Machine learning for image segmentation

Kaiwen Chang

To cite this version:

Kaiwen Chang. Machine learning for image segmentation. Machine Learning [stat.ML]. Université Paris sciences et lettres, 2019. English. NNT : 2019PSLEM058 . tel-02510662

HAL Id: tel-02510662

https://pastel.hal.science/tel-02510662

Submitted on 18 Mar 2020

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à MINES ParisTech

Machine learning based image segmentation

Apprentissage artificiel pour la segmentation d'image

Soutenue par
Kaiwen Chang
Le 10 décembre 2019

École doctorale n°621
Ingénierie des Systèmes, Matériaux, Mécanique, Énergétique - ISMME

Spécialité
Morphologie Mathématique

Composition du jury :

Nicolas Passat Professeur, Université de Reims Champagne-Ardenne	*Président*
Eva Dokladalova Professeure associée, ESIEE Paris	*Rapporteuse*
Filip Malmberg Professeur, Uppsala University	*Rapporteur*
David Legland Ingénieur de recherche, INRA	*Examinateur*
Jesús Angulo Directeur de recherche, MINES ParisTech	*Directeur de thèse*
Bruno Figliuzzi Maître-assistant, MINES ParisTech	*Maître de thèse*

MINES ParisTech | PSL

Remerciements

Cette thèse a été réalisée au Centre de Morphologie Mathématique de Mines ParisTech, sur le site de Fontainebleau. Tout d'abord je tiens à exprimer ma profonde gratitude à mon encadrant, Bruno Figliuzzi, pour son aide, sa confiance, son soutien, ses conseils et ses remarques pour améliorer ce manuscript, qui m'ont permis de beaucoup progresser pendant ces trois années. Je remercie également Jesús Angulo pour sa confiance en acceptant d'être mon directeur de thèse.

Je souhaiterais remercier aussi les membres de mon jury, M. Nicolas Passat (président du jury), Mme Eva Dokladalova (rapporteuse), M. Filip Malmberg (rapporteur) et M. David Legland (examinateur) pour leur intérêt sur mon sujet de thèse, leurs critiques constructives et leurs conseils, dans l'évaluation de cette thèse et lors de la soutenance.

Je tiens à remercier les membres du CMM, Beatriz Marcotegui, Etienne Decencière, Michel Bilodeau, Fernand Meyer, Dominique Jeulin, Petr Dokladal, Santiago Velasco-Forero, Serge Koudoro, Andres Serna, Samy Blusseau, José-Marcio Martins da Cruz et François Willot pour leur aide et leur soutien. Je remercie également Anne-Marie de Castro, la secrétaire du CMM, pour son accompagnement dans toutes les démarches administratives et sa bienveillance.

Je remercie aussi les thésards et post-doctorants que j'ai pu rencontrer durant ces trois ans : Vaia, Sebastien, Haisheng, Pierre, Amin, Jean-Baptiste, Théodore, Robin, Angélique, Nicolas, Hao, Arezki, Albane, Marine, Laure, Shuaitao, Tianyou, Aurélien, Yuan, Rémi, Eric, Leonardo, François, Elodie, Jean, David et Qinglin, qui ont rendu mon séjour à Fontainebleau plus plaisant.

Je remercie de plus l'ensemble des personnes qui ont contribué d'une manière ou d'une autre à cette thèse. J'aimerais remercier mes professeures de français, Catherine Guerrieri, Françoise Herbet-Pain, Cécile Brossaud et Liyan Sarfis, qui m'ont aidé à améliorer mon français au long des années.

Enfin, merci à toute ma famille, surtout à mes parents, pour leur soutien au cours de mes études.

Résumé

La présente thèse vise à développer une méthodologie générale basée sur des méthodes d'apprentissage pour effectuer la segmentation d'une base de données constituée d'images similaires, à partir d'un nombre limité d'exemples d'entraînement. Cette méthodologie est destinée à être appliquée à des images recueillies dans le cadre d'observations de la terre ou lors d'expériences menées en science des matériaux, pour lesquelles il n'y a pas suffisamment d'exemples d'entraînement pour appliquer des méthodes basées sur des techniques d'apprentissage profond.

La méthodologie proposée commence par construire une partition de l'image en superpixels, avant de fusionner progressivement les différents superpixels obtenus jusqu'à l'obtention d'une segmentation valide. Les deux principales contributions de cette thèse sont le développement d'un nouvel algorithme de superpixel basé sur l'équation eikonale, et le développement d'un algorithme de fusion de superpixels basé sur une adaptation de l'équation eikonale au contexte des graphes. L'algorithme de fusion des superpixels s'appuie sur un graphe d'adjacence construit à partir de la partition en superpixels. Les arêtes de ce graphe sont valuées par une mesure de dissimilarité prédite par un algorithme d'apprentissage à partir de caractéristiques de bas niveau calculées sur les superpixels.

A titre d'application, l'approche de segmentation est évaluée sur la base de données SWIMSEG, qui contient des images de nuages. Pour cette base de données, avec un nombre limité d'images d'entraînement, nous obtenons des résultats de segmentation similaires à ceux de l'état de l'art.

Abstract

In this PhD thesis, our aim is to establish a general methodology for performing the segmentation of a dataset constituted of similar images with only a few annotated images as training examples. This methodology is directly intended to be applied to images gathered in Earth observation or materials science applications, for which there is not enough annotated examples to train state-of-the-art deep learning based segmentation algorithms.

The proposed methodology starts from a superpixel partition of the image and gradually merges the initial regions until an actual segmentation is obtained. The two main contributions described in this PhD thesis are the development of a new superpixel algorithm which makes use of the Eikonal equation, and the development of a superpixel merging algorithm stemming from the adaption of the Eikonal equation to the setting of graphs. The superpixels merging approach makes use of a region adjacency graph computed from the superpixel partition. The edges are weighted by a dissimilarity measure predicted by a machine learning algorithm from low-level cues computed on the superpixels.

In terms of application, our approach to image segmentation is finally evaluated on the SWIMSEG dataset, a dataset which contains sky cloud images. On this dataset, using only a limited amount of images for training our algorithm, we are able to obtain segmentation results similar to the ones obtained with state-of-the-art algorithms.

Contents

List of figures

List of Tables

Chapter 1

Introduction

Background

Image segmentation, sometimes referred to as perceptual grouping, constitutes an important area of research in image processing. It refers to the process of partitioning an image into several regions that are perceptually similar, because they share common colors or textures, or because they correspond to objects of interest in the image. This research topic has been widely studied over the years and remains currently very active.

Segmentation results are useful in many computer vision applications. In stereo and motion estimation problems, segments often provide regions of support for computing correspondence. In higher level problems such as recognition or image indexing, segmentation techniques are classically used to solve figure-ground separation or recognition by parts problem.

It is important to note that image segmentation is a distinct task from image classification. For the latter, the seeked objective is to assign to each pixel the label of the class to which it belongs. The vision of image segmentation is more global and seeks to identify regions formed by pixels that share similar characteristics and to divide the whole image into regions corresponding to different parts in the scene. The distinction becomes blurry when considering end-to-end semantic segmentation tasks, where the segmentation is achieved through the classification of pixels.

Segmentation is also different from contour detection, which aims at finding the contours in the image. Contours indicate the existence of different objects, but not all the contours are useful depending on the scale of interest or on the vision task under consideration. Contour detection is often used as a first step for segmentation. Nevertheless, an actual segmentation is only obtained when relevant contours have been selected and when some algorithm has grouped them into closed contours.

Image segmentation is a challenging task and there is currently no comprehensive

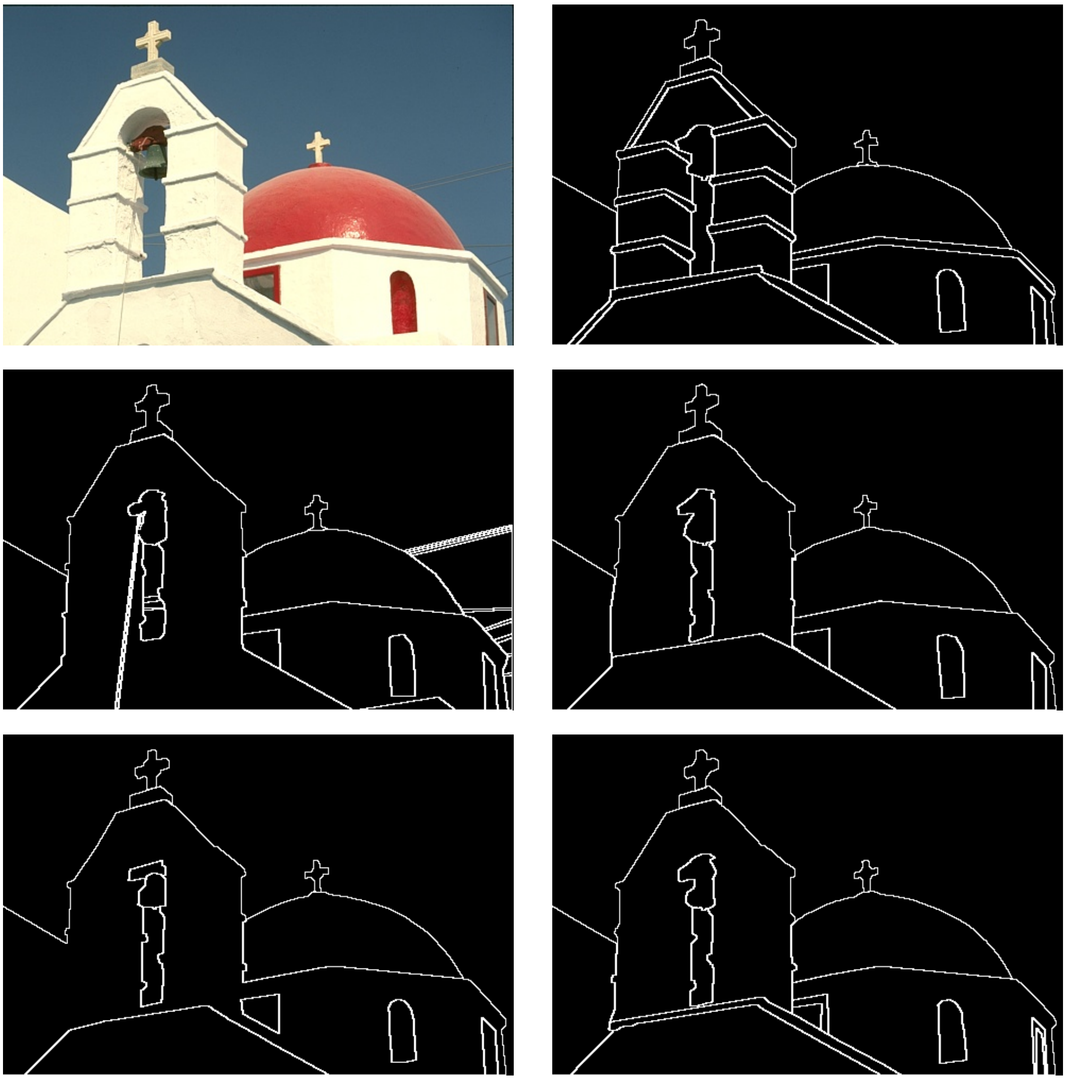


Figure 1.1 – Image 118035 in BSDS500 database [Mar01] and its five ground truth images segmented by different human subjects.

theory in this field, not least because a given segmentation is often aimed at a specific application. The methods based on gradient can be disturbed by noise or textured patterns in the image, thus producing a contour where is actually not. In addition, it is difficult to find a contour if two regions share similar colors or gray levels, or when a contour corresponds to small gradient. Moreover, the result of a segmentation depends on the applications and on the scale of the objects of interest. Even human based segmentations are characterized by a significant variability in terms of result. A hierarchical segmentation, which provides a set of segmentations adapted to distinct scales, is therefore desirable for some image processing tasks.

Literature review

A large amount of research has been conducted on image segmentation, based upon a variety of techniques including active contours, clustering, region splitting or merging, etc. These methods can be roughly classified into contour based methods and region based methods, or local methods and global methods. For contour based methods, contour detection algorithms are first used to obtain candidate contours. Then one has to find a way to close the contours as in [RFM05; Arb09]. For region based methods, the image is oversegmented into small regions, which are merged hierarchically to get a segmentation.

In this section, we briefly review the vast literature on image segmentation. We distinguish, somewhat arbitrarily, between active contours, morphological algorithms, clustering algorithms, graph-based algorithms, and machine learning based algorithms. New trends in image segmentation, including CNN based segmentation and semantic segmentation are also briefly discussed. Rather than describing in depth each approach to image segmentation, this section was written to illustrate the huge variety of mathematical tools that are classically used to perform image segmentation.

Active contours approaches

Active contour algorithms work by simulating the evolution of a closed curve on the image. These algorithms need user provided curve for initialization, and rely on energy minimization or on forces related to the local properties in the image to recover the contours of interest.

In 1988, Kass *et al.* [KWT88] proposed an algorithm based on the minimization of the energy of splines for contour detection, referred to as *snakes*. The energy functional comprises three terms: one term corresponding to some internal energy, one term leading to image forces and one term leading to external constraint forces, respectively. The internal energy term imposes a piecewise smoothing on the curve. Image forces push the curve towards relevant image features, such as lines, edges and subjective contours. An initial curve is placed near the desired contour, adding an external constraint to the snake. Nearby edges are then localized by the algorithm through the minimization of the energy term.

Malladi *et al.* presented in 1995 [MSV95] a level set approach for shape modeling. In their approach, an initial contour is provided by the user, which lies inside a shape, or encloses all the constituent shapes. Then the front propagates as the zero level set of a higher dimensional function, inward or outward in its normal direction, with a speed related to the local image gradient to find the contours of interest. Compared to snakes, this approach is able to recover shape with protusions, and the front can be easily splitted to recover more than one object.

Morphological segmentation

Morphological segmentation approaches provide well localized and closed contours, usually at the price of significant oversegmentation. A large amount of research has been conducted over the years to reduce the oversegmentation.

The archetypal morphological technique for image segmentation is the watershed algorithm, which was introduced in 1979 by Beucher and Lantuéjoul [BL79] for contours detection in gray level images. It is non-parametric and provides closed contours. The algorithm is inspired by the geographic notions of catchment basins and watersheds, and provides mathematical definitions of these notions based on the geodesic distance. The contours are defined as the watersheds of the variation function (gradient modulus) of the gray levels. Applications including fractures detection in steel and bubble detection in radiography are detailed in the original article.

In 1990, Meyer and Beucher [MB90] wrote a review on image segmentation methods based on the watershed transform and the homotopy modification. The oversegmentation problem is solved by introducing markers within the objects of interest. Lower-complete and upper complete-functions are defined to solve the undersegmentation problem due to broken contours and overlapping particles, that take into account shape information. Methods for markers design under various situations are illustrated by examples of electrophoresis gel, traffic lane segmentations and images in other domains. The watershed algorithm is also extensively used in medical imaging applications [Pas07].

Later, Beucher [Beu94] proposed a hierarchical segmentation algorithm based on the mosaic image and the waterfall algorithm to reduce the oversegmentation produced by watershed algorithm. A valued graph is constructed from the mosaic image. Each edge in the valued graph is weighted by the minimal value of the gradient on the corresponding image boundary. A hierarchical image is obtained by assigning to each catchment basin the minimum gradient of its surrounding arcs. The segmentation is computed by applying a watershed transform on the hierarchical image. The waterfall algorithm selects significant minima through reconstruction of the gray level function by the watershed. These significant minima are used as markers for the watershed. The conclusion is that the waterfall algorithm is more general than the hierarchical algorithm, and easier to use than the dynamics.

In 2007, Angulo and Jeulin [AJ07] introduced a stochastic segmentation method based on the watershed algorithm. In their method, M realizations of N random germs are generated independently according to Poisson distribution, resulting in M marker images. Next, a watershed segmentation is computed with the M marker images and a probability density function of contours is calculated using a Parzen window method. The probability density function is then segmented using a volumic watershed transform to obtain the R most significant regions. The random generation of markers helps to select contours that are robust to the variations of the segmentation conditions. Two other random germs frameworks are discussed in [AJ07], namely regionalised random germs, and uniform random germs leveling. The last framework yields the best results and outperforms the standard watershed algorithm when the aim is to segment complex images into a few regions. One of the main difficulties associated with the stochastic watershed algorithm is the computation of the probability density function for the contours. In 2014, [MH14] presented a quasi-linear algorithm enabling to compute exactly the probability density function.

Two improvements of the stochastic watershed were proposed in 2013 by Berander *et al.* [Ber13]: adding uniform noise to the input image at every iteration, and distributing the markers using a randomly placed uniform grid (with random offset and rotation). With these modifications, images with larger variabilities in region size are better segmented, the result is less sensitive to the number of markers, and the depth threshold becomes easier to set. The F-measure obtained on a database of fluorescent microscope images of nuclei was largely improved with this method.

In 2015, Meyer [Mey15] presented a hierarchical segmentation strategy based on stochastic watershed, and provided various strategies to construct hierarchies. In his study, a minimum spanning forest is used to construct a hierarchy based on prioritized markers. The strength of a contour is estimated by the level of hierarchy for which its two associated regions merge. Various features including size, contrast, orientation, color, and texture features can be considered with different hierarchies. One has nevertheless to tailor the hierarchy according to the problem at hand.

During the same year, Franchi and Angulo [FA15] proposed a fully unsupervised multi-scale approach to address the drawback of stochastic watershed of enhancing insignificant contours in complex images, referred to as bagging stochastic watershed (BSW). The gPb gradient in [Arb11] is employed when calculating the probability density functions. Probability density functions at different scales are combined to compute the BSW. This algorithm is evaluated on the BSDS dataset and compared with other stochastic watershed algorithms. The ISW approach [Ber13] achieves the best F-measure of all stochastic watershed based approaches.

Clustering algorithms

Another family of segmentation methods is based upon the notion of clustering or grouping. It is intuitive to regard segmentation as a clustering problem, since pixels with similar color and texture are more likely to belong to the same object. Mean-shift and other mode finding techniques, such as K-means and mixtures of Gaussians try to find clusters in the distribution of points in the feature space to compute the segmentation [Sze11].

In 2002, Comaniciu and Meer [CM02] proposed a general nonparametric technique for multimodal feature space analysis, which is able to delineate arbitrarily shaped clusters based on mean shift. Mean shift is defined as the difference between the weighted mean of points and the center of the density kernel. Comaniciu and Meer proved that recursive mean shift can be used to detect the modes of the density. The application of mean shift based mode detection in image segmentation with joint spatial-range domain representation overcomes the issue of gray or color clustering algorithms of oversegmenting small gradient regions. The only parameter to set is the resolution of analysis, which depends on the vision task.

Paris and Durand described in 2007 [PD07] a new fast algorithm for mean-shift image segmentation computation based on Morse theory, and introduced a way to build a segmentation hierarchy. Gaussian kernels are employed for density estimation, and the density function is computed on a regular grid. The mode extraction is done explicitly by assigning to each pixel the label of the corresponding maxima, in decreasing order of density. A persistence of boundary based on the topological persistence is defined, and clusters with boundary persistence less than a threshold are merged for simplification. A segmentation hierarchy is therefore obtained by gradually increasing the threshold. The boundary persistence can be modified to account for color information, depending on the application. PCA can be used to reduce the dimension of the feature space therefore accelerating the computation.

Clustering algorithms can also be employed for superpixels generation, as Simple Linear Iterative Clustering (SLIC) superpixels [Ach12] proposed by Achanta *et al.* based on K-means clustering algorithm. SLIC is one of the state-of-the-art superpixel algorithms. More details on this algorithm can be found in section 2.2.

Graph-based approaches

Graph based segmentation methods transform image segmentation problems into graph partitioning problems by representing an image by a graph constructed by linking adjacent pixels or regions. The aim is to segment an image into regions with low variability within each region and high variability between regions.

In 2000, Shi and Malik [SM00] proposed a novel graph-theoretic criterion to measure the effectiveness of an image partition, the normalized cut, and an efficient technique for the minimization of this criterion based on a generalized eigenvalue problem. This algorithm extracts global impression and obtains good results on static images and motion sequences.

In 2004, Felzenszwalb and Huttenlocher [FH04] introduced a graph-based image segmentation method based on pairwise region comparison. In their approach, pixels are merged according to their intensity differences across boundaries and between neighboring pixels within each region. The segmentation criteria are adaptively adjusted to take into account the variability in neighboring regions. Two different kinds of local neighborhoods, namely grid graphs and nearest neighbour graphs, are tested for graph construction. The advantage of this approach is that it preserves details in low-variability regions while ignoring details in high-variability regions.

The literature on graph based approaches, including minimum spanning tree based algorithms and graph cut algorithms, is reviewed in greater details in section 5.1.

Machine learning based methods

Learning based methods have increasingly gained popularity for performing image segmentation over the past 20 years. In these methods, machine learning algorithms, such as linear regression, support vector machines (SVM) or random forests are used to combine various cues, providing a merge probability for region merging, or a measure of similarity for graph based approaches for image segmentation. Machine learning based methods can be generalized to other kinds of images by changing the training images. The choice of features is essential for machine learning based algorithms.

In 2003, Ren and Malik [RM03] proposed a two class classification method for grouping. In this work, natural images segmented by humans are taken as positive examples. Varieties of image features from the classical Gestalt cues, including contour, texture, brightness and good continuation features are calculated for both inter-region and intra-region cases. The effectiveness of each feature is evaluated through information entropy, and the conclusion is that boundary contour is the most informative grouping cue. A logistic regression classifier is used for learning the algorithm parameters. A globally optimized segmentation is found by using a random search based on simulated annealing. The method is tested on the Corel Image base [WLW01].

In 2004, Martin *et al.* [MFM04] proposed a natural image boundaries detection method where local brightness, color, and texture features, including oriented energy, brightness gradient, color gradient, and texture gradient are employed. A methodology for benchmarking boundary detection algorithms is also defined in their study. Human-marked boundaries from the Berkeley Segmentation Dataset [Mar01] are used as ground truth. A logistic regression classifier is trained to predict the posterior probability of a pixel being on a boundary based on the combination of cues. Their results reveal the importance of texture treatment in natural images boundaries detection.

In 2006, Dollár *et al.* [DTB06] presented a novel supervised learning algorithm for edges and boundaries detection. In the proposed algorithm, an extension of probabilistic boosting trees is trained based on tens of thousands of simple features at multiple scales and orientations from large image patches centered at different locations, taking into account low, mid, and high level information. The probability for each location to belong to a boundary is calculated independently. The performance of this method on the BSDS300 dataset is comparable to that of Pb [MFM04]. Good results are also achieved in object boundary detection and rode detection applications.

A machinery for contour detection and hierarchical segmentation was proposed in 2011 by Arbeláez *et al.* [Arb11], where multiple local cues (brightness, color and texture gradients) at multiple scales and orientations are combined into a globalization framework to predict a probability of boundary. An Oriented Watershed Transform (OWT) and Ultrametric Contour Map (UCM) are then employed to close the contours and to produce a hierarchical segmentation. This algorithm significantly outperforms competing algorithms on each dataset (BSDS, MSRC, PASCAL 2008) and for each considered benchmark criterion.

In 2012, Alpert *et al.* [Alp12] presented an image segmentation approach by probabilistic bottom-up aggregation. The proposed algorithm starts from a graph where each pixel in the image is represented as a node. Pixels are then gradually aggregated to produce larger regions, according to a probabilistic model that considers the distribution of intensity difference and of texture difference to output a probability of merging adjacent regions. A graph coarsening scheme is integrated for merging. A novel evaluation scheme is proposed and this algorithm achieves a high average F-measure with the least number of fragments both in one-object and two-objects data set.

In the same year, Arbeláez *et al.* [Arb12] proposed a new design for region-based object detectors, which can integrate top-down information from a scanning-windows part model, and global appearance features (shape, color and texture). This design focuses especially on the recognition of challenging articulated categories. This algorithm achieves competitive performance on PASCAL segmentation challenge and the highest accuracy on articulated objects VOC2010 dataset.

In 2014, Arbeláez *et al.* [Arb14] proposed a unified bottom-up hierarchical image segmentation and object candidate generation algorithm named Multiscale Combinatorial Grouping (MCG). In their approach, they firstly construct a multiresolution image pyramid by subsampling and supersampling the original image. A hierarchical segmentation is then computed at each scale of the pyramid, and the hierarchies are finally aligned and combined into a single segmentation hierarchy. A classifier is trained to combine the contour strength from different scales. Compared to [Arb11], more cues including sparse coding on patches and structured forest contours are added for single scale segmentation, and an efficient normalized cut algorithm is designed to accelerate the globalization step. This approach provides state-of-the-art segmentations and object candidates on BSDS500 and PASCAL 2012 datasets.

In 2015, Dollár and Zitnick [DZ15] proposed a generalized structured learning approach for edge detection, taking advantage of inherent structures in image patches by using structured forests. This approach is different from the aforementioned methods, in the sense that it detects edges by predicting a segmentation mask for each image patch. A feature vector comprising color, gradient magnitude at various scales and orientations and differences of downsampled pixel pairs is extracted for each patch. A mapping function is designed to map the structured labels of a patch to an intermediate space where similarity can be approximated by the Euclidean distance. A structured forest classifier is trained for edge map prediction. The edge maps can be sharpened using local color cues. This approach achieves state-of-the-art edge detection results on the BSDS500 dataset and NYU depth dataset, and has realtime performance.

CNN based image segmentation

For machine learning based image segmentation, it is essential to involve global information, or information from multiple scales. The performance of the segmentation depends indeed strongly on the set of selected features. To improve the performance of segmentation, complicated handcrafted features are involved in state-of-the-art algorithms [Arb11; Arb14]. The application of convolutional neural networks on segmentation can potentially relieve us from designing and selecting features. By convolving an image with learned filters, nonlinear mapping and pooling, a feature map can be produced comprising both low, mid and high level features.

Their performance and ease of use make CNN based segmentation algorithms popular nowadays. In this chapter, we do not intend to give a thorough description of the literature on deep learning based segmentation methods, and we briefly mention a few classical articles in this section. A thorough review can be found in [Gho19].

One of the first articles on CNN based segmentation was published in 2013 by Farabet *et al.* [Far13], who presented a scene parsing approach based on a multiscale convolutional network. In their work, a three-staged ConvNet is applied to each scale of a Laplacian image pyramid for extracting features vectors. Network weights are shared across scales to enforce the scale-invariance of the learned features. Then, features maps from multiple networks are upsampled and concatenated into a map of features vectors, characterizing regions of multiple sizes centered on each pixel. The second step of the algorithm consists of a postprocessing step to ensure the spatial consistency of the scene labeling. Three strategies are tried: averaging class distribution within superpixels, minimizing the energy of a Conditional Random Field constructed from the labeling and applying a multilevel cut with a class purity criterion. This approach yielded record accuracies on the SIFT Flow dataset and on the Barcelona dataset.

R-CNN (regions with CNN features) [Gir14] was proposed in 2014 for object detection and semantic segmentation based on region proposals. Part of the architecture of R-CNN involves an AlexNet architecture pre-trained on the ILSVRC dataset for object detection. Region proposals are generated by selective search, and wrapped to the input size of the CNN. A fixed-length feature vector is then extracted for each region. A linear SVM is finally trained for each class for classification. The extension of R-CNN to segmentation also achieves state-of-the-art performance on VOC 2011. This article adapts the CNN for image classification to object detection task, and shows the effectiveness of supervised pre-training and domain specific fine-tuning, offering a practical solution for tasks with small amount of labeled data for instance object detection.

In [LSD15], fully convolutional networks were proposed for semantic segmentation, which adapt the learned representations of classification networks to achieve an end-to-end segmentation, accepting arbitrary-sized inputs. In this approach, the fully connected layers of classification networks are notably converted into convolution layers. The feature map is upsampled to the size of inputs by deconvolution. To improve the resolution of the prediction, a skip architecture is designed to combine the semantic information from a deep layer with appearance information from a shallow layer. This algorithm improves the state of the art on PASCAL VOC 2011, NYUDv2 and SIFT Flow dataset, and is quick for inference compared to patch based segmentation.

Objective of the thesis

As pointed out in the previous section, state-of-the-art segmentation algorithms make extensive use of machine learning techniques. CNN-based algorithms require to be trained on a large number of segmentation examples, for instance on image datasets including Common Objects in COntext (COCO) [Lin14] or PASCAL VOC [Eve10], which comprise tens of thousands of annotated images. Algorithms based on more traditional machine learning methods where features are mostly handcrafted, like the gPb algorithm, can be trained on smaller datasets. gPb is for instance trained on the Berkeley Segmentation Dataset, which contains 200 images for training.

In a number of practical applications, notably including images obtained in remote sensing or materials science applications, the lack of annotated images constitutes a significant issue if one wants to apply machine learning based image segmentation algorithms. A common solution for applying deep learning algorithms when few training data is available is to fine-tune an architecture of neural network pre-trained on a large database of images, a technique referred to as transfer learning in the literature. The idea behind transfer learning is that the pre-trained neural network will have identified useful features on the large dataset of images on which it is pre-trained, and that these features remain relevant for processing the new set of images. Nevertheless, due to the specificity of the images gathered in materials engineering applications or on remote sensing observations, transfer learning techniques largely remain inoperative.

In this PhD thesis, our objective is to work on the development of algorithms that can be trained to perform the segmentation of a dataset constituted of similar images with only a few annotated images as training examples. Such datasets are commonly obtained when experiments are conducted in materials science.

Our proposed methodology is based on a region segmentation approach. More precisely, starting from a superpixel partition of the image, we propose a merging algorithm which gradually merges the initial regions until an actual segmentation is obtained. The work presented in this PhD manuscript focuses on the two key aspects of the proposed methodology, namely the superpixel generation and the superpixel merging.

