* [13].

Definition 8.2.2. Let J be a cost function of a minimization problem.
Then define the *elastic net regularization* of J with parameters $\lambda > 0, \mu \in (0,1)$ by

$$\min_{\boldsymbol{x}} J(\boldsymbol{x}) + \lambda\left(\frac{1}{2}(1-\mu)\|\boldsymbol{x}\|_2^2 + \mu\|\boldsymbol{x}\|_1\right).$$

We see that Definition 8.2.2 requires two additional hyperparameters $\lambda > 0$ and $\mu \in (0,1)$. Note that in literature $\mu = 1$ is allowed as well, but we choose to omit this value to avoid singularities during later processing. The parameter λ controls the influence of regularization and also how smooth the solution will be. That is, the larger it gets, the less the actual data participates in training, while in the opposite case we rely more on the given data and only smooth a little bit. On the other hand, the parameter μ controls the trade-off between smooth and sparse solutions. So, as μ approaches 1 more emphasis is put on producing sparse vectors, while lower values allow for less sparse but smoother solutions.
Applying Definition 8.2.2 to our optimization problem given in Equation (8.2) yields the regularized problem

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{\mathcal{F}}\boldsymbol{\beta} - \boldsymbol{\Theta}\|_2^2 + \lambda\left(\frac{1}{2}(1-\mu)\|\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{\beta}\|_1\right). \tag{8.3}$$

In later steps, the hyperparameters λ and μ will be chosen to give a appropriate trade-off between smooth and sparse solutions and respecting the training data.

8.2.3. Limitation to Meaningful Parameters

An immediate observation we make is that our solution vector is an element of the space $\mathbb{R}^d$ without any restriction. However, since we aim to train weights for generating thresholds, we can constrain the feasible set further such that the resulting thresholds are valid values in the data-type-specific grayscale value range. For this limitation, we consider the feasible grayscale value value interval $[0, W]$. The constraint is then expressed as $0 \leq \theta_g + \boldsymbol{\beta}^T F(U_k) \leq W$ for all $k = 1, \ldots, M$. In matrix form, this is equivalently expressed through the convex polytope $\mathcal{C} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{C}\,\boldsymbol{x} \leq \boldsymbol{d}\}$ where

$$\boldsymbol{C}\boldsymbol{\beta} \leq \boldsymbol{d} \quad :\Leftrightarrow \quad \begin{bmatrix} -\boldsymbol{\mathcal{F}} \\ \boldsymbol{\mathcal{F}} \end{bmatrix} \boldsymbol{\beta} \leq \begin{bmatrix} \theta_g \mathbf{1} \\ (W - \theta_g)\mathbf{1} \end{bmatrix}.$$

Since the grayscale value interval includes zero and since the global threshold stems from this interval too, we conclude that the polytope $\mathcal{C}$ is nonempty as at least the point 0 is included. Additionally, it is easy to see that polytopes of this shape are closed convex sets.
Concludingly, the final optimization problem we aim to solve is given by

$$\begin{aligned} \boldsymbol{\beta}^* = \underset{\boldsymbol{\beta} \in \mathbb{R}^d}{\operatorname{argmin}} \quad & \frac{1}{2}\|\boldsymbol{\mathcal{F}}\boldsymbol{\beta} - \boldsymbol{\Theta}\|_2^2 + \lambda \left(\frac{1}{2}(1-\mu)\|\boldsymbol{\beta}\|_2^2 + \mu\|\boldsymbol{\beta}\|_1 \right) \\ s.t. \quad & \boldsymbol{C}\boldsymbol{\beta} \leq \boldsymbol{d}. \end{aligned} \tag{8.4}$$

The next section will describe the individual building blocks which together comprise the solution of that problem.

8.3. Solving the Problem

In order to solve the problem given in Equation (8.4), we first observe that this convex problem can be expressed as sum of two convex functions. We trivially rewrite it as

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta} \in \mathbb{R}^d}{\operatorname{argmin}}\, f(\boldsymbol{\beta}) + g(\boldsymbol{\beta}) \quad s.t. \quad \boldsymbol{C}\boldsymbol{\beta} \leq \boldsymbol{d},$$

where

$$\begin{aligned} f(\boldsymbol{\beta}) &= \frac{1}{2}\|\boldsymbol{\mathcal{F}}\boldsymbol{\beta} - \boldsymbol{\Theta}\|_2^2 + \frac{1}{2}\lambda(1-\mu)\|\boldsymbol{\beta}\|_2^2, \\ g(\boldsymbol{\beta}) &= \lambda\mu\|\boldsymbol{\beta}\|_1. \end{aligned} \tag{8.5}$$

The important part is that the function f is convex and differentiable, while g is convex. Therefore, it can be solved using *proximal gradient methods* which intuitively are gradient descent algorithms capable of handling non-differentiable parts.

8.3.1. Proximal Methods

Before actually solving Problem (8.4), we give a very brief overview over proximal methods. As stated, a general iterative solving algorithm performs a gradient descent-like iteration with special treatment for the non-differentiable parts.
To this end, we want to note that the regularized minimization problem *without* the constraints introduced in Section 8.2.3 could be transformed into a binary classification problem which itself is solved using an appropriate solver as demonstrated in [14].
A general scheme for solving such problems using the proximal gradient method is sketched in Algorithm 12. The depicted iteration is known as *forward-backward splitting* in literature [15]. That name is derived from the two steps in it, a *forward* step which is a gradient descent iteration, followed by the a *backward* step executing the proximal step.

Algorithm 12: General proximal gradient method solver.

Function `SolveProximal(`f, g, d`)`
  Input : Functions f and g as specified above
  Input : The dimensionality d of the solution vector
  Output: The minimizer $\boldsymbol{x}^* \in \mathbb{R}^d$
  $\boldsymbol{x}^* \leftarrow \mathbf{0}$
  $\delta \leftarrow$ step size
  while *not converged* **do**
    $\boldsymbol{x}^* \leftarrow \text{prox}_{\delta g}\left(\boldsymbol{x}^* - \delta \nabla f(\boldsymbol{x}^*)\right)$
  return $\boldsymbol{x}^*$

The method begins by initializing the solution to some arbitrary feasible vector for which we choose the zero vector and compute a problem-dependent step size. The step size can be fixed or be adapted in each iteration. Each iteration then first executes a simple gradient descent step with the current step size. Next, that intermediate result is fed to a *proximity operator* prox_g associated to a proper, lower-semicontinuous function g. We call a function proper if its codomain is $\mathbb{R} \cup \{+\infty\}$ but if it is not identically equal to $+\infty$. The proximal operator to such a function g is defined as [16]

$$\text{prox}_g(\boldsymbol{x}) = \underset{\boldsymbol{z}}{\text{argmin}} \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2 + g(\boldsymbol{z}),$$

and is a convex function with respect to $\boldsymbol{x}$. In practice, one chooses the function g in a way such that the proximal operator has an analytic solution or can be computed efficiently.
In Algorithm 12, we say that the iterations have converged if a fixed point is reached, i.e., if the solution vector $\boldsymbol{x}^*$ does not change anymore. Numerically, we check if the change of this vector between two consecutive iterations is less than a certain problem-dependent constant.

8.3.2. Solving our Problem

In the current section, we apply the introduced techniques to our problem and give an algorithm for solving it.

Gradient descent

We already saw that a proximal gradient method solver includes a regular gradient descent step. That is why we first state the gradient of f which is given by

$$\nabla f(\boldsymbol{\beta}) = \boldsymbol{\mathcal{F}}^T\left(\boldsymbol{\mathcal{F}}\boldsymbol{\beta} - \boldsymbol{\Theta}\right) + \lambda(1-\mu)\boldsymbol{\beta}.$$

Consequently, a gradient descent iteration concerning f can be expressed via

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \delta\left(\boldsymbol{\mathcal{F}}^T\left(\boldsymbol{\mathcal{F}}\boldsymbol{\beta}^{(k)} - \boldsymbol{\Theta}\right) + \lambda(1-\mu)\boldsymbol{\beta}^{(k)}\right). \tag{8.6}$$

There are various methods to estimate the step size $\delta > 0$ in order to guarantee convergence of the iterations towards the minimizer, e.g., by searching the optimal step size on a line connecting two feasible values or by simply using a constant step size. For simplicity, we focus on the latter and will give an expression for it in later sections when we discuss the convergence of the algorithm.

ℓ_1 Proximity

A proximal part remaining to solve is the function $g(\boldsymbol{x}) = \lambda\mu\|\boldsymbol{x}\|_1$ which incorporates the convex but nonsmooth ℓ_1 norm. In general, we have to solve the minimization problem in the proximity operator introduced earlier. For this specific function, it is shown in literature [15] that the according proximal operator yields a scaled soft-thresholding function.

Definition 8.3.1. Let $T > 0$ be some threshold. Define the *soft-thresholding operator* by

$$\begin{aligned} S_T \colon \mathbb{R} &\to \mathbb{R}, \\ x &\mapsto \operatorname{sgn}(x)\max\{0, |x| - T\}. \end{aligned}$$

The solution to the proximal operator for our function g then is a scaled componentwise application of the soft-thresholding operator on the solution vector, i.e.,

$$\operatorname{prox}_{\delta g}(\boldsymbol{x}) = (S_{\delta\lambda\mu}(\boldsymbol{x}_k))_{k=1}^{d}\,. \tag{8.7}$$

That procedure is also given in Algorithm 13.

Algorithm 13: Elementwise soft-thresholding.

Function `SoftThresholding(`$\boldsymbol{x}$, δ, λ, μ`)`

Input : The input vector $\boldsymbol{x} \in \mathbb{R}^d$
Input : Scalings $\delta > 0, \lambda > 0, \mu \in (0, 1)$
Output: The thresholded vector

for $k \leftarrow 1$ **to** d **do**
  $\boldsymbol{x}_k \leftarrow \operatorname{sgn}(x_k)\max\{0, |\boldsymbol{x}_k| - \delta\lambda\mu\}$
return $\boldsymbol{x}$

Polytope Constraints

However, we additionally require the generated local thresholds to lie in the feasible grayscale value range, too. In the process of solving our problem with the additional constraints, we introduce another proximal step dealing with that.
First of all, it is well-known that the indicator function

$$\iota_A(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in A, \\ +\infty, & \boldsymbol{x} \notin A, \end{cases}$$

yields $\operatorname{prox}_{\delta\iota_A}(\boldsymbol{x}) = \mathcal{P}_A(\boldsymbol{x})$ as proximal result, i.e., the proximal operator of that function is the projection onto the according set. This is easily verified as

$$\operatorname{prox}_{\delta\iota_A}(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{z}} \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{x}\|_2^2 + \delta\iota_A(\boldsymbol{z}) = \operatorname*{argmin}_{\boldsymbol{z}\in A} \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{x}\|_2^2 = \mathcal{P}_A(\boldsymbol{x})$$

holds true for all $\boldsymbol{x} \in \mathbb{R}^d$.
Similarly, the proximal operator for the indicator function of the polytope restricting the feasible set is given by the projection onto it, i.e., $\operatorname{prox}_{\delta\iota_C}(\boldsymbol{x}) = \mathcal{P}_C(\boldsymbol{x})$ where $C = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{C}\,\boldsymbol{x} \leq \boldsymbol{d}\}$ is our polytope. Since in general there does not exist a closed-form solution for the projection of a point onto a convex polytope, we use Hildreth's method for inequalities [17, p. 283] to calculate this proximal operator efficiently.
The iterative solving algorithm for this sub-problem is repeated in Algorithm 14, in which the vectors $\boldsymbol{e_i}, i = 1, \ldots, 2M$, are the standard basis elements of $\mathbb{R}^{2M}$. Note that the number of iter-

ations is limited by a constant upper limit $k_{\max}$, although in most cases the algorithm terminates early. That limit was experimentally determined to be 1024 for our applications.

Algorithm 14: Projection onto a convex polytope via Hildreth's method.

Function `ProjectOntoPolytope(`$\boldsymbol{x}, \boldsymbol{C}, \boldsymbol{d}, k_{\max} = 1024$`)`

Input : The input vector $\boldsymbol{x} \in \mathbb{R}^d$
Input : A polytope defined by $\boldsymbol{C} \in \mathbb{R}^{2M \times d}, \boldsymbol{d} \in \mathbb{R}^{2M}$ where M is the number of polytope constraints
Input : The maximum number of iterations $k_{\max}$
Output: The projected point $\boldsymbol{p} \in \mathbb{R}^d$

$\boldsymbol{p} \leftarrow \boldsymbol{x}$
$\boldsymbol{\lambda} \leftarrow \boldsymbol{0}$
$\left(n_j = \|\boldsymbol{e_j}^T \boldsymbol{C}\|_2^2\right)_{j=1}^{2M}$ `// rowwise squared` ℓ_2`-norm`
for $k \leftarrow 0$ **to** $k_{\max}$ **do**
  for $i \leftarrow 1$ **to** $2M$ **do**
    $c \leftarrow \min\left\{\lambda_i, \frac{\boldsymbol{d}_i - \boldsymbol{e_i}^T \boldsymbol{C}\boldsymbol{p}}{n_i}\right\}$
    $\boldsymbol{p} \leftarrow \boldsymbol{p} + c\,\boldsymbol{C}^T \boldsymbol{e_i}$
    $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - c\,\boldsymbol{e_i}$
  if $k > 0$ *and* $\boldsymbol{p}$ *did not change* **then**
    break
return $\boldsymbol{p}$

Hildreth also showed that if the polytope is not empty, Algorithm 14 converges [17, Theorem 3] against the solution of $\operatorname{argmin}_{\boldsymbol{z} \in \mathcal{C}} \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{x}\|_2^2$, which is exactly the required proximal operator $\operatorname{prox}_{\delta \iota_{\mathcal{C}}}(\boldsymbol{x})$.

Choosing the step size

The step size δ occuring in all steps of our optimization steps is essential for convergence of the overall algorithm. The choice of it is a well-discussed topic in convex optimization and in most cases, δ is chosen by one of the following criteria:

- The step size is constant.
- The step size is chosen by a *line search* [18] in each iteration.
- The step size is constant but a momentum [19] is multiplied for acceleration.

For simplicity, we use a constant step size. However, we emphasize that accelerations like Nesterov's momentum method [20] may be used additionally.
In our case, we set $\delta = \frac{1}{L}$ where L is a Lipschitz constant of the gradient of the function f, i.e., a constant L such that

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| \leq L \|\boldsymbol{x} - \boldsymbol{y}\|.$$

For our function f, we know that it is a linear function and thus Lipschitz continuous. Thus, such a constant exists. Even better, we can give an estimate of this constant via

$$\begin{aligned}\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\| &= \|\boldsymbol{\mathcal{F}}^T\boldsymbol{\mathcal{F}}\boldsymbol{x} + \lambda(1-\mu)\boldsymbol{x} - \boldsymbol{\mathcal{F}}^T\boldsymbol{\mathcal{F}}\boldsymbol{y} - \lambda(1-\mu)\boldsymbol{y}\| \\ &= \|(\boldsymbol{\mathcal{F}}^T\boldsymbol{\mathcal{F}} + \lambda(1-\mu)\boldsymbol{I})(\boldsymbol{x}-\boldsymbol{y})\| \\ &\leq \|\boldsymbol{\mathcal{F}}^T\boldsymbol{\mathcal{F}} + \lambda(1-\mu)\boldsymbol{I}\|_2\|\boldsymbol{x}-\boldsymbol{y}\| \\ &= \left(\|\boldsymbol{\mathcal{F}}^T\boldsymbol{\mathcal{F}}\|_2 + \lambda(1-\mu)\|\boldsymbol{I}\|_2\right)\|\boldsymbol{x}-\boldsymbol{y}\| \\ &= \underbrace{\left(\mathrm{Eig}_{\max}(\boldsymbol{\mathcal{F}}^T\boldsymbol{\mathcal{F}}) + \lambda(1-\mu)\right)}_{=:L}\|\boldsymbol{x}-\boldsymbol{y}\|.\end{aligned} \tag{8.8}$$

Thus, we obtain

$$L = \sigma_{\max}^2(\boldsymbol{\mathcal{F}}) + \lambda(1-\mu), \tag{8.9}$$

where $\sigma_{\max}(\cdot)$ is the maximum singular value of the argument matrix. The constant L is further positive since $\boldsymbol{A}^T\boldsymbol{A}$ is positive semi-definite for every matrix $\boldsymbol{A}$ and by choice of the admissible values for the regularization parameters $\lambda > 0, 0 < \mu < 1$ holds.
Our overall method will be guaranteed to converge for step sizes $0 < \delta < \frac{2}{L}$, and we choose it to have a constant value of $\delta = \frac{1}{L}$.

Overall algorithm

Given our problem rewritten as

$$\min_{\boldsymbol{\beta}} f(\boldsymbol{\beta}) + g(\boldsymbol{\beta}) + \iota_{\mathcal{C}}(\boldsymbol{\beta}), \qquad \iota_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} 0, & \boldsymbol{x} \in \mathcal{C}, \\ +\infty, & \boldsymbol{x} \notin \mathcal{C}, \end{cases}$$

one is tempted to define each iteration of the solving algorithm as *forward-backward-backward* iteration

$$\begin{cases} \boldsymbol{x}^{(0)} \in \mathbb{R}^d, \\ \boldsymbol{x}^{(k+1)} = \mathcal{P}_{\mathcal{C}}\left(S_{\delta\lambda\mu}(\boldsymbol{x}^{(k)} - \delta\,\nabla f(\boldsymbol{x}^{(k)}))\right). \end{cases} \tag{8.10}$$

Indeed, it can be shown that the iterations in (8.10) converge linearly to a unique fixed point in the convex polytope. A formal proof is given in Corollary B.4. For it to further converge against a minimizer of our target function, the following equivalences must hold. Note that in them, the sum $\nabla f(\boldsymbol{x}^*) + \partial g(\boldsymbol{x}^*)$ corresponds to the Minkowski sum between sets where $\partial g(\boldsymbol{x}^*)$ denotes the subdifferential of the function g at $\boldsymbol{x}^*$. Also, the value $\nabla f(\boldsymbol{x}^*)$ is the only element in the singleton subgradient of the differentiable function f. Furthermore, the expression $(\mathrm{Id} + \delta\partial g)^{-1}$ is a singleton relation because the minimum of proximal operators is unique [16, p.19]. Now, let us assume that Equation (8.11) holds.

$$\mathrm{prox}_{\delta\iota_{\mathcal{C}}+\delta g} = \mathrm{prox}_{\delta\iota_{\mathcal{C}}} \circ \mathrm{prox}_{\delta g} \tag{8.11}$$

Then, we have the following equivalences:

$$\begin{aligned} & 0 \in \nabla f(\boldsymbol{x}^*) + \partial g(\boldsymbol{x}^*) + \partial\iota_{\mathcal{C}}(\boldsymbol{x}^*) \\ \Leftrightarrow\, & 0 \in \delta\nabla f(\boldsymbol{x}^*) - \boldsymbol{x}^* + \boldsymbol{x}^* + \delta\partial g(\boldsymbol{x}^*) + \delta\partial\iota_{\mathcal{C}}(\boldsymbol{x}^*) \\ \Leftrightarrow\, & (\mathrm{Id} - \delta\nabla f)(\boldsymbol{x}^*) \in (\mathrm{Id} + \delta(\partial g + \partial\iota_{\mathcal{C}}))(\boldsymbol{x}^*) \\ \Leftrightarrow\, & \boldsymbol{x}^* = (\mathrm{Id} + \delta(\partial g + \partial\iota_{\mathcal{C}}))^{-1}(\mathrm{Id} - \delta\nabla f)(\boldsymbol{x}^*) \\ \Leftrightarrow\, & \boldsymbol{x}^* = \mathrm{prox}_{\delta\iota_{\mathcal{C}}+\delta g}(\boldsymbol{x}^* - \delta\nabla f(\boldsymbol{x}^*)) \\ \Leftrightarrow\, & \boldsymbol{x}^* = (\mathrm{prox}_{\delta\iota_{\mathcal{C}}} \circ \mathrm{prox}_{\delta g})(\boldsymbol{x}^* - \delta\nabla f(\boldsymbol{x}^*)) \\ \Leftrightarrow\, & \boldsymbol{x}^* = \mathcal{P}_{\mathcal{C}}\left(S_{\delta\lambda\mu}(\boldsymbol{x}^* - \delta\nabla f(\boldsymbol{x}^*))\right). \end{aligned} \tag{8.12}$$

We see that if Equation (8.11) holds true, the proposed iteration is a fixed point iteration if and only if the fixed point attained is a minimizer to the presented error function.
However, the assumption in (8.11) does **not** hold in general.

Example 8.3.2. Let $\mathcal{C} = \{(x,y)^T \in \mathbb{R}^2 \mid y \leq -2x\}$ and let f, g be as above where $\boldsymbol{A} = \boldsymbol{I}, \boldsymbol{b} = (1,3)^T$ and $\lambda = 1$ and $\mu = 1/2$. In this setting we obtain a Lipschitz constant of $L = 1 + \lambda(1-\mu) = 3/2$ and hence for our step size we get $\delta = \frac{1}{L} = 2/3$.
Now plugging everything in we observe that

$$\begin{aligned}\operatorname{prox}_{\iota_{\mathcal{C}}+g}(\boldsymbol{b}) &= \operatorname*{argmin}_{\boldsymbol{z}} \frac{1}{2}\|\boldsymbol{z}-\boldsymbol{b}\|_2^2 + \|\boldsymbol{z}\|_1 + \iota_{\mathcal{C}}(\boldsymbol{z}) \\ &= \operatorname*{argmin}_{\boldsymbol{z}\in\mathcal{C}} \frac{1}{2}\|\boldsymbol{z}-\boldsymbol{b}\|_2^2 + \|\boldsymbol{z}\|_1 \\ &= (-0.4, 0.8)^T,\end{aligned}$$

while on the other hand we have that

$$\operatorname{prox}_{\iota_{\mathcal{C}}}\left(\operatorname{prox}_g(\boldsymbol{b})\right) = \mathcal{P}_{\mathcal{C}}\left(S_{1/3}(\boldsymbol{b})\right) = (-0.9, 1.9)^T.$$

We see that in this case Equation (8.11) does not hold.

On the other hand, there are a number of cases where the naive iteration still converges against the minimizer of the target function. We will briefly discuss these cases.

Proposition 8.3.3. If $\mathcal{C}$ has a nonempty interior and

$$\partial g\left(\operatorname{prox}_{\iota_{\mathcal{C}}}(\boldsymbol{x})\right) \subset \partial g(\boldsymbol{x})$$

holds for all $\boldsymbol{x} \in \mathbb{R}^d$, then the sequence generated by Iteration (8.10) converges against the minimizer of our target function.

Proof. We notice that both $g = \lambda\mu\|\cdot\|_1$ and $\iota_{\mathcal{C}}$ are proper, convex and lower-semicontinuous functions. Let $\operatorname{int}(X)$ denote the interior of a set X. Additionally, denote by $\operatorname{dom} f$ the *effective domain*, i.e., the set of all points where the function f evaluates to a finite value.
With this notation we know that $\operatorname{int}(\operatorname{dom}\iota_{\mathcal{C}}) = \operatorname{int}(\mathcal{C})$ and $\operatorname{int}(\operatorname{dom} g) = \mathbb{R}^d$ and hence that their intersection is nonempty as well. Furthermore, since both functions are convex and $\mathcal{C}$ has a nonempty interior, it holds that $\partial(\iota_{\mathcal{C}} + g) = \partial\iota_{\mathcal{C}} + \partial g$ as shown in [21, Theorem 23.8].
Now using [22, Proposition 2.13], Equation (8.11) holds and thus the equivalences in (8.12) are fulfilled. As a consequence, the sequence generated by Iteration (8.10) converges against the minimizer of our functional. □

Furthermore, if the convex polytope which builds our feasible set is *separable*, then the naive algorithm works too.

Proposition 8.3.4. If $\mathcal{C}$ is separable, i.e.,

$$\mathcal{C} = \bigcap_{k=1}^{d} \{\boldsymbol{x} \in \mathbb{R}^d \mid \langle \boldsymbol{x}, \boldsymbol{e}_k\rangle \in C_k\},$$

where $\boldsymbol{e}_k$ is the k-th standard basis vector of $\mathbb{R}^d$ and $C_k, k = 1, \ldots, d$, are nontrivial closed intervals in $\mathbb{R}$.
Then the iteration in (8.10) converges against the minimizer of our target function.

Proof. The ℓ_1 proximity function g is separable with functions $\varphi_k(\boldsymbol{x}) = \lambda\mu|x_k|$, i.e.,

$$g(\boldsymbol{x}) \overset{\text{def}}{=} \lambda\mu\|\boldsymbol{x}\|_1 = \lambda\mu\sum_{k=1}^{d}|x_k| = \sum_{k=1}^{d}\varphi_k(\boldsymbol{x}).$$

Under the assumption that $\mathcal{C}$ is separable too, [23, Proposition 2.2] implies that Equation (8.11) holds and thus the minimum is attained using the naive fixed point iteration. □

Unfortunately, these conditions are not always fulfilled in our problem. To fix this, we adopt the *nested optimization scheme* as introduced in [23] in Algorithm 15.
In our concrete setting, we choose the following parameters:

- $(\gamma_n)_{n\in\mathbb{N}} = \delta = L^{-1}$ with L according to (8.9)
- $(\lambda_n)_{n\in\mathbb{N}} = 1$
- $\tau = 1$

The resulting iteration is shown in Algorithm 15 in which W denotes the maximum possible value of the grayscale value range $[0, W]$.

Algorithm 15: Iterative solver for Problem (8.4).

Function `FAITH_Training`($\boldsymbol{\mathcal{F}}$, θ_g, λ, μ)

Input : Feature matrix $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{M\times d}$ from seed voxel selection
Input : Global threshold $\theta_g \in \mathbb{R}$
Input : Regularization parameters $\lambda > 0, \mu \in (0, 1)$
Output: Feature weight vector $\boldsymbol{\beta} \in \mathbb{R}^d$

$\boldsymbol{C} \leftarrow \begin{bmatrix} -\boldsymbol{\mathcal{F}} \\ \boldsymbol{\mathcal{F}} \end{bmatrix}$
$\boldsymbol{d} \leftarrow \begin{bmatrix} \theta_g \mathbf{1} \\ (W - \theta_g)\mathbf{1} \end{bmatrix}$
$\delta \leftarrow \left(\sigma^2_{\max}(\boldsymbol{\mathcal{F}}) + \lambda(1-\mu)\right)^{-1}$
$\boldsymbol{\beta} \leftarrow \mathbf{0}$
while $\boldsymbol{\beta}$ *not converged* **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{\beta} - \delta \nabla f(\boldsymbol{\beta})$
  $\boldsymbol{z} \leftarrow 2S_{\delta\lambda\mu}(\boldsymbol{x}) - \boldsymbol{x}$
  while $\boldsymbol{z}$ *not converged* **do**
    $\hat{\boldsymbol{z}} \leftarrow \mathcal{P}_{\mathcal{C}}\left(\frac{1}{2}(\boldsymbol{z} + \boldsymbol{x})\right)$
    $\boldsymbol{z} \leftarrow \boldsymbol{z} + S_{\delta\lambda\mu}(2\hat{\boldsymbol{z}} - \boldsymbol{z}) - \hat{\boldsymbol{z}}$
  $\boldsymbol{\beta} \leftarrow \hat{\boldsymbol{z}}$ `// The last value of` $\hat{\boldsymbol{z}}$
return $\boldsymbol{\beta}$

Firstly, the polytope matrix and offsets are initialized as defined in Section 8.2.3. Also, the step size is computed as in Equation (8.9). The algorithm proceeds by performing a gradient descent step. Next, the nested proximal operators are evaluated by first solving the polytope proximal step, followed by soft-thresholding handling the ℓ_1 proximity. The overall procedure is continued until convergence is reached. We say that the algorithm converged if the solution vector does not change anymore between iterations, i.e., if $\|\boldsymbol{\beta}^{(k+1)} - \boldsymbol{\beta}^{(k)}\|_2 < \varepsilon$ for a small positive ε which was set empirically to 10^{-5}. We like to point out that the zero start vector is sufficient as it is included in every possible polytope in our setting.
Finally, this method converges against a minimizer of our functional.

Theorem 8.3.5.
The sequence generated by Algorithm 15 converges to a solution of Problem (8.4).

As pointed out earlier, the polytope $\mathcal{C}$ is always nonempty. Using that, a detailed proof is given in [23, Proposition 4.2].

Remark 8.3.6. Algorithm 15 uses a nested optimization scheme because after plugging in the definitions we obtain

$$\begin{aligned}\operatorname{prox}_{\iota_{\mathcal{C}}}(\operatorname{prox}_{\delta g}(\boldsymbol{x})) &= \operatorname*{argmin}_{\boldsymbol{y}} \frac{1}{2}\left\|\boldsymbol{y} - \operatorname{prox}_{\delta g}(\boldsymbol{x})\right\|_2^2 + \iota_{\mathcal{C}}(\boldsymbol{y}) \\ &= \operatorname*{argmin}_{\boldsymbol{y}} \frac{1}{2}\left\|\boldsymbol{y} - \left(\operatorname*{argmin}_{\boldsymbol{z}} \frac{1}{2}\|\boldsymbol{z} - \boldsymbol{x}\|_2^2 + \delta g(\boldsymbol{z})\right)\right\|_2^2 + \iota_{\mathcal{C}}(\boldsymbol{y}).\end{aligned}$$

Thus, in the general case we need to solve another optimization problem at each step of solving the surrounding problem. That particular scheme finds the solution to the inner problem using a classical *Douglas-Rachford split* [15] approach

$$\begin{cases}\boldsymbol{x}^{(0)} & \in \mathbb{R}^d, \\ \boldsymbol{y}^{(k+1)} & = \operatorname{prox}_{\delta\iota_{\mathcal{C}}}(\boldsymbol{x}^{(k)}), \\ \boldsymbol{x}^{(k+1)} & = \boldsymbol{x}^{(k)} + \operatorname{prox}_{\delta\|\cdot\|_1}(2\boldsymbol{y}^{(k+1)} - \boldsymbol{x}^{(k)}) - \boldsymbol{y}^{(k+1)}.\end{cases}$$

The result then is inserted in a typical forward-backward iteration.

8.4. Hyperparameter Tuning

To this end, the only remaining unknowns in Algorithm 15 are the regularization hyperparameters $\lambda > 0$ and $\mu \in (0, 1)$. In order to estimate these parameters, we use a *Grid Search* approach, i.e., we choose the best possible values from an a priori defined parameter grid.

The parameter μ controls the balance between sparsity and smoothness and hence takes values in the interval $(0, 1)$. We use the discrete parameter set $P_\mu = \{0.25, 0.33, 0.42, 0.50, 0.58, 0.67, 0.75\}$.