Thesis outline

In the first part of the manuscript, corresponding to chapters 2 and 3, we introduce two novel algorithms that can be used to compute a superpixel partition and a hierarchical superpixel partition of the image, respectively. The superpixel algorithm presented in chapter 2 makes use of the Eikonal equation to generate the superpixel partition, in a way relatively similar to the algorithm introduced by Buyssens *et al.* [Buy14b]. The algorithm compares favorably to other state-of-the-art superpixel algorithms on the Berkerley Segmentation Dataset. In chapter 3, which is somehow independent from the rest of the manuscript, we describe a segmentation algorithm based on the wavelet transform and on the watershed transform that yields a hierarchical superpixel segmentation of the image. The developed algorithms have been published at ISMM 2017 [FCF17] and ISMM 2019 [CF19] conferences.

In the second part of the manuscript, corresponding to chapters 4 and 5, we present an algorithm performing superpixels merging based upon learned similarities between adjacent regions. The superpixels merging approach is conducted in the framework of graph theory. More precisely, a region adjacency graph is computed from the superpixel partition. Each superpixel in the partition corresponds to a node in the graph, while each pair of adjacent superpixels are linked by an edge. The edges are weighted by a dissimilarity measure, which is taken to be the probability that both superpixels belong to distinct segments of the final segmentation. A machine learning algorithm is used for estimating the dissimilarity, which takes as input features various characteristics of the regions including their respective colors and textures, or the strength of the gradient at the boundary. The learning process, as well as the characteristics used to compute the similarity, are discussed in chapter 4.

In chapter 5, we propose a method using the Eikonal equation to perform the clustering of the region adjacency graph, in a similar way to the superpixel approach proposed in chapter 2. The Eikonal equation is notably adapted to the specific setting of graphs. The proposed approach is finally compared to classical approaches for performing region mergings including the normalized cut algorithm [SM00] and the segmentation by weighted aggregation [Alp12]. The comparison is conducted on the Berkeley Segmentation Dataset.

To conclude the manuscript, we present in chapter 6 the application of our methodology to a dataset of cloud images, where our approach yields state-of-the-art results with a limited number of training instances. Conclusions are finally drawn in the last chapter.

Chapter 2

Fast marching based superpixels

Superpixel algorithms refer to a class of techniques designed to partition an image into small regions that group perceptually similar pixels. Superpixel segmentations are often used as a preprocessing step for computer vision tasks, such as image segmentation [FVS09], object detection [Yan15], tracking [Wan11; YLY14], depth estimation [ZK07] or object classification, to simplify the computation for further processing or to provide computational support for features.

This chapter introduces an algorithm based on the fast marching method to compute the superpixel partition of an image. The fast marching algorithm finds several applications in mathematical morphology and stochastic geometry [DD03; Fig16; Bor18; Fig19]. In section 2.1, we discuss general properties of superpixels and we provide a review of the main state-of-the-art superpixel algorithms; in section 2.2, we elaborate on the fast marching algorithm and we describe in greater detail the proposed fast marching based superpixel (FMS) algorithm. The results of the FMS are presented in section 2.3 and compared with other superpixel algorithms. We draw conclusion and present perspectives in the last section.

Introduction

The idea of over-segmenting an image into regions for further processing existed long before the term “superpixel” was coined, for example in [MM97]. It was in [RM03] that this term was first introduced in the context of image segmentation.

Superpixel segmentations are over-segmentations which obey certain properties, including:

1. **Boundary adherence**: superpixels should preserve the boundaries in the image [Ach12; SHL17]. This property is for instance of paramount importance for segmentation applications. Several metrics have been defined in the literature to quantify boundary adherence, including *boundary recall*, which characterizes the amount of

boundary pixels that are recovered by the segmentation algorithm within a tolerance of 1 or 2 pixels, and *undersegmentation error*, which quantifies the “leakage” of the superpixels crossing an actual boundary of the image.

2. **Regularity, smoothness**: The shape of the superpixels should be regular and their boundaries should be smooth where there is not an object boundary. Obviously, algorithms returning superpixels with highly tortuous contours are more likely to exhibit high adherence to boundaries, but the resulting partition can hardly be considered to be satisfactory. Criteria including *average compactness* or *superpixels contours density* are classically used to characterize the shape of the superpixels [MDW14; Mac15; SHL17].

3. **Efficiency**: Superpixels should be fast to compute and memory efficient. Since superpixel segmentation is often used as a preprocessing step to simply further processing, its application should not consume much time, especially for real time tasks.

Literature review

A lot of research has been conducted on superpixel algorithms. In [SHL17], a thorough literature review of this field is provided, as well as a benchmark for comparison. [Wan17] provides a simple classification of these algorithms between graph-based methods and clustering based methods.

Graph based superpixels

Graph based methods build upon a graph representation of the image. In this representation, each pixel constitutes a node of the graph and all couples of adjacent pixels are linked by an edge. The weight of an edge is usually interpreted as the similarity or dissimilarity between neighboring pixels, defined by their intensity, color or spatial differences. In this way, the segmentation problem is transformed into a graph partitioning problem. Classical graph-based algorithms for superpixels generation include the normalized cut algorithm (NC) [SM00; Mal01] and the efficient graph-based segmentation algorithm (GS) [FH04].

To give a general view of graph based superpixel algorithms, we elaborate on the GS algorithm. In this approach, a graph is constructed either by connecting pixels in a 8-connected neighborhood, or by connecting nearest neighbor pixels in a feature space. In the first case, the weight of an edge is taken to be the absolute value of the intensity difference; in the second case, the weight can be chosen to be the Euclidean distance or other form of distance defined on the features space. Then, the edges of the graph are sorted in a non decreasing order according to their weights and visited in this order. At each iteration, if two nodes v_i and v_j are not in the same component, and if the weight w_{ij} is smaller than the minimum internal difference plus a threshold of each component C_i, C_j, meaning that the inter-region difference is smaller than the within-region difference, the two components are merged. The internal difference of a component is defined as the maximum weight of its minimum spanning tree. The threshold term can be designed according to the size or shape of preferred regions. This procedure is iterated until all edges are processed. The complexity of the algorithm is $O(m \log m)$, where m is the number of edges.

Some graph based algorithms produce superpixels through the optimization of an objective function. Entropy rate superpixels (ERS) [Liu11] is one of these algorithms that achieves state-of-the-art performance. An objective function based on entropy rate is defined, where entropy rate is the asymptotic conditional entropy of a random process, which measures the remaining uncertainty after observing its past trajectory. A balancing term is introduced in the objective function to favor superpixels with similar sizes. The oversegmentation is obtained by selecting edges that form a partition with K connected subgraphs. It is proved that the optimization of this objective function can be done efficiently by a lazy greedy algorithm with a complexity of approximately $O(|V| \log |V|)$, where V is the set of the graph vertices (pixels). This algorithm achieves better performance both in undersegmentation error and in recall than GS on the Berkeley segmentation benchmark [Mar01].

Clustering based superpixel algorithms

Clustering based methods for superpixel segmentation proceed by iteratively refining clusters of pixels until some convergence criterion is met. Clustering based methods notably include mean shift [CM02], watershed [VS91; BL79], turbopixel [Lev09], simple linear iterative clustering (SLIC) [Ach12; AS17] and waterpixel [MDW14; Mac15; CJ19] algorithms, respectively.

The simple linear iterative clustering (SLIC) algorithm [Ach12; AS17] is an archetypal example of clustering based algorithm. Due to its ease of use and its good performance, SLIC is ranked among the most used algorithms for computing a superpixel segmentation. In addition, the SLIC algorithm has a linear complexity, its implementation is available [Ach12] and it offers the possibility to weight the trade-off between boundary adherence and shape or size similarity.

SLIC constructs a superpixel partition by applying a k-means clustering algorithm on local patches of the image. During initialization, K cluster centers locations are selected on the image using a grid with uniform spacing S. Each pixel is then associated to the closest cluster center in the image according to a distance involving the color proximity and the physical distance between the pixel and the seed, respectively. The search for similar pixels is restricted to a neighborhood of size $2S$ by $2S$ around each cluster center. After the assignment step, the K cluster centers are updated. The color and the location associated to the cluster centers are set equal to the average color and location of the pixels of the cluster. The L^2 distance between the previous and the new locations is used to compute a residual error E. The aforementioned procedure is iterated until the error E converges. The clusters of pixels obtained after this procedure are usually not connected. A post-processing step is therefore applied to reassign the disjoint pixels to nearby superpixels.

Eikonal based region growing Clustering (ERGC) [Buy14b; BGR14; Buy14a] is a clustering based superpixel algorithm that produces superpixels by solving the Eikonal equation which describes the label propagation from initial seeds, with a velocity based on color distance between the pixel and the seed. It achieves state-of-the-art performance and is efficient to calculate, with a complexity of $O(n \log n)$, n being the number of pixels in the image. Another advantage of ERGC compared to SLIC is that no post-processing is required to obtain connected superpixels.

Other trends

Other superpixel algorithms are based on energy maximization. Superpixels Entracted via Energy-Driven Sampling (SEEDS) [Ber12] is one such algorithm that can be calculated in real time. Van den Bergh *et al.* proposed an objective function composed of two terms, one based on the color density distribution of each superpixel, the other, optional, used as a smoothing term based on the histogram of superpixel labels in a patch. This objective function is optimized by hill-climbing optimization, namely to make small local changes of superpixel labels at each iteration. The optimization is done in a hierarchical manner, by moving large blocks of pixels in early iterations, latter small block of pixels, at last single pixels to neighboring superpixels. It can achieve better boundary adherence than SLIC and ERS, and is faster to calculate. The main disadvantage of SEEDS is that it yields highly irregular boundaries.

Most superpixel algorithms make use of color and spatial information. Recently, several studies have demonstrated that using texture information could improve the superpixel performance. Xiaolin Xiao *et al.* [XGZ17] proposed to introduce both texture and gradient distance terms in SLIC's distance formula. In their work, the weight of each term is adapted to the discriminability of the features in the image. Their algorithm achieves better performances than state-of-the-art algorithms including SLIC and LSC [LC15]. Giraud *et al.* [Gir19] proposed to measure the texture distance between a pixel and a superpixel seed by considering the average distance between a square patch around that pixel and similar patches inside the superpixel found by a nearest neighbor method.

The ERGC algorithm gives promising results, but there are still some aspects worth of researching. Firstly, we can note that the position of the initial seeds do not change during the process, while more seeds might be needed for complicated regions. Secondly, it might be interesting to take the texture information into consideration to compute the image partition.

In the next subsection, we introduce a novel clustering-based algorithm for generating a superpixel partition of a given image termed fast-marching based superpixels (FMS). Following an idea originally introduced for the Eikonal-based region growing for efficient clustering algorithm (ERGC) [Buy14b; Buy14a; BGR14], we rely upon the Eikonal equation and the fast marching algorithm to assign the pixels of the images to the relevant clusters.

Eikonal equation and fast marching algorithm

A clustering based superpixel algorithm generally comprises 2 steps:

1. Initialization. A group of initial seeds should be provided as a basis for clustering.
2. Clustering. The pixels are clustered to their corresponding seeds based on the distance. The seeds can be updated after a pass of assignment. Thus the superpixel segmentation transforms to the following problem: knowing the position and properties of initial seeds, how can we assign a label to each pixel, in order for the resulting partition to have "good" properties (e.g. boundary recall, compactness, etc.).

The idea behind the Fast Marching Superpixel (FMS) algorithm is to draw an analogy between waves propagating in a heterogeneous medium and regions growing on an image at a rate depending on the local color and texture. Instead of using clustering algorithms, such as k-means clustering or DBSCAN clustering [She16], we propagate the labels of the seeds with a process that can be described by the Eikonal equation. Fast computational algorithms are available to approximate the solution of the Eikonal equation, including the fast marching algorithm [Set96; Set99a; DD03]. We elaborate on the Eikonal equation and the fast marching algorithm in this section.

Eikonal equation

The Eikonal equation is a non-linear partial differential equation which describes the propagation of waves in a medium. It finds notable applications in fields including geometrical optics or geophysics. In this section, we will first present classical results relative to the Eikonal equation on continuous domains, before studying its generalization to discretized domain.

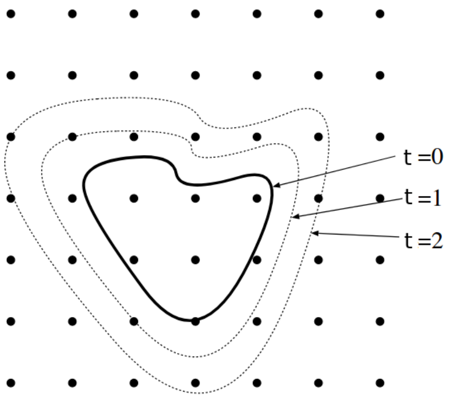


Figure 2.1 – Propagation front and level sets.

Eikonal equation on a continuous domain

Let Ω denote some open domain in $\mathbb{R}^2$. In what follows, we consider a front Γ_t propagating on Ω in the normal direction at a velocity $u := u(x)$ at every point $x \in \Omega$. Obviously, the equation describing the propagation of the front is

$$\frac{d\Gamma_t}{dt}(x,t) = u(x)\mathbf{n_\Gamma}(x), \tag{2.1}$$

where $\mathbf{n_\Gamma}$ is a unitary vector directed toward the direction normal to the propagation front.

A popular approach to solve equation (2.1) is to rely on level sets.

Definition 2.3.1. A level set $\mathcal{L}$ on $\Omega \subset \mathbb{R}^2$ is a subset of Ω where some real valued function $v : \Omega \to \mathbb{R}$ takes a constant value $c \in \mathbb{R}$

$$\mathcal{L} := \{x \mid v(x) = c\}. \tag{2.2}$$

For $t \geq 0$, the propagation front Γ_t can be represented as the level set of a function $v : \Omega \times \mathbb{R} \to \mathbb{R}$ such that

$$\forall t \geq 0, \Gamma_t := \{x | v(x,t) = 0\}. \tag{2.3}$$

Proposition 2.3.1. *Let us consider a close propagation front Γ_t and a function v such that*

$$v(x,t) \begin{cases} < 0 \textit{ if } x \textit{ is inside the area delimited by } \Gamma_t \\ = 0 \textit{ if } x \in \Gamma_t \\ > 0 \textit{ if } x \textit{ is outside the area delimited by } \Gamma_t \end{cases} \tag{2.4}$$

Then, the outward normal direction with respect to the propagation front is given by

$$\mathbf{n_{\Gamma_t}} := \frac{\nabla v(x,t)}{\|\nabla v(x,t)\|}, \tag{2.5}$$

where ∇ is the spatial gradient operator.

At this point, let us consider a trajectory $t \to y(t) \in \Gamma_t$. Necessarily, for $t \geq 0$, we have

$$v(y(t), t) = 0. \tag{2.6}$$

Hence, by differentiating with respect to t, we can easily show that the function v is the solution of the partial differential equation

$$\frac{\partial v}{\partial t}(y(t), t) + \nabla v(y(t), t) \cdot \dot{y}(t) = 0. \tag{2.7}$$

According to proposition 2.3.1, we have

$$\dot{y}(t) = u(y(t))\mathbf{n}_{\mathbf{\Gamma_t}} = u(y(t))\frac{\nabla v(x,t)}{\|\nabla v(x,t)\|}. \tag{2.8}$$

This leads to the so-called *Eikonal equation* defined for all $x \in \Omega$ by

$$\frac{\partial v}{\partial t}(x, t) + u(x)\|\nabla v(x, t)\| = 0. \tag{2.9}$$

In this work, we will present a method for solving the Eikonal equation through a stationary approach. To that end, let us denote by $x \to T(x)$ the function associating to each point $x \in \Omega$ the arrival time of the propagation front Γ_t. In mathematical terms, T is defined by the implicit equation

$$T(y(t)) = t, \tag{2.10}$$

for all trajectory $t \to y(t) \in \Gamma_t$. When we differentiate this equation with respect to t, we obtain the equation

$$u(y(t))||\nabla T(y(t))|| = 1. \tag{2.11}$$

This equation can be solved for the entire domain Ω, becoming

$$||\nabla T(x)|| = \frac{1}{u(x)}, \forall x \in \Omega. \tag{2.12}$$

Boundary conditions are usually specified on the boundary $\partial\Omega$ of the domain. One usually considers a function g defined on the boundary $\partial\Omega$ so that $t(x) = g(x)$ for all $x \in \partial\Omega$. Usually, the function g is taken to be identically 0. Hence, the stationnary Eikonal equation becomes

$$\begin{cases} ||\nabla t(x)|| = \dfrac{1}{u(x)}, \forall x \in \Omega \\ t(x) = 0, \forall x \in \partial\Omega. \end{cases} \tag{2.13}$$

For all x in Ω, the solution $t(x)$ of equation (2.13) can be interpreted as the minimal time required to travel from x to the domain boundary $\partial\Omega$. For all x in Ω, we denote by d the geodesic distance function defined by

$$d(x, \partial\Omega) = \inf_{y \in \partial\Omega} ||x - y||, \tag{2.14}$$

where $||x - y||$ is the geodesic distance between x and y. Proposition 2.3.2 below relates the distance function d to the solution of the Eikonal equation on the domain Ω.

Proposition 2.3.2. *[Set96] Let Ω be a subset of the Euclidean space $\mathbb{R}^2$. Then, the distance function $d(\cdot, \partial\Omega)$ is differentiable almost everywhere, and its gradient satisfies the Eikonal equation*

$$||\nabla d(x, \partial\Omega)|| = 1$$

with initial conditions $d(x) = 0, \forall x \in \partial\Omega$.

In other words, the Eikonal equation allows us to compute the shortest distance between any point x of the domain Ω and the boundary $\partial\Omega$.

Eikonal equation on a discretized domain

Let us consider the discretization of the Eikonal equation on a domain $\Omega \subset \mathbb{R}^2$. We assume that Ω can be discretized on a regular grid.

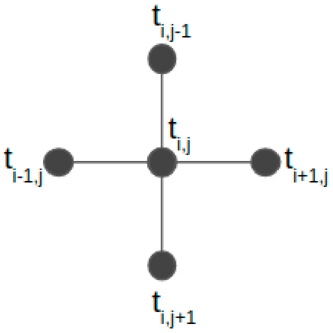


Figure 2.2 – An illustration of 4 connected neighborhood.

We rely on the finite difference approximation formula as in [Set96] to approximate the gradient term at grid point (i, j):

$$||\nabla t(i,j)|| \approx \sqrt{\max(D_{ij}^{-x},\ -D_{ij}^{+x},\ 0)^2 + \max(D_{ij}^{-y},\ -D_{ij}^{+y},\ 0)^2} = \frac{1}{u_{ij}}, \tag{2.15}$$

where u_{ij} is the velocity of the wave at (i, j). D_{ij}^{-x}, D_{ij}^{+x}, D_{ij}^{-y}, and D_{ij}^{+y} are partial derivatives, and we use the standard finite difference notation in 2D:

$$D_{ij}^{-x} = \frac{t_{ij} - t_{i-1,j}}{\Delta t}, \quad D_{ij}^{+x} = \frac{t_{i+1,j} - t_{ij}}{\Delta t} \tag{2.16}$$

For simplicity's sake, we assume that $\Delta t = 1$. Thus

$$\begin{aligned} ||\nabla t(i,j)||^2 = &\max(t_{i,j} - t_{i-1,j},\ t_{i,j} - t_{i+1,j},\ 0)^2 \\ &+ \max(t_{i,j} - t_{i,j-1},\ t_{i,j} - t_{i,j+1},\ 0)^2 \end{aligned} \tag{2.17}$$

Using the gradient discretization (2.17), the Eikonal equation becomes

$$\begin{aligned} &\max(t_{i,j} - t_{i-1,j},\ t_{i,j} - t_{i+1,j},\ 0)^2 + \\ &\max(t_{i,j} - t_{i,j-1},\ t_{i,j} - t_{i,j+1},\ 0)^2 = \frac{1}{u_{i,j}^2}. \end{aligned} \tag{2.18}$$

In this equation, the only unknown is the arrival time $t_{i,j}$ at point $p := (i, j)$. We have

$$\begin{aligned} \max(t_{i,j} - t_{i,j-1},\ t_{i,j} - t_{i,j+1},\ 0)^2 &= \\ \max(t_{i,j} - \min(t_{i,j-1}\ t_{i,j+1}), 0)^2&. \end{aligned} \tag{2.19}$$

Hence, $t_{i,j}$ is solution of the quadratic equation

$$\begin{aligned} &\max(t_{i,j} - \min(t_{i-1,j},\ t_{i+1,j}), 0)^2 + \\ &\max(t_{i,j} - \min(t_{i,j-1},\ t_{i,j+1}), 0)^2 = \frac{1}{u_{i,j}^2}. \end{aligned} \tag{2.20}$$

Eikonal equation on image domain

In this section, we present a clustering algorithm that works by solving the Eikonal equation on the image domain for clustering pixels in order to generate a superpixel partition. The image domain is interpreted here as a special case of discretized domain.

In what follows, we adopt the following notations. A pixel in image $\mathcal{I}$ is denoted by p and its coordinates by (x, y). We select N *seeds* or *cluster centers* locations $\{s_1, s_2, ..., s_N\}$ at initialization. The labels of the seeds are then gradually propagated from the labeled pixels to the unlabeled pixels according to the local velocity $\{u(p)\}$. On the domain defined by the image, the Eikonal equation therefore reads

$$\begin{cases} ||\nabla t(p)|| = \dfrac{1}{u(p)} & \forall p \in \mathcal{I} \\ t(p) = 0 \quad \forall p \in \partial\mathcal{I}. \end{cases} \tag{2.21}$$

In this expression, $u(p)$ is the local velocity at pixel p, $\partial\mathcal{I}$ corresponds to the subset of the image $\mathcal{I}$ constituted by the seeds $\{s_1, s_2, ..., s_N\}$, and $t(p)$ is the the minimal time required to travel from $\partial\mathcal{I}$ to pixel p. The velocity $u(p)$ depends on the color and the texture of both the seed and the pixel location $p := (x, y)$.

For all p in $\mathcal{I}$,

$$t(p) = t(p, \partial\mathcal{I}) = \inf_{s_i \in \partial\mathcal{I}} t(p, s_i), \tag{2.22}$$

where $t(p, s_i)$ is the minimal traveling time from s_i to p.

Fast marching algorithm

Fast methods for Eikonal equation include fast marching based methods, fast sweeping based methods and other methods. The fast marching method (FMM) is the most commonly used. It was originally introduced to solve the Eikonal equation on a continuous domain [MSV95; Set96]. The fast sweeping method (FSM) [FLZ09] is an iterative algorithm that performs Gauss-Seidel iterations with alternating sweeping ordering.

In [Gó19], nine fast methods including FMM, Fibonacci-Heap FMM(FMMFib), Simplified FMM (SFMM), Untidy FMM (UFMM), Group Marching Method (GMM), Fast Iterative Method (FIM), Fast Sweeping Method (FSM), Locking Sweeping Method (LSM) and Double Dynamic Queue Method (DDQM) are compared. The comparison is conducted under different circumstances, from simple case of empty maps to more complicated problem such as vessel segmentation.

The choice of the method depends on the application. For complex scenarios such as image segmentation, to solve the Eikonal equation with variable speed, their conclusion is that there is not an optimal method. UFMM can work well if properly tuned. SFMM is a safe choice and it is faster than FMM and FMMFib in each case.

In this research, we make use of the traditional FMM for its simplicity of implementation. Other fast methods, especially SFMM, would be worth considering to improve the speed of our algorithm.

Fast marching on a discretized domain

A possible approach to solve the Eikonal equation is to iteratively solve the nonlinear equation (2.20) at each location of the grid until some convergence criterion is met. However, this approach involves a significant amount of calculations to be conducted and can be very costly.

An alternative approach is to compute the arrival times outwards from the initial conditions by following the front propagation and computing the arrival times in increasing order. This approach leads to the fast marching algorithm and allows to solve equation (2.20) at each location of the grid in a single iteration.

The fast marching algorithm works by partitioning the points of the discretization domain Ω in three subsets during the front propagation:

— The points that have already been reached by the front are grouped into a set referred to as the *frozen* set.

— The points of the discretization grid that are adjacent to frozen points but have not been reached by the front yet are grouped in a subset referred to as the *narrow band*,

— The remaining points of the discretization grid are grouped in a subset referred to as the *far away* set.

Initialization The fast marching algorithm is initialized as follows:

1. An arrival time map is initialized: each point (i, j) in the discretization grid is associated with the arrival time $t = +\infty$, except if it belongs to the domain boundary

$\partial\Omega$. In this case, the point (i,j) is associated with the arrival time $t_{ij} = 0$.

2. All points of the discretization grid located at the domain boundary $\partial\Omega$ are added to the narrow band. Other points are labeled as *far away.* Initially, the frozen set is empty.

Iteration At each iteration, the point (i,j) of the narrow band with the smallest arrival time is extracted and labeled as *frozen.* Arrival times are computed for its neighbors in a 4-neighborhood by solving equation (2.18). Frozen points are used to compute the arrival times in other points, but their arrival time is never recomputed. We can rewrite (2.18) under the general form

$$\max(t_{ij} - t_A, 0)^2 + \max(t_{ij} - t_B, 0)^2 = \frac{1}{u_{ij}^2}, \tag{2.23}$$

where $t_A = \min(t_{i-1,j}, t_{i+1,j})$ and $t_B = \min(t_{i,j-1}, t_{i,j+1})$. By construction, (i,j) is adjacent to at least one frozen pixel. Without loss of generality, we can assume that A belongs to the frozen set and that $t_A \leq t_B$. B can be frozen, in the narrow band, or far away (in the latter case, $t_B = +\infty$). Then, we have the following result:

Proposition 2.3.3. *Equation (2.23) has a single solution t_{ij} satisfying $t_{ij} > t_A$.*

Proof. The proof is given in the appendix A.1. □

Once the arrival time t of a neighbor point (i,j) has been computed, two situations can be encountered:

— When (i,j) is in the narrow band, it has already been associated an arrival time t_{ij}^0. If the new arrival time t_{ij} is smaller than t_{ij}^0, then the arrival time is updated. Otherwise, it remains unchanged.

— When the neighbor point is far away, we add it to the narrow band with the arrival time t_{ij}.

Stopping condition The fast marching algorithm stops when the narrow band is empty.

At each iteration of the algorithm, it is necessary to extract the element of the narrow band with the smallest arrival time. The search for the smallest element in the narrow band can significantly enhance the algorithmic complexity of the fast marching approach. To reduce the complexity of the algorithm, a common solution is to store the elements of the narrow band in a binary heap. When using a binary heap structure, the complexity of the fast marching algorithm is in $O(N \log N)$, N being the number of points in the discretization grid. We refer the reader interested in more details on the fast marching algorithm implementation to the original articles [MSV95; Set96; Set99a; Set99b].

Fast Marching Superpixel (FMS)

The idea behind the FMS algorithm is to draw an analogy between waves propagating in a heterogeneous medium and regions growing on an image at a rate depending on the local color and texture. A similar idea was proposed in [Buy14b; Buy14a; BGR14]. However, both approaches differ in several aspects, including the expression for the local velocity as a function of the image content, the use of texture features or the region update during propagation.

Here we adopt the same initialization step as in SLIC, and focus on the clustering step. Instead of using clustering algorithms, such as k-means clustering or DBSCAN clustering [She16], we propagate the labels of the seeds in a process that can be described by the Eikonal equation. In this section, we describe in greater details the implementation of the FMS algorithm.

Region growing

Let us denote by K the desired number of superpixels. We initialize the algorithm by selecting N ($N < K$) seeds on a regular grid. To avoid placing a seed on a boundary, we place the seed at the local minimum of the gradient in a 3×3 neighborhood of the nodes on the grid. A similar strategy is used for selecting the initial seeds in the SLIC algorithm [Ach12]. We denote by $\mathbf{C}(p)$ the color at pixel p in the CIELAB color space. Similarly, we denote by $\mathbf{T}(p)$ a vector of features characterizing the local texture at pixel p.

Initialization The propagation algorithm is initialized as follows:

1. The arrival time map t is initialized:

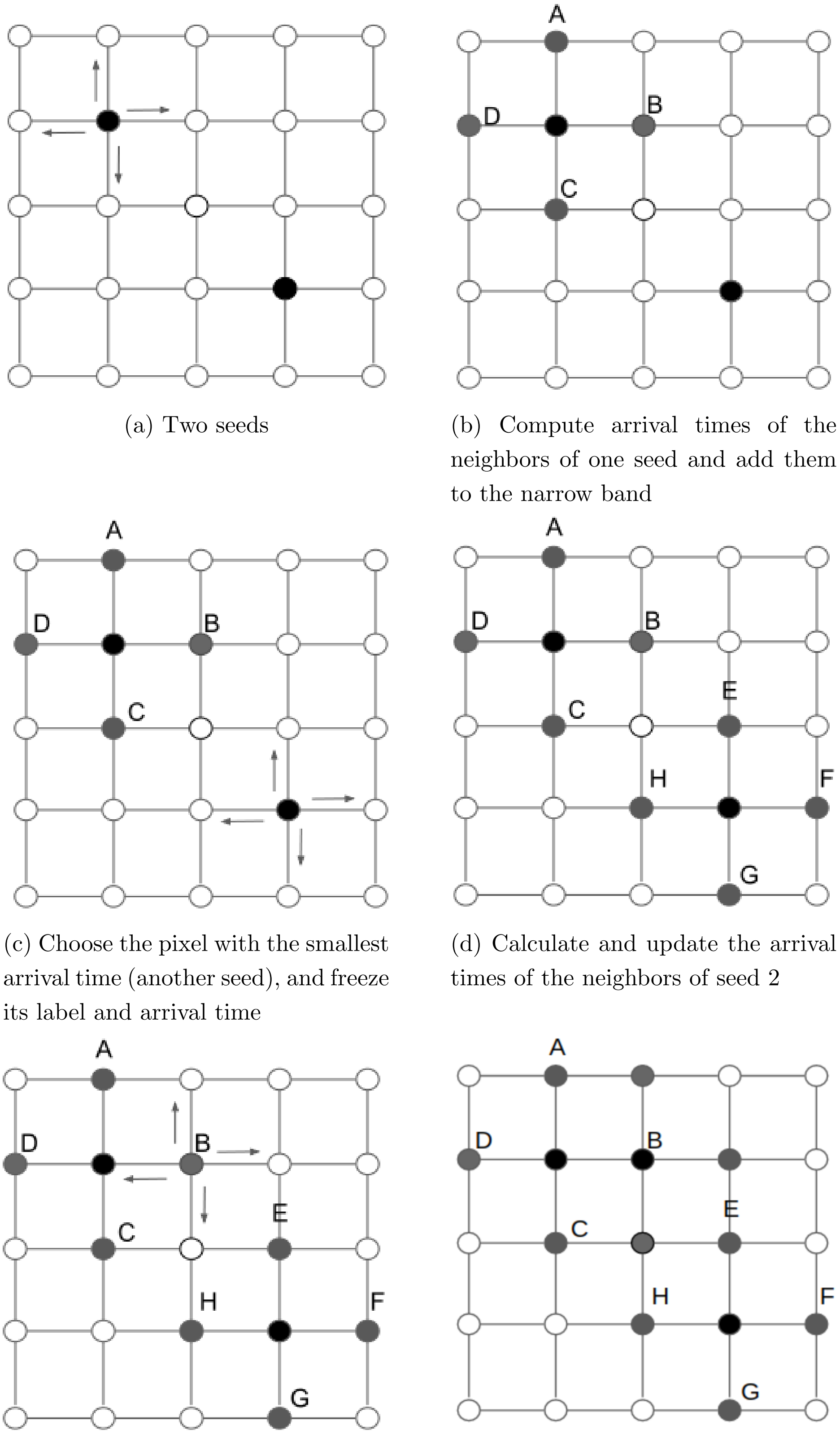
$$t(p) = \begin{cases} 0 \text{ if } p \in \partial\mathcal{I}, \\ +\infty \text{ otherwise.} \end{cases} \tag{2.24}$$

2. The label map L is initialized:

$$L(p) = \begin{cases} i \text{ if } p = s_i, \\ 0 \text{ otherwise.} \end{cases} \tag{2.25}$$

3. The pixels $\{s_1, .., s_N\}$ are grouped in a set referred to as the *narrow band*. All other pixels are labeled as *far away*.

Iteration At each iteration,



(a) Two seeds

(b) Compute arrival times of the neighbors of one seed and add them to the narrow band

(c) Choose the pixel with the smallest arrival time (another seed), and freeze its label and arrival time

(d) Calculate and update the arrival times of the neighbors of seed 2

(e) Choose the pixel with the smallest arrival time "B", and freeze the label and the arrival time of B

(f) Calculate and update the arrival times of the neighbors of B

Figure 2.3 – Illustration of the fast marching algorithm on the superpixel segmentation. Black, gray and white circles represent frozen pixels, pixels in the narrow band, and far away pixels respectively.

1. We extract the pixel $p := (x, y)$ in the narrow band with the smallest arrival time and label it as *frozen*.
2. We compute the arrival time for all neighbor pixels of p that are not frozen. To that end, we use the velocity corresponding to the label $l := L(p)$ associated to pixel p. Note that frozen pixels are used to compute the arrival times in other pixels, but their arrival times are never recomputed. Hence, the arrival time $t_{x,y}$ at a neighbor $\tilde{p} := (x, y)$ of p is solution of

   $$\begin{aligned} &\max(t_{x,y} - \min(t_{x-1,y},\ t_{x+1,y}), 0)^2 + \\ &\max(t_{x,y} - \min(t_{x,y-1},\ t_{x,y+1}), 0)^2 = \frac{1}{u_l(x,y)^2}. \end{aligned} \tag{2.26}$$

   Note that in this expression, we only consider the neighbors with label l. If a neighbor of p has a label distinct than l, then we consider its associated arrival time to be equal to $+\infty$. We introduce this slight modification of the original fast marching algorithm to efficiently keep track of the labels propagation.

   Equation (2.26) has two distinct solutions. The proof is provided in the appendix A.1. To respect the consistency of the scheme, the arrival time must be higher than the arrival time $t(p)$ associated to the point selected in the narrow band. Hence, the arrival time $t_{x,y}$ is necessarily the greater solution of equation (2.26).
3. At this point, we can encounter two distinct situations:
   - When the neighbor point (x, y) is already in the narrow band, it has necessarily been associated an arrival time $t^{old}_{x,y}$. If $t_{x,y} < t^{old}_{x,y}$, then we assign to it the arrival time $t_{x,y}$ to (x, y), as well as the label $l := L(p)$ of point p. On the contrary, if $t_{x,y} > t^{old}_{x,y}$, then the label and the arrival time at (x, y) remain unchanged.
   - When the neighbor point (x, y) is not in the narrow band, we assign to it the arrival time $t_{x,y}$, the label $l := L(p)$ and we add it to the narrow band.

Stopping condition The iteration stops when the narrow band is empty.

Figure 2.3 is a simple illustration of the fast marching based region growing with two initial seeds. At each iteration of the algorithm, it is necessary to extract the element of the narrow band with the smallest arrival time. To reduce the complexity of the algorithm, the elements of the narrow band are stored in a binary heap. After the iteration stops, each pixel of the image has necessarily been labeled, yielding a superpixel partition of the image.

Local velocity model

The algorithm described in the previous section can be used with any non-negative velocity models $u(p)$. We describe in this section the velocity model used in our research, which relies upon both color and texture information. Natural images usually contain textured regions composed of repeated texels with strong color changes. A pixel in a textured region can therefore exhibit a strong color difference with the cluster center. To account for the texturedness, we consider a velocity model incorporating both local color and texture information. Hence, we consider the distance between a pixel p and a seed s_i to be

$$D(p, s_i) = w_0 \|\boldsymbol{C}_p - \boldsymbol{C}_i\|_2 + w_1 \|\boldsymbol{T}_p - \boldsymbol{T}_i\|_2, \tag{2.27}$$

where $\boldsymbol{T}_p$ is a vector of features characterizing the local texture at pixel p, $\boldsymbol{T}_i$ is the corresponding vector of features for pixel s_i, and w_0 and w_1 are positive coefficients weighting color and texture contributions, respectively.

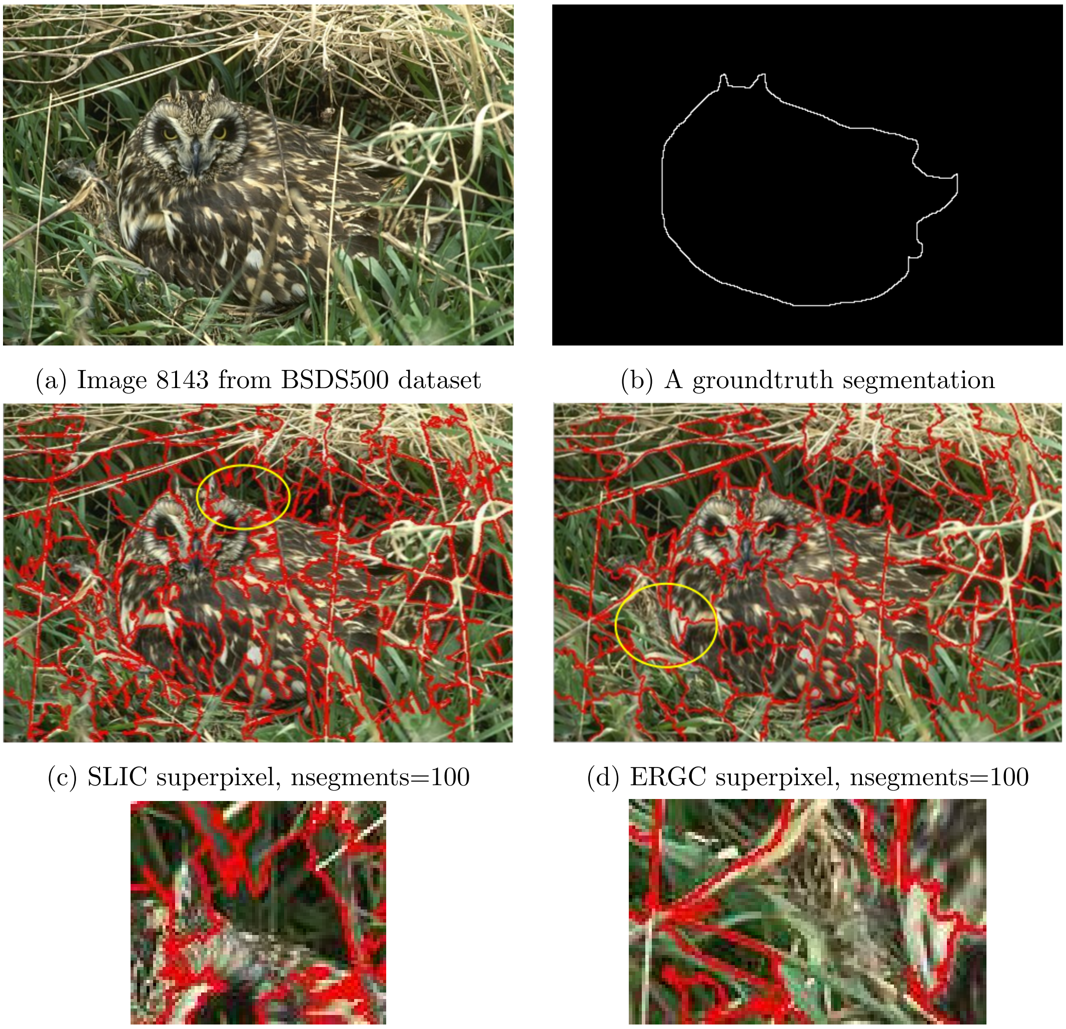
Texture

A definition of texture is given in [Skl78]: “A region in an image has a constant texture if a set of local statistics or other local properties of the picture are constant, slowly varying, or approximately periodic.” Texture reveals the spatial arrangement of gray levels of neighboring pixels. Textured regions are ubiquitous in natural images. For example, in figure 2.4, the feathers of the owl is composed of repeated pattern of brown, white and light orange feathers.

Texture information is of great importance in image segmentation algorithms ([Mal01], [RM03], [Arb11]). It is also important for superpixel segmentation. Firstly, the color variance is always high within a textured region. For SLIC algorithm, since we take the average color of pixels within a region as the color of a seed, if this average is similar to the color of an adjacent region, the pixels might be wrongly clustered to the same seeds, so that the boundary between the two regions might be missed. An example of this case is given in figure 2.4. Another reason is that some applications prefer smooth boundaries within an object, but due to color changes, the superpixel contours can be highly tortuous within a textured region.

We design the texture vector in (2.27) through texture analysis. A conventional way of performing texture analysis is to use a multi-channel filtering approach. The image is convolved with a bank of spatial filters with different ranges of frequency and multiple orientations [JF91]. One advantage of the multi-channel filtering is that it exploits differences in dominant sizes and orientations of different textures. A texture vector should integrate the information from all these channels. We expect that the texture vectors are similar within regions with the same texture, and have a large difference in regions with different textures.

We should choose a type of filter for channel characterisation of multi-channel filtering. Various filters can be used for this purpose, such as difference-of-Gaussian, Gaussian



(a) Image 8143 from BSDS500 dataset

(b) A groundtruth segmentation

(c) SLIC superpixel, nsegments=100

(d) ERGC superpixel, nsegments=100

(e) Circled region from (c)

(f) Circled region from(d)

Figure 2.4 – SLIC and ERGC superpixels for a texture image

derivatives, Laplacian of Gaussian, Laws filters and so on. [XM08] provides a thorough summary on texture analysis methods. When doing texture analysis, we prefer having good frequency properties for distinguishing between different textures; sufficient spatial resolution is also desirable to guarantee good localization of the texture boundaries. However, the bandwidth of a filter in the spatial domain and that in the frequency domain are inversely related. In this research, we use Gabor filtering for the construction of the features vector, since it achieves the optimal joint resolution in both spatial and frequency domains.

Gabor filtering

Gabor filters are widely used for texture analysis. Anil K.Jain *et al.* [JF91] designed a multi-channel filtering approach which uses a bank of even symmetric Gabor filters for channel characterisation, presented a method to choose radial frequencies and described a filter selection scheme based on image reconstruction.

We compute the local response of the image with a bank of Gabor filters to obtain the texture features. The kernel of the Gabor filters is given by the expression

$$g(x, y; f_0, \theta, \phi, \sigma_{x'}, \sigma_{y'}) = \exp\left\{ -\frac{1}{2} \left[\frac{x'^2}{\sigma_{x'}^2} + \frac{y'^2}{\sigma_{y'}^2} \right] \right\} \cos(2\pi f_0 x') \tag{2.28}$$

where

$$x' = x\cos\theta + y\sin\theta, \quad y' = -x\sin\theta + y\cos\theta$$

f_0 is the frequency of the sinusoidal wave along x' axis, θ is the angle between the x' and x axes, and σ_x, σ_y are the standard deviation of the Gaussian filter in the space domain along x' and y' axes. The relation between the frequency resolution and the space resolution is $\sigma_u = 1/(2\pi\sigma_{x'})$, $\sigma_v = 1/(2\pi\sigma_{y'})$.

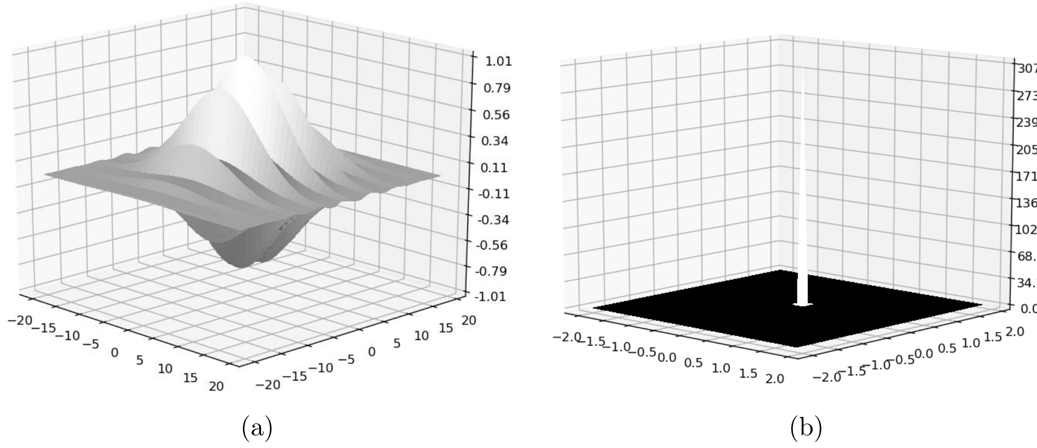


Figure 2.5 – A 2d Gabor filter in (a) spatial domain and (b) frequency domain. $\sigma_x = \sigma_y = 7.0$, $f_0 = 0.2$, $\theta = 0$.

To run the FMS algorithm, we selected four directions (0°, 45°, 90° and 135°) for the Gabor kernels of the filter banks. The frequency was set equal to $f_0 = 0.1$. Two kernels with standard deviation σ equal to 2 and 4 were used to perform the filtering. Overall, this resulted in a bank of filters containing 8 kernels. We tried more directions and other values of σ, and there was no apparent improvement of the performance of FMS on the BSDS 500 dataset.

The vector $\boldsymbol{T}_p$ stacks the 8 responses of the Gabor filtering at pixel p. The local velocity $F_i(p)$ associated with the propagation of region i is finally obtained by relying on the exponential kernel

$$u(p, i) = \exp(-w_0 \|\boldsymbol{C}_p - \boldsymbol{C}_i\|_2 - w_1 \|\boldsymbol{T}_p - \boldsymbol{T}_i\|_2). \tag{2.29}$$

The velocity is maximal and equal to 1 when the pixel and the seed share the same color and texture characteristics.

In addition to directly stacking the filter responses, we tried other methods to construct the feature vector T_p. Firstly, we clustered the response vectors of Gabor filters into textons, a small set of prototype vectors, by using k-means clustering. Then we assigned to each pixel the label of the nearest texton. When calculating the distance, we used the texture vector of the corresponding texton of p as T_p, the same for T_i. Secondly, we tried to use more directions and more scales, for example, 6 directions and 3 scales for the Gabor filtering conducted in the beginning, and then to reduce the dimension of the response vectors to 8 using a Principle Component Analysis (PCA). The response vector after dimension reduction was then stacked as T_p. However, none of the above two attempts achieved apparent improvement in performance.

Refinement

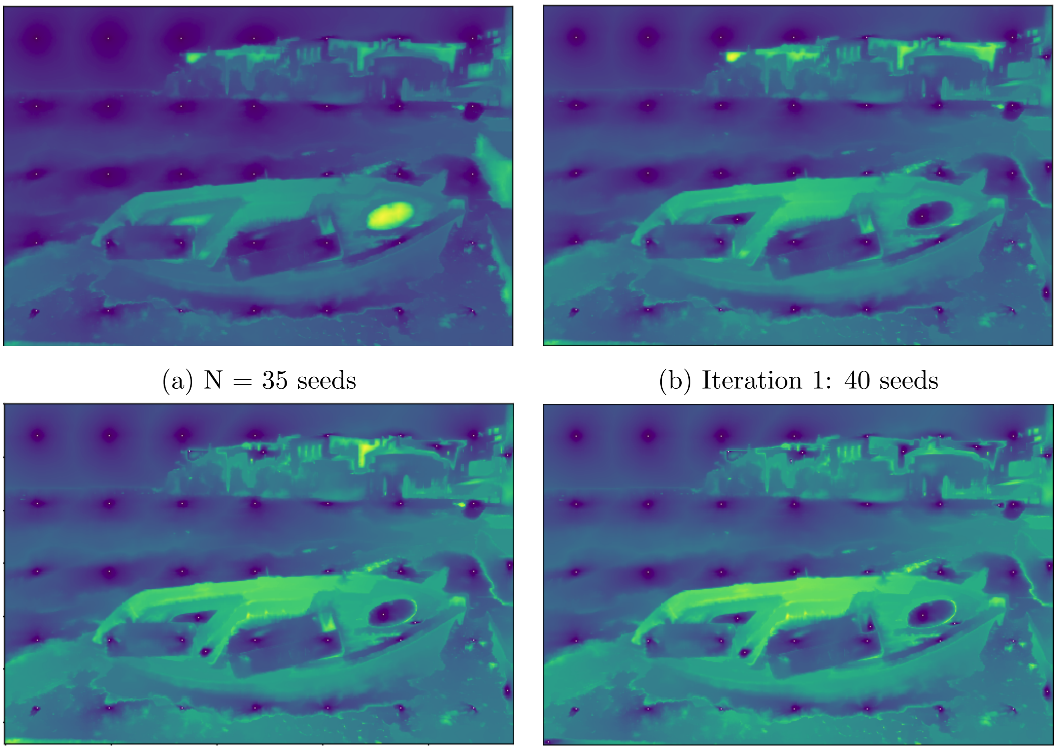
Images usually contain regions of interest over a very large range of scales. Hence, selecting the seeds according to a grid with uniform spacing might not be the optimal strategy to obtain a superpixel segmentation with good boundary adherence. In this section, we present a strategy built on the map of arrival time to refine the superpixel segmentation.

We recall that the number of desired superpixels is denoted by K. The algorithm presented in the previous sections yields a partition of the image into N connected segments corresponding to the initial seeds $\{s_1, s_2, ..., s_N\}$, as well as a map $t(p)$ storing the arrival time at each pixel p. The map $t(p)$ contains a significant amount of information that can be used to refine the superpixel partition. When the arrival time is high at some pixel p in a region $\mathcal{R}_i$ associated to seed s_i, it means that the region boundary propagating from s_i has traveled through pixels that are highly dissimilar to s_i. The arrival time found in each region $\mathcal{R}_i$ therefore constitutes an interesting criterion to assess whether or not this region should be subsequently splitted. A similar refinement strategy is used in the article [Buy14a], where the distance map is used at the end of the oversegmentation to generate new superpixels for refinement.

The refinement is conducted as follows. Let us denote by $\mathcal{B}$ the set of pixels belonging to the superpixel boundaries, and by $\delta(\mathcal{B})$ the dilated set of $\mathcal{B}$ by a disk of radius 2 pixels. Then, the maximal arrival time in region $\mathcal{R}_i$ is defined to be

$$t_i = \max_{p \in \mathcal{R}_i \cap \delta(\mathcal{B})^c} t(p), \tag{2.30}$$

where $\delta(\mathcal{B})^c$ is the complementary set of $\delta(\mathcal{B})$. To further refine the superpixel segmentation, we select the k regions with highest t_i and we add seeds at the corresponding locations before re-propagating. We exclude the pixels that belong to the region $\delta(\mathcal{B})$ to avoid implanting a seed directly on a boundary. This procedure is repeated iteratively



(a) N = 35 seeds

(b) Iteration 1: 40 seeds

(c) Iteration 2: 45 seeds

(d) Iteration 3: 50 seeds

Figure 2.6 – Illustration of the refinement on an image of the BSDS500. Each figure displays the arrival time at each pixel. At each step, new seeds are added at superpixels where the arrival time is maximal.

until the desired number K of superpixels is obtained and enables to significantly improve the boundary adherence of the resulting superpixel partition.

Experiments and discussion

In this section, we evaluate the performance of the FMS algorithm and compare it with two state-of-the-art superpixel algorithms, SLIC [Ach12] and ERGC [Buy14b; Buy14a; BGR14]. To that end, we use the Berkeley Segmentation Dataset 500 (BSDS500) [Arb11]. The BSDS500 provides an empirical basis for research on image segmentation and boundary detection. It contains 6000 hand-labeled segmentations of 500 color images from 30 human subjects and is one of the reference datasets for evaluating the performance of segmentation algorithms.

Evaluation metrics

To evaluate the performance of the algorithms, we adopt four metrics: boundary recall, undersegmentation error, compactness and contour density.

- *Boundary recall* is the ratio of the number of true positive contour pixels with regard to the number of contour pixels in the ground truth segmentation. To tolerate small localization errors, contour pixels that lie within 2 pixels from a real contour pixel are considered as true positives. High boundary recall means good adherence to boundaries.

- *Undersegmentation error* measures the leakage of a superpixel overlapping with a gound truth segment. We adopt the formulation proposed by Neubert and Protzel in [NP12].

$$\mathrm{UE}_{NP}(G,S) = \frac{1}{N}\sum_{G_i}\sum_{S_j \cap G_i \neq \emptyset} \min\{|S_j \cap G_i|, |S_j \setminus (S_j \cap G_i)|\} \tag{2.31}$$

  where S_j is a superpixel and G_i is a gound truth segment. We take the minimum of the number of overlapping pixels, and the number of non overlapping pixels within S_j as the leakage.

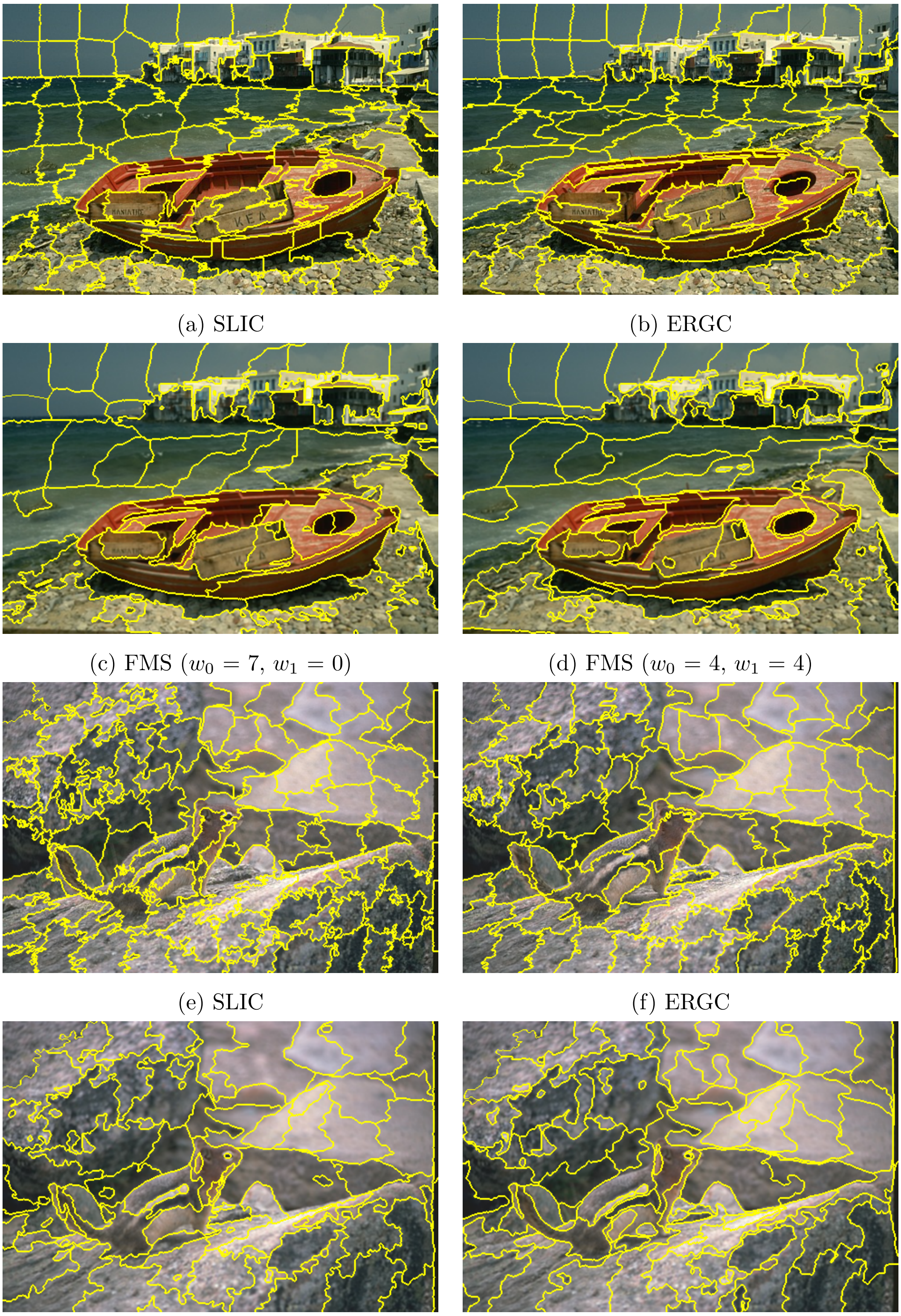
- *Compactness* is defined as the ratio of the region area with respect to a circle with the same perimeter as the superpixel, weighted by the ratio of pixel numbers inside the region [SFS12].

$$COMP(S_i) = \sum_{S_i} \frac{|S_i|}{N}\frac{4\pi A(S_i)}{P(S_i)^2} \tag{2.32}$$

  High compactness indicates a high regularity of the superpixel shapes, which is desirable for some applications.

- *Contour density* was first proposed in [Mac15]. It is defined as

$$CD = \frac{|C|}{N} \tag{2.33}$$



(a) SLIC

(b) ERGC

(c) FMS ($w_0 = 7$, $w_1 = 0$)

(d) FMS ($w_0 = 4$, $w_1 = 4$)

(e) SLIC

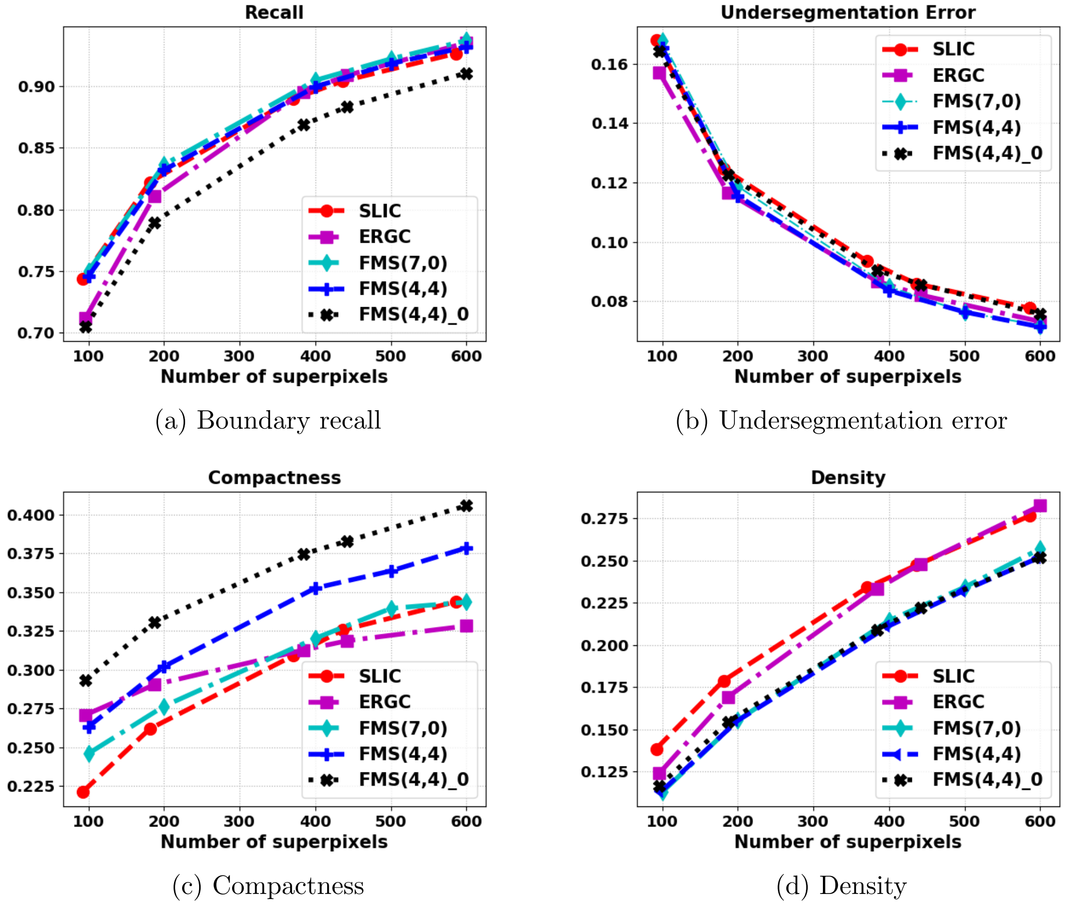
(f) ERGC

(g) FMS ($w_0 = 7$, $w_1 = 0$)

(h) FMS ($w_0 = 4$, $w_1 = 4$)

Figure 2.7 – Illustration of the superpixel segmentation on an image of the BSDS500.

where $|C|$ is the total number of pixels on superpixel contours and N is the number of pixels in the image. A low contour density tends to indicate that the tortuosity



(a) Boundary recall

(b) Undersegmentation error

(c) Compactness

(d) Density

Figure 2.8 – Comparison between SLIC, ERGC and FMS algorithms on the BSDS500. $FMS(4,4)\_0$ represents the result without refinement.

of the superpixels contours remains small. This is a desirable effect, since tortuous contours tend to artificially increase boundary recall.