To determine the value range for λ we apply the approach introduced in [24] where the authors showed that the maximum meaningful value is given by $\lambda_{\max}(\mu) = \mu^{-1}\|\boldsymbol{\mathcal{F}}^T\boldsymbol{\Theta}\|_2$. From that, a regularization path is constructed in log-space, that is, a feasible hyperparameter set given the maximum value is defined by

$$P_\lambda = \{\varepsilon\lambda_{\max}(\mu)10^{\frac{k}{N-1}\log_{10}(\varepsilon^{-1})}\}_{k=0}^{k_{\max}-1},$$

for some $\varepsilon > 0$ and a number of $k_{\max} \in \mathbb{N}$ different values. In practice, we use 16 different values which suffice as determined experimentally.

The overall hyperparameter value grid is accumulated by

$$P = P_\lambda \times P_\mu.$$

Next, we search for the best parameter combination, i.e., the tuple (λ, μ) which minimizes the prediction error. The following procedure executes this search.

1. Select λ, μ from P.
2. Perform K-fold cross validation. For each split:
   a) Split the problem in $\boldsymbol{\mathcal{F}}_{\text{train}}, \boldsymbol{\mathcal{F}}_{\text{test}}$ and $\boldsymbol{\Theta}_{\text{train}}, \boldsymbol{\Theta}_{\text{test}}$.
   b) Train the model on $\boldsymbol{\mathcal{F}}_{\text{train}}$ and $\boldsymbol{\Theta}_{\text{train}}$ for the current λ, μ yielding $\hat{\boldsymbol{\beta}}$.
   c) Predict $\hat{\boldsymbol{\Theta}} = \boldsymbol{\mathcal{F}}_{\text{test}}\hat{\boldsymbol{\beta}}$.
   d) Record prediction error $SSR_j = \|\hat{\boldsymbol{\Theta}} - \boldsymbol{\Theta}_{\text{test}}\|_2^2$.

3. Compute average error over folds $SSR_{\lambda,\mu} = \frac{1}{K}\sum_{j=1}^{K} SSR_j$.

The optimal parameter set is found by solving $\operatorname{argmin}_{(\lambda,\mu)\in P} SSR_{\lambda,\mu}$.

8.5. Segmenting Big Volumes

The preceding sections presented in the prerequisites enable us to construct an algorithm for segmenting a volume using our Feature-Adaptive Interactive Thresholding technique.

Algorithm 16: Volume segmentation using FAITH.

Input : A volume V of voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \boldsymbol{\Gamma}_{\boldsymbol{d}}$
Input : The number N_F of features to use
Input : A global threshold θ_g
Input : Voxel selection $\{\boldsymbol{x}_{\boldsymbol{\beta}}\}_{\boldsymbol{\beta}\in J}$ for $J \subset \boldsymbol{\Gamma}_{\boldsymbol{d}}$, $\#J = M$
Output: A segmented volume of voxels

$h \leftarrow$ function computing the best N_F features acc. to Section 4.6

$$\boldsymbol{\mathcal{F}} \leftarrow \begin{bmatrix} h(\boldsymbol{x}_{\boldsymbol{\beta}})^T \\ \vdots \\ h(\boldsymbol{x}_{\boldsymbol{\beta}'})^T \end{bmatrix}, \quad \boldsymbol{\beta}, \ldots, \boldsymbol{\beta}' \in J$$

`// ` λ^*, μ^* `optimal by grid search`
$\boldsymbol{\beta} \leftarrow$ `FAITH_Training(`$\boldsymbol{\mathcal{F}}, \theta_g, \lambda^*, \mu^*$`)`

parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V$ **do**
  $\theta \leftarrow \theta_g + \boldsymbol{\beta}^T\, h(\boldsymbol{x}_{\boldsymbol{\alpha}})$
  $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow T_\theta^{0,1}(\mathrm{v}(\boldsymbol{x}_{\boldsymbol{\alpha}}))$ `// Cf. Definition 2.1.1`

The procedure requires a collection of M seed voxels interactively provided by a user. That collection shall contain voxels belonging to the desired object. For an input number N_F of features to use, we select those which are best for representing environments around the seed voxels, automatically using the feature selection procedure presented in Section 4.6. Subsequently, a matrix $\boldsymbol{\mathcal{F}}$ containing the according feature vectors rowwise is constructed. The following step estimates the optimal feature weights $\boldsymbol{\beta}$ with respect to the global threshold and the computed features. The optimal regularization parameters are selected by a grid search. Finally, for each voxel a feature vector and an associated local threshold is computed. The binarization then assigns to each target voxel either the value one if the current voxel value is bigger than the local threshold, or the value zero otherwise.

8.6. Runtime and Memory Analysis

Similar to previous analyses, we show how FAITH is suited for segmenting big voxel volumes.

8.6.1. Runtime Analysis

Algorithm 16 starts by computing a selection of N_F optimal features given a seed voxel selection. As discussed in Section 4.6, the choice of optimal features has a runtime of order $\mathcal{O}(M)$ where M is the number of seed voxels. Next, for each seed voxel the selected features are evaluated and stored in a feature matrix. Clearly, the runtime of this step is linear in the number of selected voxels, i.e., it is of order $\mathcal{O}(M)$. The next step is the FAITH training procedure. It consists of a

nested training iteration where each loop has a maximum iteration count of 1024 iterations. Let m be the number of iterations necessary for the outer training loop. For each inner optimization step, denote $k_j, j = 1, \ldots, m$, its respective runtime and let $k := \max_{j=1,\ldots,m} k_j$. Thus, the runtime of this part has a complexity of $\mathcal{O}(mk)$. Although this factor will depend on M in some way, we want to note that typically the number of seed voxels is very small compared to the overall number of voxels in the volume. Finally, we apply a binarization technique for each voxel which is linear in the number of voxels.
Overall, the runtime complexity of segmentation using the presented Feature-Adaptive Interactive Thresholding technique is of order $\mathcal{O}(M + mk + N)$.

8.6.2. Memory Requirements

The initial memory requirements of Algorithm 16 are of order $\mathcal{O}(M)$ for both the feature selection and the feature computation. The training procedure itself needs to keep the matrix $\boldsymbol{C}$ and the vector $\boldsymbol{d}$ describing the convex polytope $\mathcal{C}$ in memory, as well as the solution and additional computation vectors which are all of dimensionality d which is the constant number of feature values after feature selection. The following mask-based binarization procedure requires only a small environment around each voxel in memory which is of negligible size.
Hence, the overall memory requirements scale linearly with the number of voxels provided interactively.

9. Support Vector Machine-Based Segmentation

Abstract Thresholding techniques suffer from some shortcomings, most notably that the segmented component must consist of voxels having grayscale values in the upper region of the grayscale value distribution, which is often not the case. This chapter will introduce a more sophisticated segmentation method using *Support Vector Machines* (SVMs) for classifying voxels. The following section describes what Support Vector Machines are and how they work, also listing advantages we gain by using that classifier, followed by a description of how it is used to perform voxelwise segmentation of volumetric data. This is continued by a discussion of how the model hyperparameters are tuned in our case and also how a trained model can be retrained and applied on another dataset. After that, confidence values are introduced. These are representations of how *certain* the system was at classifying a voxel to belong to the desired part. Finally, the overall segmentation procedure is explained and its amortized runtime behavior and memory requirements are analyzed.

9.1. Related Work

Support Vector Machines are well-understood classifiers. As such, they are very often used for segmentation in various forms and in a lot of applications: Segmentation of images or volumes using Support Vector Machines yields about 20 million results on a Google search, of which 600,000 entries are Google Scholar articles alone. Applications are not limited to image processing but cover nearly every possible field of research.
Comprehensive surveys are given in [25, 26].
Specifically in image or volume processing, we can identify two main uses for a SVM:

1. Classification of whole images/volumes.
2. Pixelwise/voxelwise classification.

Very often, the first type is employed. That is, interesting sub-regions of images or volumes are identified and the extracted parts then are classified using SVMs into different categories. Notable applications include Computer Aided Diagnosis systems for identifying cancer and geological evaluations (cf. [27–30]) or identification and labeling of objects found with object detection frameworks (cf. [31–34]).
On the other hand, some approaches aim for voxelwise classification, i.e., for each pixel or voxel in the image or volume, respectively, a trained model assigns a class label. This method is particularly popular in medical applications for segmenting tumors and other anomalies in magnetic resonance images as demonstrated in [35–41] to just name a few.
Nearly all existing segmentation approaches using Support Vector Machines follow a noninteractive approach, i.e., they require manually annotated images/volumes as training data. Clearly, the creation of these labeled datasets is very cumbersome and improvement by retraining is possible only to a limited extent as additional labeled data is required.
Due to this, we propose an interactive approach which utilizes very sparsely annotated training data. We conjecture that given a high demand of volumetric segmentation such interactive tech-

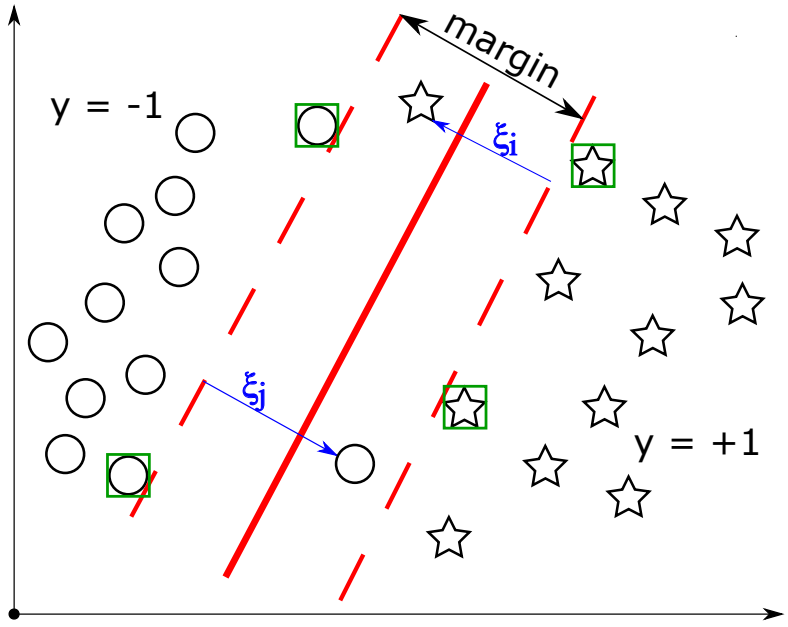


Figure 9.1.: Soft-margin SVM dataset separation.

niques gain importance, although currently only few methods exist. We want to highlight [42] and [43] which segment images and volumes, respectively, interactively, but using only color information. To the best of our knowledge, there currently do not exist approaches exploiting local geometry.

9.2. Support Vector Machines

Before we continue on how we use Support Vector Machines for voxelwise segmentation, we want to briefly recapitulate how they work and how they can be utilized for our purpose.

9.2.1. What are SVMs?

Support Vector Machines are *maximum margin classifiers*, i.e., they aim to find a hyperplane that separates two classes from each other. Denote by $\boldsymbol{x}_i \in \mathbb{R}^d$ a d-dimensional feature vector and by $y_i \in \{\pm 1\}$ its label which paired together form a sample of the training data for $i = 1, \ldots, M$. The datasets that shall be separated are indicated by different labels.
A visualization is given in Figure 9.1.
There, two clusters (circles and stars having labels -1 and $+1$, respectively) are separated by an optimal separating hyperplane (the solid red line) that maximizes the margin while simultaneously minimizing the errors made by misclassifications. The latter are depicted as data items to which a slack term $\xi_i > 0$ is assigned. The support vectors defining the decision function are surrounded by green rectangles.
This optimal separation is expressed as the optimization problem [44]

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i(\boldsymbol{w}^T\,\varphi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i = 1, \ldots, M, \end{aligned}$$

whose derivation stems from a classical paper by Vapnik [45]. It describes a *soft-margin* SVM, i.e., a maximum margin separator allowing for some classification errors in case the dataset is not fully separable. This approach further maps data elements into an often higher-dimensional feature space using a function φ.
However, we actually use a modified version of it, namely the ν-SVM. The according problem is given by [44] as

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \nu\rho + \frac{1}{M}\sum_{i=1}^{M}\xi_i \\ \text{subject to} \quad & y_i\left(\boldsymbol{w}^T\varphi(\boldsymbol{x}_i) + b\right) \geq \rho - \xi_i, \\ & \xi_i \geq 0, \quad \rho \geq 0. \end{aligned} \tag{9.1}$$

The reason we use this version is because the *hyperparameter* $\nu \in (0, 1]$ can be bounded from above by [44]

$$\nu \leq \nu_{\max} := \frac{2}{M}\ \min\{\#y_i = +1, \#y_i = -1\} \leq 1. \tag{9.2}$$

Remark 9.2.1. At this point, we want to remind the reader that a *hyperparameter* is a parameter of an optimization problem which is not trained by the solver for the problem. Instead, it requires manual optimization. On the other hand, a variable which is trained during solving of the problem is called a *parameter*.

The given problem is solved in its dual form

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \alpha_i \leq 1/M, \quad i = 1, \ldots, M, \\ & \mathbf{1}^T\boldsymbol{\alpha} \geq \nu, \quad \boldsymbol{y}^T\boldsymbol{\alpha} = 0, \end{aligned}$$

where $Q_{i,j} = y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j), i, j = 1, \ldots, M$, encodes the *kernelization*, i.e., the lifting of data elements into a high-dimensional feature space using a *kernel* k. In our application, we use a radial basis function kernel of the form

$$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$$

where $\gamma > 0$ is an additional hyperparameter.
Any training procedure applicable thus returns weights $\boldsymbol{\alpha}$ and a bias b, where the latter is calculated from the KKT conditions after calculating the weights. The derived decision function is given in Definition 9.2.2.

Definition 9.2.2. Let $M \in \mathbb{N}$ and $(\boldsymbol{x}_i, y_i), i = 1, \ldots, M$, be some training data. Furthermore, denote by k some kernel function function and let $\alpha_i, i = 1, \ldots, M$, and b be the trained weights as just explained and the trained bias term, respectively.
Define the *SVM decision function* by

$$\begin{aligned} \mathcal{S}^u \colon \mathbb{R}^d &\to \mathbb{R}, \\ \boldsymbol{x} &\mapsto \sum_{i=1}^{M} y_i\alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + b. \end{aligned}$$

Classical SVM classification predicts a label value in $\{\pm 1\}$ as shown in Definition 9.2.3.

Definition 9.2.3. Consider the setting of Definition 9.2.2.
Define the *SVM classification function* by

$$\begin{aligned} \mathcal{S}\colon \mathbb{R}^d &\to \{\pm 1\}, \\ \boldsymbol{x} &\mapsto \operatorname{sgn}\left(\mathcal{S}^u(\boldsymbol{x})\right). \end{aligned}$$

9.2.2. Why SVMs?

Having discussed the basic principle behind Support Vector Machines, the question occurs to why we choose to use a SVM instead of other classifiers.
First of all, we see that a Support Vector Machine only needs its support vectors and their associated weights in order to make a classification decision (as for all other training data elements the weights are zero). Since in practice the number of support vectors is significantly less than the count of training data instances, many computations can be avoided and we obtain a time and memory efficient procedure. Consequently, only few data items are required for an optimal decision boundary and thus often only very sparse image/volume annotations are required. This saves the very time-consuming task of manually annotating whole images or volumes as would be the case for, e.g., deep learning algorithms. Furthermore, the regularization involved makes the system robust against mis-labeled training data items while other techniques, like nearest neighbor classification, typically suffer severely from this. In addition, an advantage over other methods is the *kernelization* which allows for nonlinear transformations in an efficient way without having to design the feature mapping function φ used in Formula (9.1) explicitly. Rather, only the similarity between feature vectors in the feature space must be modeled. We do so by using a radial basis function kernel as we expect our features to group similar voxels together while distinguishing between voxels belonging to different parts, and thus obeying a Gaussian distribution. Another often stated advantage is that Support Vector Machines provide a good out-of-sample generalization. That is, bias in the training dataset can be compensated to a certain degree by appropriately tuning its hyperparameters [46].
On the other hand, Support Vector Machines naturally also have disadvantages, in particular that they assume each voxel to be independent from all others. Hence, they do not consider spatial relations between voxels on their own. Therefore, we overcome this limitation to some extent by considering neighborhoods of voxels as well. Additionally, one of the most cited disadvantages of SVMs is that their trained parameters are difficult to interpret [46, 47], if at all, since they describe a hyperplane in some feature-space defined by the according kernel function. Nevertheless, due to interpretable building blocks in form of the chosen features and kernel, misclassifications are still interpretable.

9.2.3. Voxelwise Classification

Following a description of Support Vector Machines in general, we now describe how we utilize SVMs for voxelwise classification.
As we do not assume that labeled volumes are present, we employ an *interactive* method. Hence, we require a user to mark voxels through some method and assign a label to it. Specifically in our setting, we choose simple mouse clicking where one mouse button corresponds to the *positive* label, i.e., that the selected voxel belongs to the part that shall be segmented. On the other hand, another button signals that the *negative* label shall be assigned to the selected voxel, which is interpreted as a voxel **not** belonging to the desired part. In this way, we allow for very sparse annotations while simultaneously encoding domain-specific knowledge in the labeling process. Other mechanisms, e.g., paint brush tools, are possible as well, but minimum user effort is best supported by individual voxel clicking. Next, the given initial *seed* voxel selection and

an environment size K, which is also determined by the user ad hoc, are used to extract local environments around them. So, for each seed voxel $\boldsymbol{x}_{\boldsymbol{\alpha}}$ in the collection of seed voxels a local environment of size $K \times K \times K$ centered around it, as defined in Definition 1.4.6, is extracted from the volume. From each environment features are extracted according to some feature selection. The latter might be specifically selected by the user based on *a-priori* knowledge, or automatically selected as described in Section 4.6. This results in one feature vector for each seed voxel. The collection of these vectors then is used for training a ν-SVM with its primal problem given in Problem (9.1). Having a trained system at hand, we use it for voxelwise classification. Thus, for each voxel, we again extract a local region of size $K \times K \times K$ voxels and compute the same features for it. The resulting vector is fed to the machine which classifies the voxel as *positive*, i.e., that it belongs to the desired part, or *negative* otherwise. The final result is a volume of same size where the Support Vector Machine classified each voxel accordingly.

9.3. Confidence Values

Typically, Support Vector Machines yield labels as classification result (± 1 in the binary case). However, such a hard assignment cannot assign ambiguous voxels, i.e., voxels that do not clearly belong to either class, with certainty. We already encountered a similar behavior when discussing K-Means. For the latter, we overcame that difficulty by using a probabilistic approach instead to describe how *likely* the classification of a voxel was. In order to also get such a *soft assignment* of voxels to either the positive or the negative class when using a Support Vector Machine, we use a well-known technique called *Platt scaling* [48]. Basically, Platt proposed fitting a sigmoidal model to the distribution of the decision values emitted by the function $\mathcal{S}^u$ given in Definition 9.2.2. Each value $\mathcal{S}^u(\boldsymbol{x}_i), i = 1, \ldots, M$, represents the distance from each data item to the trained margin. Now, a sigmoidal curve is fitted to these prediction values with the goal to estimate the posterior probability that the voxel belongs to the desired part. Hence, the goal is to fit a curve parameterized with values A, B such that

$$P\left(y_{\boldsymbol{x}} = 1 \mid \mathcal{S}^u(\boldsymbol{x})\right) = \frac{1}{1 + \exp\left(A\, \mathcal{S}^u(\boldsymbol{x}) + B\right)}$$

holds. Typically, the estimation of A and B is performed using a maximum likelihood approach. To improve robustness, estimation procedures often additionally perform cross validation on the dataset and choose the best parameters from it. Having both parameters estimated from the prediction values and labels of the training data, the same formula is applied to predict these probabilities. Thus, instead of a binary decision now each voxel is assigned the posterior probability that given the trained model the current voxel belongs to the component under investigation.

Definition 9.3.1. Given constants $A, B \in \mathbb{R}$ obtained via Platt scaling, redefine the SVM segmentation operator by

$$\mathcal{S}(\boldsymbol{x}) = \frac{1}{1 + \exp\left(A\, \mathcal{S}^u(\boldsymbol{x}) + B\right)}.$$

Since voxel values are discrete values in some interval depending on the data type, we assign to each voxel the computed probability scaled to $[0, 100]$ and rounded to the closest integer. We call these integers *confidence values*. The Support Vector Machine implementation we use internally uses a slightly improved method. Details can be found in [44]. The segmented volume then is better interpretable why a voxel gets selected as the derived values encode how *certain* the system was when making the decision. So, a confidence value of 100 expresses that the model is 100 percent sure that the voxel is part of the object that shall be segmented. Conversely, a value of zero encodes that it shall not be selected at all.

In addition to making the result better interpretable, the (in principle) continuous value range allows for further postprocessing, namely thresholding on confidence values. Therefore, using Platt scaling it is possible to select all voxels which are determined to belong to, e.g., 80 percent to the object we are interested in. All voxels having a confidence value less than the specified threshold will be discarded.

9.4. Hyperparameter Tuning

The ν-SVM model from Equation (9.1) and its associated dual problem depend on the hyperparameter $0 < \nu \leq \nu_{\max}$ where $\nu_{\max}$ was defined in Equation (9.2). This hyperparameter is not trained during the model estimation itself, but must be provided a priori. We define the parameter set for ν to consist of equidistant samples of that range. Specifically, we choose

$$P_\nu = \left\{ \frac{i}{100} \nu_{\max} \mid 10 \leq i \leq 100,\, i \in \mathbb{N} \right\}$$

as set of valid parameter values.
Other hyperparameters include parameters to the kernel function. In our application, we choose a radial basis function kernel with parameter $\gamma > 0$. As is common practice for estimating these, we sample the parameter set in log-space. In our application, we fix the according set to

$$P_\gamma = \{2^g \mid -6 \leq g \leq 6,\, g \in \mathbb{Z}\}\,,$$

where the boundaries for g were found experimentally.
The overall set of admissible hyperparameter values is given by $P = P_\nu \times P_\gamma$.
To estimate the best pair of hyperparameters, we perform a Grid Search using 7-fold cross validation similar to the procedure used in Section 8.4. Denote by $\boldsymbol{X} \times \boldsymbol{y} \subset \mathbb{R}^{M \times d} \times \mathbb{R}^M$ the training data. Note that the hyperparameter tuning takes place before the estimation of the Platt scaling coefficients, hence the SVM predictions used here are label values in $\{\pm 1\}$.

1. Select ν, γ from P.
2. Perform K-fold cross validation. For each split:
   a) Split the problem in $\boldsymbol{X}_{\text{train}}$, $\boldsymbol{X}_{\text{test}}$ and $y_{\text{train}}, y_{\text{test}}$.
   b) Train the model on $\boldsymbol{X}_{\text{train}}$ and y_{train} for the current ν, γ.
   c) Predict $\hat{y}$ from $\boldsymbol{X}_{\text{test}}$.
   d) Record prediction error $SSR_j = \sum_{i=1}^{N} \mathbb{1}(\hat{y}_i \neq y_{\text{test},i})$.
3. Compute average error over folds $SSR_{\nu,\gamma} = \frac{1}{K} \sum_{j=1}^{K} SSR_j$.

The optimal parameter set is found by solving $\operatorname{argmin}_{(\nu,\gamma) \in P} SSR_{\nu,\gamma}$.

9.5. Model Serialization and Iterative Training

Although our approach is highly flexible as it trains a model interactively, it is often preferable to simply apply a pretrained model. When Support Vector Machines are used, this can be accomplished by *serializing* the trained SVM, i.e., writing metadata and the trained support vectors to disk. The produced file can be read later and the loaded model can be directly applied on some dataset. Necessary metadata for reapplying a serialized SVM includes the environment size used for extracting environments around voxels and the feature selection defining what features are extracted from them. Additionally, the trained hyperparameters are serialized too. The standard

serialization provided by the implementation of our classifier simply writes the support vectors to disk. When a persisted model shall be used, its interna are read from disk. Next, the volume will be segmented according to the method described in Section 9.2.3.
While this approach can successfully apply a previously trained model on another voxel volume, the result often does not produce perfect results. Instead, additional training data need to be provided to further improve the segmentation. That new information in combination with the one read from disk then trains a new model which respects both. However, this is not possible using the standard serialization relying solely on the support vectors, for the reason that the trained hyperplane can change significantly. In detail, let $\mathcal{T}_o$ be the complete training dataset of the serialized model and denote by $\mathcal{T}_o^s \subseteq \mathcal{T}_o$ the support vectors of the serialized model. Let $\mathcal{T}_n$ be the additional training data. The optimal hyperplane maximizing the margin then depends on training data instances from the set $\mathcal{T}_o \cup \mathcal{T}_n$. However, since our model is a soft-margin SVM, misclassifications are allowed. Hence, the new optimal separating hyperplane might rely on training data instances which stem from $\mathcal{T}_o \backslash \mathcal{T}_o^s$ to maximize the margin. But the standard serialization prevents this as those instances are not persisted.
Therefore, we instead serialize all **unscaled** training feature vectors. When the model is loaded and new data items are provided interactively, both form a new training dataset which is newly scaled and used for training.

9.6. Segmenting Big Volumes

The final procedure for interactive voxel data segmentation using Support Vector Machines is depicted in Algorithm 17 in which $\mathcal{S}\colon \mathbb{R}^d \to \{0, \dots, 100\}$ denotes the SVM decision function yielding confidence values.
The procedure starts by optionally reading a serialized model and the persisted feature vectors and labels from disk. We consider this first step as a single operation of constant time complexity. Next, the iterative procedure starts where the user interactively selects additional seed voxels and according labels. Around these seed voxels local environments are extracted and feature vectors computed. With all training data combined, the Support Vector Machine is fitted and the volume is segmented. Both steps are repeated until the segmentation result is satisfactory. A final optional step serializes the model to disk so it can be reapplied.

9.7. Active Learning

Active learning is an emerging branch of machine learning which considers incremental data labeling. In detail, a model is trained on limited training information. The trained model then selects a set of yet unlabeled instances which shall be labeled by an oracle, e.g., a human. The additional labeled information now is fed back to the model which is retrained on the overall information. Additional variants of this methodology can be found in [49]. The interesting question remaining is how the model shall select the set of unlabeled samples. [49] lists several *query strategies*, including but not limited to

- *Uncertainty sampling*,
- *Query-By-Committee*, and
- *Estimated error reduction.*

Briefly summarized, uncertainty sampling selects instances which it is "uncertain" about. Query-By-Committee instead trains multiple models with different objectives on the same dataset and

Algorithm 17: Voxelwise segmentation using Support Vector Machines.

Function `SVMSegmentationImpl`(V, f, L, K, F)

Input : Input volume V of voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \boldsymbol{\Gamma}_{\boldsymbol{d}}$
Input : Seed feature vectors $f = \{f_{\boldsymbol{\beta}}\}_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}}$
Input : Associated labels $L = \{\ell_{\boldsymbol{\beta}}\}_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}}$
Input : Environment size $K \in 2\mathbb{N} + 1$
Input : Feature computation function F
Output: A segmented volume

`// Training includes hyperparameter tuning`
Train SVM using feature vectors f and labels L

parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V$ **do**
  $\boldsymbol{r} \leftarrow F(R_K(\boldsymbol{x}_{\boldsymbol{\alpha}}))$ `// Compute features`
  $\widetilde{\mathrm{v}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow \lfloor 100\, \mathcal{S}(\boldsymbol{r}) \rfloor$ `// Scale to discrete range`

Function `SVMSegmentation`(V, K, F)

Input : Input volume V of voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \boldsymbol{\Gamma}_{\boldsymbol{d}}$
Input : Environment size $K \in 2\mathbb{N} + 1$
Input : Feature computation function F
Output: A segmented volume V'

$(f_s, L_s) \leftarrow$ de-serialized feature vectors and labels `// (optional)`
$(f_i, L_i) \leftarrow ((), ())$

repeat
  `// Select seed voxels and labels on a grid` $\boldsymbol{\Gamma}_{\boldsymbol{d}'} \subset \boldsymbol{\Gamma}_{\boldsymbol{d}}$
  $f_i \leftarrow (f_i,\ \{F(R_K(\boldsymbol{x}_{\boldsymbol{\beta}}))\}_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}})$
  $L_i \leftarrow (L_i,\ \{\ell_i\}_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}})$
  $V' \leftarrow$ `SVMSegmentationImpl`$(V, (f_s, f_i), (L_s, L_i), K, F)$
until V' *is satisfactory*

Serialize model `// (optional)`
return V'

makes its decision on samples where the most models disagree with each other. Lastly, estimated error reduction aims to reduce the generalization error. That is, it adds any of the unseen data points to the training set, performs a full training and classification and decides afterwards if the selected sample is relevant.
It is easy to see that Query-By-Commitee, estimated error reduction and related methods have a high computational effort since many models must be trained over and over again until a satisfactory result is achieved. The only model computationally attractive in our setting is uncertainty sampling, which is especially suited for probabilistic classification which we aim to do already. In the field of text processing, [50] propsed an active learning approach to find the optimal SVM hyperplane by actively labeling instances closest to the currently estimated hyperplane. This principle was used in a similar fashion in [51] and [52]. The same idea, yet implemented differently, is used in [53] where the q most ambiguous samples are selected automatically in each iteration for some a priori known number q.
While the mentioned approaches all rely on reducing the space of hyperplanes separating the feature space, an alternative approach based on the probabilistic model using Platt scaling can be implemented. Since we use Platt scaling to derive the confidence values, we can formulate an active learning approach directly on these values. For confidence values as introduced in Definition 9.3.1 the highest uncertainty is indicated by a confidence value of 0.5. Thus, we can rewrite the prediction rule to resemble *uncertainties*.

Definition 9.7.1. Let $\mathcal{S}(\boldsymbol{x})$ be a confidence value according to Definition 9.3.1 and let $\delta \geq 0$. Define the *uncertainty value* $\mathcal{U}(\boldsymbol{x})$ by

$$\mathcal{U}(\boldsymbol{x}) = \exp\left(-\delta \tan\left(\pi \left|\mathcal{S}(\boldsymbol{x}) - \frac{1}{2}\right|\right)\right).$$

The parameter δ controls the locality of the uncertainty values around the critical confidence value 0.5. Hence, the larger δ gets, the more the uncertainty concentrates around a confidence of 50 percent, while lower values consider less uncertain predictions also. This trend is depicted in Figure 9.2. Applying the uncertainty function on our confidence values directly yields a new representation showing which parts of the volume are still ambiguous or less well explained. Section 12.4 describes the modified workflow on an example volume.