Results and discussion

Figure 2.8 displays the recall, undersegmentation error, compactness and density of FMS, SLIC and ERGC averaged over the 500 images of BSDS500. For a fair comparison, the parameters of SLIC and ERGC are optimized by the code of David Stutz in [SHL17]. The number of initial seeds N is set to $0.6K$ through the experiment. The superpixels constructed with these algorithms are illustrated in figure 2.7.

The parameters w_0 and w_1 for weighting the color and texture distances must be selected with caution. Overall, increasing w_0 and w_1 lead to an increase of the boundary recall. This is due to the fact that the propagation velocity becomes more sensitive to strong color and texture variations. However, for high values of the weights e.g. $w_0, w_1 > 8$, the local velocity can fall to very low values. This has the unwanted effect to deteriorate the compactness of the obtained superpixels and leads to the obtention of many small isolated superpixels corresponding to small contrasted part of the image. Hence, the choice of parameters results from a trade-off between boundary adherence performance and topological considerations. In our experiments, we selected the parameters $w_0 = 4$ and $w_1 = 4$ by relying on a grid search to obtain an optimal boundary adherence. We also computed the superpixel partition using the parameters $w_0 = 7$ and $w_1 = 0$ to evaluate the influence of texture.

In terms of recall, we can note that the FMS algorithm yields slightly better results than SLIC and ERGC algorithms. The refinement strategy also significantly improves the results. We can note that the results are better with the set of parameters $w_0 = 7$ and $w_1 = 0$ than with $w_0 = 4$ and $w_1 = 4$. The undersegmentation error is comparable to the one obtained with SLIC and ERGC. ERGC is the best performer when the number of superpixels N remains below 400, but FMS exhibits a slightly lower undersegmentation error when $N > 400$. The main advantage of the FMS algorithm is that it provides good topological metrics: the compactness of FMS is higher than the one of SLIC and ERGC, and the density of contours is significantly lower. In particular, the version of FMS with parameters $w_0 = 4$ and $w_1 = 4$ yields a boundary recall and an undersegmentation error similar to ERGC and SLIC, but with higher compactness. In addition, due to the refinement procedure, the size of the superpixels constructed with the FMS algorithm is locally adapted depending on the content of the image. One should nevertheless keep in mind that the refinement is performed at the expense of the superpixels uniformity: the refinement yields superpixels that are heterogeneous in scale and size.

In terms of complexity, if n is the number of pixels in the image, then the complexity is $O(n \log n)$ at each propagation step. Hence, if there are L refinement steps, the complexity of the algorithm is in $O(Ln \log n)$.

Finally, it is interesting to point out the main differences between the FMS and the ERGC algorithms [Buy14b; Buy14a; BGR14]. ERGC is a clustering-based algorithm that computes superpixel partitions by relying upon the Eikonal equation. A significant difference between both algorithms is that their local velocity models differ. For ERGC, the local velocity field is simply given by:

$$u(p,i) = \frac{1}{||\boldsymbol{C}_p - \boldsymbol{C}_i\|_2}. \tag{2.34}$$

In particular, the velocity is very high when the pixel and the seed have similar color characteristics. Another important difference is that ERGC only considers local color information to compute the local velocity, while FMS also relies on texture information. Finally, an additional difference between both approaches is that the cluster centers are iteratively updated according to the new pixels entering the clusters. By contrast, in the FMS algorithm, the cluster centers remain fixed.

The choice of parameters for Gabor filters is highly related to the scale of textures in the image dataset. If FMS is to be applied to a new dataset, we should first observe the textures, and modify the central frequencies, number of directions and σ accordingly.

For the choice of w_0 and w_1, the ideal solution could be to find a measure to quantify the textureness of a region, and to adjust the weights of the color and texture distances terms accordingly, to give a higher weight for the texture term in the textured regions. In an attempt to follow this approach, we clustered the responses vectors of each pixel to Gabor filters into textons and then measured the textureness by the chi-square distance of histograms of textons within a patch around a pixel. We did not see any improvement in the performances.

Conclusion and perspectives

In this chapter, we presented a fast-marching based algorithm for generating superpixel partitions of images. The FMS algorithm is evaluated on the Berkeley Segmentation Database 500, and yields results in terms of boundary recall and undersegmentation error that are comparable to the ones obtained with state-of-the-art algorithms including SLIC or ERGC, while improving on these algorithms in terms of compactness and density.

A possible extension of this work could be to incorporate gradient information in the local velocity model to better account for the presence of contours and discontinuity in the image. It would also be of interest to try to automatically estimate optimal parameters w_0 and w_1 depending on the processed image.

Chapter 3

Hierarchical segmentation based on wavelet decomposition

As discussed at length in the introductory chapter of this manuscript, image segmentation is a classical problem in image processing, which aims at defining an image partition where each identified region corresponds to some object present in the scene. The watershed algorithm is a powerful tool from mathematical morphology to perform this specific task. However when applied directly to the gradient of the image to be segmented, it usually yields an over-segmented image. To address this issue, one often uses markers that roughly correspond to the locations of the objects to be segmented. The main challenge associated with marker-controlled segmentation becomes thus the determination of the markers locations.

This chapter, somewhat independent from the rest of the manuscript, introduces a novel method to select markers for the watershed algorithm based upon multi-resolution approximations. The main principle of the method is to rely on the discrete decimated wavelet transform to obtain successive approximations of the image to be segmented. The minima of the gradient image of each coarse approximation are then propagated back to the original image space and selected as markers for the watershed transform, thus defining a hierarchical structure for the detected contours. The performance of the proposed approach is evaluated by comparing its results to manually segmented images from the Berkeley segmentation database.

Introduction

The watershed transform [BL79; MB90; VS91] is a popular algorithm based upon mathematical morphology which efficiently performs segmentation tasks. It can easily be understood by making an analogy between an image and a topography relief. In this analogy, the gray value at a given pixel is interpreted as an elevation at some location. The topography relief is then flooded by water coming from the minima of the relief. When water coming from different minima meet at some location, the location is labelled as an edge of the image. The watershed algorithm is usually applied to the gradient of the image to be segmented. Each minimum of the gradient therefore gives birth to one region in the resulting segmentation. Due to several factors including noise, quantization error or inherent textures present in the images, gradient operators usually yield a large number of minima. A well-known issue of the watershed algorithm is thus that it usually yields a severely over-segmented image as a result.

To overcome the issue of over-segmentation, a first approach is to apply the gradient operator to images that have previously been filtered. Meyer designed morphological filters, referred to as levelling filters, for this particular task [Mey98; Mey04]. Wavelet based filters have also been used to perform the filtering step. In 2005, Jung and Scharcanski proposed to rely on a redundant wavelet transform to perform image filtering before applying the watershed [JS05]. The advantage of using the wavelet transform is that its application tends to enhance edges at multiple resolutions, therefore yielding an enhanced version of the gradient. Jung subsequently exploited the multi-scale aspects of the wavelet transform to guide the watershed algorithm toward the detection of edges corresponding to objects of specified sizes [Jun07].

An alternative to overcome the over-segmentation issue is to rely on markers to perform the watershed segmentation. This strategy builds upon the assumption that it is possible to roughly determine the location of the objects of interest to be segmented. The idea is then to perform the flooding from these markers rather than from the minima of the gradient. Another approach that was considered is to select the minima of the gradient according to their importance, by considering for instance h-minima [Soi13; CR09]. Other approaches including the stochastic watershed rely on stochastic markers that are used to evaluate the frequency at which a contour appear in the segmentation [AJ07]. Finally, following a classical trend in image segmentation [FH04; Alp12], morphological algorithms have been proposed to perform a bottom-up region merging according to some morphological criteria [Beu94; MB05].

In this chapter, our aim is to present a novel method to select markers for the watershed algorithm based upon multi-resolution approximations. The main principle of the method is to rely on the orthogonal wavelet transform to obtain successive approximations of the image to be segmented. The minima of the gradient image of each coarse approximation are then propagated back to the original image space and selected as markers for the watershed transform, thus defining a hierarchical structure for the detected contours.

The outline of this chapter is as follows. We describe the proposed algorithm and state the main properties of the obtained contours hierarchy in section 3.2. In section 3.3, we evaluate the performances of the algorithm on the Berkeley segmentation database.

Conclusions are finally drawn in the last section.

Multiscale watershed segmentation

Multi-resolution approximation

A function f from $\mathbb{R}^2$ to $\mathbb{R}$ is said to be square-integrable if and only if the integral

$$\int_{\mathbb{R}^2} |f(x_1, x_2)|^2 \mathrm{d}x_1 \mathrm{d}x_2 \tag{3.1}$$

is finite. We denote by $\mathbf{L}^2(\mathbb{R}^2)$ the set of square-integrable functions. When equipped with the scalar product

$$< f, g >= \int_{\mathbb{R}^2} f(x_1, x_2) g(x_1, x_2) \mathrm{d}x_1 \mathrm{d}x_2, \tag{3.2}$$

it is well-known that $\mathbf{L}^2(\mathbb{R}^2)$ is a Hilbert space of infinite dimension.

Let us first introduce the mathematical notion of multi-resolution approximation [Mal99] which plays a central role in the proposed approach. A multi-resolution of $\mathbf{L}^2(\mathbb{R})$ is a sequence $\{V_j\}_{j\in\mathbb{Z}}$ of closed subspaces of $\mathbf{L}^2(\mathbb{R})$ satisfying the following properties

1. $\forall (j,k) \in \mathbb{Z}^2$, $f(\cdot) \in V_j \Leftrightarrow f(\cdot - 2^j k) \in V_j$,
2. $\forall j \in \mathbb{Z}$, $V_{j+1} \subset V_j$,
3. $\forall j \in \mathbb{Z}$, $f(\cdot) \in V_j \Leftrightarrow f(\cdot/2) \in V_{j+1}$,
4. $\lim_{j\to -\infty} V_j = \cap_{j=-\infty}^{j=+\infty} V_j = \{\emptyset\}$,
5. $\lim_{j\to +\infty} V_j = \mathbf{Closure}(\cup_{j=-\infty}^{j=+\infty} V_j) = \mathbf{L}^2(\mathbb{R})$.

In addition, for a sequence $\{V_j\}_{j\in\mathbb{Z}}$ to be a multi-resolution of $\mathbf{L}^2(\mathbb{R})$, there must exist a function θ in $\mathbf{L}^2(\mathbb{R})$ such that the family $\{\theta(t-n)\}_{n\in\mathbb{Z}}$ is a basis of V_0.

Multi-resolution approximations have been extensively used in computer vision since their introduction in the article [BA83] of Burt and Adelson. From this perspective, a signal of dyadic size 2^J is the orthogonal projection of a function f in $\mathbf{L}^2(\mathbb{R})$ on some space $V_J \subset L^2(\mathbb{R})$. The approximation of the signal at a resolution 2^{-j}, with $j > J$, is defined as its orthogonal projection on a subspace V_j. In higher dimensions D, e.g. for images, multi-resolution approximations of $\mathbf{L}^2(\mathbb{R}^D)$ can be obtained by considering tensorial products between subspaces: $V_j^D = V_j \otimes V_j \otimes ... \otimes V_j$.

In our algorithm, we consider a multi-resolution approximation based upon a discrete image wavelet decomposition. It is possible to define a scaling function ϕ from the Riesz basis $\{\theta(t-n)\}_{n\in\mathbb{Z}}$ of V_0. An approximation of the image at scale j is computed by projecting the approximation image at scale $j-1$ on the family $\{\phi_j(x-n)\}_{n\in\mathbb{Z}}$ of scaling

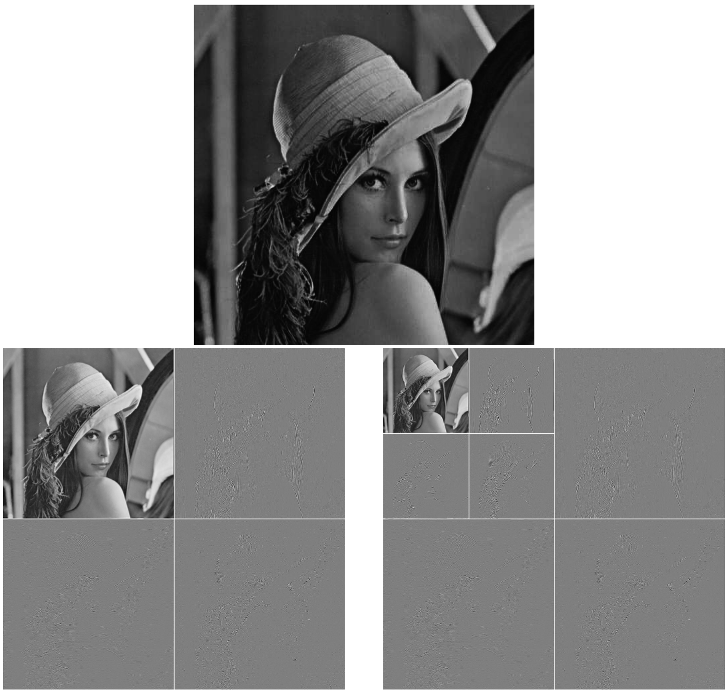


Figure 3.1 – Wavelet transform associated to Lena image for two decomposition levels. The size of the image is 512 by 512 pixels. The decomposition was performed using Daubechies wavelet with 12 vanishing moments. The approximation image corresponds to the subimage on the top left quarter. The other three subimages correspond to the projection of each approximation on a family of wavelets.

functions, where

$$\phi_j(x) = \frac{1}{\sqrt{2^j}}\phi(\frac{x}{2^j}). \tag{3.3}$$

It can be shown that the projection can be computed by relying on iterative convolutions with a low-pass filter, followed by a factor 2 sub-sampling. The discrete decimated wavelet transform of the original image is obtained by iteratively filtering the approximation image by tensorial products of a low-pass and a high-pass filters, yielding one approximation image and three details images at each iteration [Mal99].

Hierarchical contour detection

A multi-resolution approximation of an image is presented in figure 3.1, for two levels of decomposition. The multi-resolution approximation is computed using Daubechies wavelets with 12 vanishing moments. Interestingly, we can note that the main objects and contours of the original image are relatively well preserved in its approximations. However, these images contain considerably less details than the original image, making them of potential interest for segmentation. Several methods have been proposed to handle segmentation using multi-resolution approaches. In 2000, Rezaee *et al.* [Rez00] notably proposed an algorithm combining the pyramid transform and fuzzy clustering, obtaining good segmentation results on magnetic resonance images. In 2003, Kim and Kim [KK03] proposed a segmentation procedure relying on pyramidal representation and region merging.

It is straightforward to directly apply a watershed algorithm on the approximation images. However, the resolution of these images is significantly lower than the one of the original image. It is therefore difficult to establish a direct correspondence between the contours of the approximation image and the contours of the original image [KK03]. In this study, we propose to tackle this issue in a simple manner by defining multi-scale markers for performing the segmentation.

Let us consider an image I of dyadic size 2^J by 2^J pixels. We denote by L the number of decomposition levels. For l between 0 and L, we denote $I^{(l)}$ the approximation of I after l decomposition steps. The size of the image $I^{(l)}$ is therefore 2^{J-l} by 2^{J-l} pixels. By convention, $I^{(0)}$ is the original image I. $I^{(l+1)}$ is obtained by successively applying the low-pass filter associated with the wavelet transform to the rows and the columns of $I^{(l)}$ and down-sampling the result by a factor 2.

To initialize the contour detection algorithm, we first extract the minima of the gradient of the approximation image $I^{(L)}$. We obtain a sequence $\{m_1^{(L)}, ..., m_{K_L}^{(L)}\}$ of K_L markers. The locations of these markers are specified on an image $M^{(L)}$ of size 2^{J-L} by 2^{J-L}. To propagate the markers back to the original image I, we oversample the image $M^{(L)}$ by replacing each pixel $M^{(L)}(p,q)$ by an array of pixels of size 2^L by 2^L whose value is set equal to $M^{(L)}(p,q)$. Then, we apply the watershed algorithm on the image I from the image markers $M^{(L)}$, therefore obtaining a contour image $C^{(L)}$ associated to the approximation image $I^{(L)}$. We then repeat these operations at decomposition level $L-1$ to obtain a contour image $E^{(L-1)}$. We consider then the supremum image $S^{(L-1)}$ defined by

$$S^{(L-1)} = \sup(C^{(L)}, E^{(L-1)}), \tag{3.4}$$

where the supremum is defined as follows for each pixel $S^{(L-1)}[p,q]$:

$$S^{(L-1)}[p,q] = \max(C^{(L)}[p,q], E^{(L-1)}[p,q]). \tag{3.5}$$

Finally, we apply a watershed transform to the supremum image $S^{(L-1)}$ to obtain the contour image $C^{(L-1)}$ corresponding to decomposition level $L-1$. This last step is necessary to remove the thick contours that can potentially be created by the supremum between images $C^{(L)}$ and $E^{(L-1)}$. We obtain a multi-scale contour by iteratively applying the procedure described previously for all decomposition levels.

To illustrate the algorithm, the segmentation algorithm is applied to the images displayed in figure 3.2 and 3.4. The results are displayed in figure 3.3 and 3.5. We used Daubechies wavelets with 12 vanishing moments to calculate the successive multi-resolution approximations. By construction, the algorithm returns a nested sequence of contours, in

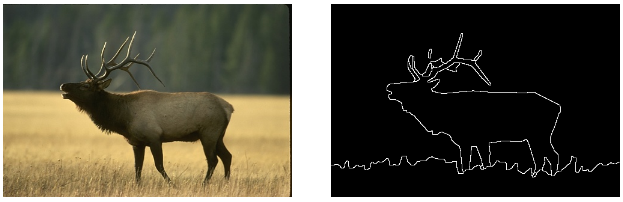


Figure 3.2 – An image from the BSDS500 used to illustrate the algorithm, along with an example of human segmentation

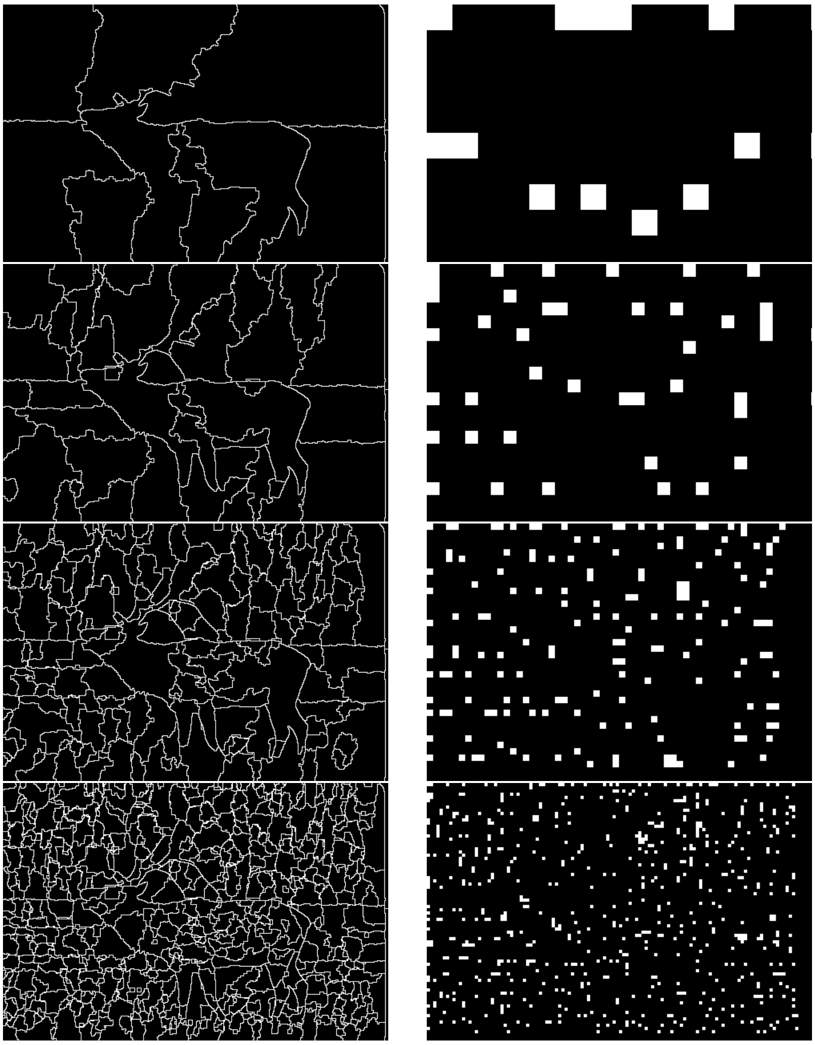


Figure 3.3 – Contour images corresponding to four decomposition levels (levels 4, 3, 2, 1 respectively) of the image presented in figure 3.2. The markers at each scale are displayed on the right images.

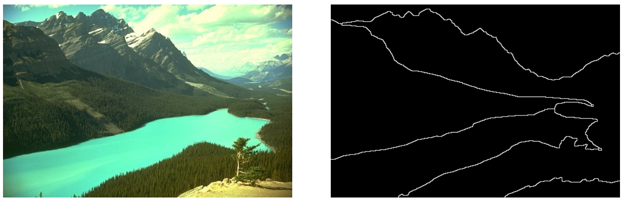


Figure 3.4 – An image from the BSDS500 used to illustrate the algorithm, along with an example of human segmentation

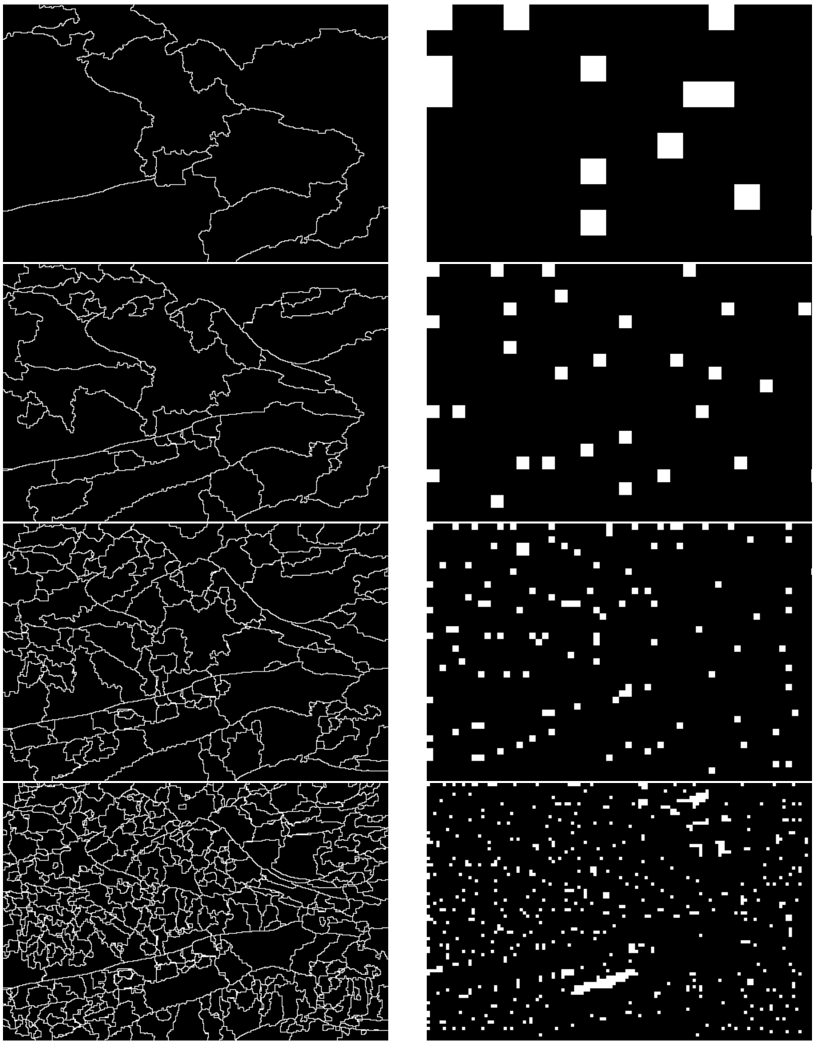


Figure 3.5 – Contour images corresponding to four decomposition levels (levels 4, 3, 2, 1 respectively) of the image presented in figure 3.4. The markers at each scale are displayed on the right images.

the sense that if a pixel of the contour image $C^{(l)}$ is labelled as a contour, then the same pixel in the contour image $C^{(l-1)}$ is also labelled as a contour. The proposed approach therefore results in a hierarchical segmentation. We can see that as expected, the contours of the largest objects tend to be extracted for the approximations with the lowest resolution. By contrast, all contours corresponding to the details of the image to be segmented appear for the approximation images of higher resolution.

Experimental results and discussion

In section 3.2, we introduced a hierarchical contour detection algorithm based upon successive multi-resolution approximations of the image to be segmented. It is of interest to assess the validity of the approach by comparing the results of the algorithm to human segmentations. To that end, we rely on the Berkeley Segmentation Database (BSD) [Mar01]. The BSD is a large dataset of natural images that have been manually segmented. At each scale of the contours hierarchy, we compare the results of the detection algorithm to the human annotations.

In our comparison, we consider two criteria. We first estimate the proportion of contours detected by the algorithm and that have also been annotated (precision). To that end, we apply a dilation of size two to the contour image corresponding to the human segmentation, and we consider the intersection between the dilated image and the contour image returned by the algorithm at each decomposition level. The dilation is applied to account for potential inaccuracy of the human contour detection. The number of pixels belonging to the intersection normalized by the number of pixels corresponding to the detected contours provides us with an estimate of the proportion of detected contours that have also been manually segmented. The second criteria that we consider is the proportion of contours that have been detected by humans and that are also detected by the algorithm (recall). At each decomposition level, we apply a dilation of size two to the contour image returned by the algorithm and we consider the intersection between the dilated image and the contour image corresponding to the human segmentation. By counting the number of pixels of the intersection normalized by the number of pixels belonging to the human detected contours, we can determine the proportion of actual contours that are returned by the detection algorithm at each scale. Note that these same criteria were employed in Chap. 2 for evaluating the Eikonal based superpixel algorithm.

We estimated both criteria on 200 images from the BSD. The results are presented in table 3.1. We can note that on average, the precision increases with the decomposition level in the contours hierarchy. This tends to assess the validity of the proposed multi-scale approach, in the sense that the contours detected with markers obtained after several decomposition levels have a significantly higher probability to correspond to hu-

	Wavelet filtering	
Dec. Level	Precision	Recall
0	0.10 ± 0.05	0.94 ± 0.03
1	0.13 ± 0.07	0.82 ± 0.07
2	0.17 ± 0.09	0.69 ± 0.11
3	0.22 ± 0.12	0.53 ± 0.14

h-minima		Alternate Sequ. Filters	
Precision	Recall	Precision	Recall
0.12 ± 0.06	0.85 ± 0.16	0.10 ± 0.04	0.91 ± 0.06
0.17 ± 0.08	0.71 ± 0.18	0.12 ± 0.05	0.69 ± 0.09
0.24 ± 0.13	0.51 ± 0.14	0.12 ± 0.05	0.46 ± 0.08
0.30 ± 0.20	0.21 ± 0.08	0.12 ± 0.05	0.22 ± 0.07

Table 3.1 – Results of the wavelet based algorithm on the BSD for each decomposition level. Precision corresponds to the average proportion of contours detected by the algorithm that have also been annotated, along with the corresponding standard deviation. Recall corresponds to the average proportion of contours that have been detected by humans and that are also detected by the algorithm, along with the corresponding standard deviation. The proportions are obtained on a database of 200 images. The results of h-minima based segmentation and alternate sequential filtering based segmentation are also presented.

man detected contours than contours directly obtained with the watershed transform. We also note that the recall of the algorithm decreases monotonously on average. This trend was to be expected, since the multi-scale approach inherently removes the contours of the smallest patterns. However, up to three decomposition levels, the recall remains higher than 0.5.

It is finally of interest to compare the results of the wavelet based markers selection to other commonly encountered methods, namely markers selection through h-minima of the image gradient and contour detection after filtering by alternate sequential filters. The difficulty here is to obtain a comparable number of segments between the distinct approaches. To that end, for each image, we select the value of h-minima and the size of structuring element for the alternate sequential filter, respectively, that yield the number of markers the closest from the one used in the wavelet based segmentation. We repeat the process for each decomposition scale. Next, we rely on the aforementioned procedure to estimate the precision and the recall of these methods. The results are summarized in table 3.1.

We note that, on average, the segmentation based upon the discrete wavelet transform performs significantly better in terms of both precision and recall than the segmentation following an alternate sequential filtering. Our interpretation of this result is that the wavelet transform better preserves the contours of the original image at the highest scales of the transform. By contrast, for structuring elements of large size yielding a number of markers similar to the one obtained with the wavelet decomposition, alternate sequential filters significantly degrade the contours of the image, which explains the poor results that are registered in terms of precision and recall. Marker selection through h-minima values is shown to yield a higher precision than the wavelet based algorithm. However, in terms of recall, the wavelet based algorithm performs significantly better on average. Both approaches remain however significantly distinct and are difficult to compare, since a segmentation based upon markers selection by h-minima is highly sensitive to the local minima of the image gradient.

Conclusion and perspectives

In this chapter, we presented a new method to select markers for the watershed algorithm based upon multi-resolution approximations. By relying on the discrete decimated wavelet transform to obtain successive approximations of the image to be segmented, we were able to define a hierarchical structure for the detected contours. We evaluated the performance of the proposed approach by comparing its results to manually segmented images from the Berkeley segmentation database. The comparison provided an empirical evidence that the contours detected for the approximation of lowest resolutions have a higher probability to correspond to human detected contours than contours detected by the classical watershed transform. A potential application of this work could be to use the saliency of the contours, defined here as the level of the pyramid at which the contour disappear, as a feature in a subsequent region merging algorithm.

Chapter 4

Learning similarities between regions

The first part of this PhD thesis was devoted to the study of superpixel algorithms. In a number of approaches to image segmentation, computing a superpixel partition of the image constitutes the first step of the segmentation process. Superpixels allow to considerably simplify the image representation. Subsequent steps of the segmentation process usually aim at grouping superpixels together to obtain an actual image segmentation. To perform the clustering of the superpixel, two key ingredients are required. First, one needs to define precisely a notion of similarity (or dissimilarity) between adjacent superpixels. Second, one needs to define a proper clustering algorithm to perform the superpixel merging using the aforementioned distance.

In this chapter, we investigate the first issue, namely the definition of a similarity measure between adjacent superpixels. While univariate criteria including for instance the difference of mean gray level or the boundary strength directly qualify to be such measures of similarity, these criteria often remain too simplistic to capture the complexity of the problem. It is therefore of interest to investigate the use of machine learning algorithms to combine low level criteria into a single similarity measure.

The outline of this chapter is as follows. First, we describe the general approach that we adopt to learn how to merge superpixels. In section 4.2, we present various kinds of features extensively used in the literature and in our work. In section 4.3, we briefly describe two machine learning algorithms, random forest and XGBoost classifiers, that we use to combine the selected features into a similarity measure for each pair of adjacent regions. The classification results are presented in section 4.4.

Introduction

The objective of image segmentation is to partition an image $\mathcal{I}$ into a family of regions or *segments* $\{S_0, S_1, ..., S_N\}$, where N is the desired number of segments, and S_i is a set of pixels within the same segment: $S_i = \{p : l(p) = i\}$, each segment being indexed with a label i.

Machine learning algorithms are generally used for classification or regression problems, and can be roughly grouped into three categories: supervised algorithms, non-supervised algorithms and semi-supervised algorithms. In this study, our work takes place within the supervised learning framework, where we have a set of training images, a test set, and sometimes a validation set at our disposal. In this framework, our objective is to find the parameters for a model that minimize the loss on training images and generalize well on test images.

The image segmentation problem can classically be formulated as a classification problem. The classification can be done per pixel, as in [Arb11], where a feature vector is calculated for every pixel. According to the ground truth, each pixel in the training images is labeled 1 if it is on the boundary or 0 if it is not, or with the probability of being a boundary pixel if multiple ground truth images are provided. Feature vectors and labels of pixels from training images are then used to train a classifier. At test stage, the feature vectors for each pixel in the test images are calculated, and predictions of the probabilities of being on the boundary are made by the classifier. The probability is interpreted as a contour strength for the pixels in the test images. Using this approach, depending on the size of the image, there can be a large number of feature vectors to be calculated and a large number of entities to be classified. In addition, the resulting contours may not be closed, which led Arbeláez *et al.* [Arb11] to propose an extension of the watershed transform, the oriented watershed transform, to partition the image into regions. Other contour based methods for computing an image segmentation include supervised learning of edges and objects boundaries [DTB06], the structured forests for fast edge detection [DZ15] and so on.

Another possible approach is to first oversegment the image into small homogeneous regions $\{R_1, R_2, ..., R_K\}$, and then to gradually merge these regions to obtain the segmentation. This is the approach adopted in this chapter. In the oversegmentation step, it is of paramount importance to preserve the boundaries. Hence, the proposed approach is only effective when applied to a superpixel with good boundary adherence properties.

Learning procedure

Instead of predicting the probability of being a boundary pixel as in [Arb11], we predict the merging probability of adjacent regions. Let us consider an image previously oversegmented into a set of superpixels $\{R_0, R_1, ..., R_K\}$. The training and test procedure is the same as in pixel based segmentation approaches, except that the superpixels provide a support for features calculation. The training procedure comprises three steps:

1. *Feature extraction*: we compute a feature vector $\mathbf{v}_m$ for each adjacent superpixel

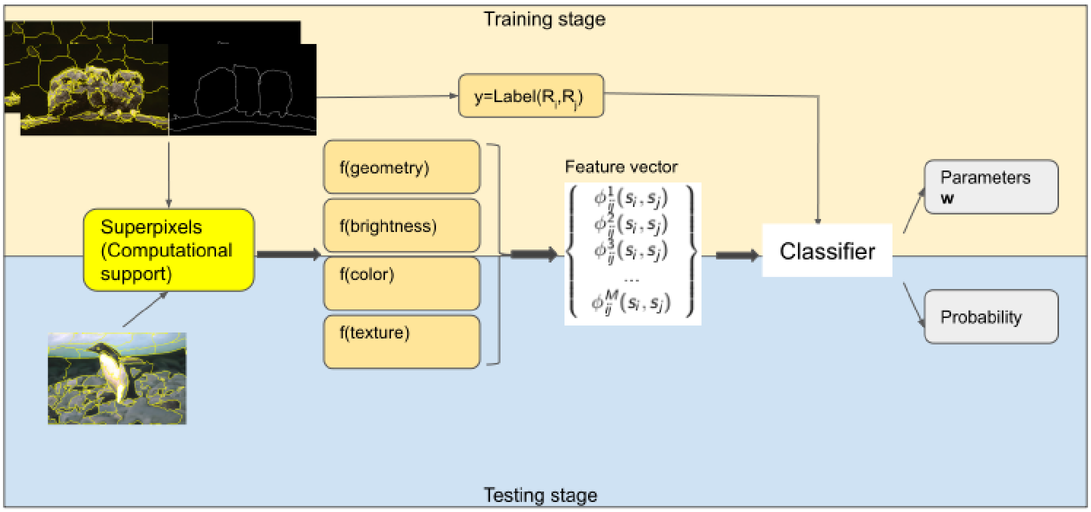


Figure 4.1 – Superpixel based machine learning for image segmentation: a schematic view

region pair (R_i, R_j).

2. *Labeling*: given a ground truth image for the segmentation task, we label each pair of adjacent regions (R_i, R_j) by 1 if they belong to the same ground truth segment g, to indicate that R_i, R_j should merge; by 0 if they belong to different ground truth segments. An alternative is to rely on a merging probability when multiple ground truth segmentations are available. Due to errors introduced by the superpixel segmentation, a superpixel can intersect several ground truth segments. In this case, we consider that the superpixel belongs to the segment with the highest overlap.
3. *Training*: the pairs $\{\mathbf{v}_m, l_m\}_{m=1}^{M}$ are used to learn parameters Θ of the model, where $\mathbf{v}_m$ is the feature vector of the region pair, l_m is the corresponding label, and M is the number of adjacent region pairs.

After the training stage, we possess a model with parameters optimized under a score, for example the F measure. In test stage, the same set of features is extracted for each pair of adjacent regions, and fed to the model to predict the merging probability of region pairs. A schematic view of the learning procedure is provided in figure 4.1.

The adjacent region pairs are arranged in a Region Adjacency graph (RAG) $\mathcal{G} = (V, E)$, where V is the ensemble of vertices, and E is the ensemble of edges. Each region is represented by a node v, and there is an edge between v_i and v_j when their corresponding regions R_i and R_j are adjacent. The predicted merging probability p_{ij} will be used as the edge weight w_{ij} for further processing, the most intuitive being thresholding, which we will talk about later in this chapter.

It is possible to generalize the model learned from the training set for the prediction of merging probability of region pairs in the test images, even if these images have highly dissimilar appearances, since contours bear things in common, including color changes, texture changes or strong gradient. Our first task is to determine features that are relevant to the classification of merging. In the next section, we describe the set of features classically used in the segmentation literature and in this research for this purpose.

Features

Features that characterize region properties are crucial for machine learning based algorithms. One can distinguish between low level features, such as the coherence of brightness, color, texture, or motion; mid-level features, like Gestalt cues (proximity, good continuation, etc); or high-level knowledge [SM00]. In state-of-the-art algorithms [Arb11], brightness, color and texture cues are combined to form a contour detector. In this section, we present color features, geometric features, contour features and texture features employed in this research.

Color features

Color features are features based on the color difference between regions. There are two means to define such features. The first one is to consider differences between statistical moments, for instance the difference of the average color values, or the color variance of the combined region. The second one is to consider histograms to characterize the difference between color distributions. Besides the conventional RGB color space, Lab and HSV color spaces are also employed in the literature for color features calculation, for example in [LST16].

Lab color space: Lab (La*b*) is the abbreviation for the three-dimensional CIELAB color space. L is the lightness, ranging from 0 (darkest black) to 100 (brightest white). a* and b* are two color channels. The a* axis represents the green/red component, ranging from green (-128) to red (127). The b* axis represents the blue/yellow color component, ranging from blue (-128) to yellow (127). The Lab color space is perceptually uniform and independent of devices. [Lab]

HSV color space: The abbreviation HSV stands for hue, saturation and value, and corresponds to a cylindrical coordinate model. The hue is in the angular dimension and represents pure colors, starting from red (0°), passing green (120°), blue (240°), and returning to red (360°). Saturation is the radial distance to the axis of the cylinder, ranging from 0 (on the axis) for white to 1 (on the surface of the cylinder) for pure color, with tint color in between. Value is the height dimension, from black (bottom) to white (top). [Hsv]

Different color spaces provide different ways for color decomposition, the employment of which might be helpful for the classification of region merging. Hence, in this research, color features are computed simultaneously in RGB, Lab and HSV color spaces. The list of the 34 color features used in this research can be found in table 4.1.

name	dimension	notation
difference of mean gray value	1	Δm_g
difference of mean of RGB, Lab and HSV channels	9	Δm_{c_k}
Euclidean distance of mean of RGB, Lab and HSV channels	3	Δ_{rgb}, Δ_{lab}, Δ_{hsv}
variance of combined region in RGB, Lab and HSV channels	9	$\sigma^2_{c_k}$
χ^2 distance of RGB, Lab and HSV histograms	15	Δh_{c_k}
χ^2 distance of flat histograms of RGB, Lab and HSV channels	3	Δh_{rgb}, Δh_{lab}, Δh_{hsv}

Table 4.1 – A summary of color features

Difference of statistics

Let us consider a pair of adjacent regions (R_i, R_j), where $R_i = \{p : (x, y) \mid p \in R_i\}$, and $R_j = \{p : (x, y) \mid p \in R_j\}$. The mean gray level within R_i is

$$m_g(R_i) = \frac{1}{|R_i|} \sum_x \sum_y Ig_{xy}, \tag{4.1}$$

where Ig is the gray level image of $\mathcal{I}$. The difference of gray level mean is

$$\Delta m_g(R_i, R_j) = |m_g(R_i) - m_g(R_j)|. \tag{4.2}$$

We calculate differences of means for the gray level image, as well as for each color channel of RGB, Lab and HSV color spaces. We can also use the Euclidean distance of the mean color:

$$\Delta_{rgb}(R_i, R_j) = \sqrt{(m_r(R_i) - m_r(R_j))^2 + (m_g(R_i) + m_g(R_j))^2 + (m_b(R_i) + m_b(R_j))^2}. \tag{4.3}$$

Variance of the combined region $R_i \cup R_j$ is employed in [Che16] as a color feature. It is calculated for each color channel:

$$\sigma^2_{c_k}(R_{i\cup j}) = \frac{1}{n} \sum_{p \in R_{i\cup j}} (c_k(p) - m_{c_k}(R_{i\cup j}))^2, \tag{4.4}$$

where $\{c_k \mid k \in [1, 9], k \in \mathbb{Z}\}$ corresponds to color channels in RGB, Lab, and HSV color spaces respectively in this order, and n is the number of pixels in the combined region $R_i \cup R_j$.

Difference of histogram

A commonly used tool for the description of the color distribution is the histogram, which represents the number of pixels of each color interval. There are several methods for histogram difference measurement. It is shown in [MFM04] that the χ^2 difference of histogram is marginally superior to the L^1 norm, and significantly better than the Mallows distance, for local boundary detection in natural images. The χ^2 difference is expressed as:

$$\chi^2 = \frac{1}{2} \sum \frac{(g_i - h_i)^2}{g_i + h_i}, \tag{4.5}$$

where $g = [g_1, g_2, ..., g_k]$, $h = [h_1, h_2, ..., h_k]$ are two histograms with the same number of bins.

In this study, we compute the χ^2 distance for the histogram of each color channel in the RGB, Lab and HSV color spaces. Especially for Lab color space, the histograms are calculated on Gaussian filtered images with $\sigma = 2.5, 5, 10$ for l channel, and $\sigma = 5, 10, 20$ for color channels as in [Arb11], taking into account information from multiple scales. The number of bins of each color channel is set to 16. The χ^2 distance of the flat histogram of a color space is also calculated. A flat histogram is the joint histogram of three color channels within a color space. For example, for the RGB color space, it is a 48 value vector joining h_r, h_g and h_b.

Geometric features

Geometric features focus on the shape and on the geometric properties of the regions. A summary of geometric features employed in this work is given in table 4.2.

name	dimension	notation
area	2	a_{min}, a_{max}
perimeter	2	p_{min}, p_{max}
solidity	3	$s_{min}, s_{max}, s_{comb}$
extent	3	$b_{min}, b_{max}, b_{comb}$
eccentricity	3	$e_{min}, e_{max}, e_{comb}$
orientation	3	$o_{min}, o_{max}, o_{comb}$

Table 4.2 – A summary of geometric features

For a pair of regions R_i and R_j, we first compute the minimal and maximal areas, as well as the minimal and maximal perimeters.

Solidity is the ratio of the number of pixels in the region with that of the convex hull of the region:

$$solid(R_i) = \frac{|R_i|}{|convex(R_i)|}.$$

The convex hull is the smallest convex set that contains all pixels in a region. We take the minimal and the maximal solidity of two regions and that of the combined region $R_i \cup R_j$ as features.

Extent is the ratio of the number of pixels in the region with that in its minimum bounding box:

$$extent(R_i) = \frac{|R_i|}{|bbox(R_i)|}.$$

The minimum bounding box is the smallest rectangle that contains all pixels in a region. It is obtained by the leftmost, rightmost, topmost and bottommost coordinates of the region. Similarly to solidity feature, this feature reveals the convexity of a region. We take the minimal and maximal extent of two regions, and the extent of the combined region $R_i \cup R_j$ as features.

Eccentricity is the eccentricity of the ellipse with the same second central moment (covariance matrix) as the region. For $\mathbf{X} = [\boldsymbol{x}, \boldsymbol{y}]$, $\boldsymbol{x}$, $\boldsymbol{y}$ being two $n \times 1$ vectors representing the x and y coordinates of the pixels in R_i respectively, the covariance matrix is

$$\mathbf{M} = (\mathbf{X} - \boldsymbol{\mu})^\top (\mathbf{X} - \boldsymbol{\mu}),$$

where $\boldsymbol{\mu} = [\boldsymbol{m_x}, \boldsymbol{m_y}]$, $\boldsymbol{m_x} = m_x \times \mathbf{1}_n$, $\boldsymbol{m_y} = m_y \times \mathbf{1}_n$, and m_x, m_y are the means of $\boldsymbol{x}$, $\boldsymbol{y}$ respectively.

To compute the eccentricity, we need to calculate the eigenvalues and eigenvectors of the aforementioned matrix $\mathbf{M}$. The eigenvectors are the directions of the axes of the ellipse, and the corresponding eigenvalues are the length of the major axis a and that of the minor axis b. The eccentricity is defined as the ratio between the focal distance and the length of the major axis

$$\rho(R_i) = \frac{c}{a} = \sqrt{1 - \left(\frac{b}{a}\right)^2},$$

where $c^2 = a^2 - b^2$, and c is the focal distance. We take the minimal and maximal eccentricity of R_i and R_j and the eccentricity of $R_i \cup R_j$ as features.

Orientation features are proposed in [NI13]. The orientation of a region is the angle between the major axis of the aforementioned ellipse and the x axis. We calculate the angle between the orientation of a region and the line l connecting the centers c_i, c_j for adjacent regions R_i and R_j. We take the minimum and maximum of these two angles, and the difference between orientations of regions as orientation features.

In our work, we use the implementation in scikit-image library [Wal14] for computing the geometric features.

Contour features

A contour separates two adjacent regions. At the boundary of two objects, there is usually a color change, thus inducing high gradient on the contour. Conversely, a contour with large gradient is generally more probable to be an actual contour. Therefore, we take the minimum and average of gradient of pixels on a contour as contour features. This feature was also considered in the waterfall algorithm [Beu94], which removes the watershed lines of lower gradient values to suppress the over-segmentation.

name	dimension	notation
contour length	1	l
mean gradient	1	g_{mean}
minimum gradient	1	g_{min}

Table 4.3 – A summary of contour features

Many recent image segmentation algorithms [RS13; NI13; LST16] take boundary features based on gPb [Arb11]. In a recent article [Che16] Cheng *et al.* proposed to use average contour strength based on holistically-nested edge detection (HED), a contour detector based on convolutional neural network as a feature.

Contour features are usually of paramount importance for image segmentation problems. Still, it is important to note that it is difficult to separate two regions with similar colors by color features and contour features.

Texture features

Texture features are of great importance for the segmentation of natural images. In textured regions, the statistics of colors is helpless to distinguish two objects with different textures. Histogram might help, but it considers only the frequency of color appearance, while lacking the spatial arrangement of colors, which is important for distinguishing between textures. In this section we discuss texture features based on Gabor filtering and SIFT and SURF keypoint descriptors. A summary of texture features is given in table 4.4.

name	dimension	notation
χ^2 distance of texton histogram at 3 scales	3	Δt_1 Δt_2 Δt_3
χ^2 distance of SURF histogram on gray and a b channels	3	Δs_g Δs_a Δs_b
textureness	1	t

Table 4.4 – A summary of texture features

Texton

A typical way of analyzing textures is through filtering. Martin *et al.* proposed in [MFM04] to cluster the responses of an image to filters with different directions and scales into a small set of prototype response vectors, which they refer to as textons.

Here we follow this approach and adopt Gabor filters (introduced in section 2.4.2) for performing the texture analysis. Gabor filters with 6 directions and 3 scales are employed. These filters are shown in figure 4.2. In [Arb11] Arbeláez *et al.* indicate that the clustering of response vectors can be done per image or on the whole database. In practice, we find that the k-means clustering on large dimensional data takes much time and memory. So in this research the clustering is done per image.

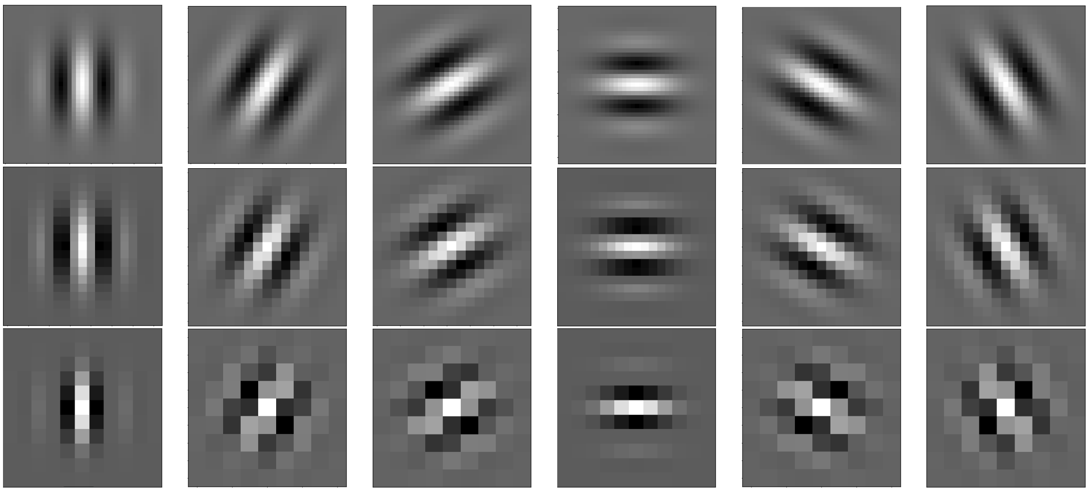


Figure 4.2 – Gabor filter kernels at 6 directions and 3 scales

The extraction of texture features works as follows:

1. We convolve the image with a set of Gabor filters, to get 18 response images. Thus, for each pixel p we obtain a 18 dimensional texture vector $\mathbf{v}_p$.
2. We cluster N texture vectors by using k-means clustering to get K textons, the most representative vectors. N is the number of pixels in image $\mathcal{I}$.
3. We affect to each pixel the index of its nearest texton (the vector with the minimum Euclidean distance).
4. We calculate the histogram of texton id inside each region, and we retain the χ^2 distance between R_i and R_j as texture feature.

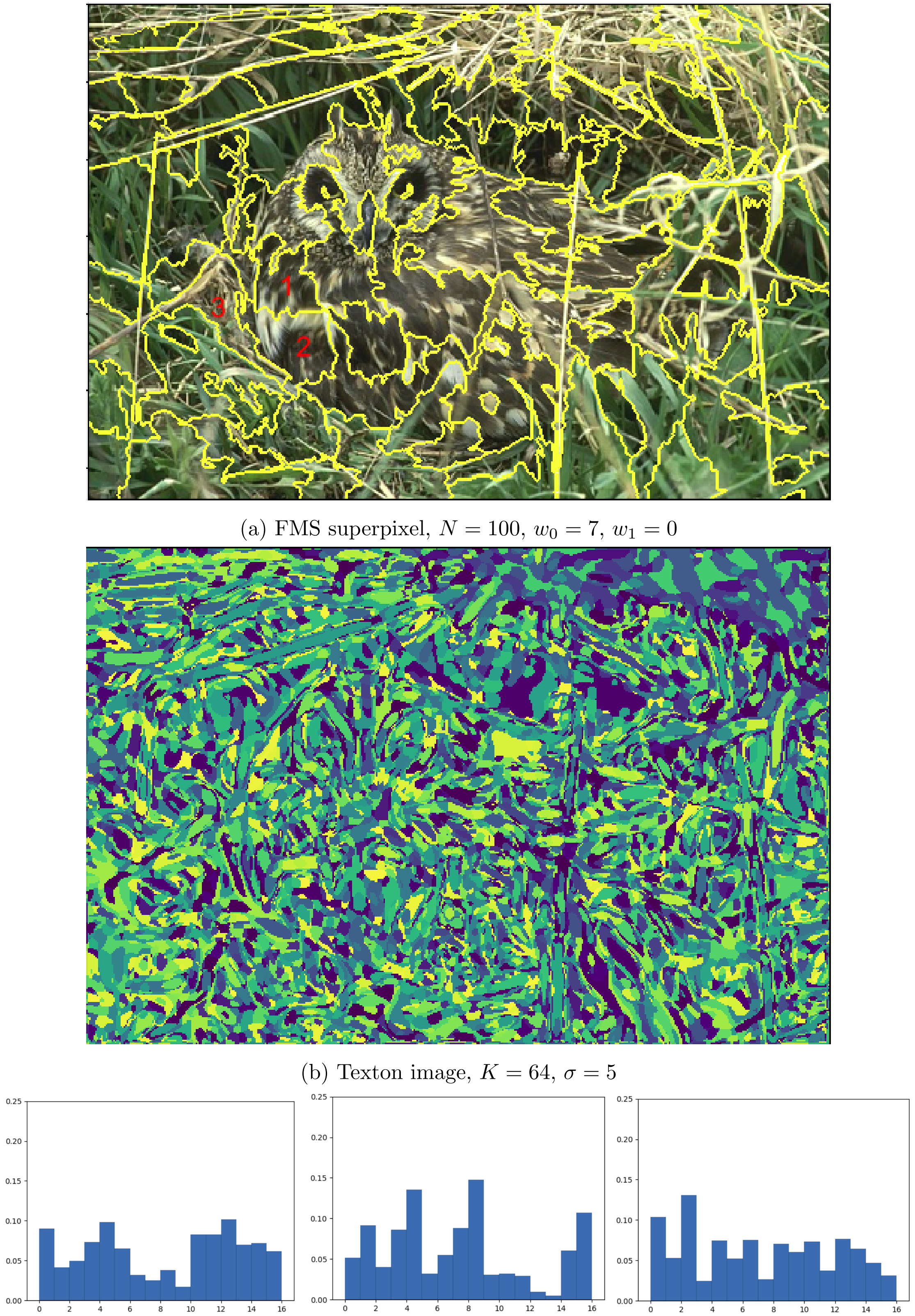
According to [Arb11], in order to take into account the information from multiple scales, the image should be smoothed with Gaussian filters with $\sigma = 5, 10, 20$ separately for the texture channel. We adopt the same processing, therefore having 3 texton images in total, and 3 texture features. An example of the texton images at 3 scales is given in figure 4.3, figure 4.4, and figure 4.5. We set the number of prototype vectors K to 64, and the number of bins for the textons histograms to 16.

We define a measure of *textureness* for a region by considering the χ^2 distance between its texton histogram at scale $\sigma = 5$ and the uniform histogram with the same bins. We take the difference of textureness between regions as a texture feature.

SIFT

We also explored the employment of SIFT (Scale-Invariant Feature Transform) descriptors for texture features. SIFT [Low04] is generally used for keypoint detection and description. In [RS13; LST16], SIFT descriptors are employed for characterizing texture. The SIFT descriptors are calculated densely on a pixel grid. Thus at a pixel on the grid we have a 128-dimensional SIFT descriptor. The rest is the same as for Gabor features, we cluster the texture vectors by k-means clustering to obtain K_s visual words. We calculate the histogram of visual words for each region, and take the χ^2 distance of two histograms of adjacent regions as the texture feature.

We tried to apply SIFT based texture features in our research. Since a SIFT descriptor is 128-dimensional, which is way larger than the dimension of a texture vector obtained by Gabor filtering, it takes much more time at the clustering step than the latter. Therefore we tried another commonly used keypoint descriptor SURF (Speeded-Up Robust Features). As its name suggests, the SURF descriptors are more computationally efficient than SIFT. Since a SURF descriptor has 64 dimensions, the clustering time is also reduced to a large extent compared to SIFT. We take the χ^2 distance of SURF visual words histograms at gray level image and a and b color channels of Lab color space as texture features.