9.8. Runtime and Memory Analysis

We conclude the current chapter by a brief analysis of the amortized runtime and memory requirements of Algorithm 17.

9.8.1. Runtime Analysis

We consider the first step of Algorithm 17 as a single I/O operation and thus of constant time complexity. Although the runtime of reading the feature vectors from disk conceptually depends on the number of serialized vectors, the reading operations are fast enough to disregard that fact. The following segmentation procedure first trains an SVM. In many applications, the runtimes are reported to be of quadratic or cubic complexity in the number of samples while being linear in the feature vector dimensionality[1] which somewhat coincides with the solving complexity reported in the description of the implementation we use [44]. A worst-case runtime analysis given in [54] concludes a computational complexity of $\mathcal{O}(M^5 \log M / \varepsilon_M)$ where M is the number of

[1] This heavily depends on the given dataset, these estimations stem from `https://scikit-learn.org/stable/modules/svm.html#complexity`.

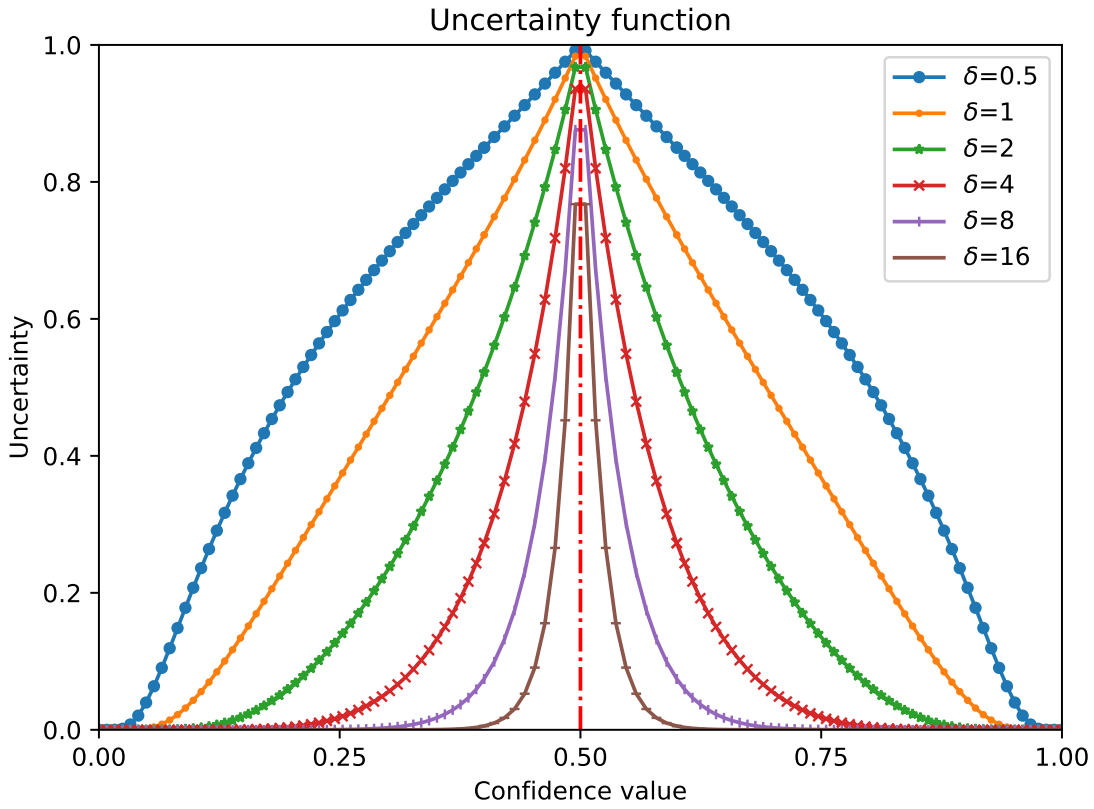


Figure 9.2.: Uncertainty function for several locality parameters.

training data items and ε_M is some value depending on M which is involved in the estimation of the number of iterations. However, [55, p.10] reports that typically solvers are of quadratic or cubic order with respect to the number of seed voxels, depending on the regularization hyper-parameter. Due to a hyperparameter tuning, one can actually expect the runtime to be multiplied by $\#P/N_T$ where P is the set of admissible hyperparameters and N_T is the number of threads that can be used, as all sub-problems during the grid search can be solved independently from each other in parallel. However, we emphasize that the number M describes the quantity of training samples which were selected interactively. Thus, we expect M to be much smaller than the number of voxels N of a volume in practice. After training, the segmentation step is executed where each voxel is classified by the trained machine. Clearly, that step is of linear runtime. The presented instructions are repeated until the result is satisfactory which needs to be decided by the user, effectively multiplying the aggregated complexity by the number of iterations $k \in \mathbb{N}$.
Overall, we conclude that our segmentation procedure based on Support Vector Machines has a complexity of $\mathcal{O}(k \operatorname{poly}(M) N)$ where typically $\deg \operatorname{poly}(M) \leq 5$. Under the assumption that the training data is selected interactively and thus M is much smaller than N, the runtime effectively decreases to $\mathcal{O}(kN)$.
Hence, the proposed algorithm is approximately linear in the number of voxels and depends on the iterations until the result is good enough. Experiments hint that even the number of iterations is often very small to achieve good results, making the algorithm even more suitable for processing big data sets.

9.8.2. Memory Requirements

Regarding the memory requirements, Algorithm 17 needs to keep the training data available at all times, i.e., it needs $\mathcal{O}(Md)$ memory where M is the number of training data items and d is their dimensionality. The segmentation phase only needs a small environment around the current voxel in memory, similar to memory analyses made for other algorithms in this thesis.
Overall, we need to keep the support vectors and the current region in memory, thus effectively having a memory complexity of $\mathcal{O}(Md)$. Therefore, the proposed algorithm is well suited for handling big voxel volumes.

10. Multiresolution Segmentation

Abstract The segmentation algorithms presented thus far are designed to consume as few memory as possible and to be time-efficient in the sense that they effectively have linear runtime complexity with respect to the number of voxels. However, a linear runtime for voxelwise classification still implies that probably costly operations are executed for each individual voxel. Moreover, that action is often redundant since typically large areas of a voxel volume will not be selected as segmented voxels at all. As an example, consider a computed tomography scan of a car, in which some components only appear in few regions, e.g., the springs which only occur near the tires. Hence, most regions of voxels in such a scan will not get selected as being part of the object under investigation anyway. Exploiting this knowledge, the current chapter proposes a general multiresolution segmentation framework which aims to decrease algorithm runtimes significantly with only minor deficiencies in terms of segmentation performance.
First, a brief overview over existing approaches to multiresolution volumetric segmentation is given. Next, our framework is introduced in a general way by explaining algorithmic details to multiresolution segmentation common to all techniques proposed in this thesis. The following section applies the framework to FAITH and our SVM procedure presented in preceding chapters. Finally, the asymptotical runtime behavior and memory consumption are estimated for when a multiresolution approach is applied on top of a voxelwise segmentation method.

10.1. Related Work

The described challenges regarding long runtimes of voxelwise algorithms do occur increasingly as the available data size and measurement capabilities are increasing. Hence, research was already conducted to decrease the runtime using multiresolution methods. Note that we focus on voxelwise segmentation here, i.e., the creation of voxelwise masks and thus do *not* consider previous work on nonvoxelwise methods over multiple resolution levels.
At this point we want to briefly recapitulate recent advances in segmentation with convolutional neural networks. They have proven to provide excellent segmentation results and they provide a multiresolution approach inherently. Such networks also do extract features across multiple scales on their own. A survey about their uses in medical imaging which especially includes segmentation of organs like the prostate is given in [56]. However, we already mentioned the main disadvantage of neural networks in our scenario as they typically require a big training dataset, which is often not available in our use cases.
Closer to our field of applications are multiresolution approaches that perform segmentation using a dedicated classifier trained on explicit features, as they most likely can achieve good performance even with few training data items. Similar to the field of Support Vector Machine-based segmentation, most approaches follow a noninteractive approach, i.e., they use *a priori* annotated training data to train some classifier. Here, we want to specifically cite [57] and [58], respectively, when mentioning a multiresolution segmentation approach in 2D. The first paper describes features across multiple scales using a Wavelet and Curvelet-based analysis, and using a Support Vector Machine on that feature vectors to classify images if they contain breast tumors. Contrary, [58] deals with a more general video segmentation toolchain which acts similar to our approach. However, it segments images using a variant of an Expectation Maximization-based

method on each resolution level and predicts pixel values by a weighted sum of the individual predictions. The hierarchy is implemented by a discrete wavelet transform of an image. They also propose the usage of a three-dimensional wavelet analysis operating on a stack of images in a video which unfortunately scales quadratically with the number of pixels in each image.
When dealing with three-dimensional data which explicitly excludes videos that are actually 2+1-dimensional images, i.e., image stacks, we especially want to cite [59] and [60]. [59] actually implements a 3D segmentation procedure for segmenting prostates from ultrasound images by training three SVMs for each volume axis. For each one, features are extracted using a multiresolution analysis via the discrete wavelet transform. After training, to each voxel three confidence values are assigned and a probabilistic vote decides from that values if the voxel is segmented. On the other hand, Bhalerao et al. [60] proposed a somewhat similar method compared to the one we will introduce in subsequent sections. As such, they construct a Gaussian pyramid and perform a segmentation procedure on the coarest level. However, they perform full classification and then propagate the results from the coarsest down to all resolution levels and adaptively refine segmentation boundaries. Specifically for Support Vector Machine-based segmentation, we further want to highlight [61]. Although dealing with object detection rather than pixelwise segmentation, their work incorporates several resolution levels and processes them simultaneously with a dedicated SVM for each scale in a sliding window approach. The results are then fed into a neural network where each neuron itself is a Support Vector Machine which at its output neuron yields the final classification.
Regarding *interactive* segmentation of voxel volumes in a multiresolution manner, even less work was published to date. The few examples include [62] in which the authors propose an interactive active shape model which extracts features from multiple scales, but processes the data in its original format. To our best knowledge, the method available closest to our approach is [63]. In it, the authors propose both an interactive and a multiresolution approach for segmentation of retinal image data in 3D. However, the biggest difference is that they limit their features to statistical as well as simple gradient-based features and that they compute feature vectors as weighted averages of features over all different scales. Hence, they still compute the features for each voxel on each scale instead of propagating only relevant regions.
The following sections will introduce a new multiresolution segmentation procedure well suited for different kinds of segmentations, be it based on thresholding (cf. FAITH) or on Support Vector Machines. Additionally, the framework introduces *multiresolution volumes* which are views on voxel data interpreting it on coarser resolutions without the need to persist that data in memory or disk. We will also see the benefits by choosing interesting regions on a coarse level and only considering the selected ones on finer scales.

10.2. Multiresolution Volumes

In context of multiresolution segmentation, we need to operate on several resolution levels of a volume. In the current section we will introduce the notion of a *multiresolution volume* as a generalization of volumes as introduced in Definition 1.4.1. Notationally, the following sections will deal with $1 \leq \ell \leq \ell_{\max} \in \mathbb{N}$ resolution levels. In this notation, 1 represents the *coarsest* resolution level while on the other hand $\ell_{\max}$ denotes the *finest* resolution level.

10.2.1. Definition

Before we formally define multiresolution volumes, we first introduce the notion of a *sampling policy*. Such a policy defines how a voxel value on a coarser resolution level is computed from the voxels it covers on a higher resolution. For this, assume that we are given representations $V^{\ell}, V^{\ell'}$

for resolution levels $1 \leq \ell < \ell' \leq \ell_{\max}$ on associated grids $\boldsymbol{\Gamma}_{\boldsymbol{d}}^{1} \subseteq \boldsymbol{\Gamma}_{\boldsymbol{d}}^{\ell} \subset \boldsymbol{\Gamma}_{\boldsymbol{d}}^{\ell'} \subseteq \boldsymbol{\Gamma}_{\boldsymbol{d}}^{\ell_{\max}} = \boldsymbol{\Gamma}_{\boldsymbol{d}}$.
Following the definition a sampling policy, these representations are introduced formally.
In the following, denote by $\mathrm{p}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ the position function attributed to a volume on resolution level ℓ. Hence, for each resolution level ℓ, the function $\mathrm{p}^{\ell} \colon V^{\ell} \to \boldsymbol{\Gamma}_{\boldsymbol{d}}^{\ell}, \boldsymbol{x}_{\boldsymbol{\alpha}} \mapsto \boldsymbol{\alpha}$ maps a voxel of a volume on a certain scale to its position within its volume, i.e., relative to its grid.

Definition 10.2.1. Denote by $\ell_{\max} \in \mathbb{N}$ the number of resolution levels available for some volume V. Let $V^{\ell'}$ be the volume on a resolution level $\ell' \in \{1, \dots, \ell_{\max}\}$ and accordingly consider V^{ℓ} the volume on a coarser level $\ell \in \{1, \dots, \ell'\}$.
The *set of covered voxels* $\mathfrak{S}_{\ell'}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ by a voxel $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V^{\ell}$ is defined by

$$\mathfrak{S}_{\ell'}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \left\{ \boldsymbol{x}_{\boldsymbol{\beta}} \in V^{\ell'} \;\middle|\; \mathrm{p}^{\ell'}(\boldsymbol{x}_{\boldsymbol{\beta}}) = 2^{\ell'-\ell}\, \mathrm{p}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}}) + \boldsymbol{k},\, |\boldsymbol{k}| \leq 2^{\ell'-\ell} - 1,\, \boldsymbol{k} \in \mathbb{N}^3 \right\}.$$

Its cardinality is given by $\#\mathfrak{S}_{\ell'}^{\ell}(\cdot) = 8^{\ell'-\ell}$ voxels.

Having the notion of covered voxels at hand, we now can define what a sampling policy is.

Definition 10.2.2. Let $\boldsymbol{x}$ be any voxel of a volume on a coarse resolution level ℓ and denote by ℓ' a finer resolution level, i.e., $1 \leq \ell \leq \ell' \leq \ell_{\max}$.
A *sampling policy* is a mapping

$$\begin{aligned} \Upsilon_{\ell'}^{\ell} \colon V^{\ell} &\to \mathbb{R}_{\geq 0} \\ \boldsymbol{x}_{\boldsymbol{\alpha}} &\mapsto \mathrm{v}(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \Upsilon\left(\mathfrak{S}_{\ell'}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}})\right), \end{aligned}$$

which computes a voxel value from the set of its covered voxels according to some function Υ.

Conceptually, a sampling policy according to Definition 10.2.2 is intended to compute the value of a voxel $\boldsymbol{x}_{\boldsymbol{\alpha}}$ on a coarser level ℓ from its covered voxels $\mathfrak{S}_{\ell'}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ on a higher resolution ℓ' as introduced in Definition 10.2.1. Note that a sampling policy might also use the position of the voxels, even though we use the voxel values only.
Depending on the application, different sampling policies can be of advantage, e.g., a discrete downsampling which computes the new voxel value by simply choosing the value of a specific voxel of the covered region. However, a discrete downsampling often discards important information. To keep that information available in the downsampled volume, we instead use an averaging strategy as given in Definition 10.2.3.

Definition 10.2.3. Let $\boldsymbol{x}_{\boldsymbol{\alpha}}$ be any voxel of a volume on a coarse resolution level ℓ and let $\mathfrak{S} := \mathfrak{S}_{\ell'}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ be the set of voxels on level ℓ' covered by $\boldsymbol{x}_{\boldsymbol{\alpha}}$.
Then the *averaging sampling policy* computes its value from its covered voxels by calculating their arithmetic mean, i.e.,

$$\Upsilon_{\ell'}^{\ell}(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \Upsilon(\mathfrak{S}) = \frac{1}{\#\mathfrak{S}} \sum_{\boldsymbol{y} \in \mathfrak{S}} \mathrm{v}(\boldsymbol{y})$$

for resolution levels $1 \leq \ell \leq \ell' \leq \ell_{\max}$.

We emphasize again that alternatively many different sampling policies can be used, examples include choosing the maximum of the covered voxel values or applying filters, e.g., Gaussian filtering.
Using sampling policies, we can interpret a volume at different resolution levels and combine these into a *multiresolution volume*.

Definition 10.2.4 (Multiresolution Volume)**.**
Let $(\mathbf{\Gamma}_{\boldsymbol{d}}, f_v, V)$ be a volume according to Definition 1.4.1 for which $\ell_{\max} \in \mathbb{N}$ resolution levels are available. We call this the volume at the finest resolution level.
This induces coarser volumes

$$\left(\mathbf{\Gamma}^{\ell}_{\boldsymbol{d}} = \mathbf{\Gamma}_{\lfloor 2^{\ell-\ell_{\max}}\boldsymbol{d}\rfloor},\ f^{\ell}_v,\ V^{\ell} = \{\boldsymbol{x}_{\boldsymbol{\alpha}} \mid \boldsymbol{\alpha} \in \mathbf{\Gamma}^{\ell}_{\boldsymbol{d}}\}\right), \quad \ell = 1, \dots, \ell_{\max} - 1.$$

The voxel value function f^{ℓ}_v computes its values now based on the specified sampling policy, thus the value function changes to

$$\begin{aligned} \mathrm{v}^{\ell} \colon V^{\ell} &\to \mathbb{R}_{\geq 0} \\ \boldsymbol{x}_{\boldsymbol{\alpha}} &\mapsto f^{\ell}_v(\boldsymbol{\alpha}) := \Upsilon^{\ell}_{\ell_{\max}}(\boldsymbol{x}_{\boldsymbol{\alpha}}), \end{aligned}$$

which uses the original voxel value function in turn.
A *multiresolution volume* for $\ell_{\max}$ resolution levels is the tuple

$$\left(V^1, \dots, V^{\ell_{\max}}\right),$$

encompassing the volume interpretations for all levels.

With the interpretation of a volume on different scales, not only do the voxel values change between resolution levels, but also the dimensions of the volume. Hence, we obtain new iterable Regions of Interest.

Definition 10.2.5. Let $K \in 2\mathbb{N}+1$ be an odd environment size and let $\boldsymbol{d} \in \mathbb{N}^3$ be the dimensions of a given volume on its finest resolution level. Define $\boldsymbol{K}_2 := \lfloor K/2 \rfloor \mathbf{1}$.
Then, the iterable region of the volume at resolution level $\ell = 1, \dots, \ell_{\max}$ is given similar to Definition 1.4.5 by

$$\left(\boldsymbol{K}_2,\ \left\lfloor 2^{\ell-\ell_{\max}}\boldsymbol{d} \right\rfloor - 2\boldsymbol{K}_2\right),$$

where the floor operation is applied componentwise.

A question remaining is what the maximum number of resolution levels for a given volume is. Fortunately, the answer can be given explicitly.

Lemma 10.2.6. Let $K \in 2\mathbb{N} + 1$ be an odd environment size and let $(d_1, d_2, d_3) \in \mathbb{N}^3$ be the dimensions for a given volume on the finest resolution level.
Then the maximum number of resolution levels for that volume is

$$\ell_{\max} = 1 + \left\lfloor \log_2 \frac{d_{\min}}{K} \right\rfloor,$$

where $d_{\min} = \min_{j=1,2,3} d_j$ is the smallest volume axis dimension.

Proof. Define $K_2 := \lfloor K/2 \rfloor$. We immediately see that $2K_2 = K - 1$ holds since K is odd. For the coarsest resolution level $\ell = 1$ to yield a valid volume, the iterable Region of Interest at that level must span at least one voxel. The smallest dimension of the iterable RoI at the coarsest resolution is given in Definition 10.2.5 by $\lfloor 2^{1-\ell_{\max}} d_{\min} \rfloor - 2K_2$.
Hence, we have

$$\begin{aligned} & 1 && \leq 2^{1-\ell_{\max}} d_{\min} - 2K_2 \\ \Leftrightarrow\ & 0 && \leq 2^{1-\ell_{\max}} d_{\min} - K \\ \Leftrightarrow\ & 2^{\ell_{\max}} K && \leq 2 d_{\min} \\ \Leftrightarrow\ & \ell_{\max} && \leq \log_2 \frac{2 d_{\min}}{K}. \end{aligned}$$

Therefore, the largest integral value for this expression is

$$\ell_{\max} = \left\lfloor \log_2 \frac{2d_{\min}}{K} \right\rfloor = 1 + \left\lfloor \log_2 \frac{d_{\min}}{K} \right\rfloor .$$

□

Although this is the largest feasible number of resolution levels, we want to note that in practice one chooses that limit in a way such that the object to be segmented is still recognizable.

10.2.2. Switching Resolution Levels

The presence of multiple views implies that at certain points during the execution of algorithms using multiresolution volumes, the execution switches from one scale to another. Typically, it jumps from a coarse level to the next finer one, as we will discuss in later sections.
In that process, we almost always want to transfer Regions of Interest between scales. The following Lemma ensures that this lifting is well defined by the presented transfer rule.

Lemma 10.2.7. Let $(\boldsymbol{o}, \boldsymbol{d}) \in \mathbb{N}_0^3 \times \mathbb{N}^3$ be an arbitrary valid Region of Interest contained in the iterable RoI of a volume at resolution level $\ell \in \{1, \dots, \ell_{\max} - 1\}$.
Additionally, let $\boldsymbol{K}_2 = \lfloor K/2 \rfloor \, \mathbf{1}$. Define a RoI $(\boldsymbol{o}', \boldsymbol{d}') \in \mathbb{N}_0^3 \times \mathbb{N}^3$ through

$$\begin{aligned} \boldsymbol{o}' &= 2\boldsymbol{o} - \boldsymbol{K}_2, \\ \boldsymbol{d}' &= 2\left(\boldsymbol{d} + \boldsymbol{K}_2\right). \end{aligned}$$

Then that region is contained in the iterable RoI of the volume at resolution level $\ell + 1$.

Proof. In the following, the floor operation is applied on vectors componentwise, as well as the relation operator.
For brevity, let $\boldsymbol{d}_V^{(l+1)} = \lfloor 2^{\ell+1-\ell_{\max}} \boldsymbol{d}_V \rfloor$ be the dimensions of the volume V at resolution level $\ell + 1$ calculated from the dimensions $\boldsymbol{d}_V$ of the volume at the finest resolution level.
Since the given RoI is contained in the iterable Region of Interest at level ℓ by assumption, the constraints

$$\boldsymbol{K}_2 \leq \boldsymbol{o} \qquad \text{and} \qquad \boldsymbol{o} + \boldsymbol{d} \leq \left\lfloor 2^{-1} \, \boldsymbol{d}_V^{(\ell+1)} \right\rfloor - \boldsymbol{K}_2$$

are fulfilled. Then, by applying the definition of the new region we get

$$\boldsymbol{o}' = 2\boldsymbol{o} - \boldsymbol{K}_2 \geq 2\boldsymbol{K}_2 - \boldsymbol{K}_2 = \boldsymbol{K}_2$$

and

$$\boldsymbol{o}' + \boldsymbol{d}' = 2\boldsymbol{o} - \boldsymbol{K}_2 + 2\boldsymbol{d} + 2\boldsymbol{K}_2 = 2\left(\boldsymbol{o} + \boldsymbol{d}\right) + \boldsymbol{K}_2 \leq 2\left\lfloor 2^{-1} \boldsymbol{d}_V^{(\ell+1)} \right\rfloor - \boldsymbol{K}_2 \leq \boldsymbol{d}_V^{(\ell+1)} - \boldsymbol{K}_2$$

componentwise. Thus, the necessary constraints also hold for the next resolution level, i.e., the scaled RoI $(\boldsymbol{o}', \boldsymbol{d}')$ is contained in the iterable RoI of the volume at level $\ell + 1$. □

As a small remark, the previous lemma enables us to omit costly boundary checks in an implementation.

10.2.3. Practical Considerations

Without stepping to deep into actual implementation issues, we still want to make a few remarks about implications of the above formulations for concrete realizations in a given programming language.
The most important thing here is that the volumes are **not** actually downsampled and persisted. Instead, each voxel value at some resolution level is computed ad hoc from the original volume

using a problem-dependent sampling strategy. Thus, only *iterators* to the original data source at a given location must be accessible. However, this gives rise to a small caveat itself: Implementations should *cache* these iterators for efficient access. But one must be careful not to use file-based iterators, where a region is read from disk ad hoc. That is, because when caching all necessary iterators on a resolution level $\ell \in \{1, \ldots, \ell_{\max}\}$, actually $8^{\ell_{\max}-\ell}$ file handles would be opened, which is not supported by many operating systems, e.g., Windows. Therefore, we recommend using a more advanced technique like memory mapped files.
Another minor issue regards the dimensions of a multiresolution volume at its finest level. For computing the coarser representations *on-the-fly*, the sampling policy considers all voxels on a finer level covered by the one on the coarse level. Considering the coarsest scale, each voxel consequently covers actually $2^{\ell_{\max}-1}$ voxels along each volume direction. Hence, if the dimensions of a volume at its finest scale do not match this *box* exactly, some voxels are not considered and should not be considered for computation. Implementations using iterators need to keep track of this behavior for correct iterator movement. To compensate this, we introduce *virtual dimensions* for this case. From a given limit of resolution levels $\ell_{\max}$ and the dimensions of the original voxel data, new dimensions are computed which fit the box description just explained.

Example 10.2.8. Suppose that the original volume had dimensions of $(11, 7, 4)$ and further assume that the limit of resolution levels set is $\ell_{\max} = 3$. This is feasible as it is possible to apply three scales to the minimum dimension in one direction being 4 voxels.
The dimensions to fit the *box* requirement need to be multiples of the minimum dimension 4, thus the virtual dimensions introduced are $(8, 4, 4)$.
Starting from this, the dimensions per scale are updated for all coarser levels.

Lastly, we want to make a statement about index calculations and iterator movement. Depending on the volume dimensions and resolution levels, it occurs frequently that incrementing an iterator on a coarse level, i.e., moving it to the next voxel, does not result in a simple increment of the finer level pointers. Instead, multiple slices, i.e, planes, are often skipped at once, even though on the coarse level the offset is just one voxel. Thus, we need to make additional index calculations on each iterator movement starting from an iterator at some three-dimensional position:

1. Compute a linearized index from the three-dimensional index considering the standard traversal and the virtual dimensions.
2. Increment the one-dimensional index by the specified offset.
3. Calculate the updated three-dimensional position.
4. Update the iterators on different scales to that position.

Although this implies a high amount of operations to be executed at each iterator movement, we observed only a moderate runtime increase induced by them. These steps are visualized in Figure 10.1. There, an iterator movement by one voxel on the coarser level induces a physical increment by 22 voxels on the actual volume.

10.2.4. Relation to Wavelet-Based Approaches

In our literature survey we already encountered the usage of wavelets to construct a multiresolution hierarchy. Conceptually, the multiresolution volume definition presented earlier in this chapter can be endowed with wavelet technology too by choosing a wavelet lowpass filter as sampling policy. In a way, the averaging sampling policy presented in Definition 10.2.3 mimics

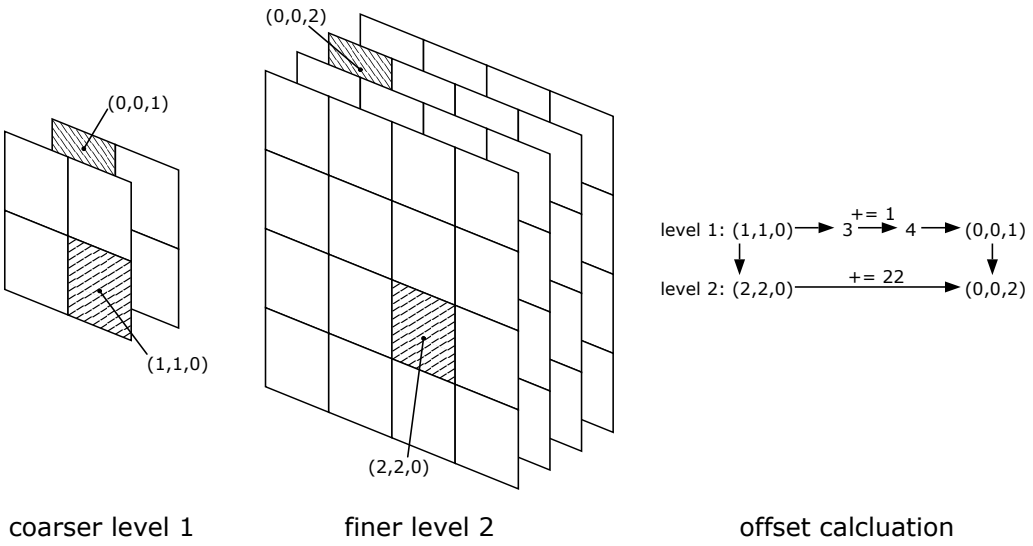


Figure 10.1.: Multiresolution indices increment calculation.

the approximation part of Haar wavelets. Thus, conceptually the consideration of coarser resolution levels is similar to viewing the lowpass wavelet coefficients of a discrete wavelet transform. However, we still like to follow a more general approach. Although a wavelet-based technique possibly could transform larger parts of the volume more efficiently, the approximated wavelet volume views need to be persisted in some way, which is exactly what we try to avoid here. On the other hand, an *on-the-fly* wavelet-based lowpass filtering is probably just as efficient as our simple procedure.

10.3. General Interactive Multiresolution Segmentation

We now combine the notion of multiresolution volumes with voxelwise classificators to define a general algorithm for interactive segmentation in a multiresolution setting.
To accomplish that, we first describe the necessary interfaces to the segmentation model, i.e., a classifier used for determining if a voxel belongs to the part under consideration. When discussing applications in a later section, we will cover how these traits are implemented with regard to our thresholding routine FAITH (cf. Chapter 8) and our Support Vector Machine-based technique proposed in Chapter 9. After that, we are going to describe the framework and segmentation procedure. Then we will also take a look at two algorithmic details concerning aggregation and propagation of interesting sub-regions. The section ends with a description how our supervised segmentation methods we just referred to are included in said framework.

10.3.1. Model Interfaces

To keep things general in this chapter, we require that whatever voxelwise classifier shall be used across multiple scales must obey a certain interface.
Firstly, such a model must provide means to be trained from the interactively selected collection of voxels and the local regions around them of a specified environment size. Given a list of input Regions of Interest, one classifier is trained on it for each resolution level, where the voxel regions are scaled to the according scale. The implemented training interface should also include the (de-)serialization of a model, if required.
Additionally, a model must be applicable on a voxel. In detail, each model must provide some functionality to compute a voxel value from a local environment, which itself may be a confidence

value, e.g., in the Support Vector Machine case. In pseudocode, we denote the applying of a model m on a region of size K around a voxel $\boldsymbol{x}$ by $m(R_K(\boldsymbol{x}))$.
Lastly, each model further specifies a *confidence threshold*. A confidence threshold is a value deciding when the predicted value of a classifier signals that the currently processed voxel does not belong to the desired component and shall be discarded from further analysis. Conversely, if the prediction produces a value bigger than that threshold, it is considered for further analysis and potentially creates a Region of Interest which again is traversed on a finer resolution. Hence, when only certain parts shall be segmented from a big scan, many regions are skipped early, thus increasing the runtime efficiency of our method.

10.3.2. The Framework

Now we proceed with the main topic of this chapter, namely the general interactive multiresolution segmentation framework. Without further ado, the procedure is given in Algorithm 18.
The main procedure has a structure similar to previous segmentation techniques depicted in Algorithms 16 and 17. In detail, we assume a priori information in form of a multiresolution volume according to Definition 10.2.4, an environment size as well as a function being able to compute features from regions and a collection of models, one for each resolution level.
During execution, a collection of interactively selected voxels is accumulated. Depending on the underlying model type, one also collects labels indicating if a voxel belongs to the component under focus. Both the fixed function parameters and the iteratively acquired voxel input are fed to an implementation routine which we will describe shortly and which yields a segmented volume of the same size as the input volume on its finest resolution. The presented steps are repeated until the segmentation result is satisfactory, just as it was introduced in previous algorithms. Given its input, the routine implementing the actual segmentation extracts local environments of size $K \times K \times K$ around each seed voxel on each resolution level. The positions of the seed voxels are scaled appropriately to the different scales. Next, a separate model is trained for each level. The training typically considers the extracted Regions of Interest and extracts features from them. Since each model is likely to encounter a different representation of voxel regions, the feature selection might vary if the latter is determined automatically as explained in Section 4.6. Depending on the application, the feature selection can also be fixed across scales. Next, confidence thresholds are extracted from all but the finest models. These are either pre-determined by the underlying classifier, e.g., for FAITH, or estimated from the training data, e.g., when using our Support Vector Machine-based segmentation method. The following section contains a detailed explanation about them.
Then the actual segmentation procedure starts. Generally, on each but the last level *candidate RoIs* are searched for. Those are sub-regions in the volume on the current scale which are likely to contain voxels which belong to the part that shall be segmented. Their construction is also going to be described in the following section. Thus, first we search for candidate RoIs on the coarsest scale. Since no previous information is given in this case, we traverse the whole volume on that level and construct intersting sub-regions which are identified by the associated model m_1 and its corresponding confidence threshold ρ^1. Having obtained a set of candidates, we proceed on all intermediate levels by considering these RoIs scaled to the next resolution level and searching for candidates in them using the next model.
At the finest resolution level, a set of candidates remain. Now, we make use of the model trained on this scale to classify each voxel in the remaining RoIs and write the result to the according voxel in the target volume.

Algorithm 18: General interactive multiresolution segmentation.

Function `MRSegmentationImpl`$(V, X, y, K, \boldsymbol{F}, \boldsymbol{m})$

Input : Multiresolution volume $V = \left(V^1, \ldots, V^{\ell_{\max}}\right)$
Input : Seed voxel collection $X = (\boldsymbol{x}_{\boldsymbol{\beta}})_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}}$
Input : Optional seed voxel labels $y = (y_{\boldsymbol{\beta}})_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}}$
Input : Environment size $K \in 2\mathbb{N} + 1$
Input : Feature computation functions $\boldsymbol{F} = (F_\ell)_{\ell=1}^{\ell_{\max}}$
Input : Classifiers $\boldsymbol{m} = (m_\ell)_{\ell=1}^{\ell_{\max}}$
Output: A segmented volume

Extract Regions $R_K(\boldsymbol{x}_{\boldsymbol{\beta}})$ with possibly associated labels $y_{\boldsymbol{\beta}}$, $\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}$ on all levels
Train each m_ℓ on these regions using $F_\ell, 1 \leq \ell \leq \ell_{\max}$ on these regions
Get confidence thresholds $\boldsymbol{\rho} = \left(\rho^\ell\right)_{\ell=1}^{\ell_{\max}-1}$ from $\boldsymbol{m}$

```
// Find candidate RoIs on coarse levels
```
$\boldsymbol{C}_0 \leftarrow \{(\boldsymbol{0}, \boldsymbol{d}^1)\}$
for $\ell \leftarrow 1$ **to** $\ell_{\max} - 1$ **do**
  $\boldsymbol{C}_\ell \leftarrow$ `SearchCandidates`$(V^\ell, \boldsymbol{C}_{\ell-1}, m_\ell, \rho^\ell)$

```
// Voxelwise segmentation on finest level
```
for $C \in \boldsymbol{C}_{\ell_{\max}-1}$ **do**
  $V_C \leftarrow$ voxels in RoI C
  parfor $\boldsymbol{x}_{\boldsymbol{\alpha}} \in V_C$ **do**
    $\widetilde{\mathrm{v}}^{\ell_{\max}}(\boldsymbol{x}_{\boldsymbol{\alpha}}) \leftarrow m_{\ell_{\max}}(R_K(\boldsymbol{x}_{\boldsymbol{\alpha}}))$

Function `MRSegmentation`$(V, K, \boldsymbol{F}, \boldsymbol{m})$

Input : Multiresolution volume $V = (V^1, \ldots, V^{\ell_{\max}})$
Input : Environment size $K \in 2\mathbb{N} + 1$
Input : Feature computation function $\boldsymbol{F} = (F_\ell)_{\ell=1}^{\ell_{\max}}$
Input : Classifiers $\boldsymbol{m} = (m_\ell)_{\ell=1}^{\ell_{\max}}$
Output: A segmented volume V'

$X \leftarrow ()$
$y \leftarrow ()$ `// labels (optional)`
repeat
  `// Select seed voxels on a grid` $\boldsymbol{\Gamma}_{\boldsymbol{d}'} \subset \boldsymbol{\Gamma}_{\boldsymbol{d}}^{\ell_{\max}}$
  $X \leftarrow (X, (\boldsymbol{x}_{\boldsymbol{\beta}})_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}})$
  $y \leftarrow (y, (y_{\boldsymbol{\beta}})_{\boldsymbol{\beta} \in \boldsymbol{\Gamma}_{\boldsymbol{d}'}})$ `// (optional)`
  $V' \leftarrow$ `MRSegmentationImpl`$(V, X, y, K, \boldsymbol{F}, \boldsymbol{m})$
until V' *is satisfactory*
return V'

10.3.3. Confidence Thresholds and RoI Fusion

So far we described a general algorithm for interactive and iterative multiresolution segmentation. The current section explains two major parts in Algorithm 18 in detail which have not been covered yet. First, we take a closer look at confidence thresholds and explain what they are used for. Then we show how individual candidate voxels are fused together to form Regions of Interest.

Confidence Thresholds

Confidence thresholds play a major role during multiresolution segmentation in the sense as they describe when a voxel is considered a *candidate voxel*. A voxelwise segmentation method decides for each voxel at some resolution level, if it is part of the object that shall be segmented. When applying our FAITH thresholding procedure, the classification yields either some foreground value if the model decides positively about the current voxel, or a background value otherwise. On the other hand, the Support Vector Machine-based segmentation instead produces confidence values, i.e., discrete values ranging from zero to 100 indicating how likely the voxel belongs to the desired part.
A *confidence threshold* now decides when the classification of a voxel being interesting is high enough such that the voxels spanned on the next finer resolution level shall be considered too.

RoI Fusion

The last missing detail concerns the fusion of candidate voxels into *candidate RoIs*. In order to keep a linear-like runtime per resolution level, we follow a *mark-and-sweep*-like approach, which is a technique sometimes used in the implementation of programming languages.

1. **Mark**: Accumulate interesting candidate voxels.
2. **Sweep**: Fuse the voxels collected thus far into Regions of Interest.

The overall procedure is given in Algorithm 19.
In the second procedure, we search for candidate RoIs on the next finer resolution level. We obtain them by considering a candidate RoI inherited from the previous resolution level, scaled to the current scale according to Lemma 10.2.7. For each voxel inside that region the previously trained model decides whether it is a candidate voxel. These are then fused to Regions of Interest depending on a fusion criterion explained next. After each voxel inside this region was processed, we fuse all remaining candidate voxels into regions too. Then, the resulting RoIs are fused pairwise if they have enough overlap. In particular, they are combined if one contains the other. The minimum overlap percentage is a tunable parameter which was experimentally set to 0.02. Finally, each remaining candidate RoI is scaled to cover the according set of voxels on the next finer resolution level according to Lemma 10.2.7.
The *fusion* method adds a new candidate voxel to some internal collection $\boldsymbol{\Omega}$. Additionally, if the sweep phase sets in, we fuse the accumulated voxels by a simple connectedness criterion.

Definition 10.3.1. A candidate voxel $\boldsymbol{\omega}$ is *connected* to a RoI $C = (\boldsymbol{o}, \boldsymbol{d}) \in \mathbb{N}_0^3 \times \mathbb{N}^3$ if

$$\boldsymbol{o} - \mathbf{1} \leq \mathrm{p}(\boldsymbol{\omega}) \leq \boldsymbol{o} + \boldsymbol{d}$$

holds componentwise.

The accumulated voxels are fused into candidate RoIs whenever the sweep phase sets in, which is periodically invoked. In our algorithm, we make use of a simple heuristic which executes the

Algorithm 19: RoI fusion during multiresolution segmentation.

Function `Fuse`($\boldsymbol{\Omega}$, $\boldsymbol{C}$, *fuseAnyway*)
- **Input :** Candidate voxel collection $\boldsymbol{\Omega}$
- **Input :** Candidate RoI collection $\boldsymbol{C}$
- **Input :** Boolean `fuseAnyway` indicating unconditional fusion
- $N_{\text{sweep}} \leftarrow 2^{20}$
- **if** *fuseAnyway or* $\#\boldsymbol{\Omega} = N_{sweep}$ **then**
  - **foreach** $\boldsymbol{\omega} \in \boldsymbol{\Omega}$ **do**
    - $C_{\boldsymbol{\omega}} \leftarrow (\mathrm{p}(\boldsymbol{\omega}), \mathbf{1})$
    - fused $\leftarrow$ false
    - **foreach** $C \in \boldsymbol{C}$ **do**
      - **if** $C_{\boldsymbol{\omega}}$ *is fusable with* C **then**
        - $C \leftarrow C \cup C_{\boldsymbol{\omega}}$
        - fused $\leftarrow$ true
        - **break**
    - **if** *not fused* **then**
      - $\boldsymbol{C} \leftarrow (\boldsymbol{C}, C_{\boldsymbol{\omega}})$
  - $\boldsymbol{\Omega} \leftarrow ()$

Function `SearchCandidates`(V^ℓ, $\boldsymbol{C}_{\ell-1}$, m_ℓ, ρ^ℓ)
- **Input :** Volume interpretation at level ℓ
- **Input :** *Parent* candidate RoIs $\boldsymbol{C}_{\ell-1}$
- **Input :** Model m_ℓ for level ℓ
- **Input :** Confidence threshold ρ^ℓ at level ℓ
- **Output:** Candidate RoIs $\boldsymbol{C}$
- $\{\boldsymbol{\Omega}_j \leftarrow ()\}_{j=1}^{N_T}$ `// Candidate voxels per thread`
- $\{\boldsymbol{C}_j \leftarrow ()\}_{j=1}^{N_T}$ `// Candidate RoIs per thread`
- **foreach** $C \in \boldsymbol{C}_{\ell-1}$ **do**
  - Consider voxels $\boldsymbol{X}$ of C in V^ℓ
  - **parfor** $\boldsymbol{x} \in \boldsymbol{X}$ **do**
    - **if** $m_\ell(R_K(\boldsymbol{x})) > \rho^\ell$ **then**
      - $t \leftarrow$ index of current thread
      - $\boldsymbol{\Omega}_t \leftarrow (\boldsymbol{\Omega}_t, \boldsymbol{x})$
      - `Fuse`($\boldsymbol{\Omega}_t$, $\boldsymbol{C}_t$, *false*)
- $\boldsymbol{\Omega} \leftarrow \bigcup_{j=1}^{N_T} \boldsymbol{\Omega}_j$
- $\boldsymbol{C} \leftarrow \bigcup_{j=1}^{N_T} \boldsymbol{C}_j$
- `Fuse`($\boldsymbol{\Omega}$, $\boldsymbol{C}$, *true*) `// Fuse remaining candidate voxels`
- Fuse candidate RoIs if enough overlap
- `// Scale candidate RoIs acc. to Lemma 10.2.7`
- **foreach** $(\boldsymbol{o}, \boldsymbol{d}) := C \in \boldsymbol{C}$ **do**
  - $\boldsymbol{o}' \leftarrow 2\boldsymbol{o} - \boldsymbol{K}_2$
  - $\boldsymbol{d}' \leftarrow 2(\boldsymbol{d} + \boldsymbol{K}_2)$
  - $C \leftarrow (\boldsymbol{o}', \boldsymbol{d}')$
- **return** $\boldsymbol{C}$

load #1	search #1	···	···
	load #2	search #2	···

Table 10.1.: Schematic RoI fusion pipeline

fusion process whenever a certain number of voxels, e.g., 2^{20}, were accumulated. If in the fusion process a voxel is connected to an existing Region of Interest, then it is fused into it, making the RoI potentially bigger. If there is no such region, a new one of dimension $\boldsymbol{d} = \mathbf{1}$ is created and appended to the list of RoIs.

Advanced fusion
The proposed fusion scheme is simple and efficient. However, we assume that better strategies improve the result in the sense that less or better fitting RoIs are produced.
Conceptually, we are interested in constructing an error function

$$\mathcal{E}(C, \boldsymbol{\omega}) = |\mathrm{v}(\boldsymbol{\omega}) - \mathbb{E}[C]| + \mathrm{shape}(C, \boldsymbol{\omega}),$$

which measures how much error is introduced to C when the candidate voxel $\boldsymbol{\omega}$ is added to it. The error is composed of the deviation of the grayscale value of $\boldsymbol{\omega}$ from the mean grayscale value in the RoI, and additionally takes the shape distortion denoted by $\mathrm{shape}(C, \boldsymbol{\omega})$ into account.
The best candidate RoI C^* for including the candidate voxel then is the one minimizing that error, i.e.,

$$C^* = \operatorname*{argmin}_{C \in \boldsymbol{C}} \mathcal{E}(C, \boldsymbol{\omega}).$$

In case the voxel should not belong to any of the existing regions, i.e., if $\mathcal{E}(C^*, \boldsymbol{\omega}) > T$ for some threshold T, we create a new one containing just this voxel for now.
Care has to be taken when choosing a shape distortion functional. A natural idea would be to check how well the shape present in the region is extended by the candidate voxel. However, in case that the shape is not aligned along volume axes, e.g., when considering a skewed plane, the shape can possibly be extended very well but the resulting region gets unnecessarily big. One example for a distortion functional is the change of *isotropy*. Consider the current RoI C and the RoI C' enhanced with the candidate voxel $\boldsymbol{\omega}$. From these, compute the isotropy features f_s and f'_s, respectively, according to Definition 4.4.7. Then, the functional is given by $\mathrm{shape}(C, \boldsymbol{\omega}) = |f_s - f'_s|$. Currently, we did not take this error-based fusion into account, but we conjecture that when using a proper shape functional less and better-suited Regions of Interest can be produced.

10.3.4. Practical Considerations

Similar to Section 10.2.3, we want to give a very brief insight into aspects that need to be considered when implementing the fusion procedure. Specifically, we focus on the points of RoI division and a pipeline mechanism to improve the overall efficiency.
By RoI division we simply mean that in case of very large candidate Regions of Interest produced by our method they should be subdivided in multiple regions of smaller size which can be handled in memory. Of course, this is only meaningful when using a pipeline mechanism. The mentioned pipeline mechanism is a common technique which separates I/O actions from/to disk and computations into different threads. Hence, costly file operations can be executed while another thread of execution proceeds with its calculation. Schematically, Table 10.1 depicts how this works. Hence, we load a region. While one thread processes the loaded region, another one already loads the next one, and so on. Such a pipeline is used in our application if and only if we do *not* have random access on the volume, e.g., if we stream the volume directly from file.

10.4. Applications

The proposed framework is designed to be as general and modular as possible, i.e., it allows for arbitrary voxelwise classification algorithms used for segmentation across multiple scales. The only requirements are that these classifiers must implement the interface explained in Section 10.3.1. Now we integrate the voxelwise segmentation procedures FAITH and the Support Vector Machine-based method into our multiresolution scheme, which we will describe in the current section. Hence, we briefly explain how these algorithms implement the interface.

10.4.1. FAITH

We start with the Feature-Adaptive Interactive Thresholding technique. Firstly, it provides a procedure for training a model on each resolution level as explained in Algorithm 15. Applying a model on a local environment around a voxel is further given in Algorithm 16 which computes a local threshold and emits the value one if the voxel value is bigger than this threshold, or zero otherwise.
Concerning the confidence threshold, it should be chosen such that any value emitted bigger than it marks a voxel as a candidate voxel. When using FAITH, only two values are emitted, either zero or one. Clearly, candidates shall be the voxels which get assigned a value of one. Hence, the confidence threshold is zero. If in future work our thresholding procedure is extended to yield confidence values similar to our SVM-based method, one computes a confidence threshold as shown in the following section.

10.4.2. SVM

Similarly, our Support Vector Machine classifier can be trained on seed voxel information and applied to regions around voxels, cf. Algorithm 17. However, this segmentation operator produces confidence values in $[0, 1]$, where implementations emit values in the discrete range $\{0, \dots, 100\}$. Intuitively, we then choose a confidence threshold of 50 percent. In case the dataset is separable in feature space, that value would select all voxels which are somewhat likely to belong to the desired part, whereas it discards all voxels which are unlikely to be interesting.
We refine the stated idea to a more data driven approach. Let $\mathcal{T}^\ell = \mathcal{T}_+^\ell \cup \mathcal{T}_-^\ell$ be the training data given for each resolution level $1 \leq \ell \leq \ell_{\max}$ and denote by $\mathcal{S}^\ell$ the SVM classification operator emitting confidence values as specified in Definition 9.3.1 for level ℓ.
Then, we choose a confidence threshold ρ^ℓ such that the misclassifications are minimized, i.e.,

$$\rho^\ell = \operatorname*{argmin}_{\rho \in [0,1]} \frac{1}{\#\mathcal{T}_+^\ell} \sum_{\boldsymbol{x} \in \mathcal{T}_+^\ell} \mathbb{1}(\mathcal{S}^\ell(\boldsymbol{x}) < \rho) + \frac{1}{\#\mathcal{T}_-^\ell} \sum_{\boldsymbol{x} \in \mathcal{T}_-^\ell} \mathbb{1}(\mathcal{S}^\ell(\boldsymbol{x}) > \rho),$$

where $\mathbb{1}(b)$ evaluates to 1 if b is true, or to 0 otherwise. In practice, we start by a value of 50 percent and find the minimizer by a simple search in both directions. A question remaining is, if there actually exists a minimizer.

Definition 10.4.1. Let $\mathcal{T}$ be the training data and let $y_{\boldsymbol{x}} \in \{\pm 1\}$ be the label associated to a voxel $\boldsymbol{x} \in \mathcal{T}$. Denote by $\mathcal{S}$ the SVM classification function defined in Definition 9.3.1. Define the *probabilistic hinge loss function* by

$$g(\rho) = \frac{1}{\#\mathcal{T}} \sum_{\boldsymbol{x} \in \mathcal{T}} \max\{0, y_{\boldsymbol{x}}(\rho - \mathcal{S}(\boldsymbol{x}))\}, \quad \rho \in [0, 1].$$

The value ρ is called a *confidence threshold*.

First of all, we see that $y_{\boldsymbol{x}}(\rho - \mathcal{S}(\boldsymbol{x}))$ is linear in ρ and thus convex. Since the maximum and sum of convex functions are convex too, we conclude that g is a proper, convex function. As such, there exists a global minimum.
We can describe its shape even further.

Lemma 10.4.2. Let g be the above function and let ρ be some confidence threshold. Then $g \geq 0$ and $g(\rho) = 0$ if and only if the dataset is separable.

Proof. First, we notice that g is trivially nonnegative. Furthermore, we see that

$$g(\rho) = 0 \Leftrightarrow y_{\boldsymbol{x}}(\rho - \mathcal{S}(\boldsymbol{x})) \leq 0 \Leftrightarrow \begin{cases} \mathcal{S}(\boldsymbol{x}) \geq \rho, & y_{\boldsymbol{x}} = +1, \\ \mathcal{S}(\boldsymbol{x}) \leq \rho, & y_{\boldsymbol{x}} = -1, \end{cases} \Leftrightarrow \begin{cases} \mathcal{S}(\boldsymbol{x}) \geq \rho, & \boldsymbol{x} \in \mathcal{T}_+, \\ \mathcal{S}(\boldsymbol{x}) \leq \rho, & \boldsymbol{x} \in \mathcal{T}_-. \end{cases}$$

holds. This implies that g is zero on some interval $[\rho_1, \rho_2] \subset [0, 1]$ if and only if no misclassifications occur on that interval, which is the case if and only if the dataset is separable. □

10.5. Runtime and Memory Analysis

We conclude the chapter by providing an analysis of the asymptotic runtime and memory behavior of a multiresolution segmentation.

10.5.1. Runtime Analysis

Before analyzing the asymptotic runtime of the actual multiresolution segmentation procedure, we notice that we repeat the process until the result is satisfactory, effectively multiplying any runtime by the number $k \in \mathbb{N}$ of iterations. As in previous analyses, that number is typically very small.
In each such iteration, at first local environments around the seed voxels are extracted from the input volume, all happening on multiple resolution levels. Neglecting implementation details, we consider this process to be linear in the number of seed voxels, i.e., to have a runtime complexity of $\mathcal{O}(M)$ per resolution level. Following the mentioned extraction process, one model is trained on each scale. The complexity of the fitting procedure depends on the model, which we will denote by $\mathcal{O}(\mathcal{T})$ which is often polynomial in the number of seed voxels, i.e., $\mathcal{O}(\mathcal{T}) \approx \mathcal{O}(\mathrm{poly}(M))$ where typically $\deg \mathrm{poly}(M) \leq 3$ as in the SVM case. Considering the estimation of confidence thresholds as described in the previous section, we approximate the runtime behavior by being linear in the number of seed voxels. The basis for this is formed by the analysis of the confidence threshold computation for our SVM-based classifier shown in Section 10.4.2, in which we search for the optimal threshold by evaluating an error function for each seed voxel, and for each possible threshold value in the discrete range $\{0, \ldots, 100\}$. Hence, in this case we make at most $100M$ computations which is linear in M. So, the runtime thus far is of order $\mathcal{O}\left(\ell_{\max}(M + \mathcal{T})\right)$.
Starting with the actual segmentation procedure, the first step in the function MRSegmentationImpl in Algorithm 18 needs to traverse the coarsest scale of the given multiresolution volume. With reference to Definition 10.2.4, that step thus is of complexity $\mathcal{O}(2^{3(1-\ell_{\max})}N)$, where $\ell_{\max}$ is the number of resolution levels provided, the number N denotes the count of voxels in the overall volume and the factor 3 stems from the sampling along all three axes. While traversing the volume on its coarsest resolution, candidate RoIs are collected. That accumulation process first remembers a constant number of candidate voxels which is independent of the problem size and fuses them into Regions of Interest depending on a connectivity criterion. The necessary runtime to achieve the fusion depends linearly on the quantity of generated candidate regions and is also linear in the number of candidate voxels, which is however independent of both M and N as argued before. Overall, denoting by N_C^1 the number of generated regions on

the coarsest level during accumulation which clearly depends on the size of the considered sub-volume at the current resolution level, the fusion has a runtime of $\mathcal{O}(N_C^1)$. A more sophisticated fusion procedure might increase the runtime further. Combining both estimates, the candidate RoI collection at the coarsest resolution level is of complexity $\mathcal{O}(2^{3(1-\ell_{\max})}\, N\, N_C^1)$. For each resolution level $1 < \ell < \ell_{\max}$, subsequently inside of each candidate RoI acquired on the previous scale we search for a new set of regions. Hence, each iteration searches for candidates in $N_C^{\ell-1}$ regions and emits a total of N_C^ℓ new candidates. Accumulating all intermediate steps, their runtime is of complexity $\mathcal{O}\left(\sum_{\ell=2}^{\ell_{\max}-1} N_C^{\ell-1} N_C^\ell\right)$. The final segmentation step performs a voxelwise classification on all remaining Regions of Interest, which is linear in the number of voxels inside each region, i.e., of order $\mathcal{O}(\sum_{C\in\boldsymbol{C}_{\ell_{\max}-1}} \#C)$.
We conclude that the multiresolution segmentation procedure as introduced in Algorithm 18 which uses some voxelwise classifier having a training procedure runtime of order $\mathcal{O}(\mathcal{T})$ is of overall runtime complexity

$$\mathcal{O}\left(k\left(\ell_{\max}(M+\mathcal{T}) + \sum_{\ell=1}^{\ell_{\max}-1} N_C^{\ell-1} N_C^\ell + \sum_{C\in\boldsymbol{C}_{\ell_{\max}-1}} \#C\right)\right),$$

where $N_C^0 = 2^{3(1-\ell_{\max})}N$.

Remark 10.5.1. The multiresolution segmentation is intended to be used if relatively small parts shall be found in huge voxel datasets. Under this assumption, the resulting number of candidate RoIs is typically small and the Regions of Interest themselves are small in size too. Then, the runtime effectively decreases to a full scan over the coarsest level followed by a $\log \ell_{\max}$-like runtime due to the tree-like cascading of regions. Regarding the size, we consider the scan of a Ford Fiesta described in the following chapter. The overall scan consists of $3,171,818,496$ voxels. Targeting the segmentation of the four springs located near the tires, the remaining RoIs on the finest level encompass $6,281,221$ voxels in total, which amounts to roughly 0.2 percent of the overall voxel count. In this example, we omitted several very small voxel regions which still are selected. On the other hand, if the desired object encompasses the majority of all voxels in the scan, then we expect a multiresolution approach to be slower since we create Regions of Interest on each scale which cover the same information but increase in size after switching to the next finer resolution level. This worst case then increases the runtime to a polynomial order, thus then the regular segmentation techniques introduced in previous chapters having linear runtime are preferable. In practice the user performing the interactive segmentation shall decide if either the single- or the multiresolution approach should be applied.

10.5.2. Memory Requirements

Regarding the memory consumption, the only thing necessary to be kept in memory at all times are the support vectors for each model which includes at most M seed voxels and, depending on the underlying classifier, also M associated labels. Considering we store a feature vector of dimensionality d for each seed voxel, the data occupies $\mathcal{O}(\ell_{\max}\, M\, d)$ memory. During the search for candidate RoIs, the sub-volumes are traversed in the usual local fashion, thus their memory consumption is negligible. The regions themselves are represented just by their origin and their dimensionality. Thus, although the number of candidate regions can get rather large, but the memory consumption stays low. For the final segmentation step we traverse the sub-volumes built by considering only the remaining Regions of Interest of the volume in the usual manner and hence only need constant memory.
Overall, the memory consumption for a multresolution segmentation is of order $\mathcal{O}(\ell_{\max}\, M\, d)$.

References

[1] A Karthikeyan and M Valliammai. "Lungs Segmentation using Multi-level Thresholding in CT Images". In: *IJECSE* (2012).

[2] Selin Uzelaltinbulat and Buse Ugur. "Lung tumor segmentation algorithm". In: *Procedia Computer Science* 120 (2017), pp. 140–147.

[3] Walter Jentzen et al. "Segmentation of PET Volumes by Iterative Image Thresholding". In: *Journal of nuclear medicine : official publication, Society of Nuclear Medicine* 48 (Feb. 2007), pp. 108–14.

[4] Maliheh Ahmadi et al. "Image segmentation using multilevel thresholding based on modified bird mating optimization". In: *Multimedia Tools and Applications* (Apr. 2019).

[5] A. R Pulagam, V. K. R Ede and R. B. Inampudi. "Segmentation of Airways in Lung Region Using Novel Statistical Thresholding and Morphology Methods". In: *Biomed Pharmacol* (Jan. 2017), pp. 10–14.

[6] S. Anitha and T. R. Ganesh Babu. "An Efficient Method for the Detection of Oblique Fissures from Computed Tomography images of Lungs". In: *Journal of Medical Systems* 43.8 (June 2019), p. 252.

[7] Shengjun Zhou et al. "Segmentation of the hip joint in CT volumes using adaptive thresholding classification and normal direction correction". In: *Journal of the Chinese Institute of Engineers* 36.8 (2013), pp. 1059–1072.

[8] K.J. Batenburg and J. Sijbers. "Adaptive thresholding of tomograms by projection distance minimization". In: *Pattern Recognition* 42.10 (2009), pp. 2297–2305.

[9] Wayne Niblack. *An Introduction to Digital Image Processing*. Birkeroed, Denmark, Denmark: Strandberg Publishing Company, 1985.

[10] J. Sauvola and M. Pietikäinen. "Adaptive Document Image Binarization". In: *Pattern Recognition* 33.2 (2000), pp. 225–236.

[11] Carl-Fredrik Westin et al. "Using Local 3D Sstructure for Segmentation of Bone from Computer Tomography Images". In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. June 1997, pp. 794–800.

[12] C.H. Li and C.K. Lee. "Minimum cross entropy thresholding". In: *Pattern Recognition* 26.4 (1993), pp. 617–625.

[13] Hui Zou and Trevor Hastie. "Regularization and variable selection via the elastic net". In: *Journal of the Royal Statistical Society, Series B* 67 (2005), pp. 301–320.

[14] Quan Zhou et al. "A Reduction of the Elastic Net to Support Vector Machines with an Application to GPU Computing". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, pp. 3210–3216.

[15] Patrick Combettes and Valérie R. Wajs. "Signal Recovery by Proximal Forward-Backward Splitting". In: *Multiscale Modeling & Simulation - MULTISCALE MODEL SIMUL* 4 (Jan. 2005).

[16] Neal Parikh and Stephen Boyd. "Proximal Algorithms". In: *Found. Trends Optim.* 1.3 (Jan. 2014), pp. 127–239.

[17] Gabor T. Herman and Arnold Lent. "Iterative reconstruction algorithms". In: *Computers in Biology and Medicine* 6.4 (1976), pp. 273–294.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004.

[19] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. "Learning Representations by Back-Propagating Errors". In: *Neurocomputing: Foundations of Research*. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699.

[20] Yuri E. Nesterov. "A method of solving a convex programming problem with convergence rate $\mathcal{O}(\frac{1}{k^2})$". In: *Dokl. Akad. Nauk SSSR* 269.3 (July 1983), pp. 543–547.