(a) FMS superpixel, $N = 100$, $w_0 = 7$, $w_1 = 0$

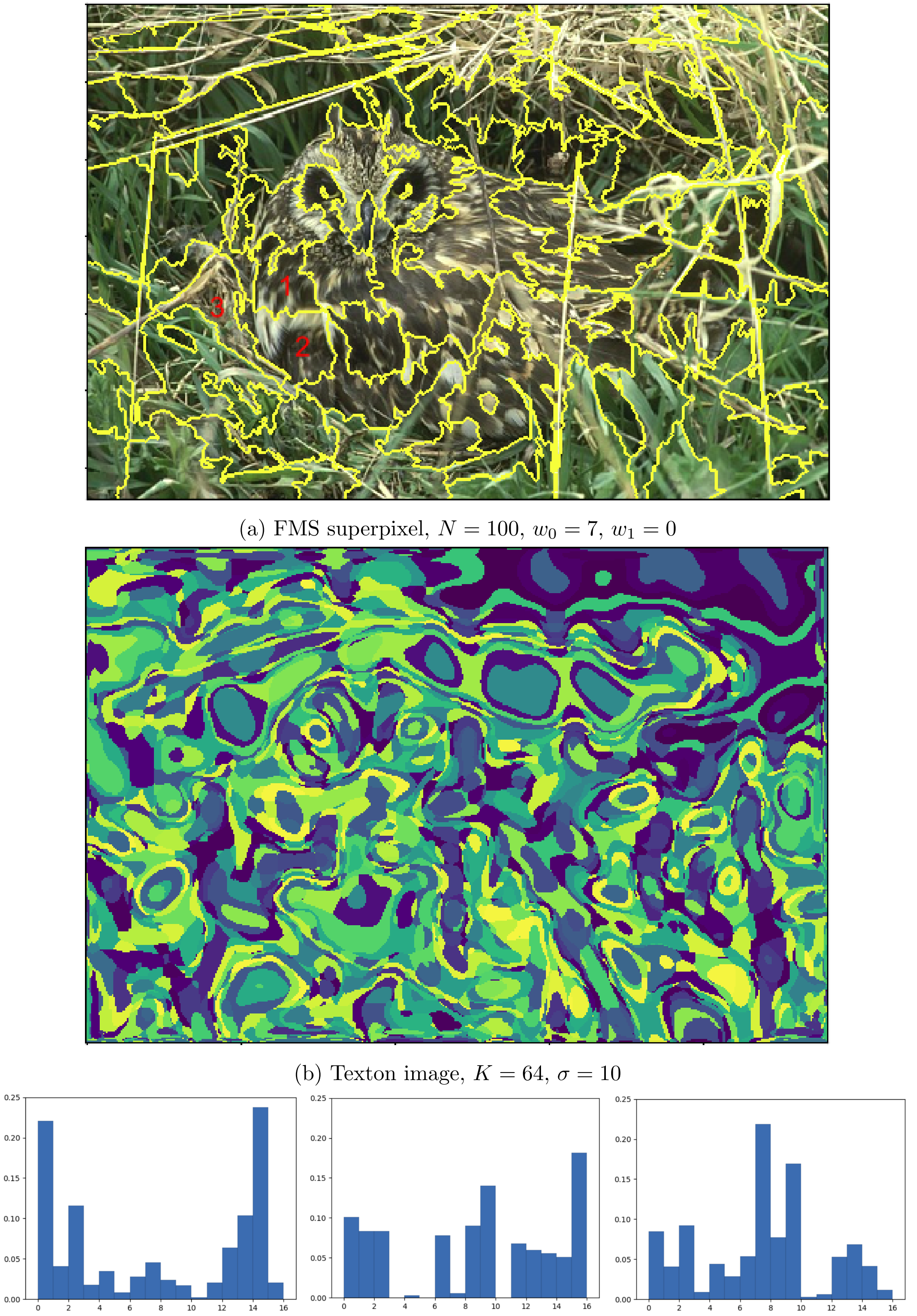
(b) Texton image, $K = 64$, $\sigma = 5$

(c) Texton histogram for region 1

(d) Texton histogram for region 2

(e) Texton histogram for region 3

Figure 4.3 – An example of texton image at scale $\sigma = 5$



(a) FMS superpixel, $N = 100$, $w_0 = 7$, $w_1 = 0$

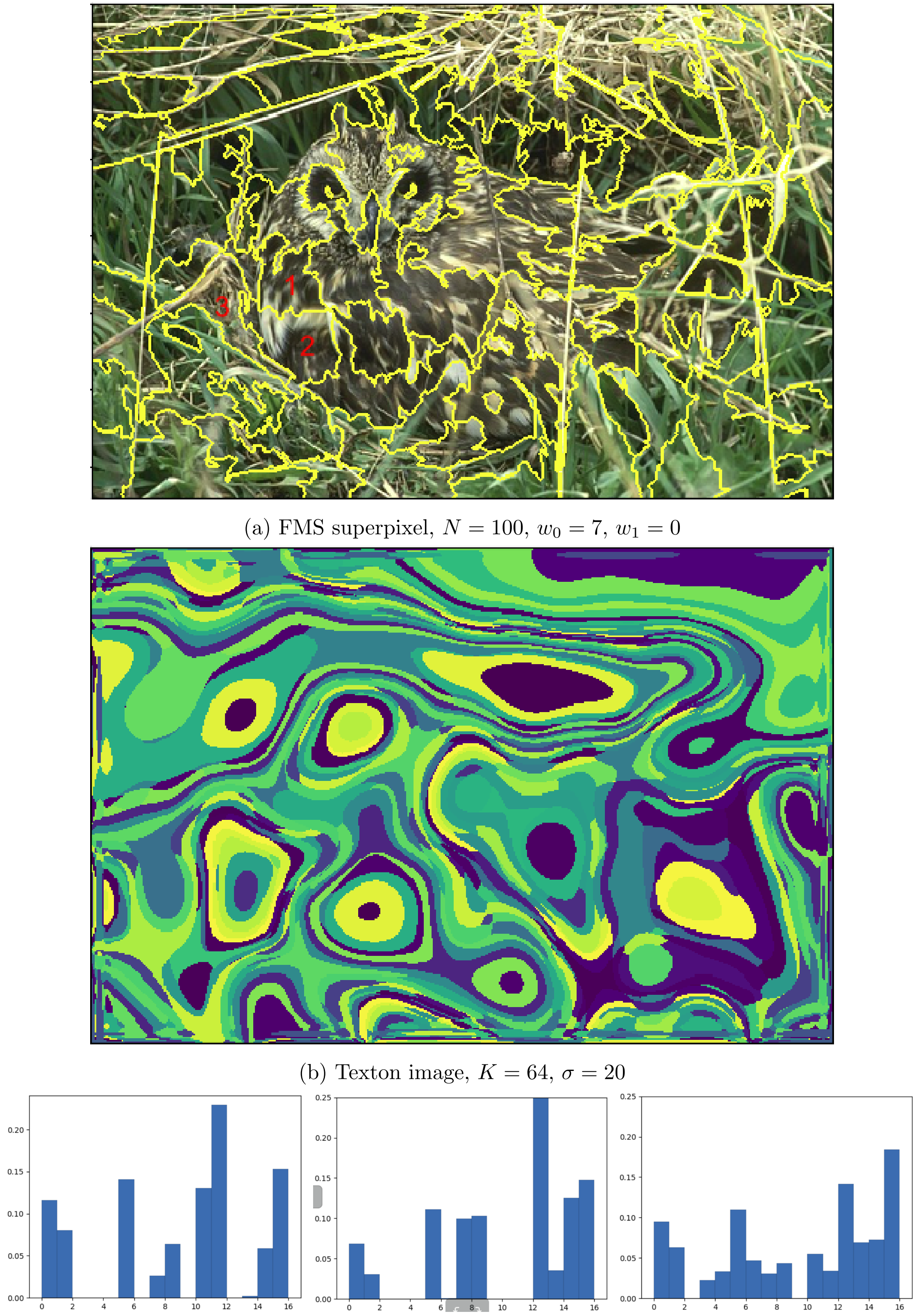
(b) Texton image, $K = 64$, $\sigma = 10$

(c) Texton histogram for region 1

(d) Texton histogram for region 2

(e) Texton histogram for region 3

Figure 4.4 – An example of texton image at scale $\sigma = 10$



(a) FMS superpixel, $N = 100$, $w_0 = 7$, $w_1 = 0$

(b) Texton image, $K = 64$, $\sigma = 20$

(c) Texton histogram for region 1

(d) Texton histogram for region 2

(e) Texton histogram for region 3

Figure 4.5 – An example of texton image at scale $\sigma = 20$

Classifiers

A classifier is a model that can make classifications having the features as input. There are many kinds of classifiers, including logistic regression, SVM and so on. Our research focuses on two decision tree-based classifiers: random forest and XGBoost. In this section we introduce these two algorithms.

A classifier learns information from the training set. For tree based classifiers, the information is contained in the structure of the trees. A *learning algorithm* fits the parameters Θ using training examples $\{\mathbf{x}_i, l_i\}_{i=1}^{N}$, where the labels are known. The classifier seeks to model the posterior probability distribution over world state, given the observed feature vector, and to be able to infer the posterior probability $p(\mathbf{c}|\mathbf{v})$ of a class knowing the feature vector.

Data preparation A feature vector $\mathbf{v}_f$ is extracted for each pair of adjacent regions (R_i, R_j) in the image. A ground truth segmentation $G = \{g_1, g_2, ..., g_{n_g}\}$ is provided, where n_g is the number of segments. We label a vector 1 ($l = 1$) if R_i, R_j should be merged ($\exists g_k \in G, \text{where } R_i \subset g_k, \text{and } R_j \subset g_k$), or 0 ($l = 0$) if they belong to different ground truth segments.

We refer to the classical references [HTF09] for the description of decision trees, and [Bre01] for random forest.

Decision tree

A decision tree partitions the feature space into a set of disjoint regions $\{R_i\}_{i=1}^{T}$, by making recursively binary partitions. These regions are represented by leaves (terminal nodes) of the tree. T represents the number of leaves of the tree. Each leaf (region) corresponds to a unique class c_i. The predictive rule is

$$\mathbf{x} \in R_i \Rightarrow f(\mathbf{x}) = c_i.$$

A classification tree can be viewed as:

$$T(\mathbf{x}, \Theta) = \sum_{j=1}^{T} c_j I(\mathbf{x} \in R_j), \tag{4.6}$$

where $\Theta = \{R_j, c_j\}_{j=1}^{T}$ and $I(\mathbf{x} \in R_j)$ is the indicator function of region R_j.

A tree is constructed by a greedy algorithm. Starting with all data points, we seek to determine at each step a splitting variable j and a splitting point s, by minimizing the sum of the losses of the left and right nodes after the split:

$$\min_{j,s}\left[\min_{c_L}\sum_{\mathbf{x}_i\in R_L(j,s)} l(y_i,\hat{y}_i)+\min_{c_R}\sum_{\mathbf{x}_i\in R_R(j,s)} l(y_i,\hat{y}_i)\right], \tag{4.7}$$

where l is the loss function, $R_L = \{\mathbf{x}_i \mid x_{ij} \leq s\}$ and $R_R = \{\mathbf{x}_i \mid x_{ij} > s\}$, x_{ij} represents the jth feature of the features vector $\mathbf{x}_i$, c_L and c_R are the class adopted in R_L and R_R respectively.

For a node m, the observations in R_m are classified to the class $k(m) = \arg\max_k \hat{p}_{mk}$, where the most observations belong. $\hat{p}_{mk}$ is the proportion of observations in class k:

$$\hat{p}_{mk} = \frac{1}{N_m}\sum_{\mathbf{x}_i\in R_m} I(y_i = k),$$

where N_m is the number of observations in region R_m. For binary classification, the classes k are simply the elements of the set $\{0, 1\}$.

The impurity measure is employed as the loss function l in (4.7). Two commonly used measures are:

- Gini index: $\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}) = 2p(1-p)$
- Cross-entropy: $-\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk} = -p\log p - (1-p)\log(1-p)$

where p is the proportion of one class in node m in case of binary classification.

To split a node m, we iterate through all the features:

1. For each feature we iterate through all the examples to find the splitting point, and take the feature and splitting point that minimize (4.7).
2. We split the observations in node m to the left and right node according to the adopted splitting rule.
3. We assign to R_L and R_R its corresponding class c_L and c_R.

The partition is repeated on all regions (leaves in the tree).

The tree is constructed in the training stage. At test stage, a feature vector extracted from a test image go through the tree, and is assigned to a leaf according to the splitting rules. The corresponding class of that leaf is taken as the predicted class.

One advantage of decision tree classifiers is their good interpretability. One disadvantage is that they are prone to overfitting. Various pruning methods can be used to avoid overfitting. Other approaches including random forests work by growing several trees in parallel.

Random forests

Definition 4.3.1. A random forest is a classifier consisting of a collection of tree-structured classifiers $\{h(\mathbf{x}, \Theta_k), k = 1, 2, ...\}$, where the Θ_k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input $\mathbf{x}$. [Bre01]

Random forests are an ensemble of trees. The examples from the training set are sampled uniformly with replacement at the construction of each tree, a procedure called **bagging**. Bagging, also known as bootstrap aggregating, is a commonly used algorithm to reduce variance. We bag repeatedly on the training data to build K trees. For the kth tree, we sample n_k training examples with replacement $\{\mathbf{X}_k,\ \mathbf{y}_k\}$, and train a decision tree $T(\mathbf{x}; \Theta_k)$ using these examples. Each tree is grown independently and is identically distributed.

The randomness of random forests also lies in the random selection of features. At each split, m ($m \leq M$) features are randomly selected for candidates, where M is the total number of features. It is recommended in [HTF09] to use $m = \sqrt{M}$ for classification. The aim of this random selection is to reduce the correlation between trees, thus reducing the generalization error.

For a regression random forest, the prediction is

$$\hat{f}_{rf}^{K}(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} T(\mathbf{x}; \Theta_k), \tag{4.8}$$

which is the average of the prediction of each tree.

For a classification random forest, at the test stage, each tree votes, and the class of the majority vote is taken as the predicted class.

The improvement of random forests compared to decision trees is merely the variance reduction, while the bias is the same as for individual trees. Random forests are robust to noise, and they are less likely to overfit. This algorithm runs faster than boosting trees, since the trees in a forest can be developed in parallel.

Hyperparameters tuning

In this work, we use the implementation of random forest in scikit-learn [Ped11]. The main hyperparameters are:

— *class_weight*: weights associated with classes. We choose “balanced”, since the input data is highly imbalanced, with much more 1s than 0s. A weight inversely proportional to the class frequencies is associated to each class to balance the importance of both classes.

- *n_estimators*: number of trees in the forest. A sufficient number of trees are needed to reduce the variance, after which the performance does not improve with the increase of tree numbers.
- *max_features*: number of features to consider at each split. We adopt "sqrt", which means the square root of the total number of features.
- *max_depth*: maximum depth of the tree. A higher depth needs more computation.
- *min_samples_split*: minimum number of samples required to split a node.
- *min_samples_leaf*: minimum number of samples required to be a leaf node.

We tune *max_depth*, *min_samples_split* and *min_samples_leaf* to control the complexity of the constructed trees. A grid search can be done to find the optimal hyperparameters. In practice, we start from initial values, then search on each hyperparameter separately. This iteration can be done several times until there is no obvious improvement. We evaluate the performance of a classifier by its AUC-ROC score.

AUC-ROC score: AUC-ROC is the area under the ROC curve. ROC (Receiver Operating Characteristic) is a curve that plots true positive rate against false positive rate:

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN},$$

where TPR, FPR stand for true positive rate and true negative rate, TP, TN, FP, and FN stand for true positive, true negative, false positive and false negative respectively.

The ROC characterizes the ability of a classifier to separate negative and positive instances. A classifier with high AUC is desirable, meaning that it can achieve a high true positive rate while the false positive rate is low.

In this segmentation problem, the ROC is more appropriate than the precision recall curve to evaluate a classifier, since it is more important to lower the false positive rate. We remind that class 0 signifies "not merge", which means that there is a contour between two regions. It is desirable to have a high recall of real contours, thus low false positive rate. We can make compromise on the TPR, since false contours can be merged in later processing. The precision recall curve only focuses on the correct classification of positive instances. Another possible solution is to inverse the labels of the training examples, and use the precision recall curve.

XGBoost

XGBoost (Extreme Gradient Boosting) is proposed in [CG16], and is a state-of-the-art tree boosting system. In this subsection, we refer to [HTF09] for the description of boosting trees and [CG16] for XGBoost.

Boosting trees

Boosting fits an additive expansion in a set of basis functions. For boosting trees, the basis functions are individual trees. Boosting trees can be expressed as a sum of trees:

$$f_M(\mathbf{x}) = \sum_{m=1}^{M} T(\mathbf{x}; \Theta_m), \tag{4.9}$$

where Θ_m includes split variables, split points, and the predictions at the leaves of the trees. The parameters $\{\Theta_m\}_{m=1}^{M}$ are obtained by minimizing a loss function over the training data

$$L(\mathbf{f}) = \sum_{i=1}^{N} l(y_i, f(\mathbf{x}_i)), \tag{4.10}$$

where $f(\mathbf{x})$ is a series of trees, l is the loss function and N is the number of training examples. The training can be formulated as the optimization problem

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}), \tag{4.11}$$

where $\mathbf{f} \in \{0, 1\}^N$ are the predictions $f(\mathbf{x}_i)$ made at training data points:

$$\mathbf{f} = \{f(\mathbf{x}_i), f(\mathbf{x}_2), ..., f(\mathbf{x}_N)\}.$$

The direct optimization of the loss function (4.10) is infeasible. Stagewise additive modeling is commonly used for the construction of boosting trees, where the trees are built sequentially. Each tree seeks to reduce the residual error of the previous trees.

Stagewise additive modeling Boosting trees can be constructed by forward stagewise additive modeling, adding new basis functions without modifying the trees built before. When constructing the m-th tree, the parameters Θ_m can be obtained by

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i; \Theta_m)) \tag{4.12}$$

knowing the previous $m-1$ trees, where $\hat{\Theta}_m = \{R_j, c_j\}_{j=1}^{J_m}$, R_j is a leaf of the m-th tree, c_j is the class predicted by leaf j, J_m is the number of leaves in the m-th tree, and $f_{m-1}(\mathbf{x})$

is the current model with $m-1$ trees:

$$f_{m-1}(\mathbf{x}) = \sum_{t=1}^{m-1} T(\mathbf{x};\Theta_t). \tag{4.13}$$

The procedure of the forward stagewise modeling to construct boosting trees is described in Algorithm 1.

Algorithm 1 Forward Stagewise Additive Modeling

```
1. Initialize $f_0(\mathbf{x}) = 0$.
2. For $m = 1$ to M:
    (a) Compute
            $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + T(\mathbf{x}_i;\Theta_m))$
    (b) Set
            $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + T(\mathbf{x},\Theta_m)$
```

The construction of an individual tree can be divided into two steps: finding the splitting variables and splitting points of a region, and calculating the prediction values in each region.

— Finding c_j given R_j: to assign a class to a given leaf, for classification problem we can take the class to which the most observations falling in the leaf belong.

— Finding R_j: to find the split variables and split points for a region under a certain loss.

Compared with random forests, an advantage of boosting trees is that they can reduce bias. They are however much slower to train since boosting trees are built sequentially.

XGBoost

Having a training data set with N examples and K features $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$ ($\mathbf{x}_i \in \mathbb{R}^K$, $y_i \in \mathbb{R}$), the prediction of regression boosting trees is given by

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{m=1}^{M} f_m(\mathbf{x}_i),\ f_m \in \mathcal{F}, \tag{4.14}$$

where $\mathcal{F} = \{f(\mathbf{x}) = w_{q(\mathbf{x})}\}$ ($q : \mathbb{R}^K \to T$, $\mathbf{w} \in \mathbb{R}^T$), q is the structure of a tree that maps a feature vector to the index of a leaf, and T is the number of leaves in the tree. f_m is composed of the tree structure q and of the weight associated to each leaf $\mathbf{w} = \{w_1, w_2, ..., w_T\}$, where w_i is the score on the i-th leaf.

The set of functions $\mathcal{F}$ can be learnt by minimizing the regularized objective

$$\begin{aligned} L(\phi) &= \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{m=1}^{M} \Omega(f_m) \\ \text{where } \Omega(f) &= \gamma T + \frac{1}{2}\lambda \left\| \mathbf{w} \right\|^2, \end{aligned} \tag{4.15}$$

where l is the loss function, and $\Omega(f)$ is a regularization term used for reducing the trees complexity as well as smoothing the weights in order to avoid over-fitting.

The model is trained using stagewise additive modeling (Algorithm 1). Having the prediction given by the previous $m-1$ trees $\hat{y_i}^{(m-1)}$, we need to minimize the following equation to select the parameters of the m-th tree:

$$L^{(m)} = \sum_{i=1}^{N} l(y_i, \hat{y_i}^{(m-1)} + f_m(\mathbf{x}_i)) + \Omega(f_m),$$

of which the second order approximation is

$$L^{(m)} \simeq \sum_{i=1}^{N} [l(y_i, \hat{y}_i) + g_i f_m(\mathbf{x}_i) + \frac{1}{2} h_i f_m^2(\mathbf{x}_i)] + \Omega(f_m),$$

where $g_i = \partial_{\hat{y}^{(m-1)}} l(y_i, \hat{y}_i)$, $h_i = \partial^2_{\hat{y}^{(m-1)}} l(y_i, \hat{y}_i)$ are the first and second order gradient respectively. We remove the constant term to simplify the objective

$$L^{(m)} = \sum_{i=1}^{N} [g_i f_m(\mathbf{x_i}) + \frac{1}{2} h_i f_m^2(\mathbf{x}_i)] + \Omega(f_m). \tag{4.16}$$

Replacing Ω by its expression in equation (4.15) yields

$$\begin{aligned} L^{(m)} &= \sum_{i=1}^{N} [g_i f_m(\mathbf{x_i}) + \frac{1}{2} h_i f_m^2(\mathbf{x}_i)] + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T, \end{aligned} \tag{4.17}$$

where $I_j = \{i \mid q(\mathbf{x}_i) = j\}$ is the set of index of examples that fall in leaf j.

If the structure of the tree is fixed, the optimal weight w_j^* of leaf j is obtained by setting $\partial_{w_j} L^{(m)} = 0$

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}. \tag{4.18}$$

The corresponding loss is

$$L^{(m)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \tag{4.19}$$

To construct a tree, we start from the root and iteratively split the node greedily. This loss can be used to select from the possible split points. The loss reduction by splitting a node is

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma, \tag{4.20}$$

where I_L, I_R are sets of indices of examples that are attributed to the left and right node respectively after the split; $I = I_L \cup I_R$ contains all examples before the split.

Shrinkage and feature sampling are used to prevent overfitting. Other insights such as weighted quantile sketches are used to enable the algorithm to solve large scale machine learning problems with far fewer resources. For details of this algorithm please refer to [CG16].

Hyperparameter tuning

We use the implementation of XGBoost package[1] on Github. The main hyperparameters are:

— *scale_pos_weight*: this hyperparameter is used to balance the weights of positive and negative examples. The typical value is the number of negative instances over the number of positive instances.

— *n_estimators*: number of trees. A large number of trees can reduce the loss, but greatly increases the training time.

— *max_depth*: maximum depth of each tree.

— *gamma*: minimum loss reduction required to split a node.

— *subsample*: subsample ratio of training instances.

— *colsample_bytree*: subsample ratio of columns (features) for each tree.

— *learning_rate*: step size shrinkage.

According to the documentation, *max_depth* and *gamma* are able to reduce overfitting by controlling the model complexity, while *subsample* and *colsample_bytree* reduce overfitting by adding randomness in a similar way to the random forest algorithm.

We search for one hyperparameter at a time, and iterate through different hyperparameters. Iterations are repeated until the AUC-ROC score does not apparently increase. There are other hyperparameters interesting to adjust. Here, we restrict ourselves to the most commonly used ones.

[1] https://github.com/dmlc/xgboost

Experiment results and discussion

To evaluate quantitatively the classification step, we tested the performance of aforementioned classifiers on the BSDS500 dataset. FMS superpixels were used to provide calculation support for features. For each adjacent region pair (R_i, R_j), a feature vector of 66 features was extracted, including color features, geometry features, contours features and texture features discussed in section 4.2. The label of a region pair was computed according to the ground truth segmentations. As there are multiple ground truth segmentations for one image, we took the merging probability p (corresponding to the number of times that R_i and R_j belong to the same ground truth segment divided by the number of ground truth segmentations), and assigned to (R_i, R_j) $l = 1$ if $p > 0.5$, and otherwise $l = 0$. Before concatenating features of different images, we standardized each feature by subtracting its mean value and dividing it by its standard deviation.

The 100 validation images were used for hyperparameter tuning. At test stage, feature vectors were extracted for each pair of adjacent regions of the test images. The classifier took the feature vectors as input, and predicted the merging probability of the corresponding pair of adjacent regions. From that point, a straightforward (yet too basic) method to obtain an actual segmentation is to gradually merge the regions according to their merging probabilities.

Results

We trained both a random forest classifier and a XGBoost classifier when the number of superpixels is 500 and 800. The corresponding ROC curves are plotted in figure 4.6 to compare them quantitatively.

We can see from figure 4.6 that with the same number of superpixels, XGBoost achieves a higher AUC than a random forest classifier. For each kind of classifier, the one trained with 800 superpixels achieves a higher AUC than the one trained with 500 superpixels. One possible reason is that more training instances are available with more superpixels. When it comes to training time, the hyperparameter tuning of XGBoost takes much more time than random forest, since the trees in the forest can be constructed in parallel, while boosting trees are constructed sequentially since the construction of a tree depends on all its preceding trees.

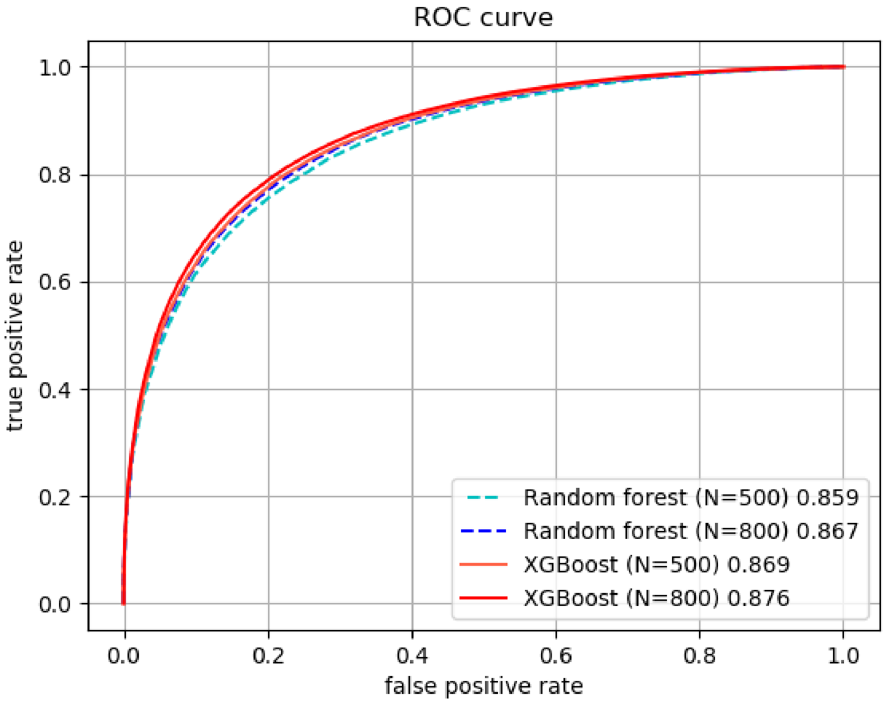


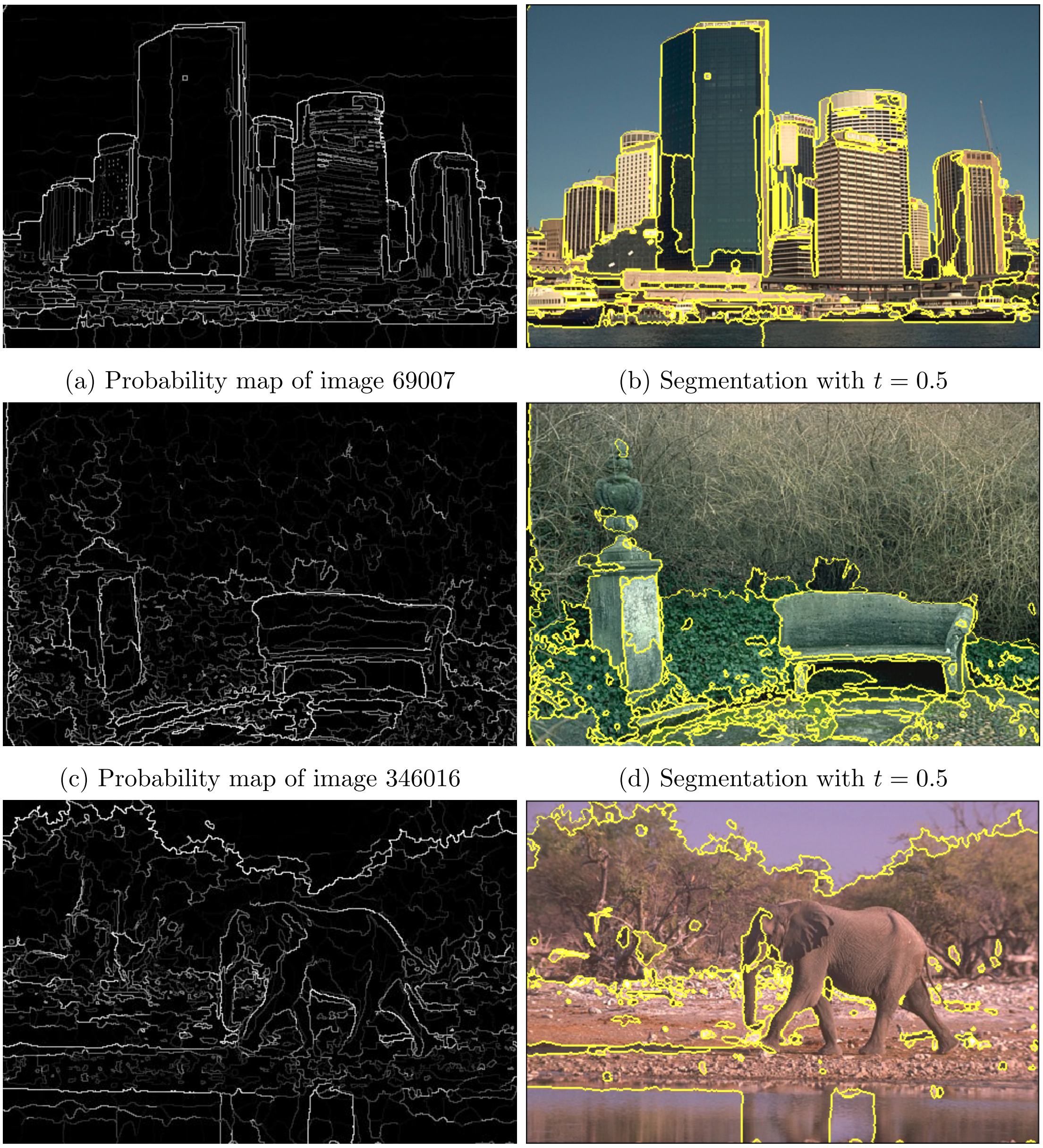
Figure 4.6 – ROC curves of random forest classifiers and XGBoost classifiers trained with 500 and 800 FMS superpixels.The last number at each line of the legend shows the AUC of the corresponding ROC curve.

RAG thresholding

We evaluate the effectiveness of the classifier on image segmentation through thresholding on the Region Adjacency graph (RAG). The RAG $\mathcal{G} = (V, E)$ of an image where V is the ensemble of vertices, and E is the ensemble of edges, is constructed by taking each region as a vertex v, and linking the vertices whose corresponding regions are adjacent in the image. Let us denote by p_{ij} the predicted merging probability of an adjacent region pair (R_i, R_j), then the probability of existence of the contour separating them is $1 - p_{ij}$, which is used as the weight w_{ij} of the edge e_{ij}. We set a threshold t, and merge the region pairs whose contour existence probability is lower than t, by cutting the corresponding edges in the RAG. An intuitive segmentation is produced in this way.