[21] R.T. Rockafellar. *Convex Analysis*. Princeton Landmarks in Mathematics and Physics. Princeton University Press, 1970.

[22] Samir Adly, Loïc Bourdin and Fabien Caubet. "On a decomposition formula for the proximal operator of the sum of two convex functions". In: *Journal of Convex Analysis* 26 (Jan. 2018).

[23] Caroline Chaux, Jean-Christophe Pesquet and Nelly Pustelnik. "Nested Iterative Algorithms for Convex Constrained Image Recovery Problems". In: *SIAM Journal on Imaging Sciences* 2.2 (2009), pp. 730–762.

[24] Rob Tibshirani, Trevor Hastie and Jerome Friedman. "Regularized Paths for Generalized Linear Models Via Coordinate Descent". In: *Journal of Statistical Software* 33 (Feb. 2010).

[25] Janmenjoy Nayak, Bighnaraj Naik and H Behera. "A Comprehensive Survey on Support Vector Machine in Data Mining Tasks: Applications & Challenges". In: *International Journal of Database Theory and Application* 8.1 (2015), pp. 169–186.

[26] Mayank Chandra and S. Bedi. "Survey on SVM and their application in image classification". In: *International Journal of Information Technology* (Jan. 2018).

[27] K. B. Vaishnavee and K. Amshakala. "An Automated MRI Brain Image Segmentation and Tumor Detection using SOM-Clustering and Proximal Support Vector Machine Classifier". In: *2015 IEEE International Conference on Engineering and Technology (ICETECH)*. 2015, pp. 1–6.

[28] Mingjun Song and Daniel Civco. "Road Extraction Using SVM and Image Segmentation". In: *Photogrammetric Engineering & Remote Sensing* 70.12 (Dec. 2004), pp. 1365–1371.

[29] N. Abdullah, U. K. Ngah and S. A. Aziz. "Image Classification of Brain MRI Using Support Vector Machine". In: *2011 IEEE International Conference on Imaging Systems and Techniques*. 2011, pp. 242–247.

[30] Hari Babu Nandpuru, S. S. Salankar and V. R. Bora. "MRI Brain Cancer Classification Using Support Vector Machine". In: *2014 IEEE Students' Conference on Electrical, Electronics and Computer Science*. 2014, pp. 1–6.

[31] Z. Song et al. "Contextualizing Object Detection and Classification". In: *CVPR 2011*. 2011, pp. 1585–1592.

[32] Hongming Zhang et al. "Object detection using spatial histogram features". In: *Image and Vision Computing* 24.4 (2006), pp. 327–341.

[33] Bernd Heisele et al. "Hierarchical classification and feature reduction for fast face detection with support vector machines". In: *Pattern Recognition* 36.9 (2003), pp. 2007–2017.

[34] Philipp Michel and Rana El Kaliouby. "Real Time Facial Expression Recognition in Video Using Support Vector Machines". In: *Proceedings of the 5th International Conference on Multimodal Interfaces*. Association for Computing Machinery, 2003, pp. 258–264.

[35] J. Zhou et al. "Extraction of Brain Tumor from MR Images Using One-Class Support Vector Machine". In: *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*. 2005, pp. 6411–6414.

[36] Jie Lu et al. “Automatic Liver Segmentation in CT images based on Support Vector Machine”. In: *Proceedings of 2012 IEEE-EMBS International Conference on Biomedical and Health Informatics*. 2012, pp. 333–336.

[37] Zhiqiang Lao et al. “Computer-Assisted Segmentation of White Matter Lesions in 3D MR Images Using Support Vector Machine”. In: *Academic Radiology* 15.3 (2008), pp. 300–313.

[38] Stefan Bauer, Lutz-P. Nolte and Mauricio Reyes. “Fully Automatic Segmentation of Brain Tumor Images Using Support Vector Machine Classification in Combination with Hierarchical Conditional Random Field Regularization”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2011*. Springer Berlin Heidelberg, 2011, pp. 354–361.

[39] Jianguo Zhang et al. “Tumor Segmentation from Magnetic Resonance Imaging by Learning via one-class support vector machine”. In: *International Workshop on Advanced Image Technology (IWAIT ’04)*. Singapore, Singapore, Jan. 2004, pp. 207–211.

[40] E. Ricci and R. Perfetti. “Retinal Blood Vessel Segmentation Using Line Operators and Support Vector Classification”. In: *IEEE Transactions on Medical Imaging* 26.10 (2007), pp. 1357–1365.

[41] S. Ruan et al. “Tumor segmentation from a multispectral MRI images by using Support Vector Machine classification”. In: *2007 4th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. 2007, pp. 1236–1239.

[42] Bérengère Mathieu, Alain Crouzil and Jean-Baptiste Puel. “Interactive multiclass segmentation using superpixel classification”. In: (Oct. 2015). arXiv: 1510.03199.

[43] Xing Zhang et al. “Interactive liver tumor segmentation from ct scans using support vector classification with watershed”. In: *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society* (2011), pp. 6005–6008.

[44] Chih-Chung Chang and Chih-Jen Lin. “LIBSVM: A Library for Support Vector Machines”. In: *ACM Transactions on Intelligent Systems and Technology* 2 (May 2011), 27:1–27:27.

[45] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20 (Sept. 1995), pp. 273–297.

[46] Laura Auria and Rouslan A. Moro. *Support Vector Machines (SVM) as a Technique for Solvency Analysis*. Discussion Papers of DIW Berlin 811. DIW Berlin, German Institute for Economic Research, 2008.

[47] Amr Mohamed. “Comparative Study of Four Supervised Machine Learning Techniques for Classification”. In: *International Journal of Applied Science and Technology* 7.2 (June 2017).

[48] John C. Platt. “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods”. In: *ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, 1999, pp. 61–74.

[49] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009.

[50] Simon Tong and Daphne Koller. “Support Vector Machine Active Learning with Applications to Text Classification”. In: *Journal of Machine Learning Research* 2 (Mar. 2002), pp. 45–66.

[51] Jan Kremer, Kim Steenstrup Pedersen and Christian Igel. “Active Learning with Support Vector Machines”. In: *WIREs Data Mining and Knowledge Discovery* 4.4 (2014), pp. 313–326.

[52] Lei Wang, Kap Luk Chan and Zhihua Zhang. “Bootstrapping SVM active learning by incorporating unlabelled images for image retrieval”. In: *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2003, pp. 629–634.

[53] Giona Matasci, Devis Tuia and Mikhail Kanevski. "SVM-Based Boosting of Active Learning Strategies for Efficient Domain Adaptation". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 5.5 (2012), pp. 1335–1343.

[54] Wan Zhang and Irwin King. "A Study of the Relationship Between Support Vector Machine and Gabriel Graph". In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*. Vol. 1. 2002, pp. 239–244.

[55] Léon Bottou et al. "Support Vector Machine Solvers". In: *Large-Scale Kernel Machines*. 2007, pp. 1–27.

[56] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical Image Analysis* 42 (2017), pp. 60–88.

[57] Yang Wang and Miaomiao Yin. "Multiresolution and Multiscale Geometric Analysis based Breast Cancer Diagnosis using weighted SVM". In: *Proceedings of the 2015 International Conference on Mechanical Science and Engineering*. Atlantis Press, Mar. 2016, pp. 373–378.

[58] Mohammed Abdel-Megeed M. Salem. "Multiresolution Image Segmentation". PhD thesis. Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, 2008.

[59] Hamed Akbari and Baowei Fei. "3D ultrasound image segmentation using wavelet support vector machines". In: *Med Phys* 39.6 (June 2012), pp. 2972–2984.

[60] C. C. Reyes Aldasoro and A. Bhalerao. "Volumetric Texture Segmentation by Discriminant Feature Selection and Multiresolution Classification". In: *IEEE Transactions on Medical Imaging* 26.1 (2007), pp. 1–14.

[61] Jérôme Pasquet et al. "An efficient multi-resolution SVM network approach for object detection in aerial images". In: *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. July 2015, pp. 1–6.

[62] Marleen de Bruijne et al. "Interactive segmentation of abdominal aortic aneurysms in CTA images". In: *Med Image Anal.* 8.2 (June 2004), pp. 127–138.

[63] Alfred R. Fuller et al. "Segmentation of Three-dimensional Retinal Image Data". In: *IEEE Transactions on Visualization and Computer Graphics* 13 (2007).

Part V.

Evaluation and Experiments

But thou hast arranged all things by measure and number and weight.

Holy Bible, Wisdom 11:20

Table of Contents

After introducing several segmentation approaches in the preceding chapters, we continue with an evaluation of these methods on selected datasets.
First, we perform a quantitative evaluation. That is, we execute our Support Vector Machine-based segmentation approach in an *a priori* defined experimental setup mimicking the proposed interactive workflow. For the results we compute several common metrics which will show that our segmentation can achieve a good segmentation performance on the selected datasets. Additionally, we are going to monitor the influence of our multiresolution approach on these metrics. On the other hand, the second chapter in this part evaluates the approaches qualitatively. There, we will see segmentation results from many different concrete application fields which exemplary shows the high flexibility and performance of our techniques. Furthermore, we will see that we obtain good results both on small and big volumes.

11. Quantitative Evaluation

Abstract The current chapter evaluates our proposed Support Vector Machine-based segmentation technique quantitatively. Hence, on selected volumes where ground truth data is available, the interactive workflow is emulated and the resulting segmentation is evaluated according to several performance metrics. First, these metrics and the selected datasets are described. Next, the segmentation performance is measured. Finally, the influence of our multiresolution approach on the segmentation performance is measured in the same way.

11.1. Performance Metrics

On a predefined setup which will be described shortly, we evalute the results according to the following metrics:

- Intersection over Union
- Precision
- Recall
- F_1 score

All of these metrics are computed from the commonly known Type I and Type II errors summarized in Table 11.1 which is often called a *confusion matrix* [1]. In that table the term *actually positive/negative* refers to a voxel belonging (*positive*) or not belonging (*negative*) to the desired object that shall be segmented. On the other hand, *Classification positive/negative* denotes if a voxel is selected by our algorithm to be or not to be part of the segmented volume.

Now, we can define our metrics mentioned above. First, the *Intersection over Union* (IoU, aka *Jaccard Index*) is defined as the rate of all voxels being selected both by the segmentation as well as by the ground truth, i.e., the true positives, related to the union of both volumes [2]. In terms of the errors just described, it is expressed as

$$\text{IoU} = \frac{tp}{tp + fp + fn}.$$

Next, we measure the *precision* and *recall*. The precision considers all voxels which our algorithm classified as relevant and counts how many of these voxels are actually positive according to the ground truth data. The recall instead computes how many of the actually positive voxels defined by the ground truth information our algorithm selected as relevant too. We follow the definition given in [1] and thus we have

$$\text{precision} = \frac{tp}{tp + fp}, \quad \text{recall} = \frac{tp}{tp + fn}.$$

	Actually positive	Actually negative
Classification positive	true positive (tp)	false positive (fp)
Classification negative	false negative (fn)	true negative (tn)

Table 11.1.: A confusion matrix.

Scan	Env. size	Features used	Voxel removing
Ford Fiesta	7	Grayscale + Orientation	$K = 5, \eta = 15$
Piston	5	Grayscale + Orientation	$K = 5, \eta = 18$
Big Piston	5	Grayscale	$K = 5, \eta = 18$
TPA	3	Grayscale + Orientation	$K = 3, \eta = 18$
ME-163, Rumpf	7	Grayscale + Plane fit	-
ME-163, Ring	7	Grayscale + Distance Histogram	-
Tasterwald	7	Grayscale + Local Binary Pattern	-

Table 11.2.: Experimental setup and parametrization.

Finally, the popular F_1 score combines both the precision and recall in a single value as it is the harmonic mean of them, i.e.,

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

For all of these four metrics we note that the higher the values, the better the result is.

Remark 11.1.1. As a small remark, we want to note that we prefer the F_1 score metric over the precision alone, as it includes both the precision and recall in a natural way. In general, the latter indicators should be interpreted together only.

A final statistic we evaluate is the *Receiver Operator Characteristic* (ROC). The ROC is a diagram depicting how well a classifier behaves by plotting the true positive rate over the false positive rate. In our application, we vary a confidence value threshold from zero to 100, extract the necessary component which we will describe in detail in the following section, and compute the metrics on that component. In such a graphic, the optimal classifier performance is a curve which almost instantaneously rises to the top left corner of the diagram and then stays constant [1].

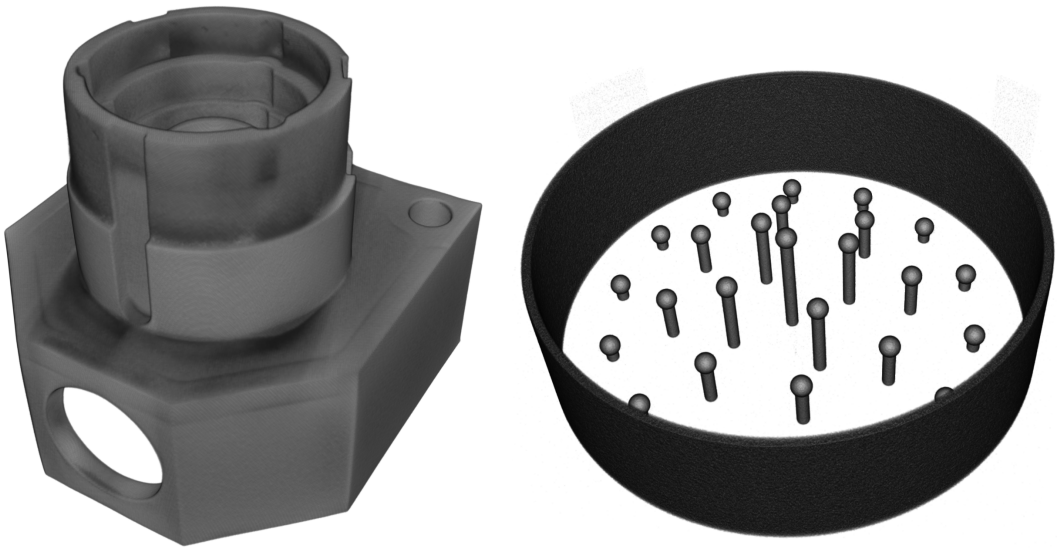
11.2. Experiment Setup

Before we continue with the experimental results, we first want to describe our setup of examples as shown in Table 11.2, and also describe the tested volume datasets and some challenges associated with them.

Remark 11.2.1. In a real world application, the user would typically specify an initial threshold to remove measurement noise. This can speed up the segmentation considerably, but it might also change the evaluation score if relevant voxels are set to zero during thresholding. Therefore, in the quantitative evaluation here, we did not threshold anything and classified each individual voxel of the input volume.

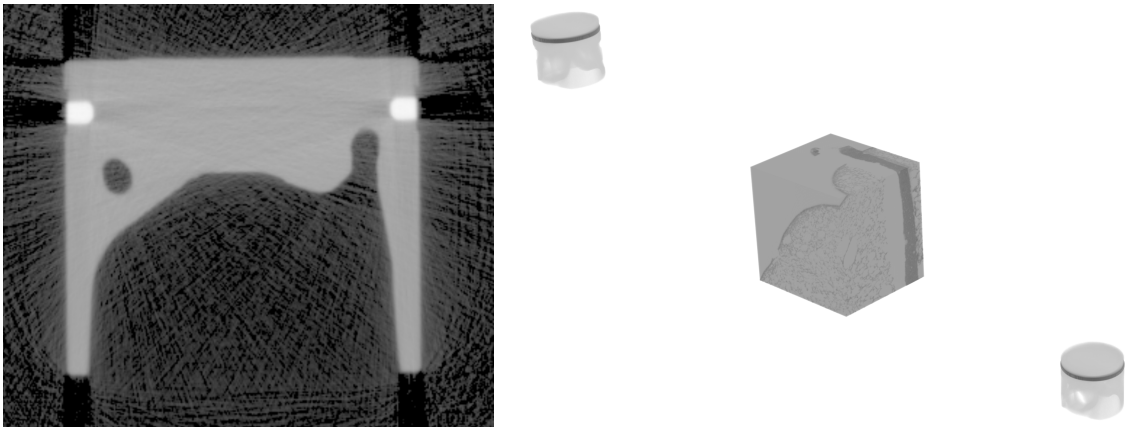
The first volumes we test are the scans “TPA” and “Tasterwald”, both of which are calibrated test volumes whose scans were provided through the Fraunhofer CT in Measurement Technology (CTMT) and are depicted in Figures 11.1a and 11.1b, respectively. Those scans are very simple and typically can be segmented quite well by simple thresholding. The challenge on these two datasets is the measurement noise which decreases the grayscale values on some regions in the volume data to the extent that thresholding away the noisy parts even introduces holes in the thresholding result.

Next, we consider a simple motor piston produced by Mahle whose dataset was created by the Fraunhofer EZRT in Fürth, of which a slice is shown in Figure 11.2a. We artificially created a bigger dataset by inserting two pistons into one scan while we also mixed in a small region of a typical Bavarian meatloaf in a bun, traditionally called “Leberkässemmel”. The grayscale



(a) Voxel dataset “TPA”. (b) Voxel dataset “Tasterwald”.

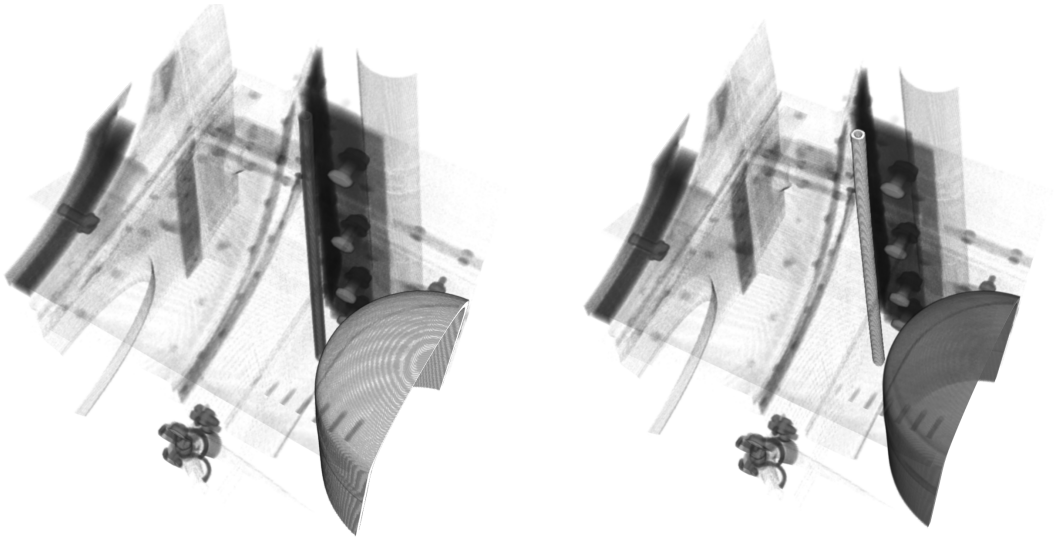
Figure 11.1.: Calibrated test volumes.



(a) Slice of scan “Piston”, contrast enhanced. (b) Voxel dataset “BigPiston”.

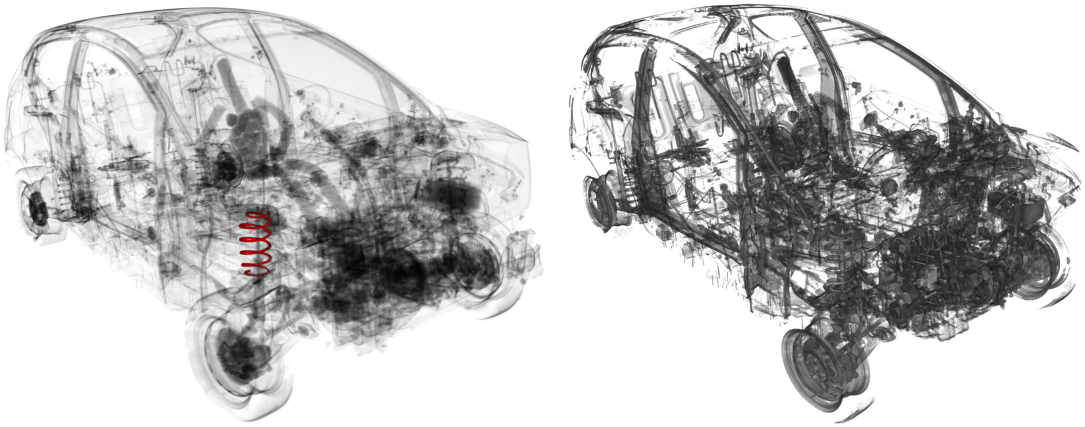
Figure 11.2.: Motor piston scans.

values of the latter were adjusted so the components are not solely distinguishable by their voxel values. We called the result “Big Piston”, which is depicted in Figure 11.2b. For the piston, we want our segmentation to both disregard the measurement noise as well as segmenting the piston without the iron ring in the upper half. As a remark, we want to note that in this particular case a good result can be achieved using both a lower and an upper thresholding technique, yet we still compare to a simple binarizing thresholding method as the latter is arguably the most used segmentation procedure. For the “BigPiston” data we face the same challenge, that scan is used mostly as a benchmark dataset as it is of substantially bigger size, namley increasing from about 174 MB to roughly 11GB. In it, we set the background, i.e., every voxel not occupied by both pistons or the meatloaf in a bun by a constant value of one, so it does not get skipped by our algorithm automatically.
The two following scans listed in our experimental setup both are two components of a Region of Interest from a CT scan of a World War II fighter jet called *Messerschmitt Me 163 Komet* built



(a) “ME-163, Rumpf” [1] (lower right). (b) “ME-163, Ring” [1] (upright tube, middle).

Figure 11.3.: Components of the ME-163.



(a) Original data, one marked spring. (b) Thresholding only.

Figure 11.4.: Voxel dataset “Ford Fiesta”.

in Germany in 1941[1]. The part “Rumpf” is a cut-off of a rounded cap cylindrical part while the “Ring” piece is a small tube inside the aircraft. Both pieces are placed within a $512 \times 512 \times 512$ Region of Interest inside the actual scan. Both elements are depicted in Figures 11.3a and 11.3b, respectively. For the part “Rumpf” we admit that simple thresholding can perfectly segment that element. On the other hand, for the “Ring” piece we need to at least apply both lower and upper thresholding, but we are better off using our Support Vector Machine-based segmentation approach as the following results will show.
We finish our test data description with one of our favorite scans showing an entire car, namely a crashed Ford Fiesta. The scan is roughly six Gigabytes in size and is comprised of many elements of similar material but of different shape. For the quantitative evaluation we want to focus on the four springs of that car located near the tires for our segmentation. An image depicting the scan where one such spring is highlighted is given by Figure 11.4a. The challenge faced here is that simple thresholding will not yield favorable results as shown in Figure 11.4b, but our

[1]Data courtesy of the *Deutsches Museum*.

novel technique will instead. We also like to note that the marked spring in Figure 11.4a was the only one we learned characteristics from, i.e., we marked just a *single* spring in our interactive procedure and we will end up with a segmentation of *all four* springs in subsequent experiments. Table 11.2 lists all quantitatively examined CT scans and also the environment size and the feature set used for testing. The latter two are determined automatically by searching a fixed set of combinations and choosing the one achieving the highest Intersection over Union score.
Just as in our framework described earlier, the searched components are extracted from the result of our Support Vector Machine-based segmentation by further binarizing with a threshold, followed by an optional removal of isolated voxels according to Definition 2.3.2 and finally selecting the components as identified by a Connected Components Analysis. During the search for the optimal environment size and feature set, the binarizing threshold is set to 50 percent, which is the optimal separation threshold in a separable problem setting. The voxel removing however uses the parameters also seen in Table 11.2.
After the optimal K and F parameters are estimated for a scan, the segmentation is invoked another time. From the result, we estimate the Receiver Operator Characteristic curve, i.e., we vary the binarization threshold from zero to 100 inclusive, extract the components from the volume according to the previously described procedure and track their scores. Additionally, we keep track of the best binarizing threshold, i.e., the threshold that produces the highest IoU score. Finally, we compare the result to the score achieved by plain binarizing thresholding and subsequent component extraction. We also track the influence of our multiresolution approach on both the execution times and the scores.

11.3. Results

Regarding the results of our segmentation, we summarize the scores according to the previously discussed metrics in Table 11.3 where Precision is abbreviated as *Prec.* In it, we marked each cell green where that classifier produced better results and red otherwise. On two occations, we marked them as yellow where the comparisons yielded very similar results.
All results were obtained by our purely CPU-based implementation inside an internal framework developed at the Fraunhofer EZRT, applied to the data on a Dell Precision Tower 3620. The latter contains an Intel® Core™ i7-770H processor with eight logical cores with an average clock rate of 4.2 GHz. It further is equipped with 32 GiB of main memory in which all of our tested volumes fit entirely two times (for an input and an output volume), thus we load them directly into memory. We like to repeat at this point that our algorithms can be applied to volumes bigger than main memory. But in these cases consistent and correct runtime measurement is hard to impossible to achieve. Therefore, for measurements we consider cases where the data fits into RAM only.

	SVM				Naive			
Scan	**IoU**	**Prec.**	**Recall**	**F_1**	**IoU**	**Prec.**	**Recall**	**F_1**
Ford Fiesta	0.757	0.970	0.775	0.862	0.011	0.011	0.831	0.022
Piston	0.980	0.980	1.000	0.990	0.675	0.676	0.999	0.806
Big Piston	0.985	0.988	0.997	0.992	0.786	0.787	0.998	0.880
TPA	0.989	0.992	0.997	0.994	0.730	0.730	1.000	0.844
ME-163, Rumpf	0.949	0.963	0.985	0.974	0.971	1.000	0.971	0.985
ME-163, Ring	0.950	0.965	0.984	0.974	0.017	0.017	1.000	0.033
Tasterwald	0.883	0.999	0.884	0.934	0.520	0.641	0.734	0.684

Table 11.3.: SVM segmentation result scores.

Given the datasets from industrial tomography for which we actually had labeled ground truth data accessible we clearly see that our SVM-based segmentation approach is superior on most scans on most metrics, with the exception of the scan "ME-163, Rumpf" which is a single part of the fight jet ME-163 which is easily segmented by thresholding. Nevertheless, the achieved scores are still quite high. Furthermore, we discussed earlier that both precision and recall shall be best interpreted together in form of the F_1 score, regarding which we still are mostly superior. From the presented results we want to explicitly take a look at the "Ford Fiesta" scan as an example where our segmentation procedure yields a significant performance increase. Both methods use a specified segmentation procedure - SVM vs thresholding - followed by the component extraction procedure described in the setup. As both work with the same parametrization, we can indeed assume that the main influence is due to the different segmentation method. Now, in case of naive thresholding, a bigger part of the chassis remains which does not significantly reduce after voxel removing and component extraction as many components are still connected then, cf. Figure 11.4b. On the other hand, our Support Vector Machine-based segmentation technique uses local geometry and structure to distinguish between different regions which discards bigger parts of the chassis and further breaks up the connections between unrelated objects. Thus, the subsequently applied algorithm can indeed isolate individual components.
A similar thing occurs during segmentation of the "ME-163 Ring" scan where the desired tube is connected to several other parts locally, which cannot be resolved totally by binarizing thresholding. For the piston and its bigger version, we want to mention that we only compared against simple binarizing thresholding according to Definition 2.1.2. A thresholding scheme employing both a lower and an upper threshold can produce excellent results here too when combined with our postprocessing chain.
Finally, we also like to mention the single case were our segmentation performs "worse" compared to thresholding, although the achieved scores are still around 95 percent. We believe that this observed minor decrease in performance is due to individual voxels not being selected for segmentation although they should, based on local information around the borders of the object. The far simpler binarization technique applies quite well as the considered part consists of voxels having values in the upper grayscale value range. However, that advantage does no longer apply in general as can be seen on the "Ring" scan where the desired objects stems from the same Region of Interest and where its grayscale values do not belong to the upper voxel value region.
An alternative view on the performance of our Support Vector Machine-based segmentation procedure is by considering their ROC curves, depicted in Figure 11.5. It also shows that our classifier performs very well on most cases while on less successful curves like the on attributed to the "Ford Fiesta" scan still have a steep ascension as it should be although not until the very upper left corner. We believe that the minor *bump* in the ROC of the "Big Piston" scan is due to the sensitivity of these voxelwise performance metrics on individual misclassified voxels.
Overall our methods perform quite well, especially on well behaved data, i.e., data which fit well towards individual geometric features. The following example demonstrates this by explicitly computing the feature vector representation for two different ideal geometries and their difference.

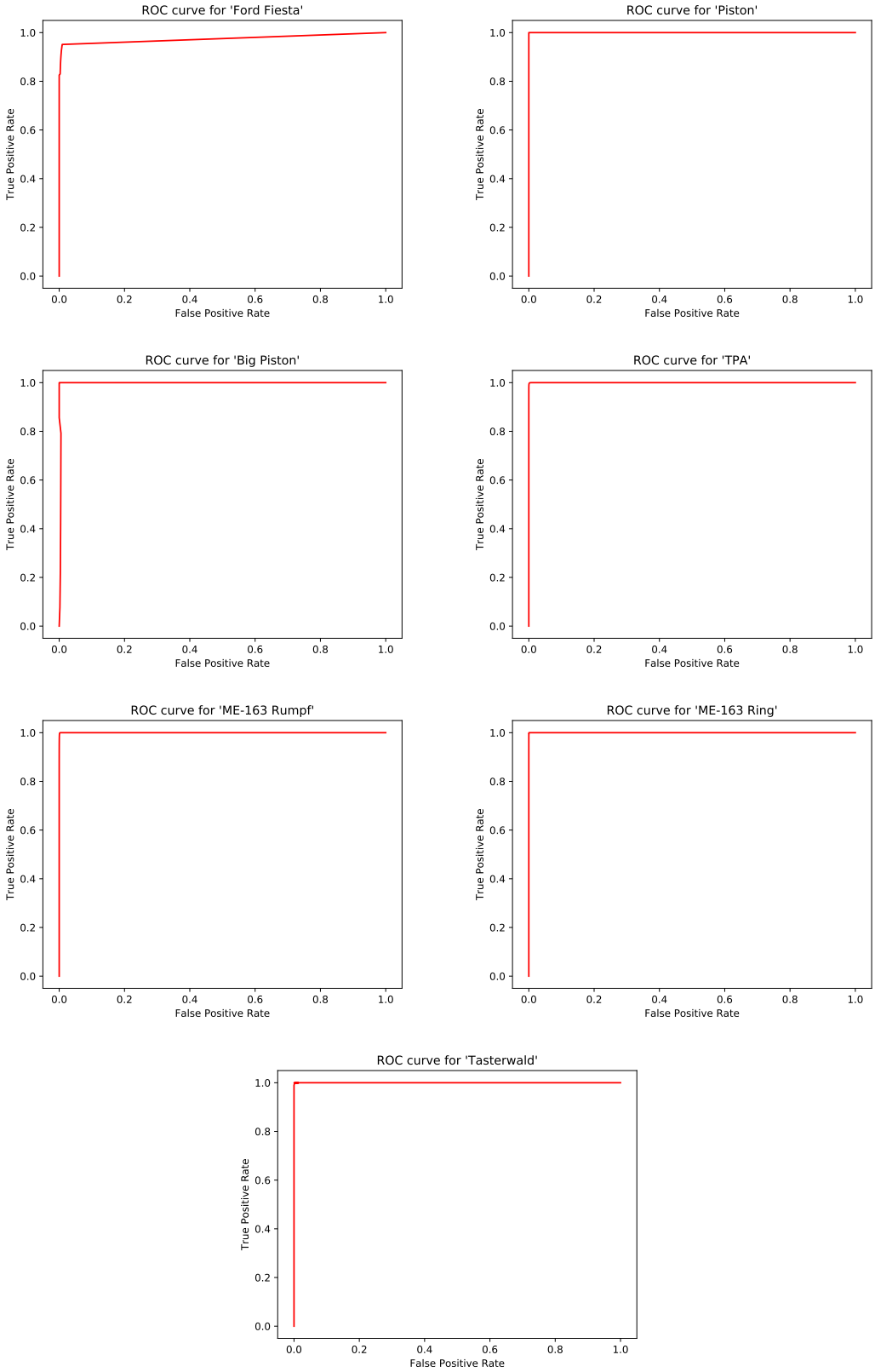


Figure 11.5.: ROC curves of our test cases.