In figure 4.7, we draw a probability map for three test images, where the existence probability of contours obtained by a XGBoost classifier are plotted, as well as the segmentations obtained by RAG thresholding.

We observe that from figure 4.7 the thresholding segmentation works well with a high quality probability map as in the first image. But it fails in the third image since the probability of contours of the elephant is smaller than that of the small regions on the ground. The contours of the small regions in the second and third image have high



(a) Probability map of image 69007

(b) Segmentation with $t = 0.5$

(c) Probability map of image 346016

(d) Segmentation with $t = 0.5$

(e) Probability map of image 107045

(f) Segmentation with $t = 0.5$

Figure 4.7 – Examples of probability maps and segmentations obtained by RAG thresholding. Images on the left are probability maps showing the existence probability of contours. Images on the right are segmentations obtained by merging regions with contour probability lower than a threshold $t = 0.5$.

existence probabilities that we can not get rid of without losing the contours of objects of interest by using simple RAG thresholding.

In figure 4.8, we draw the precision recall curves of RAG thresholding segmentation under different thresholds on the training set and test set. We sort the edge weights in increasing order for each image, and set the threshold to be a certain percentage of the edge weights. The optimal threshold is the one that gives the highest F-score on training

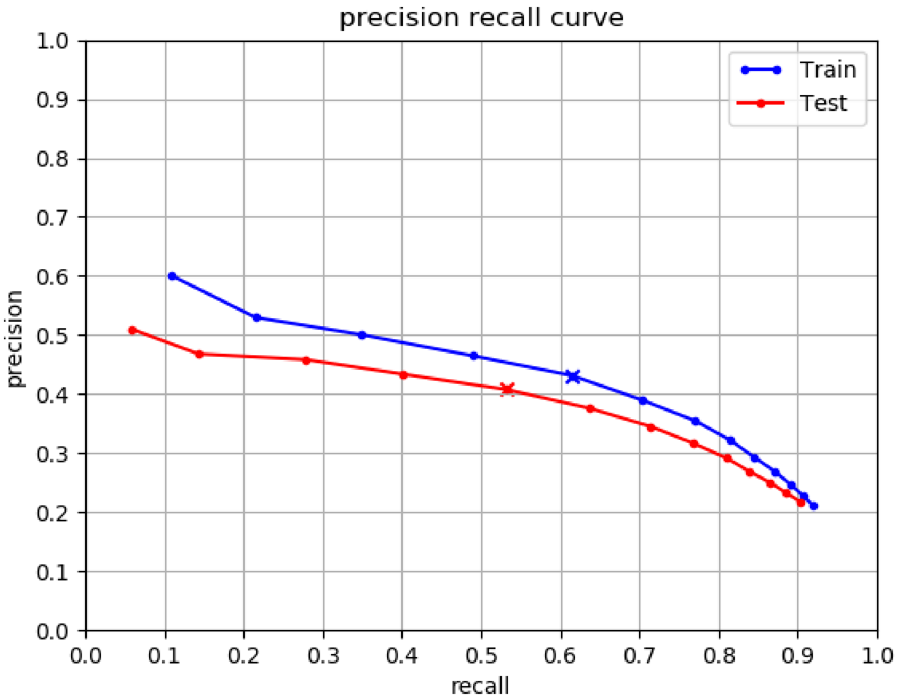


Figure 4.8 – Precision recall curves on training set and test set of RAG thresholding segmentation for XGBoost classifier with 800 superpixels. The point with highest F-score is marked by 'x' on the curves.

set. The highest F-score on the training set 0.507 is achieved when the percentage is 0.6, and the corresponding threshold is the weight separating 60% edges with weights lower than the threshold against the other 40%. The F-measure at this threshold for test set is 0.462. The region metrics, namely covering, Rand index and variation of information are 0.532, 0.677, 1.442 respectively. The definitions of these region metrics can be found in section 5.4.3.

The precision and recall of the RAG thresholding are not satisfactory. We would like to improve the precision while not decreasing too much the recall. Hence, in the next chapter we will elaborate on a graph partition approach that serves this goal.

Feature importance

The relative importance of each feature is plotted for the random forest classifier and the XGBoost classifier in figure 4.9 and figure 4.10 respectively. For a tree, at each split, the improvement of the split criterion is attributed to the feature chosen as the splitting feature. For each feature, these improvements are accumulated over a tree and averaged over all the trees as a measure of importance. For more details about the feature importance please refer to [HTF09].

We see that with the random forest classifier, the importance of features decreases drastically, while for the XGBoost classifier, the decrease in importance is more steady. For the random forest classifier, the color features under different color spaces are of the greatest importance. The contour features (mean gradient, minimum gradient), and the geometric features (convex ratio, bbox ratio, area and perimeter) come next. The color variance feature are less important compared to other color features. The SURF texture features of a,b channels and the texture feature of textons at the first scale are of little importance.

For XGBoost classifier, the mean gradient is the most important feature. The distance of SURF histograms comes the third. Color features are still important features, while other features, including texture features, contour features and geometric features have a greater importance when compared to the random forest classifier.

For both classifiers the eccentricity features and orientation features are of the least importance. That is possibly because in the original article they are involved for the segmentation of neural tissue in microscopy images, where geometric difference between different objects is more significant, which is not necessarily the case for natural images. For real time applications these features can be eliminated to accelerate feature computation.

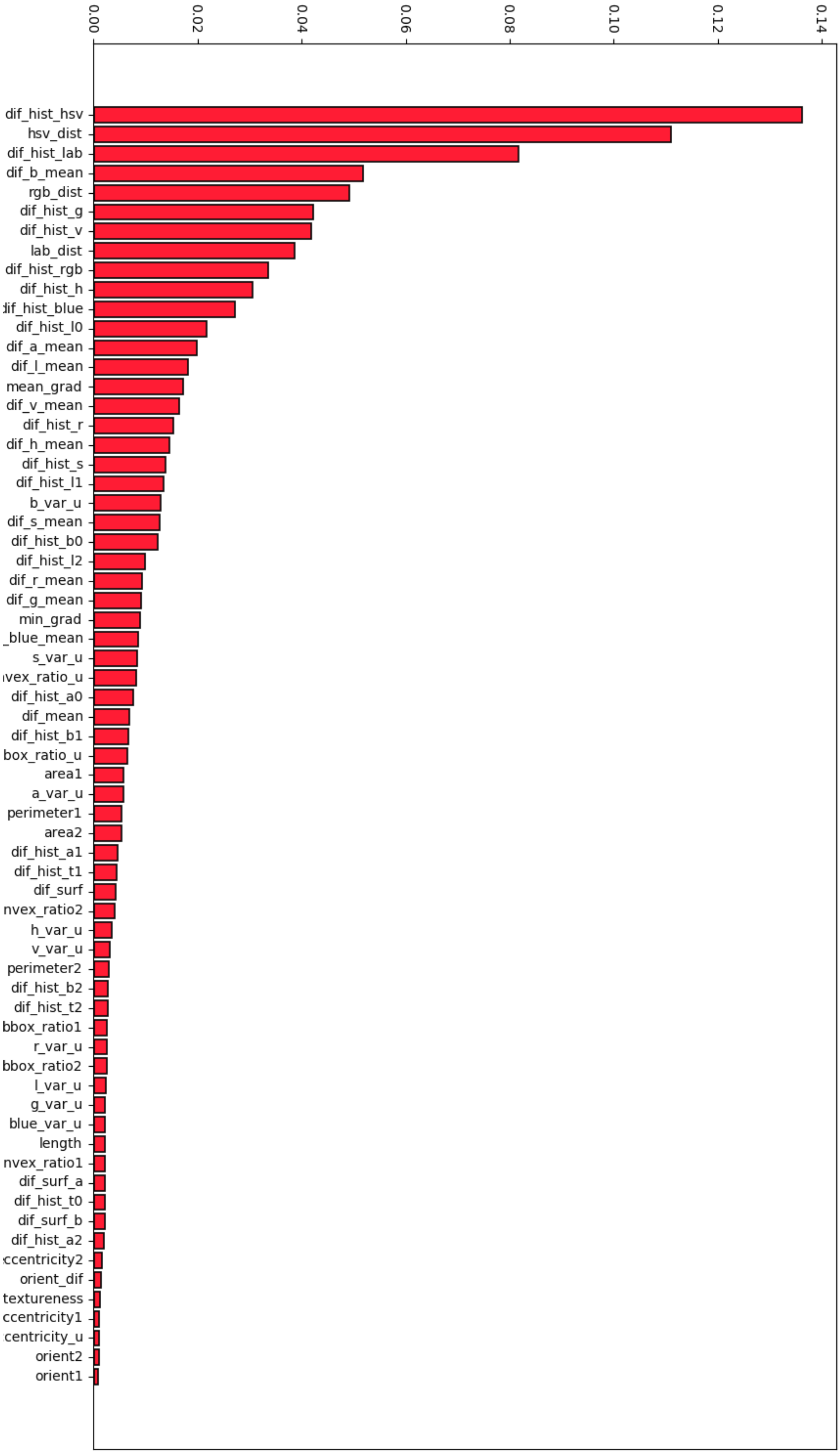


Figure 4.9 – Feature importances of RF classifier, $N = 800$

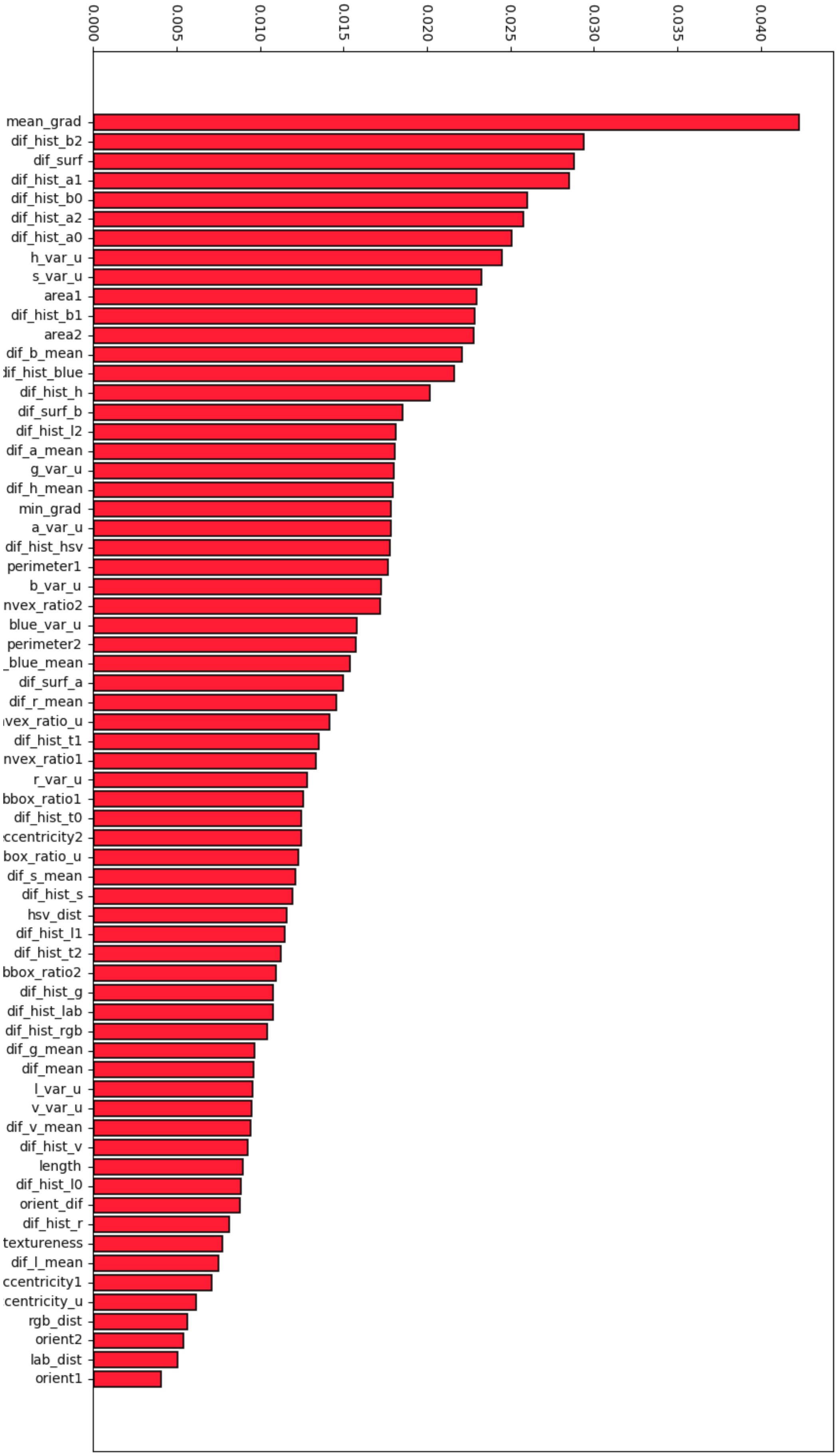


Figure 4.10 – Feature importances of XGBoost classifier, $N = 800$

Conclusion and perspectives

In this chapter, we introduced two machine learning algorithms, namely random forest and XGBoost, that were used to assess whether or not two adjacent superpixels should be merged. The classification task present several inherent difficulties:

- In the merging classification, the dataset is highly imbalanced. It is therefore difficult to correctly classify the negative class since the number of negative examples is lower. In addition, it is important to have a low false positive rate, since the negative instances correspond to the contours, and are therefore directly related to the recall metric. In [HG09], a summary of approaches to deal with imbalanced dataset is provided. In our study, we weighted the loss of each instance by a parameter inversely proportional to their class frequencies. Another possibility is to over sample the minority class, or under sample the majority class for training.
- Only local features are used in merging classification. Therefore, the lack of global information could explain the difficulty to retain the contours of objects with color similar to the background (for example the elephant in figure 4.8). In [Arb11] spectral clustering is applied to globalize the contour strength. High level features, such as the geometric context [HEH05], also help the segmentation [RS13; LST16]. Instead of using handcrafted features, a possible improvement of the proposed method could therefore be to use features extracted from convolutional neural networks, which achieves the state-of-the-art performance in many vision problems.

  When designing the contour features, for example the mean of gradient, we could also directly use contour strength by other algorithms where global information is involved, for example the gPb gradient [Arb11], as has been done in [RS13; LST16]. The output of a deep learning based contour detection algorithm, for example the holistically nested edge detection [XT15], providing highly reliable contour probabilities, can also be used for contour feature calculation, as proposed in [Che16].

  To deal with the globalization problem, we could make use of the information of neighboring regions, for example by considering a Markov random field on the region adjacency graph.
- Finally, it is worth noticing that during the learning process, the classification of each instance is made independently. The measure of performance of a classifier is not directly related to the quality of segmentation. In [Jai11] it is proposed to use reinforcement learning for image segmentation, with a reward function directly related to a segmentation metric, such as the Rand index.

Chapter 5

Region merging

Region adjacency graph representation is a key ingredient of several segmentation algorithms including the normalized cut algorithm [SM00] or the waterfall algorithm [Beu94]. In this chapter, we present a novel clustering method for partionning the adjacency graph based upon the fast marching algorithm and the dissimilarity measure introduced in chapter 4.

The outline of this chapter is as follows. First, we review the literature related to graph partition for image segmentation. Next, we recall the main properties of the fast marching algorithm and we describe how this algorithm can be adapted to work on graph structures. Then, we describe the use of the fast marching algorithm to partition the region adjacency graph associated to an image. The graph partitioning yields a coarsened oversegmentation of the image from its superpixels by efficiently clustering adjacent superpixels. The fast marching based algorithm is finally compared with similar algorithms including the normalized cut and SWA.

Literature review

We can represent an image by a graph by taking its pixels or a group of pixels, for example superpixels or regions generated by an oversegmentation algorithm, as nodes, linking adjacent pixels or regions by an edge, and choosing a measure of similarity or dissimilarity as edge weight. An advantage of graph based segmentation is that it takes into account spatial information since adjacent regions or pixels are connected.

Graph based approaches for image segmentation

A significant amount of research has been conducted on graph partition for image segmentation. A survey of the literature can be found in [PZZ13].

Minimum spanning tree based algorithms

The idea of relying on graph representations for segmenting images dates back to the early years of image processing. A first family of methods rely upon the minimum spanning tree (MST) of the region adjacency graph. The minimum spanning tree of a graph, sometimes called shortest spanning tree (SST), is a tree that spans the nodes of the graph with the smallest weights. The MST of a connected, undirected graph is a subset of its edges connecting all the vertices together, without any cycles and with the minimum possible total edge weight. Efficient algorithms are available to compute the MST of a graph, including Kruskal's algorithm [Kru56].

In 1971, Zahn presented a graph clustering method based on the minimum spanning tree [Zah71]. The minimum spanning tree of a graph is computed by Kruskal's, Prim's or Dijkstra's algorithm. The clustering is formed by iteratively breaking the inconsistent edges in MST. An inconsistent edge is an edge whose weight is significantly larger than the average weight of nearby edges. The significance can for example be measured by the ratio between the edge weight and the average weight. Urquhardt [Urq82] proposed to normalize the weight of each edge by the smallest weight among the set of all edges incident on the vertices touching that edge. Despite bringing some improvement, the normalization still fails to deal with intra-region variability issues.

Later, Morris *et al.* proposed a hierarchical image segmentation algorithm based on graph representations of images and on the shortest spanning tree [MLC86]. In their work, an image is mapped to a graph by representing each pixel by a node, and by linking the nodes to their 4-connected or 8-connected nearest neighboring pixels by edges. The weight of each edge is set to be the absolute value of the difference in intensity between the pair of pixels that it links. Then the SST of the graph is constructed. A hierarchical segmentation is obtained by iteratively cutting the SST edge with the highest weight, which corresponds to the maximum contrast observed in the image.

This approach presents strong limitations: in high variability regions, textured regions for instance, differences in intensities between adjacent pixels belonging to the same region can significantly exceed those observed between adjacent pixels belonging to distinct segments. Hence, a major issue with MST method is that they tend to split the region with the most variability into multiple segments and to conversely merge regions that should correspond to distinct segments. Another problem is that two very different regions may remain connected if they have similar neighboring pixels. According to Morris, these problems result from the lack of global information. Three possible variants are provided to deal with these problems. One solution is to update after each split the pixels intensity to the average intensity of pixels within the same region, an approach called recursive SST in this article.

In the efficient graph based image segmentation algorithm [FH04], the maximum weight of the MST of a region is employed to measure the internal difference of a region. Two regions are merged if the minimum of inter-region differences is larger than the weight of the edges linking them plus a threshold. More details on this algorithm can be found in section 3.2.

Graph cut algorithms

Several graph based algorithms have been proposed based upon the notion of *graph cut*. The *cut* between two subgraphs A and B is defined as

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{5.1}$$

where u, v are nodes belonging to A and B respectively, and w is a measure of similarity between nodes. Graph cut based segmentation algorithms produce segmentation by optimizing a cost function based on the notion of cut, which maximizes the dissimilarity between different clusters of nodes, or maximizes the similarity of nodes within the same cluster, or both. From this perspective, image segmentation is recast as a spectral clustering problem and can be handled with efficient clustering algorithms [Lux07].

In 1993, Wu and Leahy [WL93] proposed a data clustering approach based on graph theory, and demonstrated its application to image segmentation. In this approach, the segmentation is performed by minimizing the cost function equation (5.1), and is therefore called minimal cut. Overall, Wu and Leahy's method yields interesting segmentation results for some images. However, the minimum cut criterion tends to favor cutting small subsets of isolated nodes in the graph. It is indeed straightforward to see from equation (5.1) that the cut increases with the number of edges linking the subsets of the partition.

A very popular approach for defining a cut criterion is the normalized cut formulation proposed by Shi and Malik [SM97; SM00]. It is a criterion that comprises both the association within a cluster and the disassociation between clusters. The exact optimization of this criterion is NP-complete. However, it is possible to obtain an approximate solution by relaxing some assumptions of the problem. The segmentation is obtained by solving a generalized eigenvalue problem to minimize the normalized cut. The eigenvector with the second smallest eigenvalue bipartitions the graph into two parts. The partition can be done recursively to further partition the graph. The normalized cut algorithm overcomes the disadvantage of minimal cut of favoring small regions. However, it tends to split large homogeneous regions.

In the original article introducing the method, an exponential kernel involving the negative brightness distance is employed for the similarity function. In a subsequent article, Malik *et al.* proposed a similarity measure accounting for texture in the normalized cut framework [Mal01]. In their approach, the texture of each image is described using *textons*. The textons are obtained by clustering the outputs of filter responses using the k-means algorithm. For images without texture, the dissimilarity is computed using the intervening contour method introduced in Leung and Malik [LM98].

The normalized cut algorithm is also used to introduce global information in contour based segmentation. The Gaussian directional derivatives of eigenvectors for normalized cut minimization are employed as the spectral component for the boundary detector in the gPb algorithm developed by Arbelaez *et al.* [Arb11], which constitutes one of the state-of-the-art algorithms for performing image segmentation. We notice that the graph is constructed differently by connecting pixels within a circle with radius r.

Inspired by algebraic multigrid (AMG) solvers of minimization problems of heat or electric networks, Sharon *et al.* [SBB00] introduced in 2000 a fast multiscale segmentation algorithm called segmentation by weighted aggregation (SWA), providing an approximate solution for normalized cut minimization with a time complexity linear in the number of pixels. The graph of adjacent pixels is coarsened recursively by a weighted aggregation procedure, where a smaller set of representative pixels or blocks are selected at each scale. The couplings between blocks can be directly derived from finer levels, or can be related to the internal statistics of the block. At last, a top down process provides the segmentation. This algorithm is completed in [SBB01; Gal03]. Details of this algorithm can be found in section 5.4.2.

Alpert *et al.* [Alp12] presented in 2012 a probabilistic bottom-up aggregation approach for image segmentation, employing the coarsening strategy of SWA for region mering, of which the performance is comparable to the state of the art. The probability of adjacent regions belonging to the same segment is obtained by a “mixture of experts”-like model integrating intensity and texture cues weighted by a term based on the sparseness of the gradient histogram. The conditioning probability of merging knowing a certain cue is obtained by Bayes formula. Pixels are merged gradually to larger regions according to the merging probability. A notable difference from the original SWA algorithm is that the edge weight is recalculated at each coarser level according to the probabilistic model in this approach.

Finally, graph-based approaches have also been demonstrated to work well in the context of fuzzy segmentation [MLN09].

Multi-stages algorithms

In a bottom-up merging segmentation algorithm, the region size grows larger as the merge goes on. For a similarity measure based on a classifier, the segmentation performance is therefore expected to decrease since the classifier is only trained with small regions of the original oversegmentaiton, and is therefore not appropriate for classifying larger regions. A large region may for instance be less convex. Similarly, the texture features may be not important for small regions, while more important for large regions. Therefore, there is a need to take into consideration the influence of region size on the probability prediction. Multi-stages algorithms come up where the classifier is retained in multiple stages to serve this purpose.

Image segmentation by cascaded region agglomeration (ISCRA) algorithm [RS13] is an algorithm based on region merging by cascaded classifiers that achieves state-of-the-art performances. A classifier is firstly trained on the superpixels, then the regions are merged according to the probability predicted by this classifier. Then, the merge stops and a new classifier is trained for larger regions. The number of remaining regions can be used as the stopping criterion. In this case, the training and merging steps repeat until the desired number of segments is reached. 60 stages are considered in the original version of ISCRA algorithm.

A machine learning based agglomerative segmentation method is presented in [NI13] where training examples are collected at multiple stages. A flat learning stage is first conducted to obtain a classifier to predict the merging probability of adjacent region pairs. This probability can be used as a merge priority function (MPF) that indicates the merging order at the next stage. At the next stage, region pairs are processed in descending order of the MPF. Region pairs are labeled compared with the ground truth, and merged if they belong to the same ground truth segment. The weights of the edges incident on the new region are recalculated after each merge to ensure that the influence of the change in region size is taken into account. Training examples from all previous stages are concatenated for training in the current stage, and the newly obtained classifier is employed to decide the merging order of the next stage. The performance of the classifiers converges after several stages. This method surpasses the gPb by Arbeláez *et al.* [Arb11] on BSDS500 in terms of region metrics.

Eikonal equation on a graph

A graph is a mathematical structure used to describe a set of objects that are pairwise related. The objects are represented by *vertices* (also called *nodes* or *points*) and relationships between pair of vertices are represented by *edges* (also called *links*).

Shortest path in a graph

Definition 5.2.1. Let k be some integer in $\mathbb{N}$. A path p of length k in a graph $\mathcal{G}$ is a sequence of vertices $\{v_0, v_1, ..., v_k\}$ such that, for all $i = 0, ..., k-1$, there exists an edge $e := (v_i, v_{i+1})$ linking v_i and v_{i+1}. The weight $w(p)$ of the path $p := \{v_0, v_1, ..., v_k\}$ is the sum of the weights of all edges along the path:

$$w(p) := \sum_{i=0}^{k-1} w(v_i, v_{i+1}). \tag{5.2}$$

Let u and v be two arbitrary vertices in $\mathcal{G}$. We denote by $\mathcal{P}(u,v)$ the set of all paths in $\mathcal{G}$ between u and v. Then, the shortest path weight $\hat{w}(u,v)$ between u and v is

$$\hat{w}(u,v) = \begin{cases} \min_{p \in \mathcal{P}(u,v)} w(p) & \text{if } \mathcal{P}(u,v) \neq \emptyset \\ \\ +\infty & \text{otherwise.} \end{cases} \tag{5.3}$$

Accordingly, the shortest path $\hat{p}$ between u and v is the path with minimal weight between u and v:

$$\hat{p}(u,v) = \begin{cases} \arg\min_{p \in \mathcal{P}(u,v)} w(p) & \text{if } \mathcal{P}(u,v) \neq \emptyset \\ \\ \emptyset & \text{otherwise.} \end{cases} \tag{5.4}$$

Note that obviously, since $\mathcal{G}$ is undirected, if $p := \{v_1, v_2, .., v_k\}$ is the shortest path between v_1 and v_k, then $q := \{v_k, .., v_2, v_1\}$ is the shortest path between v_k and v_1.

Eikonal equation

The fast marching algorithm was originally introduced to solve the Eikonal equation on a continuous domain [MSV95; Set96]. The Eikonal equation is a non-linear partial differential equation which describes the propagation of waves in a medium. It finds notable applications in fields including geometrical optics or geophysics. Classical results relative to the Eikonal equation on continuous domains are presented in section 2.3.1. In this section, we study the generalization of the Eikonal equation to undirected graph structures.

Eikonal equation on an undirected graph

In this section, our aim is to propose a way to extend the continuous Eikonal equation to the setting of graphs. To that end, let us consider an undirected graph $\mathcal{G} := (V, E)$. Unless otherwise stated, the graphs considered in this section will always be path-connected. We assume that each edge (i, j) in E is associated a weight w_{ij}, and that $t : V \to \mathbb{R}$ is a function defined on the set V of all vertices of the graph.

Definition 5.2.2. If v is a vertex in V, we denote by $\mathcal{N}_v$ the set of all neighbor vertices. For all vertices u in $\mathcal{N}_v$, we define the derivative of t at the vertex v with respect to u to be:

$$Dt(u, v) := w_{uv}(t_u - t_v). \tag{5.5}$$

Using this definition, it becomes possible to define an equivalent formulation of the Eikonal equation adapted to graph structures. In the continuous setting, the Eikonal equation relates the L^2 norm of the local gradient to the local velocity u in the open domain Ω

$$u(x)||\nabla t(x)|| = 1, \forall x \in \Omega. \tag{5.6}$$

The equivalent formulation for a graph structure should therefore read

$$u(v)||\nabla t(v)|| = 1, \forall v \in V, \tag{5.7}$$

where $u(v)$ denotes a local velocity associated to node v. From now on, we will assume the velocity term to be constant:

$$\forall v \in V, u(v) = 1. \tag{5.8}$$

The difficulty is therefore to replace the gradient used in the continuous setting by a gradient adapted to the graph setting. To that end, we propose the following construction. Let v be some node in the graph and $\mathcal{N}_v$ the set of its neighbor vertices. We consider a front that propagates from seed vertices over the entire graph. Let us assume that the propagation front reaches at least one element in $\mathcal{N}_v$ before reaching v. This is necessarily the case when v is not a germ from which the front propagates. Then, we specify the following requirements for the arrival time at vertex v:

1. The arrival time $t(v)$ must satisfy the following set of inequalities:

   $$t(v) \leq t(u) + \frac{1}{w_{uv}}, \tag{5.9}$$

   for all vertices u in $\mathcal{N}_v$ already reached by the propagation front.

2. There must exist one particular vertex $\hat{u} \in \mathcal{N}_v$ such that

   $$t(v) = t(\hat{u}) + \frac{1}{w_{\hat{u}v}}. \tag{5.10}$$

Putting these requirements together, we obtain the single equation

$$\|\nabla t(v)\| := \max_{u \in \mathcal{N}_v} w_{uv}(t(v) - t(u))^+, \tag{5.11}$$

where $(t(v) - t(u))^+ := \max(t(v) - t(u), 0)$.

Two remarks can be formulated regarding equation (5.11). First, we can notice the key role played by the weights $\{w_{uv}, u \in \mathcal{N}_v\}$. In the continuous setting, due to the regular discretization grid, the spatial step used for approximating the gradient is constant. In the graph setting, the weights of the edges can be interpreted as similarity measures. Hence, the quantities $\{w_{ij}^{-1}, (i,j) \in E\}$ can naturally be interpreted as *distances* between the node. When two adjacent vertices are highly similar, the weight of the corresponding edge is large, which yields in turn a small distance between these nodes. Second, it is interesting to notice that the heuristic approach used to obtain expression (5.11) leads to a formulation of the Eikonal equation on the graph using the L^∞ norm. In the continuous setting, the Eikonal equation was defined on $\mathbb{R}^2$, which naturally carries an Euclidean distance. For the similarity graph setting, more intuitive distances can be obtained using the heuristic approach presented above.