Example 11.3.1. For the sake of this example, consider two voxel regions $V_1 := R_{13}(\boldsymbol{x}_{\boldsymbol{\alpha}})$ and $V_2 := R_{13}(\boldsymbol{x}_{\boldsymbol{\beta}})$ around voxels $\boldsymbol{x}_{\boldsymbol{\alpha}}$ and $\boldsymbol{x}_{\boldsymbol{\beta}}$, respectively. Each region is of dimension $13 \times 13 \times 13$ voxels. Now, let V_1 contain a perfect line having a diameter of one voxel across the center voxel, i.e., $\boldsymbol{x}_{\boldsymbol{\alpha}+(0,j,0)} = v > 0$ for $j = -6, \ldots, 6$. Contrary, assume V_2 contains a small fully filled cube in its center, i.e., $\boldsymbol{x}_{\boldsymbol{\beta}+(i,j,k)} = v > 0$ for $i, j, k = -2, \ldots, 2$. All remaining voxels in these regions shall be zero. Clearly, the two regions are not distinguishable by their voxel values since they are identical.
Now, we compute the linearity feature descriptors f_1^L, f_2^L as introduced in Definition 4.4.2. After scaling, the computed values are

$$\begin{aligned} f_1^L = (\ & 1.00, -1.00, 0.97, -0.06, -0.06, -0.06, -0.06, -0.06, -0.06, -0.06, \\ & -0.06, -0.06, -0.06, -0.06, -0.06, -0.06, -0.06, -0.06, -0.06)^T, \\ f_2^L = (-1.00, \ & 1.00, 5.42, -0.37, -1.21, 0.19, -0.91, -0.19, -0.57, -0.24, \\ & -0.56, -0.16, -0.25, -0.26, -0.20, -0.10, -0.35, -0.01, -0.28)^T. \end{aligned}$$

where we rounded the values to two decimals.
Their distance (cf. Section 3.5) is given by $\|f_1^L - f_2^L\|_2 = 5.55$ which given the scales of the feature values suffices to separate the feature vectors in feature space. Indeed, an experiment using this setting verified this and produced a nearly perfect segmentation. The only misclassified points were the boundary points of the line, as in local regions centered around them the geometry changes to a point-like structure.

Combining both analyses shows that our segmentation algorithm is not only able to operate on big voxel volumes but can also reach a highly precise result. Consequently, our algorithm will also turn out to be useful in future research for creating ground truth data, e.g., for neural networks, with less human overhead.

11.4. Influence of Multiresolution Segmentation

We complete the quantitative analysis by examining the influence of our multiresolution approach to our Support Vector Machine-based segmentation, both in terms of runtime and achieved scores. For each basic configuration listed in Table 11.2, we applied the SVM-based segmentation across $\ell_{\max} = 2, 3, 4$ scales and tracked both their scores and their runtime speedups relative to the single resolution approach. The score development is shown in Table 11.4 where its score values are rounded to three decimals. The same development trends are depicted in Figures 11.6, 11.7, 11.8 and 11.9 for all considered voxel datasets.
Considering the Intersection over Union metric, we notice that bigger and homogeneous objects achieve approximately the same IoU score over multiple resolution levels. On the other hand, when fine structures or especially curved edges are part of the object, cf. springs in a car or the curved surface of the *Rumpf* scan, we observe that the score decreases. Especially in these cases, we observed that geometric details and also the overall shape of the desired parts get distorted considerably during downsampling. Hence, some necessary voxels cannot be identified as candidate voxels and thus are rejected for further classification on subsequent resolution levels.
Concerning the precision, we notice quite some visual variations across scales, but closer inspection of the ordinate suggests that the variation is rather minuscle with the maximum variation being about three percent. Overall, the precision values seem to be rather stable.
Regarding the recall and F_1 score metrics, we observe a quite similar trend as when discussing the IoU metric, although the decreases are smaller.

Scan	$\ell_{\max}$	IoU	Precision	Recall	F_1	Speedup w / wo training
Ford Fiesta	1	0.757	0.970	0.775	0.862	-
	2	0.765	0.964	0.787	0.867	1.76 / 2.79
	3	0.688	0.975	0.700	0.815	1.77 / 4.79
	4	0.660	0.978	0.670	0.796	1.07 / 2.89
Piston	1	0.980	0.980	1.000	0.990	-
	2	0.953	0.953	1.000	0.976	2.47 / 3.62
	3	0.955	0.956	1.000	0.977	2.36 / 5.31
	4	0.954	0.959	0.999	0.976	1.91 / 4.54
Big Piston	1	0.985	0.988	0.997	0.992	-
	2	0.985	0.988	0.997	0.992	3.22 / 3.98
	3	0.983	0.993	0.990	0.992	3.07 / 4.51
	4	0.978	0.990	0.988	0.989	2.29 / 3.48
TPA	1	0.989	0.992	0.997	0.994	-
	2	0.985	0.986	0.998	0.992	1.46 / 1.51
	3	0.990	0.997	0.993	0.995	2.35 / 2.64
	4	0.981	0.983	0.998	0.991	1.40 / 1.52
ME-163, Rumpf	1	0.949	0.963	0.985	0.974	-
	2	0.921	0.971	0.947	0.959	2.87 / 3.16
	3	0.827	0.965	0.852	0.705	4.28 / 5.62
	4	0.668	0.966	0.684	0.801	3.65 / 5.00
ME-163, Ring	1	0.950	0.965	0.984	0.974	-
	2	0.961	0.989	0.972	0.980	3.36 / 3.51
	3	0.905	0.981	0.922	0.950	10.13 / 13.16
	4	0.845	0.973	0.865	0.916	8.01 / 10.70
Tasterwald	1	0.883	0.999	0.884	0.934	-
	2	0.847	0.999	0.847	0.917	3.79 / 3.84
	3	0.822	0.997	0.723	0.902	4.12 / 4.23
	4	0.671	0.999	0.671	0.803	20.56 / 24.84

Table 11.4.: Multiresolution score development and speedups.

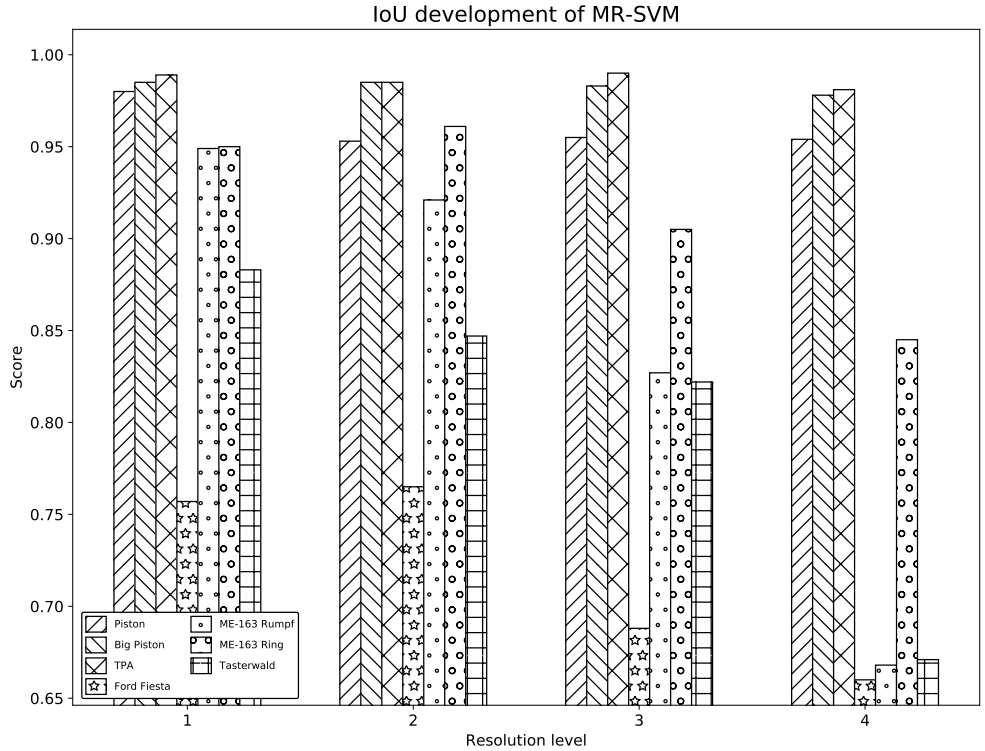


Figure 11.6.: Multiresolution IoU development graph.

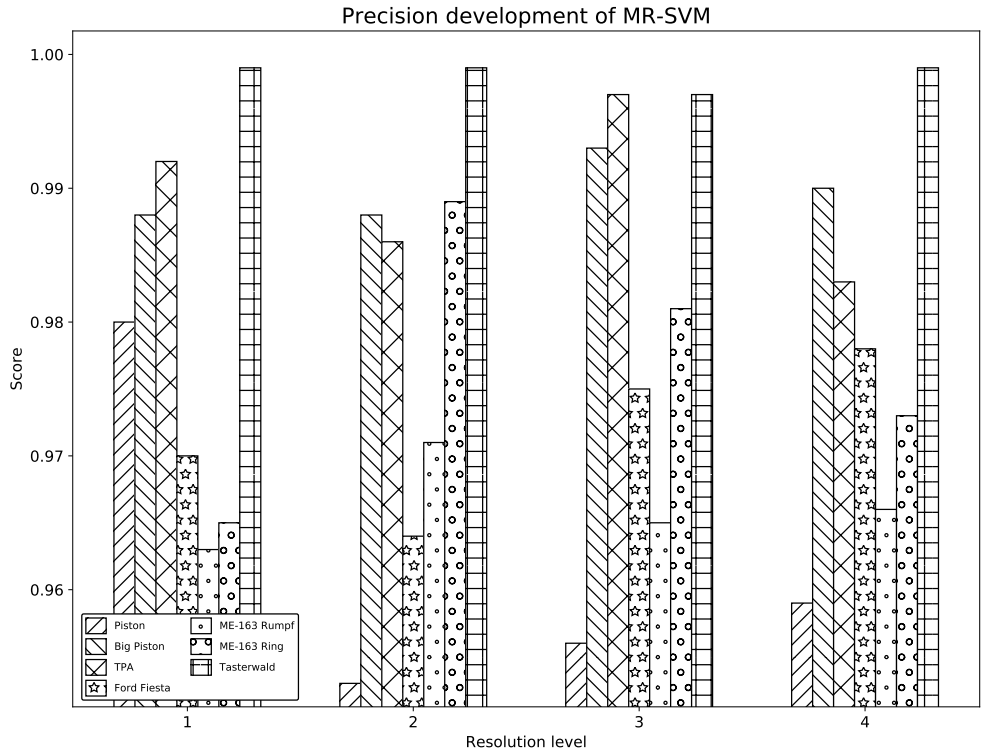


Figure 11.7.: Multiresolution Precision development graph.

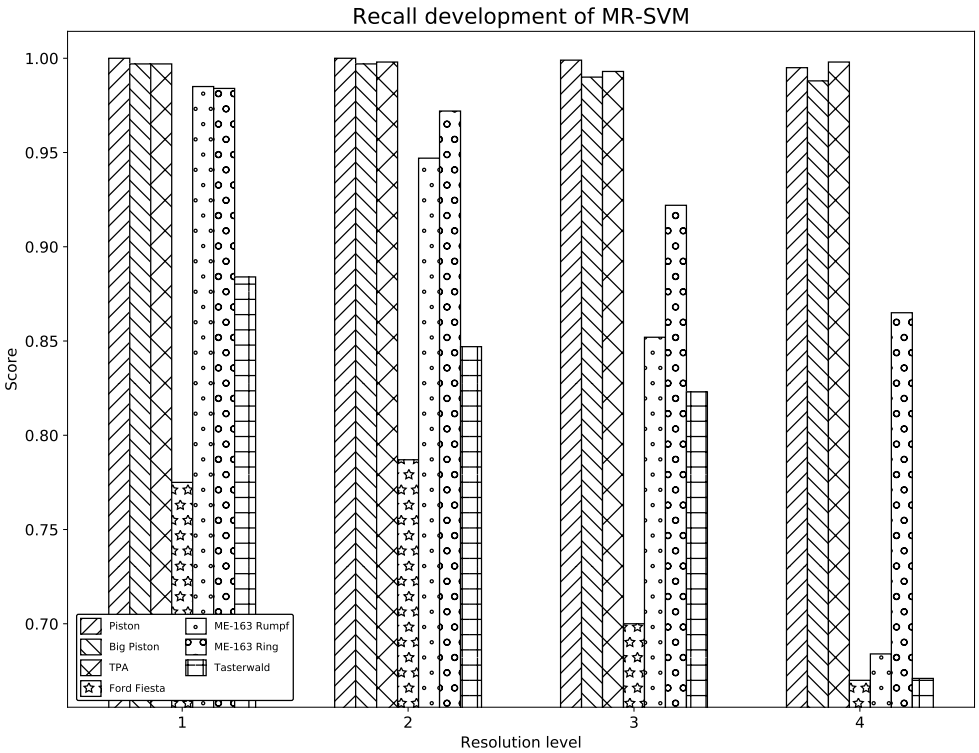


Figure 11.8.: Multiresolution Recall development graph.

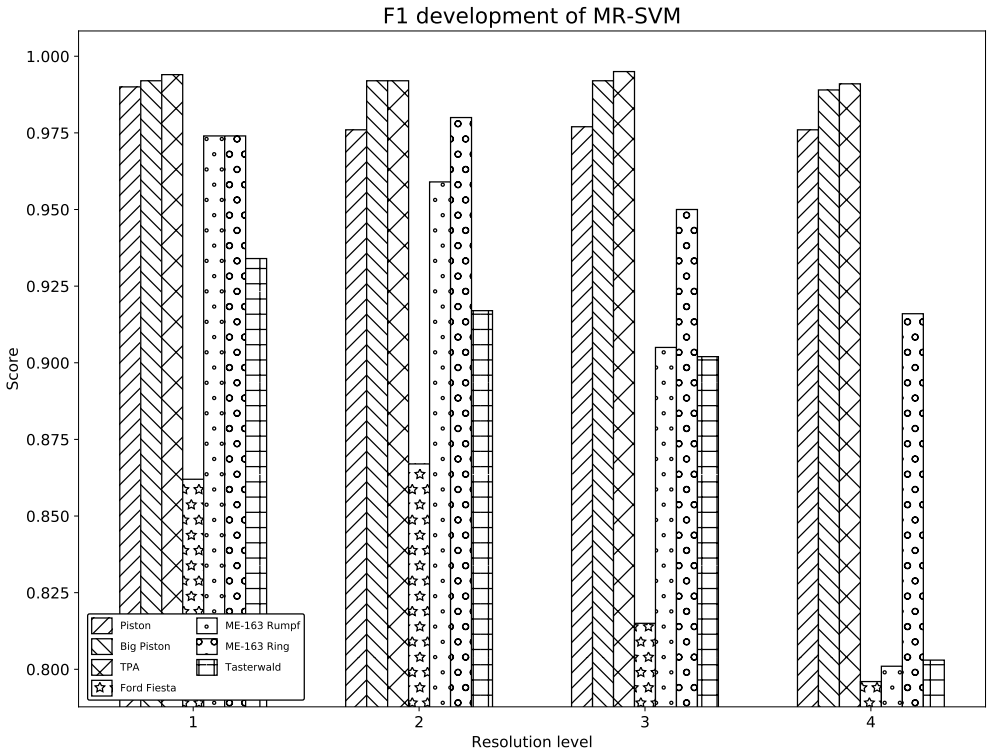


Figure 11.9.: Multiresolution F_1 score development graph.

Overall, applying our multiresolution approach equipped with our Support Vector Machine-based segmentation technique to the configurations shown in Table 11.2 keeps good segmentation results, where the performance decreases slightly depending on the dataset.
As the recall metric hints, it decreases if less voxels belonging to the desired part are selected. On the other hand, the precision shows that most of the voxels remaining after our workflow, are correct, i.e., are part of the component under investigation. Overall, it seems that the classifier correctly locates the wanted objects and their geometric counterparts (cf. the other three springs) but with an increasing number of resolution levels some voxels of these objects are lost. Reasons for this may be of two kinds. First, important details and thus voxels are rejected at coarser levels due to downsampling effects and hence are not considered for further classification on finer levels. Secondly, we use the same postprocessing chain for each resolution level. But the result of segmentation with four resolution levels might require different postprocessing parametrizations than when applying our method to a single scale only.

After focussing on the segmentation performance, at this point we want to actually show the main point of interest when multiresolution approaches are used, namely the decrease of runtime. When measuring the improvement of our multiresolution approach over the regular linear iteration, i.e., for the case when $\ell = 1$, we firstly observed the speedup of the segmentation only, i.e., we start with already trained models and measured the classification phase only. A plot of the speedups over different scales for our test datasets is given in Figure 11.10. However, in the actual application one has to additionally consider that now multiple models need be trained, hence the training time is increased to roughly $\ell_{\max}$ times the single resolution algorithm's training time. The overall runtime behavior including all training times are given in Figure 11.11. Both graphs depict the ordinate on a logarithmic scale. The detailed speedups relative to the single resolution iteration are given in the last column of Table 11.4 for both configurations, which are rounded to three decimals. At this point we want to note that external influences including the framework's memory management affect the runtime too and alternative strategies might improve results even further.
We see that the runtime is improved in every case, while clearly the biggest improvement is achieved if the searched components are small relative to the scan size, e.g., the small orbs of the "Tasterwald" scan, yielding a speedup of up to 20. In such scans, the trend shows that then the speedup is improved for a higher number of scales. However, for other scans which consist largely or entirely of the desired component, the speedups stay mostly constant or even decrease. This is due to the same RoIs being selected on each resolution level, thus the same information needs to be processed every time. Therefore, the same amount of work is done multiple times, increasing the runtime.
Considering also the additional training runtime decreases the speedups, but fortunately *not* by an order of magnitude. Thus, on average the multiresolution version still outspeeds its single resolution counterpart.

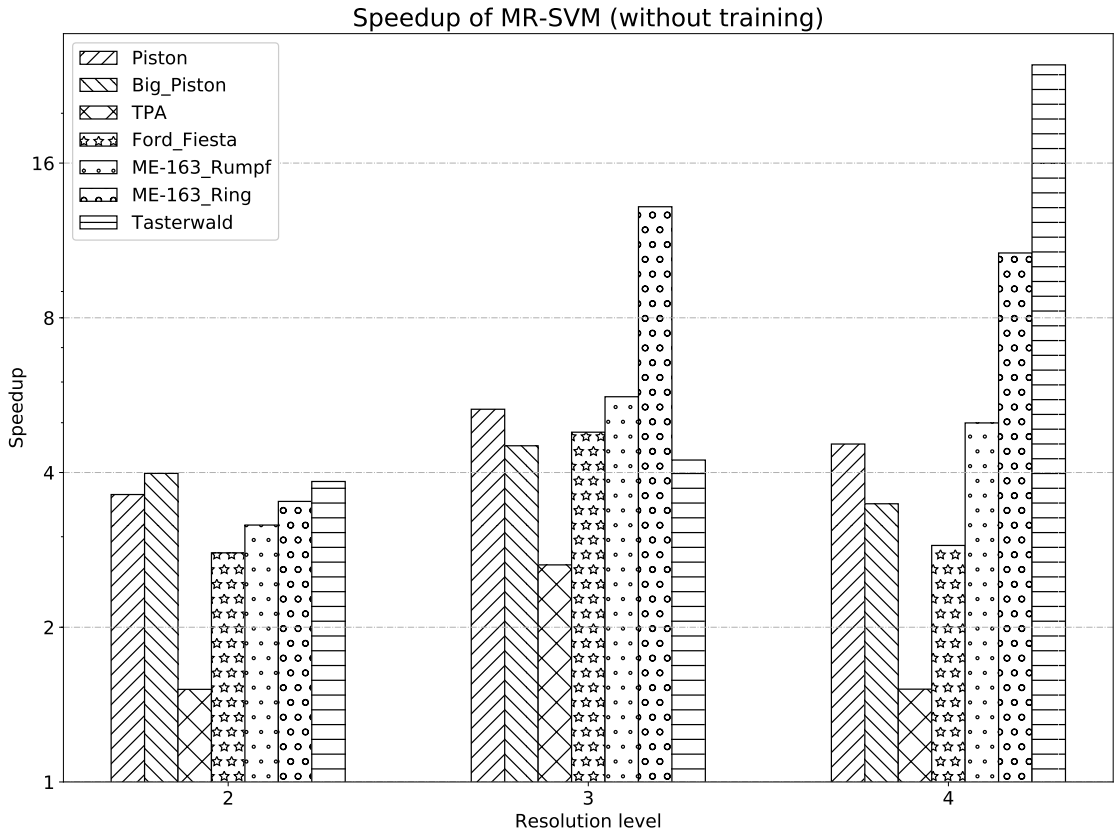


Figure 11.10.: Speedups over multiple resolution levels, segmentation only.

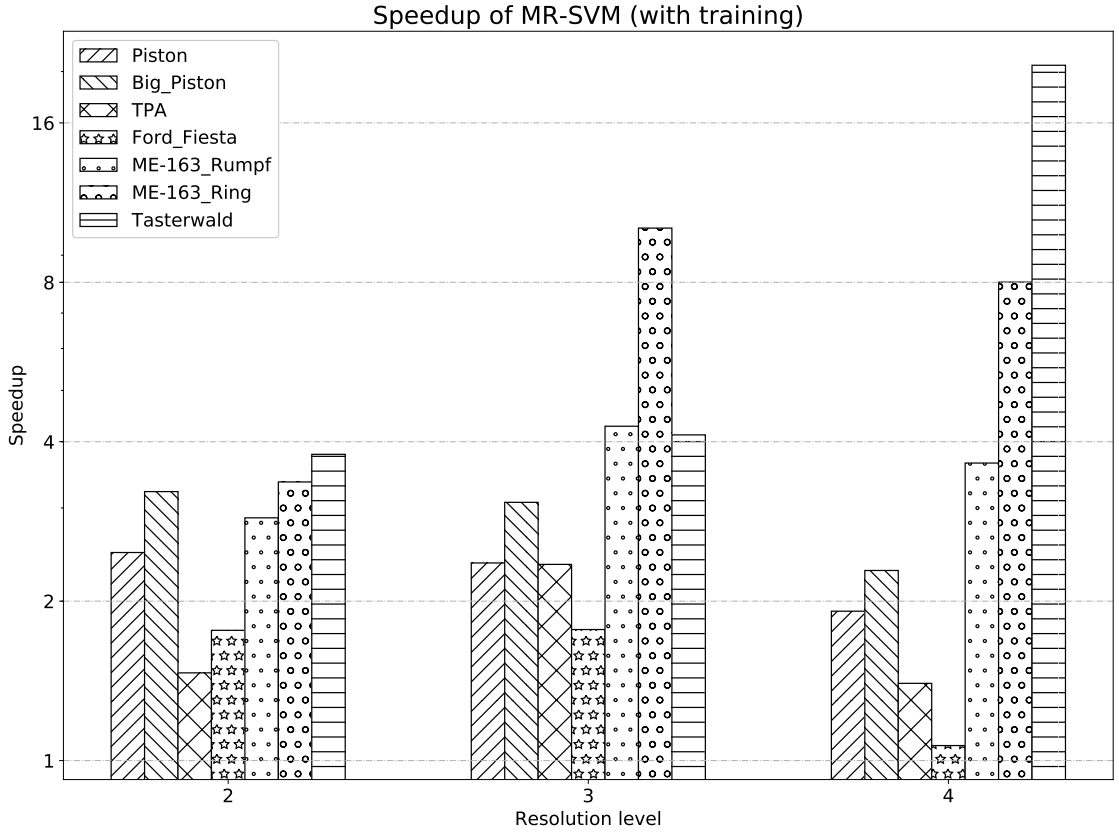


Figure 11.11.: Speedups over multiple resolution levels, including training.

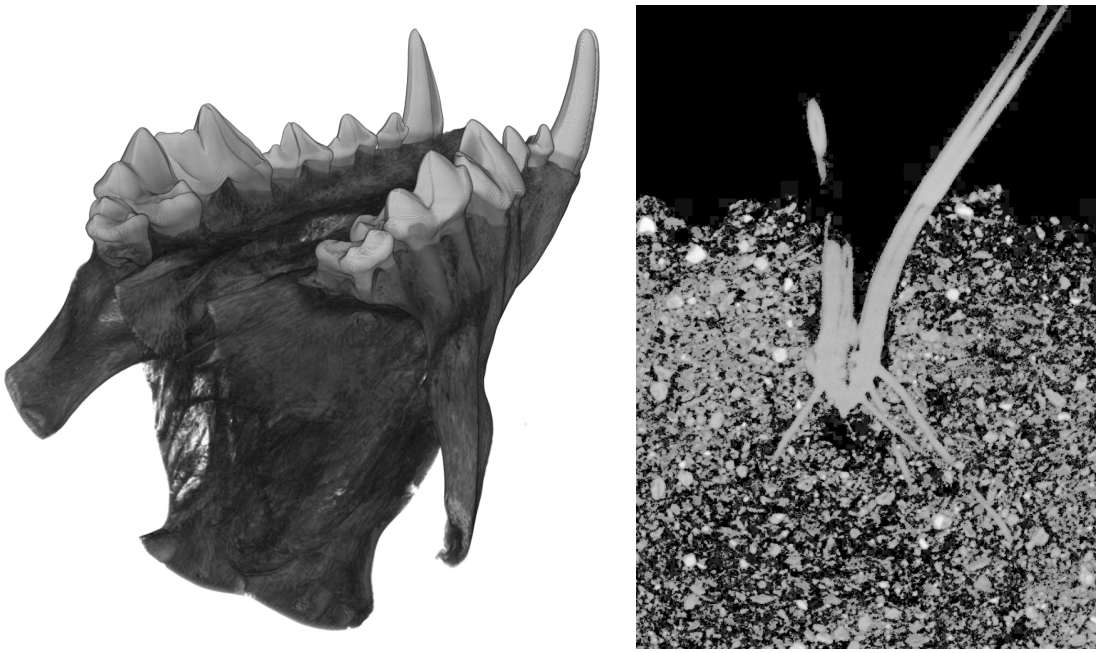
12. Qualitative Evaluation

Abstract Contrary to the preceding chapter, we now focus on a qualitative evaluation. The additional data used there is described first. Subsequent sections apply sampling-based clustering, adaptive thresholding and the proposed Support Vector Machine-based segmentation on a broad variety of different real world datasets. The produced results are visualized and show the high versatility of our methods. Finally, we will briefly discuss the influence of the number of seed voxels and the influence of grayscale value scaling when pretrained models are applied.

12.1. Data

Before continuing to qualitative results we like to introduce the reader to the datasets used for evaluation using the different techniques. All of the voxel volumes described shortly were generated by the Fraunhofer Development Center for X-ray Technology (EZRT) in Fürth. If necessary, additional copyright information is provided.
Regarding clustering techniques, we use both a motor piston as well as a jaw of a wolf which both are suited for clustering since both scans are composed of few individual materials. The piston data is the same as used earlier in quantitative evaluation and is depicted in Figure 11.2a, while the jaw scan labeled "Canis Lupus" can be viewed in Figure 12.1a.
On the other hand, applying FAITH requires that the desired component is the one having the highest grayscale values since it is a binarization technique. Therefore, we selected again the Canis Lupus scan to extract the denser part of its teeth. Additionally, we apply the method to the upper part of a voxel volume obtained by scanning a peruvian mummy, where the extracted Region of Interest is shown in Figure 12.2b[1]. There, the goal is to extract the skull while also keeping thin internal structures which would be lost by simple thresholding.
Finally, the remaining scans examined are segmented using our Support Vector Machine-based approach. There, we consider data from a variety of different fields and we will see that although the domains are highly different very good results can still be obtained using interactive input. We start with the industrial domain in which we aim on the one hand to segment the four springs found near the tires by only marking one, and on the other hand, we extract the windshield using the same technique. These components stem from a scan of a Ford Fiesta which we already used in our chapter on quantitative evaluation, an image is given in Figure 11.4a, in which the sole spring used for the first segmentation case is highlighted. Next, we again consider the peruvian mummy but this time we aim at extracting different components, the entire scan is visualized in Figure 12.2a. Lastly, the Support Vector Machine-based segmentation is used to extract the roots of a wheat plant. The latter are especially difficult to segment since the scan is low in contrast and the surrounding earth is often connected and similar in structure. A slice image of said plant inside its pot is given in Figure 12.1b.
Especially in the context of our method based on a SVM, we briefly examine the influence of the number of seed voxels. Finally, we conclude by showing the retraining capabilities of that technique on a crash car example.

[1] Courtesy of Lindenmuseum Stuttgart.



(a) Voxel dataset “Canis Lupus”.

(b) Slice of scan “Wheat”, contrast enhanced.

Figure 12.1.: Wolf jaw and wheat plant data.

(a) Voxel dataset “Mummy”.

(b) Voxel dataset “Mummy Head”.

Figure 12.2.: Peruvian mummy datasets, courtesy of the Lindenmuseum in Stuttgart.

Scan	#Cluster	#Geometric Features	Env. size
Piston	5	0	-
		2	5
Canis Lupus	8	0	-
		2	7

Table 12.1.: Clustering Setup.

Scan	Threshold	#Levels	#Env. size	#Geometric Features
Canis Lupus	24415	3	5	2
Mummy Head	1541	1	7	

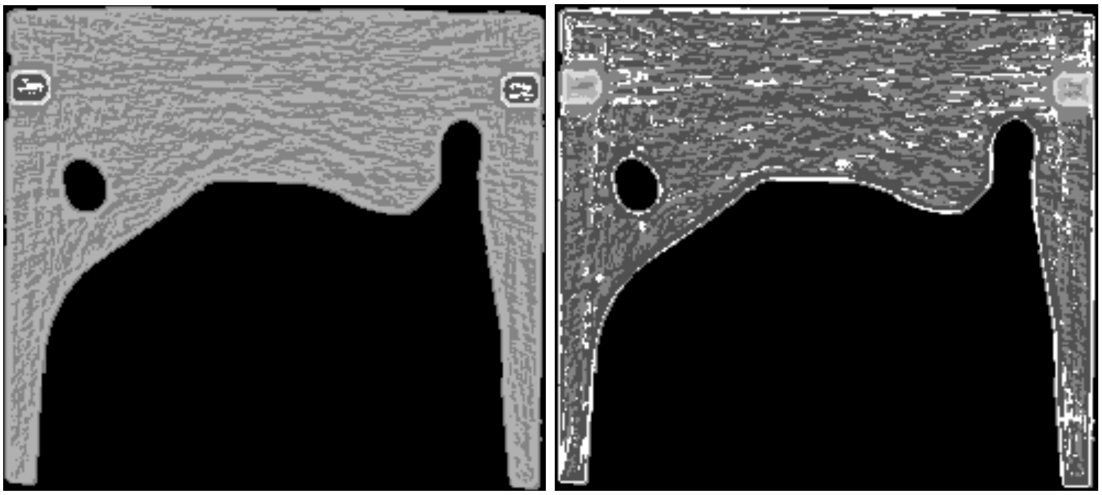
Table 12.2.: FAITH Setup.

12.2. Clustering

We begin the qualitative evaluation by applying our modified clustering techniques to both the piston and wolf jaw scan, the configurations used are listed in Table 12.1. In our experiments when using Gaussian Mixture Models we did not use the Akaike Information Criterion since we want to enforce exactly the specified number of clusters.
Regarding clustering applied to the "Piston" scan, the results are given in Figure 12.3. In Figures 12.3a and 12.3b the results of sampling-based K-Means clustering are depicted. We see that both configurations capture the iron ring well, while on the other hand K-Means significantly suffers from the measurement noise in the scan. The Gaussian Mixture Model-based clustering performs significantly better. The only disadvantage when using features is that the features capture local structure which is corrupted by noise, while on the other hand they enhance contours. Both results are shown in Figures 12.3c and 12.3d.
The results of sampling-based clustering applied to the Canis Lupus scan are depicted in Figure 12.4. A similar behavior as when clustering the piston dataset can be seen in Figures 12.4a and 12.4b. Again, this behavior is improved by using Gaussian Mixture Models as shown in Figures 12.4c and 12.4d.
Overall, we see that depending on the configuration a good clustering can be obtained using our modified algorithm based on a stratified random sample. But, our modification enables the processing of large volumes.

12.3. FAITH

We proceed by showing the results when applying the Feature-Adaptive Interactive Thresholding technique developed in Chapter 8. The configurations used are listed in Table 12.2. For each scan, the threshold is chosen in a way such that the desired component is selected by binarization as good as possible. Regarding the Canis Lupus scan, we focus on the teeth, while for the skull of the mummy we desire to segment the skull bones.
First, we consider the wolf jaw, whose results are depicted in Figure 12.5. Figure 12.5a shows the original slice where we can clearly identify both the upper teeth as well as the round bone structures below. Simple binarization thresholding results in Figure 12.5b where the jaw embedding the teeth is thresholded away, but at the same time most of the teeth are lost. Applying FAITH restores most of the teeth while some borders of the jaw bone are kept. However, the new technique selects the teeth way better.



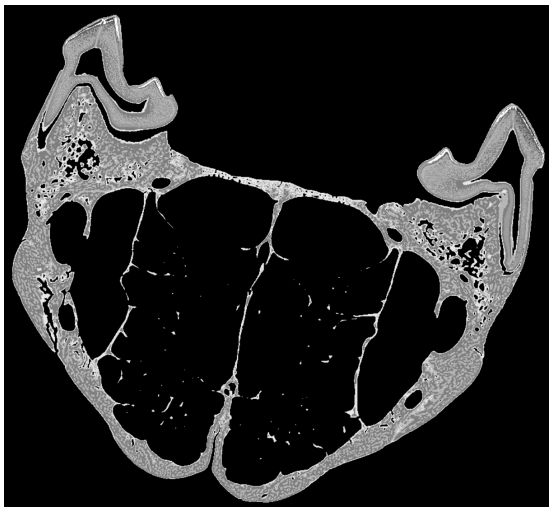
(a) K-Means, configuration 1.

(b) K-Means, configuration 2.

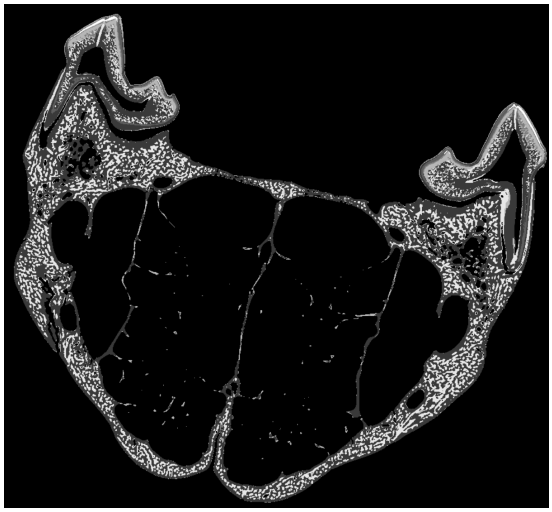
(c) GMM, configuration 1.

(d) GMM, configuration 2.

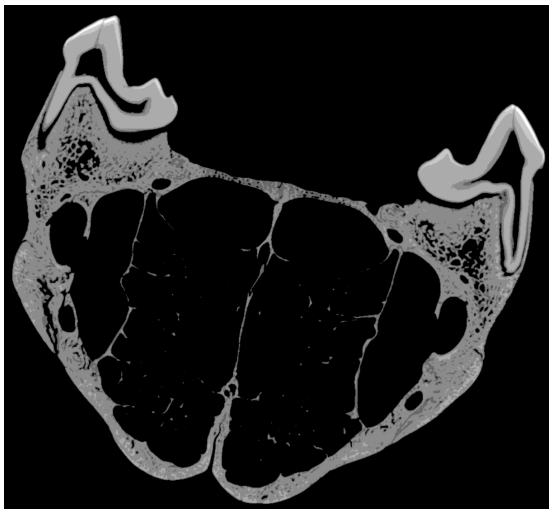
Figure 12.3.: Clustering techniques applied to “Piston”.



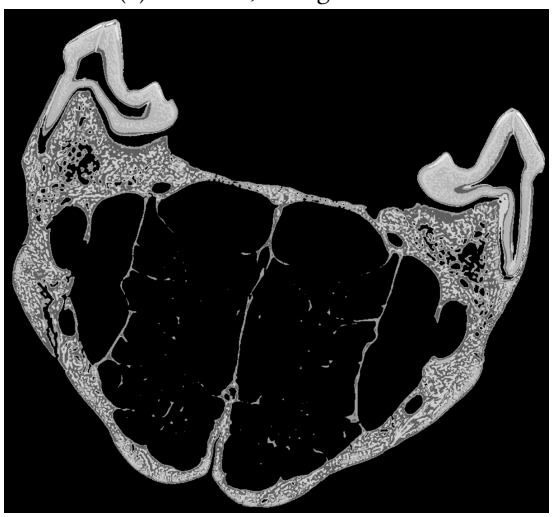
(a) K-Means, configuration 1.



(b) K-Means, configuration 2.

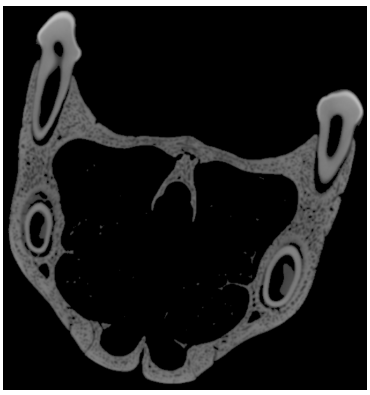


(c) GMM, configuration 1.



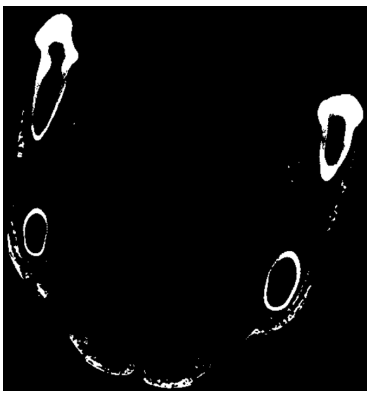
(d) GMM, configuration 2.

Figure 12.4.: Clustering techniques applied to “Canis Lupus”.



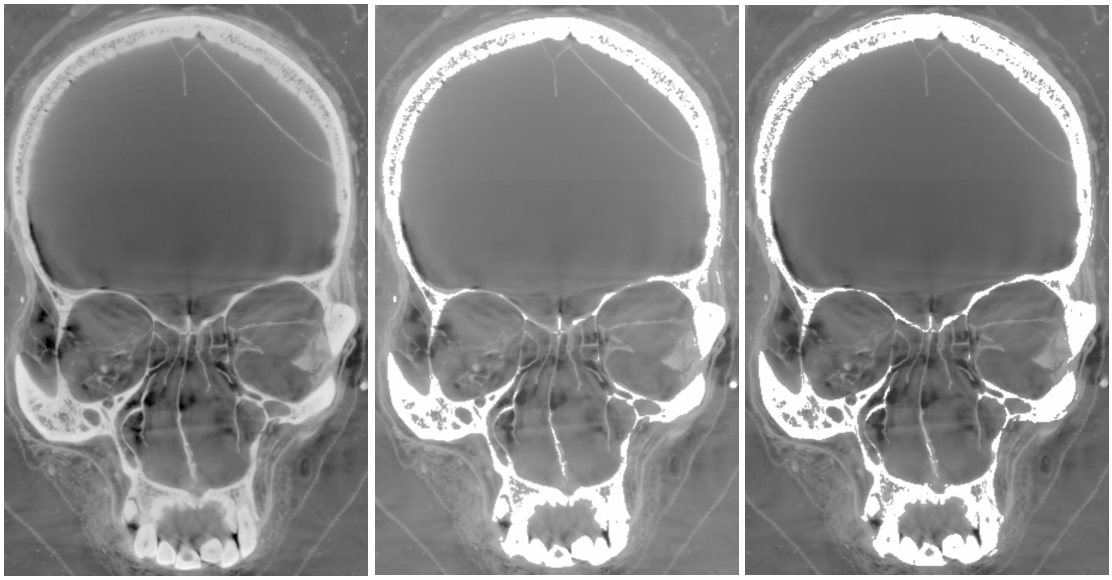
(a) Original scan.

(b) Simple binarization.



(c) FAITH applied.

Figure 12.5.: FAITH applied to “Canis Lupus”.



(a) Original scan. (b) Simple binarization. (c) FAITH applied.

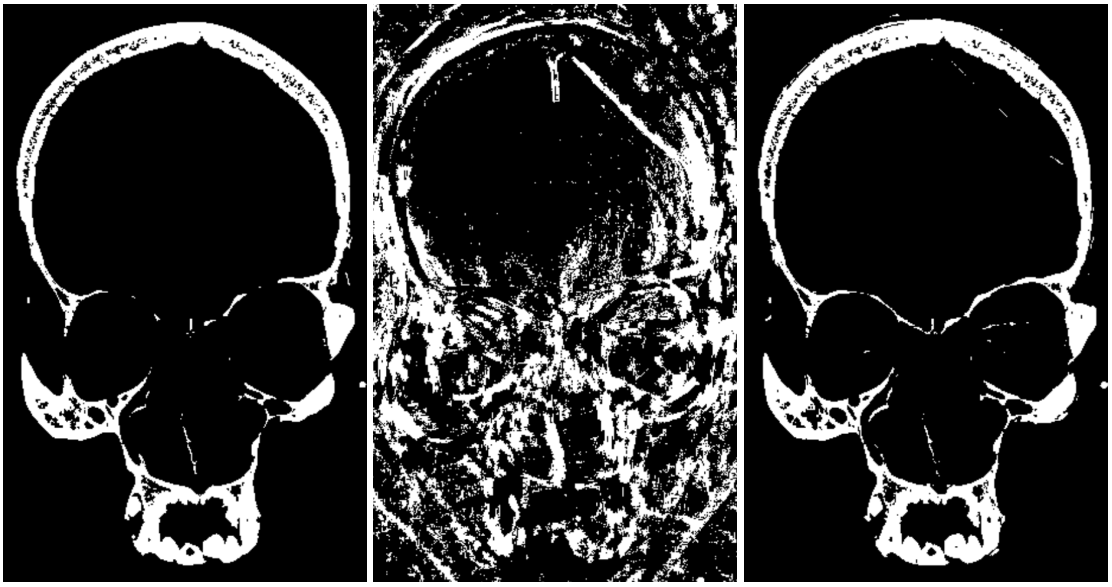
Figure 12.6.: FAITH applied to the mummy skull, results overlaid in white

Next, we apply the procedure to the mummy skull where the original slice image is given in Figure 12.6a. Plain binarization yields most of the skull which we see overlaid over the original data in Figure 12.6b. However, many thin detailed structures especially the top of the eye socket are thresholded away. The original data shows the thin structures. Applying FAITH improves upon this significantly as one can see in Figure 12.6c.

Lastly, we like to show the influence of the individual steps introduced in Chapter 8 on the example of the mummy skull dataset. Figure 12.7a shows the effect of plain thresholding which discards thin structures at the top of the eye sockets. Including features to improve upon this deficiency using a least squares approximation introduced in Section 8.2.1 results in severe artifacts as shown in Figure 12.7b. However, applying regularization and limiting to valid thresholds significantly improves this intermediate result and further restores the thin structures lost due to plain thresholding.

12.4. SVM-Based Segmentation

In the last section of our qualitative evaluation we apply our Support Vector Machine-based segmentation described in Chapter 9 to segment many different objects in order to show the versatility of that method. The used configurations are given in Table 12.3. In each case, a data-dependent skip threshold was applied.
We start in the industrial use case by segmenting the springs as well as the windshield of the crashed Ford Fiesta. Both results are depicted in Figure 12.8. Regarding the windshield, we see the segmented volume overlaid in white on a three-dimensional rendering of the car in Figure 12.8a. The extracted component is largely intact and we specifically used the plane fit feature. Considering the springs, we marked individual seed voxels of the one spring visualized in Figure 11.4a and aimed to extract all four springs. Figure 12.8b shows a slice with segmented voxels overlaid in white. Although the slice contains two springs only, all four were found.
We continue by applying our segmentation procedure to a dataset including a peruvian mummy. For most examples, we applied grayscale value and orientation features and also used several



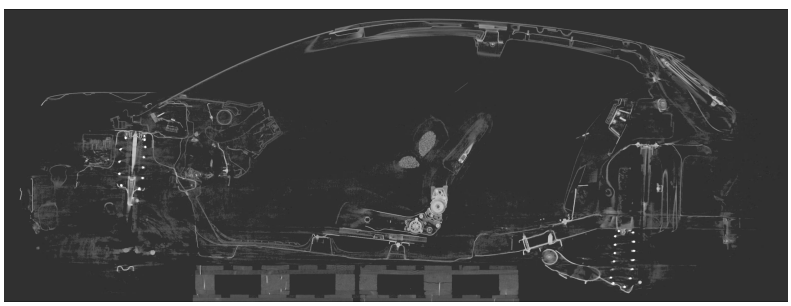
(a) Simple binarization. (b) Least squares approximation. (c) Full FAITH procedure applied.

Figure 12.7.: Individual FAITH steps applied to the skull dataset.

Scan		**Feature set**	K	$\ell_{\max}$	**#Seed Voxels**
					pos / neg
Ford Fiesta	Springs	Gray + Orient	3	1	65 / 81
	Windshield	Gray + Orient + Plane			73 / 96
Mummy	Corn cob	Gray + Orient	7	3	127 / 82
	Bracelets			2	242 / 128
	Tool			1	50 / 44
	Skull		5		336 / 100
	Ropes	Gray + Orient + Line	7		101 / 128
Piston		Gray + Orient	5		50 / 72
Wheat roots			7		79 / 39

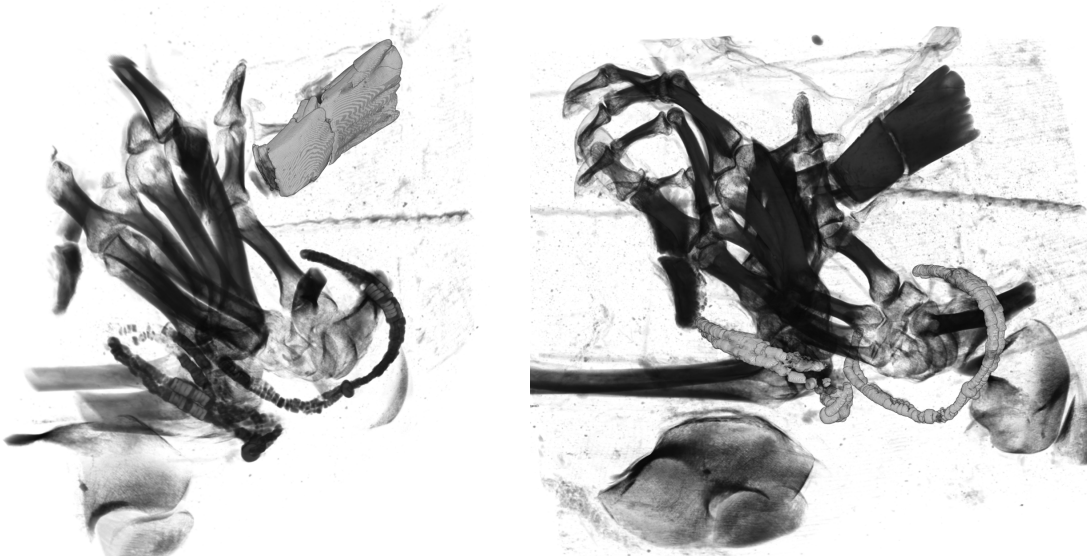
Table 12.3.: SVM segmentation configurations.

(a) Ford Fiesta: Windshield.



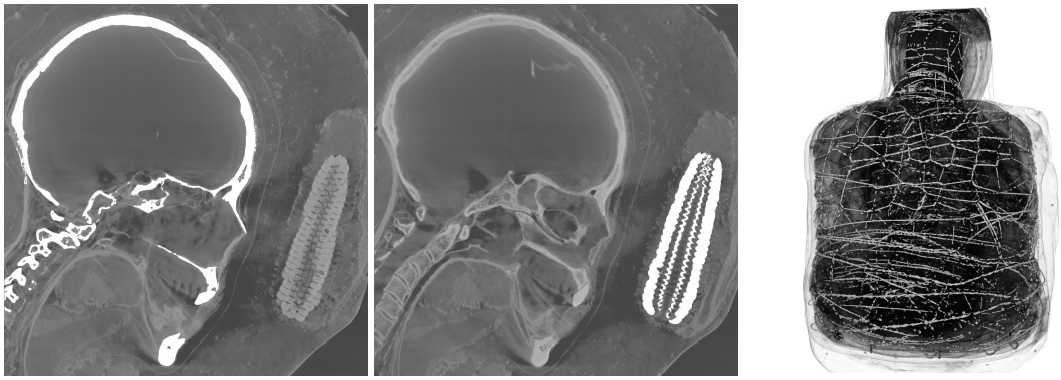
(b) Ford Fiesta: Springs.

Figure 12.8.: Ford Fiesta segmentations, results overlaid in white.



(a) Mummy: Tool. (b) Mummy: Bracelets.

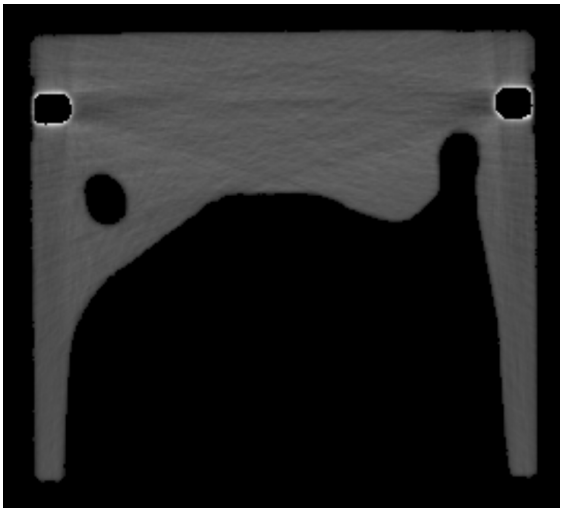
Figure 12.9.: Peruvian mummy, results overlaid in white.



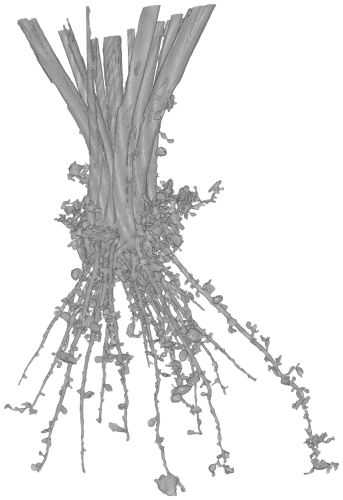
(a) Mummy: Skull. (b) Mummy: Corn cob. (c) Mummy: Ropes.

Figure 12.10.: Peruvian mummy, results overlaid in white.

resolution levels. But, when segmenting the ropes holding bandages together we additionally use line fit features. All results related to the mummy scan are depicted in Figures 12.9 and 12.10, the two original datasets can be found in Figure 12.2. We see that good segmentation results can be obtained using the technique proposed in Chapter 9 and its multiresolution pendant.
Next, the same method is used to segment the aluminium piston and disregarding the iron ring on top. Furthermore, the roots of a wheat plant are extracted with the same technique. These roots are especially difficult to segment as their shape and grayscale values change as the roots go deeper in the earth. Both results are shown in Figure 12.11. We see that the roots are mostly kept, while still many small stones remain in the result volume as they are locally indistinguishable from the roots.
Finally, we apply the same algorithm to archaeological data of a Roman chest armor piece which was found near the German town Kalkriese, the depicted data belongs to the local museum. Archaeologists strongly believe that this armor is a relic of the Battle of the Teutoburg Forest. The scan was made from a big block of earth, a slice is shown in Figure 12.12a. The segmentation result is depicted in Figure 12.12b in 3D viewed from above. In both figures, one can clearly

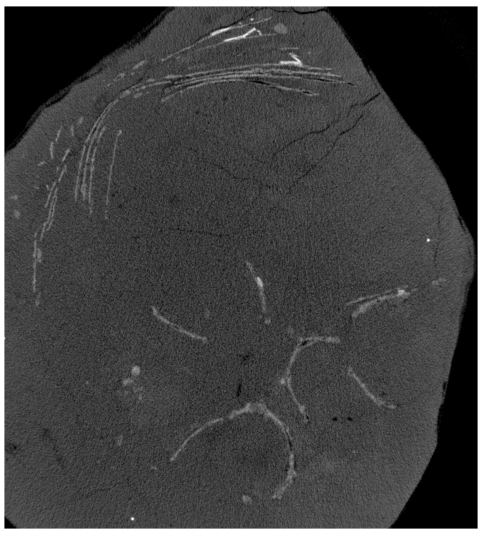


(a) Piston without iron ring.



(b) Wheat roots.

Figure 12.11.: Piston and wheat roots.



(a) Varus armor original slice.

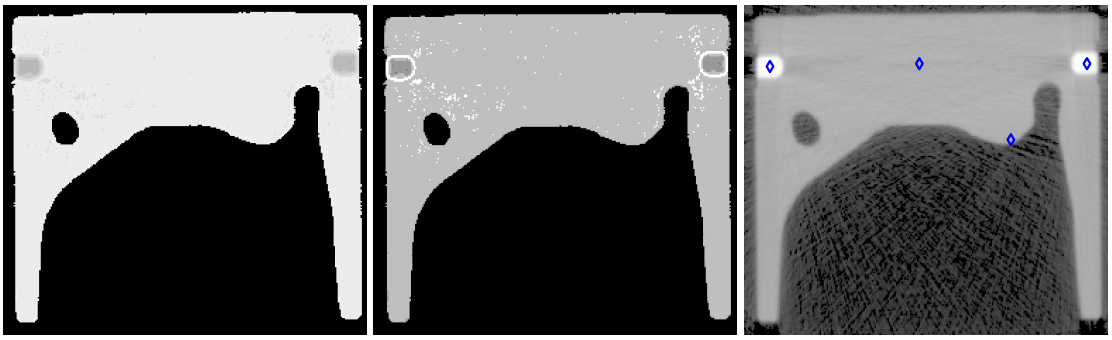
(b) Segmented armor piece.

Figure 12.12.: Roman armor piece.

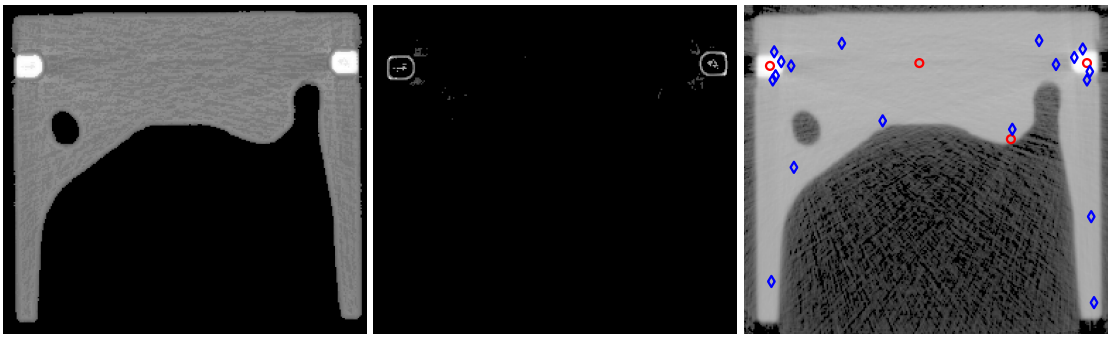
identify the individual segments of the armor and also some fasteners. This dataset also proved hard to segment due to the low contrast and the poor condition the armor is in.

Active Learning by Uncertainty Sampling

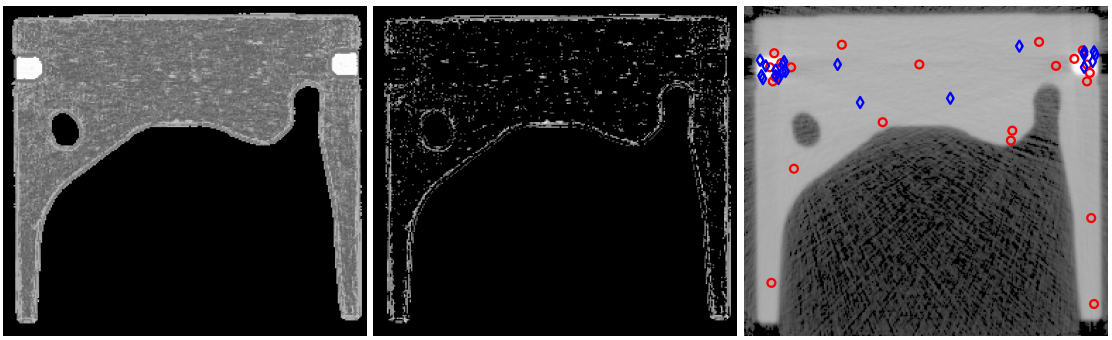
The overall evaluation focussed on confidence values thus far. In this section we briefly examine uncertainty sampling as an active learning approach which is based on these confidences. For this we applied an uncertainty function given in Definition 9.7.1 on the intermediate segmentation results obtained from our well-known piston dataset, from which we aimed to segment the ring part. The parametrization was identical for each iteration and specifically used an environment size of five voxels and uses the grayscale and orientation features. The measurement noise outside of the piston was removed via thresholding with value 4020 given the overall grayscale value range [0,65535]. The uncertainty locality parameter δ was set to 2.
The following evaluation considers the obtained confidence values as well as the computed uncertainty values. The latter served as graphical input where the next additional seed voxels shall be chosen. In the display of the seed voxel selection, we mark the previous iteration voxels, i.e., the voxels that led to the shown results, with red circles. The voxels which were selected according to the uncertainty map are drawn as blue diamonds. The overall set of training voxels for each iteration consists of the union of previously used voxels (the red circles) and the newly selected ones (the blue diamonds). In all images attributed to an iteration, brighter colors denote higher values. Additionally, the contrast was enhanced in all images.
Figure 12.13 shows the progression of all four executed uncertainty sampling iterations. In the very first iteration we selected only four seed voxels, cf. Figure 12.13c. After executing our SVM-based segmentation algorithm with the stated parametrization, we obtained confidence values and uncertainty values depicted in Figure 12.13a and Figure 12.13b, respectively. We observe that the results are not satisfactory as the confidence values of the piston ring are even smaller than the values of the piston itself. Additionally, the uncertainty map shows that the overall uncertainty is still quite high, especially at the contact surfaces of the ring and the piston, as expected. Given this knowledge, we select additional seed voxels in uncertain regions, i.e., both inside of each component and also at the contact surfaces. This selection is marked in Figure 12.13f. The identically parametrized segmentation algorithm then yields new confidence values in Figure 12.13d and uncertainties in Figure 12.13e. We immediatly see that the confidence values are much better suited for selection of the steel ring. The uncertainty confirms this as now the contact surfaces are still a source of segmentation ambiguity while the majority of the volume can be assigned to its respective class with high probability. To improve the results further we again select additional seed voxels at the contact surfaces. We decided to also include some seed voxels at the interior of the piston since the confidence there is still significant. Figure 12.13i displays the new selection. The results of this third iteration are depicted in Figures 12.13g and 12.13h, respectively. While the confidences do not change significantly and still match the ring well, the uncertainty map clearly shows the improvement over ambiguities as the steel ring is explained well now. Only the borders of the piston remain. Therefore, we select additional voxels at the border, cf. Figure 12.13l. The final uncertainty map is shown in Figure 12.13k which is zero at the majority of points, hence only very few points left seem uncertain. This signals that not much improvement by further seed voxel selection can be expected here from an active learning point of view. Simple thresholding on the final confidence values in Figure 12.13j yields the part under investigation. Overall, 59 seed voxels contributed to the shown results.
Concludingly, we saw that an uncertainty map as computed from the confidence values obtained by the segmentation is useful to improve segmentation results. However, we also experienced that the confidence and uncertainty values should be interpreted simultaneously for best results.



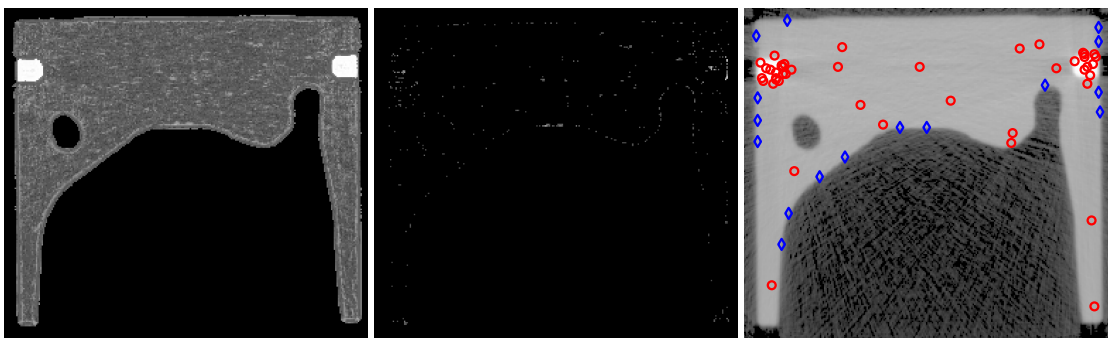
(a) Confidence values #1. (b) Uncertainty values #1. (c) Voxel selection #1.



(d) Confidence values #2. (e) Uncertainty values #2. (f) Voxel selection #2.

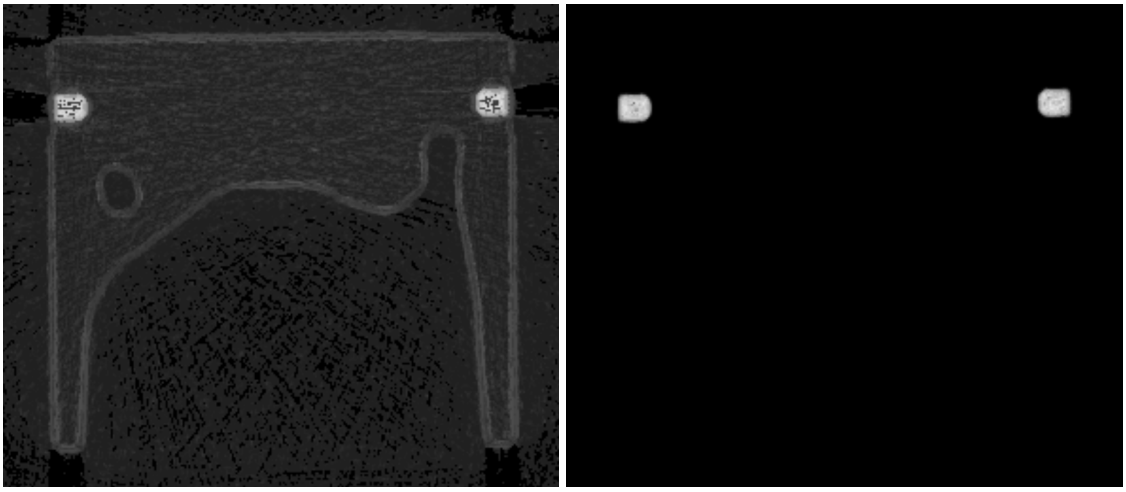


(g) Confidence values #3. (h) Uncertainty values #3. (i) Voxel selection #3.



(j) Confidence values #4. (k) Uncertainty values #4. (l) Voxel selection #4.

Figure 12.13.: Progression of uncertainty sampling on piston data.



(a) Iron ring, one voxel per class. (b) Iron ring, multiple voxels per class.

Figure 12.14.: Seed voxel count influence on iron ring.

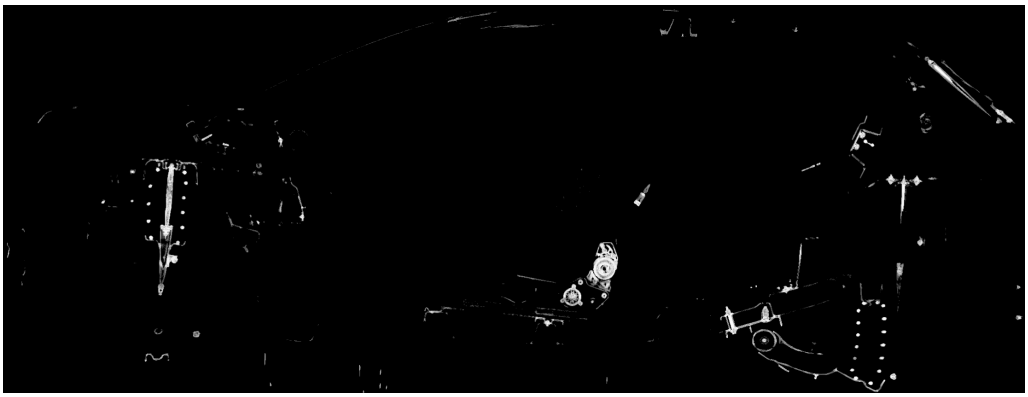
Influence of Seed Voxel Count

Thus far, we trained a Support Vector Machine on several seed voxels for each class. Now, we briefly examine the influence of the number of seeds on the segmentation performance.
We start by segmenting the iron ring from the piston dataset. The minimimum number of seed voxels is one seed per class, hence we selected one positive labeled voxel belonging to the iron ring and three negative labeled voxels from the aluminium piston, the styrofoam holder and the background corrupted with measurement noise. A slice of the segmented volume is shown in Figure 12.14a, which shows that the model already assigned high confidences to the iron ring but the other components still remain in the target volume, although the confidence values are small. In a second configuration, we used 13 positive and 30 negative labeled seed voxels. The result is depicted in Figure 12.14b which shows that the unwanted components are discarded better while at the same timekeeping the iron ring.
A similar comparison regarding the segmentation of the springs of the Ford Fiesta scan is shown in Figure 12.15. The few voxel configuration is composed of three voxels of a spring and three voxels of non-spring components, the result is shown in Figure 12.15a. We see that many non-spring components, e.g., parts of the chassis, are present in the result. This is considerably improved by adding more seed voxels as depicted in Figure 12.15b where in total 44 positive and 69 negative seeds were selected.
In general, we see that the more information in form of seed voxels the model is trained on, the higher is the probability that a better segmentation is achieved. Theory supports this as Steinwart [3, Corollary 1] proved that given a certain choice of the problem's penalty hyperparameter and using a Gaussian kernel, Support Vector Machines are *universally consistent* on all compact subsets of $\mathbb{R}^d$. Intuitively, a universally consistent classifier gets better the more training data it can learn from.

Retraining Models

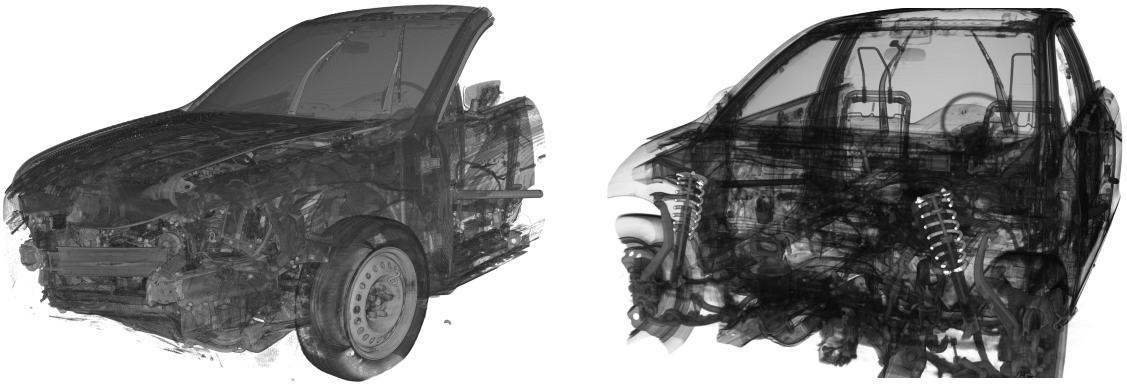
We conclude the presented qualitative evaluation by applying a previously trained Support Vector Machine-based model onto another dataset. One could evaluate this on many different scans and the quality of the original model also impacts the quality when reapplying it on a different volume. Instead, we focus on a single dataset, namely the voxel volume showing a crashed Ford

(a) Springs, three voxels per class.



(b) Springs, multiple voxels per class.

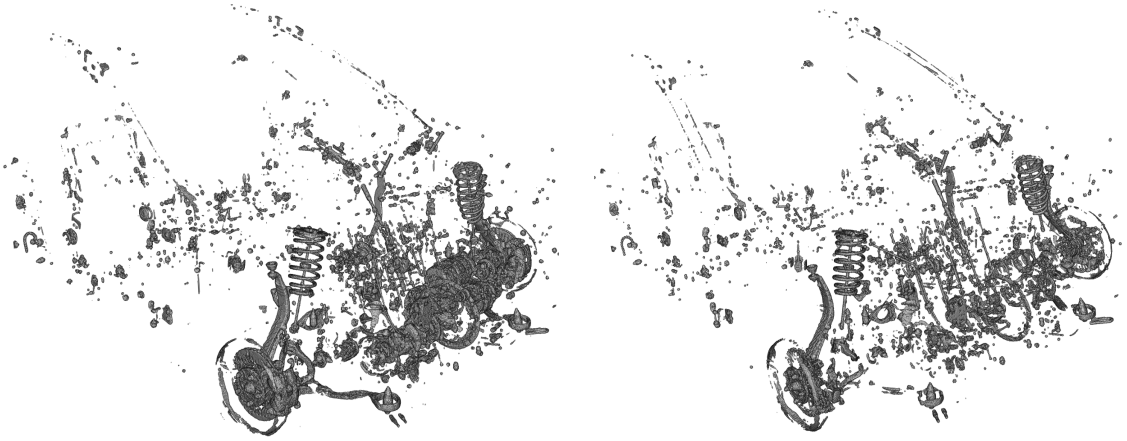
Figure 12.15.: Seed voxel count influence on Ford Fiesta springs.



(a) Original data, zoomed into its front.

(b) Result, overlaid and cropped.

Figure 12.16.: “Honda Accord”, original data and result of retrained segmentation.



(a) Applied to unscaled data, color corrected.

(b) Applied to scaled data, color corrected.

Figure 12.17.: Retraining a pretrained model on a different scan.

Fiesta, and check the influence of scaling to the results. Regarding the model, we use a SVM trained a priori on this scan to find the springs of the car. Next, we apply it to another crash car scan, namely that depicting a Honda Accord shown in Figure 12.16a. We also selected few additional seed voxels from the Honda dataset to adapt the trained model to the new data. However, one major point not discussed yet is the influence of the actual grayscale value distribution, since it can have considerable impact on the computed features. Thus, we apply the same model with the same seed voxel selection to the unprocessed Honda voxel value data, cf. Figure 12.17a, as well as to a derived dataset where we linearly scaled the voxel values such that the voxel values belonging to the springs approximately match between these scans, cf. Figure 12.17b. Both images show the unprocessed output of the SVM-based segmentation procedure. We observe that scaling the grayscale distribution accordingly improved the result obtained using a Support Vector Machine by discarding additional voxel regions which do not contain relevant voxels. However, this behavior does not influence the final result after postprocessing shown in Figure 12.16b where we cropped the volume to highlight the segmented springs. Nevertheless, appropriate scaling is expected to improve the segmentation results in general.

References

[1] Jesse Davis and Mark Goadrich. "The Relationship between Precision-Recall and ROC Curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. Association for Computing Machinery, 2006, pp. 233–240.

[2] Jeroen Bertels et al. "Optimizing the Dice Score and Jaccard Index for Medical Image Segmentation: Theory and Practice". In: *Medical Image Computing and Computer Assisted Intervention – MICCAI 2019*. Springer International Publishing, 2019, pp. 92–100.

[3] Ingo Steinwart. "Support Vector Machines are Universally Consistent". In: *Journal of Complexity* 18.3 (2002), pp. 768–791.

Part VI.

Conclusions and Outlook

Um, can you repeat the part of the stuff where you said all about the... things?

Homer Simpson, The Simpsons

Table of Contents

At this final part of this thesis we recapitulate our contributions to the field of industrial volumetric segmentation. Finally, in the following chapter we briefly give an outlook of points in our framework where further research should be conducted.

13. Conclusion

We finally recapitulate our contributions to the field of industrial volumetric segmentation. Specifically, in the first chapter we discussed limitations of existing methods when applied in our setting. After this, we formally introduced how to represent a three-dimensional voxel volume and, most importantly, the description of a *local environment* centered around an iterable voxel. Next, we discussed the benefits of comparing histograms by their Wasserstein distance instead of the regular Euclidean distance. In that chapter we further introduced an embedding which transforms histograms into vectors such that the Euclidean distance of the latter approximates the Wasserstein distance of the original histograms. Additionally, we gave algorithms for efficiently computing this embedding. In Chapter 4 we described a set of features that are computed from local environments around voxels. On the one hand, we augmented existing descriptors based on the grayscale values or the texture of regions and conditionally enhanced them to describe three-dimensional environments. On the other hand, we proposed new features specifically comparing regions to geometric primitives like planes or lines. We further aimed to make the defined descriptors robust against noise and recapitulated a method to compute the selection of features, which best describe the regions, automatically. After explaining features computed from local regions, we focussed on unsupervised segmentation of voxel volumes. There, we defined a novel sampling strategy to achieve a representative random sample from the voxel volume. Having such a sample, we continued to present the two well-known data clustering algorithms K-Means and Gaussian Mixture Models in Chapters 6 and 7, respectively. For both methods, we specifically focussed on their initialization and ways to avoid degeneracies. Additionally, we combined them with our random sampling procedure which induces a constant memory consumption independent of the size of the processed voxel dataset, and hence makes them feasible to be applied on volumes without size restrictions. Both techniques also consider the features introduced in Chapter 4 or alternatively cluster volumes purely on their grayscale value distribution. We further gave brief analyses over the asymptotic runtime behavior and memory consumption showing their effective linear runtime with respect to the volume voxel count. In Part IV, we derived three novel segmentation techniques which learn their necessary information from very few training data *interactively*. That is, user interaction provides domain expert knowledge in form of seed voxels and conditionally labels. Then, classifiers are trained on this information. First, we extended a thresholding procedure which incorporated feature information by allowing multiple features to have influence on the result in Chapter 8. Next, we used a Support Vector Machine model as a voxelwise classifier based on user interaction and local information. Then, we provided a general voxelwise multiresolution segmentation framework in which we embedded the previous two mentioned interactive methods. For all techniques, we provided details how they can be used for voxelwise segmentation of volumes without size restrictions. Finally, we evaluated the introduced segmentation procedures on selected voxel volume datasets. For the quantitative evaluation, we applied a deterministic setup mimicking an interactive Support Vector Machine-based segmentation including postprocessing steps on the mentioned volumes for which ground truth was available. The results indicate that the segmentation is almost always superior to classical thresholding which is still the most used method for volumetric segmentation. We also saw that applying the multiresolution counterpart of our procedure can speed up the volume processing considerably while the segmentation

performance decreased only slightly. The qualitative evaluation instead applied all methods on several datasets and the achieved results show the versatility of our methods.

The main advantages of our contributions are both the generality and the interactivity. The generality refers to the fact that all of our methods operate on general voxel volumes and hence are not tailored towards CT or MRI scans. Furthermore, we do not focus on specific parts we want to segment, e.g., the liver in medical scans, either. Instead, we do not impose any prior knowledge on the parts that shall be segmented. The interactivity instead implicitly provides that knowledge. It further enables a user to segment *unique scans*, i.e., scans of objects where no comparable volumes exist. The best examples in this category are objects of cultural heritage, cf. our peruvian mummy, or crashed car prototypes. We further saw that our segmentation procedures yield good to excellent segmentation results, which proposes the new idea of using them to produce ground truth data for other classifying systems like neural networks. The creation of such ground truth data is cumbersome and requires an enormous effort by domain experts. However, we claim that this process can be sped up significantly using our methods and requires only little manual editing. Recently, we applied this to create ground truth data by segmenting pieces from toy containers and successfully trained a deep neural network which automatically detects these pieces at different positions and in different orientations in other containers. Overall, we believe that the different techniques introduced in this thesis will serve applications by allowing for voxelwise segmentation of big volumes of many different domains. The capabilities presented are likely to benefit the creation of ground truth data for different classification systems or even be directly used for extraction and visual inspection of digitized three-dimensional objects.

14. Outlook

We like to point out a few points in this thesis where further research could be conducted to improve the segmentation performance and usability of our techniques.
First, in Chapter 3, we conjectured that when only very few training samples are available, the segmentation decision can be improved when using the Wasserstein distance instead of the regular Euclidean distance to resolve ambiguities, e.g., if the decision value for SVM classification is close to the hyperplane. Then, the Wasserstein distance might assign the item with more confidence by reducing the distance to the assigned cluster. While this behavior seems reasonable, more sophisticated numerical experiments shall be performed to empirically verify this. Also, the engineering of features is a task which can be continued endlessly. While we covered the main geometric properties of local environments, improvements of the presented features are certainly possible and worthwile. Additionally, we limited ourselves to purely local features. However, if application-dependent global information is given, it might be incorporated into the feature vector to allow for better classification.
Specifically considering FAITH, a future extension of that method might fit a sigmoidal function to the decision values similar to Platt scaling. The modified algorithm then would yield probabilities which in turn enables confidence thresholding as when applying our SVM-based method. Concerning SVMs, an important open problem is an efficient serialization with respect to model retrainability. That is, it should be investigated what training data instances need to be serialized in order to make the model retrainable instead of serializing all data points. Considering our multiresolution framework, several heuristics were introduced to efficiently handle candidate RoIs. There, one might improve especially the fusion criterion by the idea of an shape error functional already stated in that chapter. The remaining challenge is not only to find a good functional but also one which computes the introduced shape error on a small set of locally processed information only.
On a technical level, one deficiency of our methods lies within its nature, namely that they classify voxelwise. This is exactly our intention, but it severely affects the actual runtime of algorithms, and while they are of asymptotical linear runtime with respect to the input voxel count, processing each voxel can take a long time. There, further research should be done in two natural ways: grid and GPU computing. By grid computing, we mean that big volumes should be partitioned with necessary overlap and distributed among a computing cluster or grid. After the voxelwise classification of the partitions, the results then shall be fused back together into one result volume. While this is not a hard problem to solve, it is not supported by the framework in which our methods are implemented thus far. However, extensions should easily be able to implement this. Since our algorithms further act locally and assign computed values to each individual voxel, a natural idea would be to derive efficient GPU computations of the presented features. This can speed up both the training phase as well as the classification phase considerably. The latter can further be performed fully on the device since the classification rule of a Support Vector Machine is simply composed of feature vectors and effectively dot produts. Finally, we want to emphasize that both approaches can be combined easily. That is, after the model is trained, the volume is partitioned, distributed among the nodes in the cluster and classified on the GPU devices there.
Overall, there is still room for improvement upon our novel techniques which can not only improve their segmentation but also the runtime performance and interactivity.

Appendices

A. Reservoir Sampling Has Uniform Probability

In this section, we prove that the typical reservoir sampling algorithm draws a uniformly distributed sample from a population, i.e., that each population element has equal probability of getting selected. Although it is well-known and several short proofs for this can be found in literature, we give a detailed explanation for the sake of completeness following the proof found in [1].

Lemma A.1. Given a population X, Algorithm 4 produces a uniformly distributed random sample Ω, i.e. each population item $x_i, i \in \{1, \dots, N\}$, has the probability

$$P(x_i \in \Omega) = \frac{M}{N}$$

of being chosen to be in the reservoir.

Proof. Denote Ω_t to be the random sample after iteration $t \in \mathbb{N}$.
We show that $P(x_i \in \Omega_t) = M/t$ for all $t \geq i$.
We proof this by induction over t. For the base case, we distinguish between two cases.
If $i \leq M$, then the base case is $t = M$ where all elements are added to the sample always, i.e.,

$$P(x_i \in \Omega_t) = P(x_i \in \Omega_M) = 1 = M/t.$$

On the other hand, if $i > M$, the base case is $t = i$ for which we get

$$P(x_i \in \Omega_t) = P(x_i \in \Omega_i) = P(r_i < M) = M/i = M/t,$$

where $r_i \sim \mathcal{U}([1, i])$ as specified by the algorithm.
In this setting, our induction hypothesis is that

$$P(x_i \in \Omega_t) = M/t.$$

for some $t \geq i$. For the induction step, consider the time point $t + 1$. Given our algorithm, we compute the probability that the item x_{t+1} replaces an item x_i in the sample by

$$\begin{aligned} P(x_{t+1} \text{ replaces } x_i \mid x_i \in \Omega_t) &= P(x_{t+1} \text{ is selected to be added })P(x_i \text{ is selected to be removed}) \\ &= \frac{M}{t+1}\frac{1}{M} \\ &= \frac{1}{t+1}. \end{aligned}$$

Plugging in, we obtain

$$\begin{aligned} P(x_i \in \Omega_{t+1}) &= P(x_i \in \Omega_t)P(x_i \text{ is not replaced at time } t+1 \mid x_i \in \Omega_t) \\ &= P(x_i \in \Omega_t)\left(1 - P(x_{t+1} \text{ replaces } x_i \mid x_i \in \Omega_t)\right) \\ &= \frac{M}{t+1}. \end{aligned}$$

Finally, after a full scan of the input population, the resulting probability is given by

$$P(x_i \in \Omega) = P(x_i \in \Omega_N) = \frac{M}{N}.$$

□

B. Convergence of Naive FAITH Iterations

Here we show that the naive FAITH iteration

$$\begin{cases} \boldsymbol{x}^{(0)} \in \mathbb{R}^d, \\ \boldsymbol{x}^{(k+1)} = \mathcal{P}_{\mathcal{C}}\left(S_{\delta\lambda\mu}(\boldsymbol{x}^{(k)} - \delta\,\nabla f(\boldsymbol{x}^{(k)}))\right). \end{cases}$$

converges against a unique fixed point, although that fixed point need not be the true minimizer of the FAITH target function as shown in Example 8.3.2.

First, we need an auxiliary lemma.

Lemma B.1. Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be any proper, lower semi-continuous convex function.
Then the proximal operator for f is *nonexpansive*, i.e.,

$$\|\operatorname{prox}_f(\boldsymbol{x}) - \operatorname{prox}_f(\boldsymbol{y})\|_2 \leq \|\boldsymbol{x} - \boldsymbol{y}\|_2$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$.

This is fulfilled as proximal operators are firmly nonexpansive and thus also nonexpansive. A detailed explanation is given in in [2, p.355, Lemma 2].

Now, we prove the nonexpansive properties of all steps of the iteration.

Lemma B.2. Let $\mathcal{C} = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{C}\,\boldsymbol{x} \leq \boldsymbol{d}\} \neq \emptyset$ be a nonempty convex polytope defined by $\boldsymbol{C} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{d} \in \mathbb{R}^m$. Further, let $t > 0$ and denote the soft-thresholding operator with threshold t by

$$S_t(x) = \operatorname{sgn}(x)\ \max\{0, |x| - t\}.$$

Then, for all $x, y \in \mathbb{R}^n$ we have

$$\|\mathcal{P}_{\mathcal{C}}(\boldsymbol{x}) - \mathcal{P}_{\mathcal{C}}(\boldsymbol{y})\|_2 \leq \|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{16.1}$$

$$\|S_t(\boldsymbol{x}) - S_t(\boldsymbol{y})\|_2 \leq \|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{16.2}$$

Proof. As reasoned in Section 8.3.2, both the projection onto our polytope and the soft-thresholding operation are solutions to proximal operators. Hence, by Lemma B.1 both are nonexpansive operators and thus the claimed inequalities hold. □

Moreover, we show the contractivity of our gradient descent method for the differentiable part f of our objective function.

Lemma B.3. Let $f(x) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \frac{1}{2}\lambda(1-\mu)\|\boldsymbol{x}\|_2^2$ be the differentiable part of our function where $\lambda > 0, \mu \in (0, 1)$.
Then, the gradient descent method with a constant step size of $0 < \delta < \frac{2}{L}$ (with L being the Lipschitz constant of the gradient of f) is a contraction.

Proof. Let $0 < \delta < \frac{2}{L}$ be the step size with L as defined in Equation (8.9).
First, we notice that $\nabla^2 f = \boldsymbol{A}^T\boldsymbol{A} + \lambda(1-\mu)\boldsymbol{I}$. Since for any matrix $\boldsymbol{A}$ the product $\boldsymbol{A}^T\boldsymbol{A}$ is positive semi-definite and $\lambda(1-\mu)\boldsymbol{I}$ strictly positive definite, the combination $\nabla^2 f$ is strictly positive definite, i.e., all eigenvalues are positive.
We can explicitly compute the largest eigenvalue as

$$\operatorname{Eig}_{\max}(\nabla^2 f) = \operatorname{Eig}_{\max}\left(\boldsymbol{A}^T\boldsymbol{A} + \lambda(1-\mu)\boldsymbol{I}\right) = \operatorname{Eig}_{\max}(\boldsymbol{A}^T\boldsymbol{A}) + \lambda(1-\mu) \overset{(8.8)}{=} L,$$

where the second equality holds as the addition with a scaled identity matrix shifts all eigenvalues by the scaling, and hence in particular also the largest one.

Now, let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ be arbitrary with $\boldsymbol{x} \neq \boldsymbol{y}$.
Define

$$h\colon [0,1] \to \mathbb{R}^d, \qquad t \mapsto t\boldsymbol{x} + (1-t)\boldsymbol{y},$$

and further define

$$g\colon [0,1] \to \mathbb{R}^d, \qquad t \mapsto h(t) - \delta \nabla f(h(t)).$$

From this we derive

$$g'(t) = \left(\boldsymbol{I} - \delta \nabla^2 f(h(t))\right)(\boldsymbol{x} - \boldsymbol{y}).$$

It follows that

$$\begin{aligned}
\|\boldsymbol{x} - \delta\nabla f(\boldsymbol{x}) - \boldsymbol{y} + \delta \nabla f(\boldsymbol{y})\|_2 &= \|g(1) - g(0)\|_2 \\
&\leq \sup_{\xi \in (0,1)} \|g'(\xi)\|_2 \\
&\leq \sup_{\xi \in (0,1)} \|\boldsymbol{I} - \delta \nabla^2 f(h(\xi))\|_2 \|\boldsymbol{x} - \boldsymbol{y}\|_2 \\
&= \underbrace{\|\boldsymbol{I} - \delta \nabla^2 f\|_2}_{=:\rho} \|\boldsymbol{x} - \boldsymbol{y}\|_2.
\end{aligned}$$

Note that in the mean value theorem for vector-valued functions equality does not hold in general, but the stated inequality is valid.
Trivially, $\rho \geq 0$. Additionally, we already discussed that the eigenvalues of $\nabla^2 f$ are elements of the interval $(0, L]$. Thus, the eigenvalues of $\boldsymbol{I} - \delta\nabla^2 f$ are contained in $[1 - \delta L, 0)$. This leads to the estimation

$$\rho = \|\boldsymbol{I} - \delta \nabla^2 f\|_2 = |1 - \delta L| < 1.$$

Overall, we conclude that

$$\|\boldsymbol{x} - \delta\nabla f(\boldsymbol{x}) - \boldsymbol{y} + \delta \nabla f(\boldsymbol{y})\|_2 \leq \rho \|\boldsymbol{x} - \boldsymbol{y}\|_2, \tag{16.3}$$

where $\rho \in [0,1)$ is independent of $\boldsymbol{x}$ and $\boldsymbol{y}$, which shows that this is a contraction. □

Finally, we accumulate this three auxiliary statements into one result.

Corollary B.4. Let $\boldsymbol{x} \in \mathbb{R}^d$ be any arbitrary starting point. Let f be the function as before. Additionally, assume given factors $0 < \delta < \frac{2}{L}$, $\lambda > 0$ and $\mu \in (0,1)$ as above.
Then, the iteration

$$\boldsymbol{x}^{(k+1)} = \mathcal{P}_{\mathcal{C}}\left(S_{\delta\lambda\mu}\left(\boldsymbol{x}^{(k)} - \delta \nabla f(\boldsymbol{x}^{(k)})\right)\right)$$

builds a contraction.

Proof. With application of our lemmas before we get

$$\begin{aligned}
&\|\mathcal{P}_{\mathcal{C}}\left(S_{\delta\lambda\mu}\left(\boldsymbol{x} - \delta\nabla f(\boldsymbol{x})\right)\right) - \mathcal{P}_{\mathcal{C}}\left(S_{\delta\lambda\mu}\left(\boldsymbol{y} - \delta\nabla f(\boldsymbol{y})\right)\right)\|_2 \\
&\qquad \overset{(16.1)}{\leq} \|S_{\delta\lambda\mu}\left(\boldsymbol{x} - \delta\nabla f(\boldsymbol{x})\right) - S_{\delta\lambda\mu}\left(\boldsymbol{y} - \delta\nabla f(\boldsymbol{y})\right)\|_2 \\
&\qquad \overset{(16.2)}{\leq} \|\boldsymbol{x} - \delta\nabla f(\boldsymbol{x}) - \boldsymbol{y} + \delta\nabla f(\boldsymbol{y})\|_2 \\
&\qquad \overset{(16.3)}{\leq} \rho\|\boldsymbol{x} - \boldsymbol{y}\|_2
\end{aligned}$$

with $\rho \in [0,1)$.
Thus, the iteration is a contraction using the Banach fixed-point theorem. □

References

[1] Tim Roughgarden and Gregory Valiant. *CS168: The Modern Algorithmic Toolbox, Lecture #13: Sampling and Estimation*. Online under `http://timroughgarden.org/s17/l/l13.pdf`. May 2017.

[2] Martin Burger, Alex Sawatzky and Gabriele Steidl. "First Order Algorithms in Variational Image Processing". In: *Splitting Methods in Communication, Imaging, Science, and Engineering*. Springer International Publishing, Jan. 2016, pp. 345–407.