Finally, boundary conditions are also required to properly define the Eikonal equation on the graph. To define these boundary conditions, we select a subset $\{v_1, ..., v_k\}$ of k vertices in V and we require that $\forall i = 1, ..., k$, $t(v_i) = 0$.

In the previous section, we defined the shortest path distance between two vertices u and v in $\mathcal{G}$ to be:

$$\hat{w}(u,v) = \begin{cases} \min_{p \in \mathcal{P}(u,v)} w(p) & \text{if } \mathcal{P}(u,v) \neq \emptyset \\ \\ +\infty & \text{otherwise.} \end{cases} \tag{5.12}$$

Building upon this definition, we can define the shortest path between any vertex v of the graph and the subset $\partial\mathcal{G} := \{v_1, ..., v_k\}$ to be

$$d(v, \partial\mathcal{G}) = \min_{i=1,\dots,k} \hat{w}(v, v_i). \tag{5.13}$$

Proposition 5.2.1 is the analog of Prop. 2.3.2 for undirected graphs.

Proposition 5.2.1. *Let $\mathcal{G}$ be an undirected, weighted graph. Then, the shortest path $d(\cdot, \partial\mathcal{G})$ is the solution of Eikonal equation*

$$||\nabla d(v, \partial\mathcal{G})|| = 1$$

with boundary conditions $d(v_i) = 0, \forall i = 1, ..., k$.

According to proposition 5.2.1, it is possible to partition a graph $\mathcal{G}$ into $K \geq 1$ subgraphs by relying on the Eikonal equation. To that end, we start by selecting K vertices $\{v_1, ..., v_k\}$ of $\mathcal{G}$. Then, solving the Eikonal equation on $\mathcal{G}$ with boundary conditions set to be $t(v_i) = 0, \forall i = 1, ..., k$ allows to compute the distance $w(v, \partial\mathcal{G})$ of the shortest path linking each vertex $v \in V$ to the closest vertex in the subset $\partial\mathcal{G} := \{v_1, ..., v_k\}$. Since the graph $\mathcal{G}$ is path-connected, for $i = 1, ..., k$, the subsets

$$\mathcal{C}_i = \{v \in V, w(v, v_i) = w(v, \partial\mathcal{G})\} \tag{5.14}$$

constitute a partition of $\mathcal{G}$ into K connected subgraphs. We will subsequently use this approach to compute a partition of the region adjacency graph associated to a superpixel segmentation and to coarsen the segmentation.

Fast marching algorithm

Efficient algorithms have been proposed in the literature to solve Eikonal equation, including the fast sweeping method [Zha04] or the fast marching algorithm [Set96]. The discussion of the fast marching algorithm in the continuous setting can be found in section 2.3.2. In this section, we discuss the generalization of the fast marching algorithm to undirected graph structures.

Fast marching algorithm on a graph

In this section, we describe the generalization of the fast marching algorithm to undirected graph structures. Let us consider an undirected, path-connected graph $\mathcal{G} := (V, E)$. We assume that each edge (i, j) in E is associated a weight w_{ij} and that $t : V \to \mathbb{R}$ is a function defined on the set V of all vertices of the graph.

For an undirected graph $\mathcal{G}$, an equivalent formulation of equation (2.23) can be obtained by setting

$$\max_{u \in \mathcal{N}_v} w_{uv}(t(v) - t(u))^+ = 1. \tag{5.15}$$

Boundary conditions are imposed by selecting a subset $\partial\mathcal{G} := \{v_1, ..., v_k\}$ of k vertices in V and setting $t(v_i) = 0, \forall i = 1, ..., k$.

As for continuous domains, instead of iteratively solving for each vertex of $\mathcal{G}$ until convergence, we follow the front propagation in the graph to compute the arrival times. To that end, we partition the vertices of $\mathcal{G}$ into three distinct subsets:

- Vertices that have already been reached by the front are grouped into a set referred to as the *frozen* set.
- Vertices that are adjacent to frozen points but have not been reached by the front yet are grouped in a subset referred to as the *narrow band.*
- The remaining vertices are grouped in a subset referred to as the *far away* set.

Initialization The fast marching algorithm is initialized as follows:

1. An arrival time map is initialized: each vertex v in the graph $\mathcal{G}$ is associated the arrival time $t = +\infty$, except if it belongs to the graph boundary $\partial\mathcal{G}$. The vertices $\{v_i, i = 1, ..., k\}$ are associated the arrival times $t(v_i) = 0$.
2. All vertices belonging to $\partial\mathcal{G}$ are added to the narrow band. Other vertices are labeled as *far away.* Initially, the frozen set is empty.
3. To keep track of the shortest paths between each vertex in $\mathcal{G}$ and the boundary $\partial\mathcal{G}$, we finally assign the label i to each vertex in $\{v_i, i = 1, ..., k\}$. All other vertices are labeled 0.

Iteration At each iteration, the vertex v of the narrow band with the smallest arrival time is extracted and labeled as *frozen*. Here, we introduce a slight modification to the classical fast marching approach to keep track of the shortest path during the front propagation. Let us assume that the shortest path between v and $\partial\mathcal{G}$ is the path $p := (v_i, v)$ for some $i \in \{1, ..., k\}$. Then, v necessarily belongs to the region

$$\mathcal{C}_i = \{u \in V, w(u, v_i) = w(u, \partial\mathcal{G})\}. \tag{5.16}$$

At this point, we compute the arrival times for each neighbor of v, by considering that the arrival time at a neighbor node u is

$$\tilde{t}(u) := \begin{cases} t(u) & \text{if } u \in \mathcal{C}_i \\ \infty & \text{otherwise.} \end{cases} \tag{5.17}$$

Arrival times are then computed by solving equation (5.15). Again, frozen vertices are used to compute the arrival times in adjacent vertices, but their arrival time is never recomputed. Once the arrival time t of a neighbor point u has been computed, two situations can be encountered:

- When (i, j) is in the narrow band, it has already been associated an arrival time t^0_{ij} and it is assigned to one of the regions $\mathcal{C}_j, j = 1, ..., k$. If the new arrival time t_{ij} is less than t^0_{ij}, then the arrival time is updated and the vertex u is assigned to the region $\mathcal{C}_i$. Otherwise, the vertex remains unchanged.
- When the neighbor vertex u is far away, we add it to the narrow band with the arrival time t_{ij} and we assign u to the region $\mathcal{C}_i$.

Stopping condition The fast marching algorithm stops when the narrow band is empty.

The stability of the fast marching algorithm is proven in the appendix A.3.

Application to superpixels merging

Let I be a color image. We denote by $I(p, q)$ the color of the pixel located at position (p, q) in the image. For 8-bit RGB images, $I(p, q)$ is a 3-dimensional vector taking its values in the set $\{0, 1, ..., 255\}$. We assume that I is represented as the union of disjoint superpixels $\{\mathcal{S}_i, i = 1, ..., N\}$:

$$I = \cup_{1 \leq i \leq N} \mathcal{S}_i, \tag{5.18}$$

with $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ if $i \neq j$.

It is convenient to associate to the superpixel partition of I a graph referred to as its region adjacency graph $\mathcal{G}$. The region adjacency graph is a representation of the image

I as an undirected graph, whose vertices $v_i, i = 1, ..., N$ are associated to the superpixels $\mathcal{S}_i, i = 1, ..., N$. Two vertices v_i and v_j are linked by an edge of the graph if and only if the corresponding superpixels $\mathcal{S}_i$ and $\mathcal{S}_j$ share a common boundary in the image. In the following, we will adopt the classical notation $\mathcal{G} := (V, E)$ when referring to the region adjacency graph, where V is the set of all vertices in $\mathcal{G}$ and E the set of all edges.

We can specify a weight for each edge in E by defining a function $w : E \to [0, 1]$ which associates to the edge e_{ij} joining vertices v_i and v_j a quantity $w_{ij} \in [0, 1]$. In what follows, we will assume that the weight w_{ij} can be interpreted as a dissimilarity measure between vertices v_i and v_j. Since each vertex in $\mathcal{G}$ corresponds to regions in the original image I, the dissimilarity measure characterizes in fact the dissimilarity between adjacent superpixels in the image. Classical choices of dissimilarity measures include the difference between the average colors of adjacent superpixels, or the average or minimal amplitude of the gradient on the boundary between the adjacent superpixels. In this section, we will assume that, for each pair $(\mathcal{S}_i, \mathcal{S}_j)$ of adjacent superpixels, we evaluated the probability p_{ij} that $\mathcal{S}_i$ and $\mathcal{S}_j$ belong to the same segment of the image. Obviously, the higher the merging probability, the higher is the similarity betweens superpixels $\mathcal{S}_i$ and $\mathcal{S}_j$. Hence, by considering the quantity

$$w_{ij} = \exp(-p_{ij}), \tag{5.19}$$

we define a dissimilarity measure between vertices v_i and v_j of $\mathcal{G}$. When not stated otherwise, we will assume that the weights of the region adjacency graph are obtained according to equation (5.19).

Algorithm

The merging algorithm proposed in this section works by iteratively solving the Eikonal equation on the region adjacency graph of the superpixel segmentation $\mathcal{S}$ of I and adapting the boundary conditions. Let us assume that we start from an initial segmentation $\mathcal{S}^0$ containing N superpixels. Typical values for N are in the order of 500 or 800 superpixels. Our aim is to significantly reduce the number of segments in the image to a value around $50 - 100$. The superpixel merging is conducted as follows:

1. To initialize the algorithm, K vertices $\{v_1, v_2, ..., v_K\}$ are chosen randomly in the region adjacency graph.

2. The Eikonal equation is solved for the region adjacency graph with boundary conditions $t(v_i) = 0, \forall i = 1, ..., K$. After this step, the graph is clustered into K separated subgraphs $\{\mathcal{G}_i, i = 1, ..., K\}$. For $i = 1, ..., K$, we denote by V_i and E_i the set of the vertices and of the edges of $\mathcal{G}_i$, respectively.

3. For each subgraph $\mathcal{G}_i, i \in \{1, ..., K\}$, we search for the edge e_i in E_i with maximal weight w_i. We denote by $n_{0,i}$ and $n_{1,i}$ the vertices in V_i linked by e_i. Then, we select the subgraph $\mathcal{G}_j$ whose maximal internal weight is the highest and we add the nodes

$n_{0,j}$ and $n_{1,j}$ to the boundary conditions.

4. We solve the Eikonal equation for the region adjacency graph with the updated boundary conditions $t(v_i) = 0, \forall i = 1, ..., K+2$, where $v_{k+1} = n_{0,j}$ and $v_{k+2} = n_{1,j}$.
5. We iterate between steps 3 and 4 until some stopping criterion is met.

Stopping criterion Two distinct stopping criteria can be used in the algorithm. A first stopping criterion consists of stopping the algorithm iterations when a specified number of segments are obtained. The advantage of this approach is that it enables to control the number of segments obtained in the final segmentation. However, this approach can yield a segmentation with segments still containing highly dissimilar superpixels.

A second stopping criterion consists of specifying a probability threshold t and of iterating between steps 3 and 4 until no subgraph contains weights higher than this threshold. The threshold can be fixed in an adaptative manner based upon the set of weights as observed on the adjacency graph. Here, we propose the following procedure to select the threshold value:

1. Sort the edges increasingly according to their weight.
2. Arbitrarily select a proportion of edges considered to be actual contours, and set the corresponding weight as threshold.

This method for selecting the threshold has also the advantage that it partly allows to control the number of segments in the final segmentation.

It is interesting here to discuss the refinement procedure proposed in step 4. The refinement is conducted by progressively adding new seeds and re-propagating from the new set of seeds on the whole graph. Hence, the Eikonal equation is solved a significant number of times. One possibility to improve the efficiency of the algorithm is to add more seeds at each iteration to reduce the number of iterations. The main issue with this approach is that when multiple nodes are selected at each step of the algorithm, these nodes often tend to belong to adjacent regions in the image. Hence, after re-propagation, the corresponding regions can be over-segmented. To prevent this, it is therefore preferable to add the new germs one by one. Since the complexity of the fast marching is in $O(N \log(N))$, N being the number of vertices in the graph, the computation time is not critical in our problem. Hence, we considered that the best strategy was to progressively update the set of seeds.

Finally, it is interesting to note that despite the refinement step, the segmentation results remain dependent on the initial choice of germs. Since the germs are initially randomly selected, it can lead to regions of the image that are artificially over-segmented. To reduce over-segmentation and to diminish the influence of the initial seeds selection, a simple post-processing step is conducted at the end of the algorithm.

The post-processing is conducted by first extracting, for each pair of adjacent regions, the maximal merging probability observed at the boundaries separating these regions. This yields a dissimilarity measure between adjacent regions. The pairs of regions are then processed by increasing order of dissimilarity. When the dissimilarity between a pair of adjacent regions is below the threshold specified for the refinement step, these regions are merged. During the procedure, a region can only be merged once, to avoid creating regions of the adjacency graph containing edges with a weight greater than the threshold. The post-processing allows to significantly reduce the number of clustered regions while only slightly decreasing the boundary recall.

Comparison with normalized cut and SWA

In this section, we conduct a quantitative comparison of the Eikonal based merging algorithm with two similar existing algorithms in the literature, namely the normalized cut and the segmentation by weighted aggregation (SWA) algorithm. First, we recall the normalized cut and SWA algorithms.

Normalized cut

In this section, we assume that the image to be segmented is represented using a region adjacency graph $\mathcal{G}$.

Definition

It is straightforward to note that a graph $\mathcal{G} = (V, E)$ can be partioned into two distinct subsets A and B such that $A \cup B = \mathcal{G}$ and $A \cap B = \emptyset$ by removing all edges connecting the two subsets. Let us assume that each edge linking adjacent regions is weighted according to the dissimilarity between these regions. In this case, it is easy to define a global measure of dissimilarity between the subsets A and B by considering the so-called *cut* cost

$$cut(A, B) = \sum_{i \in A, j \in B} w_{ij}, \tag{5.20}$$

where w_{ij} denotes the weight of the edge linking region $i \in A$ and $j \in B$. Within this framework, one can define the optimal bipartioning of a graph to be the one that minimizes the *cut* cost. A considerable amount of work has been conducted to propose a number of cut formulations.

Optimization

First, let us introduce a few notations. For a given subset A of the graph, let us define the total connection from the nodes in A to all nodes in the graph to be

$$assoc(A, V) = \sum_{i \in A, j \in V} w_{ij}. \tag{5.21}$$

More generally, the total connection from the nodes in a subset A of the graph to the nodes of a subset B is defined to be

$$assoc(A, B) = \sum_{i \in A, j \in B} w_{ij}. \tag{5.22}$$

The idea of Shi and Malik is to use as disassociation measure the *normalized cut* (*Ncut*), defined by

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}. \tag{5.23}$$

Let us consider a weighted undirected graph $\mathcal{G} = (V, E)$. Given some partition of the vertices into two subsets A and B, we can introduce the $|V|$-dimensional vector x such that

$$x_i = \begin{cases} 1 & \text{if} \quad i \in A, \\ -1 & \text{if} \quad i \in B. \end{cases} \tag{5.24}$$

We define the total connection from node i to the other nodes by

$$d(i) = \sum_j w_{ij}. \tag{5.25}$$

Finally, we introduce the $N \times N$ symmetrical matrix $\mathbf{W}$ defined by $W(i, j) = w_{ij}$, and the $N \times N$ diagonal matrix $\mathbf{D}$ defined by

$$D(i, j) = \begin{cases} d_i & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.26}$$

and we consider the quantities

$$k = \frac{\sum_{x_i > 0} d_i}{\sum_i d_i} \tag{5.27}$$

and $b = \frac{k}{1-k}$. Then, in this case, it can be shown (see [SM00]) that minimizing the normalized cut is equivalent to solving the optimization problem

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D} - \mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \text{ subject to } \mathbf{y}_i \in \{1, -b\} \text{ and } \mathbf{y}^T\mathbf{D}\mathbf{1} = 0. \tag{5.28}$$

In equation (5.28), if we relax $\mathbf{y}$ to take real values instead of the two discrete values $\{1, -b\}$, the optimization problem equation (5.28) is equivalent to solving the generalized eigenvalue system

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}. \tag{5.29}$$

More precisely, the second smallest eigenvector of the generalized eigensystem equation (5.29) is the real valued solution to the normalized cut problem.

Due to the relaxation, the eigenvectors computed using this approach can take continuous values. Hence, we need to define a way to partition them in two parts. Several methods have been proposed to partition the eigenvectors. In the original article of Shi and Malik, the authors suggest to select l evenly spaced splitting points in the range of values spanned by the eigenvector coefficients, to compute the l corresponding values of the $Ncut$ and to select the one yielding the minimal $Ncut$. An alternative to this method is to rely on the k-means algorithm to partition the eigenvector coefficients.

Algorithm

In summary, when applied to image segmentation, the normalized cut algorithm proceeds as follows:

1. Given an image, compute a weighted graph $\mathcal{G} = (V, E)$ where the nodes V correspond to the pixels and the edges E link all pairs of adjacent pixels. Set the weight of each edge to be some measure of similarity between the nodes.
2. Compute the matrices $\mathbf{D}$ and $\mathbf{W}$ as defined above, and solve the generalized eigensystem (5.29) for eigenvectors with the smallest eigenvalues.
3. Use the eigenvector with the second smallest eigenvalue to bipartition the graph.
4. Iterate until the number of segments is as required.

The normalized cut framework provides a lot of freedom for setting a dissimilarity measure between adjacent regions. In the original article [SM00], Shi and Malik define the similarity matrix $\mathbf{W}$ by relying upon brightness and spatial proximity, to get

$$W_{ij} = e^{-\frac{||F_i - F_j||^2}{\sigma_I^2}} e^{-\frac{||X_i - X_j||^2}{\sigma_X^2}} 1_{[0,r]}(||X_i - X_j||). \tag{5.30}$$

Only neighboring pairs of pixels lying at a distance smaller than some threshold r are considered. Usually, we find $r = 5$ pixels in the literature.

Finally, it is worth noting that the normalized cut algorithm can be employed to perform the merging of regions. Exemple code is for instance available in the scikit-image library. In this context, the normalized cut algorithm is employed on the region adjacency graph constructed from the image partition. Dissimilarity measures between adjacent regions can be obtained in a multiplicity of ways, for instance by considering the average value of the gradient on the common boundary, the difference between their mean color intensity, etc. When a training database is available for learning the segmentation, as it is the case with the Berkeley Segmentation Dataset, an additional possibility is to try to learn a probability of merging for two adjacent regions, which can be used to compute a similarity measure.

Segmentation by weighted aggregation

The segmentation by weighted aggregation (SWA) algorithm [SBB00] was initially proposed in an attempt to reduce the computational cost of the normalized cut approach. The algorithm is essentially based upon the normalized cut approach of Shi and Malik and propose to reduce the complexity of the approach by considering a multiscale version of the region adjacency graph. The runtime of this algorithm is linear in the number of pixels.

We have a graph $\mathcal{G} = (V, E)$, where V is the ensemble of nodes (pixels), E is the ensemble of edges between pixels. For a segment $S^{(m)} = (v_{m_1}, v_{m_2}, ..., v_{m_{n_m}}) \subseteq V$, we associate to it a state vector $\mathbf{u}^{(m)} = (u_1, u_2, ..., u_N)$,

$$u_i = \begin{cases} 1, & \text{if } v_i \in S^{(m)} \\ 0, & \text{if } v_i \notin S^{(m)}. \end{cases} \tag{5.31}$$

where $u_i \in \mathbb{R}$, N is the number of pixels. Let us denote by $\mathbf{W}$ the weight matrix, $\mathbf{W}(i, j) = w_{ij}$, where w_{ij} is the coupling between nodes i and j; and $\mathbf{L}$ is the Laplacian matrix $\mathbf{L} = \mathbf{D} - \mathbf{W}$, where $\mathbf{D}$ is the diagonal matrix defined in (5.26). Similarly to the normalized cut algorithm, the aim of SWA is to minimize the energy

$$\Gamma(\mathbf{u}) = \frac{\sum_{i>j} w_{ij}(u_i - u_j)^2}{\sum_{i>j} w_{ij} u_i u_j} = \frac{\mathbf{u}^T L \mathbf{u}}{\frac{1}{2}\mathbf{u}^T W \mathbf{u}} \tag{5.32}$$

which is the sum of weight along the boundary of a segment, normalized by the sum of its internal weight.

The time complexity of the normalized cut algorithm is $O(N^{\frac{3}{2}})$. SWA provides an approximated solution to this optimization by recursive coarsening, of which the runtime is linear in the size of the image. It is inspired by Algebraic Multigrid (AMG) solvers, initially used in physical systems of heat or electric networks. The graph is coarsened scale after scale by selecting a set of representative nodes at each scale. The optimization problem is solved at the coarsest scale, and the interpolation matrix at each scale are used to relate this graph partition to its corresponding segment at the finest scale.

The central problems are how to choose the representative nodes at each scale, how to obtain the weight matrix of a coarse scale knowing that at the finest scale, and how to relate the state vector obtained at a coarse scale back to the finest scale.

Region adjacency graph coarsening

Let us denote by $\mathcal{G}^{[0]}$ the original region adjacency graph of the image. At the pixel level, $\mathcal{G}^{[0]} := (V^{[0]}, E^{[0]})$, where $V^{[0]}$ is the set of all pixels in the image and $E^{[0]}$ contains the edges linking any pixel to its 4 closest neighbors. We assume that the edges e_{ij} are weighted by the similarity w_{ij}, for example the probability p_{ij} that regions i and j belong to the same segment.

The SWA algorithm proceeds by iteratively constructing a sequence of smaller graphs $\mathcal{G}^{[1]}, \mathcal{G}^{[2]}, ...$ and by computing interpolation matrices enabling to recover $\mathcal{G}^{[i-1]}$ from the coarser graph $\mathcal{G}^{[i]}$. The graph is recursively coarsen using a *weighted aggregation* procedure by selecting seeds.

At each step s of the algorithm, the graph $\mathcal{G}^{[s]}$ is constructed by selecting nodes from the coarser graph $\mathcal{G}^{[s-1]}$. Following the notations used in [Alp12], let us denote by $\mathcal{C}$ the set of the selected nodes in $V^{[s-1]}$ and by $\mathcal{C}^c$ its complementary. The selection is performed by sequentially extracting the nodes in $V^{[s-1]}$, and by adding it to $\mathcal{C}$ when the following criterion is met:

$$\frac{\sum_{j \in \mathcal{C}} w_{ij}}{\sum_{j \in V^{[s-1]}} w_{ij}} < \psi, \tag{5.33}$$

where ψ is some threshold usually taken equal to 0.2. When the criterion is met, it indicates that the current node selected in $V^{[s-1]}$ is strongly dissimilar to the nodes already in $\mathcal{C}$, and has therefore to be added to the subsequent scale description. Obviously, this approach is strongly dependent on the order over which the nodes are processed. Alpert *et al.* suggest to extract the nodes in $V^{[s-1]}$ by decreasing size. Once a subset $\mathcal{C}$ of nodes is selected in $V^{[s-1]}$, we can set $V^{[s]} := \mathcal{C}$. We are however still left with the computation of the similarity matrix between the selected nodes. The similarity matrix computation is based upon the interlevel interpolation method described below.

Interlevel interpolation

For each node $i \in \mathcal{C}^c$, let us denote by $\mathcal{N}_i = \{j \in \mathcal{C}, w_{ij} > 0\}$ its coarse neighborhood. Now, let us define the interpolation matrix $\mathbf{T} := \mathbf{T}^{[s-1][s]}$:

$$T_{ij} = \begin{cases} w_{ij} / \sum_{k \in \mathcal{N}_i} w_{ik} & \text{if } i \in \mathcal{C}^c, j \in \mathcal{N}_i \\ 1 & \text{if } i \in \mathcal{C}, j = i \\ 0 & \text{otherwise} \end{cases} \tag{5.34}$$

The matrix $\mathbf{T}^{[s-1][s]}$ can interpolate between the graphs at scales s and $s-1$, respectively. T_{ij} can be interpreted as the likelihood of the node i belonging to the aggregate j. The process of relating the coarse to fine nodes is called *weighted aggregation*, since a node at scale s is an aggregate of a set of strongly coupled nodes at scale $s-1$.

Knowing the state vector $\mathbf{u}^{[s]}$, $\mathbf{u}^{[s-1]}$ can be obtained by using the interpolation matrix:

$$\mathbf{u}^{[s-1]} \approx \mathbf{T}\mathbf{u}^{[s]} \tag{5.35}$$

Substitute $\mathbf{u}^{[s-1]}$ by its approximation in (5.35), we have

$$\frac{\mathbf{u}^{[s-1]T}\mathbf{L}\mathbf{u}^{[s-1]}}{\frac{1}{2}\mathbf{u}^{[s-1]T}\mathbf{W}\mathbf{u}^{[s-1]}} \approx \frac{\mathbf{u}^{[s]T}\mathbf{T}^{T}\mathbf{L}\mathbf{T}\mathbf{u}^{[s]}}{\frac{1}{2}\mathbf{u}^{[s]T}\mathbf{T}^{T}\mathbf{W}\mathbf{T}\mathbf{u}^{[s]}} \tag{5.36}$$

Hence, the affinity matrix $\mathbf{W}^{[s]}$ at scale s can be computed from the affinity matrix $\mathbf{W}^{[s-1]}$ at scale $s-1$:

$$\mathbf{W}^{[s]} = \mathbf{T}^{T}\mathbf{W}^{[s-1]}\mathbf{T}. \tag{5.37}$$

We obtain a similar expression for the Laplacian matrix $\mathbf{L}^{[s]}$.

The couplings at a coarser level can be computed by the relation equation (5.37), knowing the initial weight matrix $\mathbf{W}$. The simplest way to construct $\mathbf{W}$ is to consider the intensity difference and set $w_{ij} = e^{-\alpha|I_i - I_j|}$, I_i, I_j being the average intensity of the corresponding region of the node v_i, v_j. In [Alp12], Sharon *et al.* proposed a Bayesian approach that integrates both color intensity and texture cues using a mixture of experts-mike model to compute the merging propabilities p_{ij} between adjacent pixels, and use it as the similarity w_{ij}. The SWA algorithm can be easily adapted to work with any method for computing the merging probability.

Besides being directly derived from the previous level, the weight $w_{ij}^{[s]}$ at level s can also be modified to incorporate the statistics of each block, for example the average intensity level [SBB01]. The couplings can also be recalculated at each scale based on the region properties, without inheriting from the previous level. More details can be found in [Alp12].

During the coarsening process, the pixels are aggregated into a small number of most representative clusters step by step. After aggregating to the coarsest level, the state vector $\mathbf{u}$ is propogated back to the finest scale by using the relation equation (5.35), thus returning the segmentation.

Results and discussion

In this section, we present the results of the fast marching algorithm used for superpixels merging in this study. To evaluate the results, we rely on the Berkeley Segmentation Dataset (BSDS500) as in previous chapters. Here, our objective is to start from a superpixel partition of a specified image, containing roughly 500 superpixels, and to merge the superpixels to reduce their number around $60-80$. The resulting oversegmentation contains a "reasonable" number of segments and can constitute a good support to apply a classification algorithm relying on higher level features to complete the segmentation. The complete segmentation approach will be illustrated on a database of cloud images later in the manuscript.

Experiments

The approach that we propose starts from an initial superpixel segmentation of the considered image. Here, we employ the fast marching based superpixel algorithm described in chapter 2 in the manuscript. However, any superpixel algorithm, for instance SLIC, could be employed instead. In our evaluation, we start from a superpixel segmentation containing exactly $K = 500$ superpixels. The next step in our approach is to compute a region adjacency graph (RAG) for our image. Each region of the image is associated to a node of the RAG, and each pair of adjacent regions is represented by an edge, weighted by the probability that both regions should belong to the same segment.

To evaluate the performance of our algorithm with respect to other existing methods in the literature, we adapted two classical algorithms, namely the normalized cut (Ncut) and the segmentation by weighted aggregation (SWA) algorithms to our weighted RAG setting. It is straightforward to apply the original Ncut algorithm to our image description. The Ncut algorithm is indeed defined for undirected graphs whose edges carry a measure of similarity between adjacent nodes. Here, the edges of the RAG used to represent the image are weighted by their probability of belonging to the same segment, which obviously constitutes some similarity measure. Hence, the Ncut algorithm as originally formulated in the framework of graphs can directly be applied to our problem. To compute the Ncut segmentation, we directly rely on the implementation available in the library scikit-learn.

It is also relatively straightforward to adapt the original SWA approach of Galun *et al* [Gal03] to the RAG description adopted here. In the original SWA algorithm, probabilities of merging are iteratively computed in a hierarchical manner from contour and color cues. At each scale, the SWA combines the information of the previous scales to come up with new contour and color cues, that are subsequently transformed into a similarity matrix between adjacent regions in the image.

Here, as described previously in the manuscript, we directly rely on a classifier (either a random forest classifier or a gradient boosting algorithm) taking different cues as input features to evaluate the probability of merging each pair of adjacent regions. To compare our approach to the SWA algorithm, we therefore replace the original probability model by our XGBoost classifier. Starting from the resulting region adjacency graph, the SWA algorithm proceeds by coarsening the graph. To that end, the graph vertices are grouped into two subsets $\mathcal{C}$ and $\mathcal{C}^c$. Initially, the subset $\mathcal{C}$ is empty. Then, the vertices of the graph are processed one by one and added to $\mathcal{C}$ if they are "strongly dissimilar" to their neighbors already in $\mathcal{C}$. More precisely, if i is the node currently being processed, we compute the quantity $S(i)$ defined by:

$$S(i) = \frac{\sum_{j \in \mathcal{N}_i \cap \mathcal{C}} p_{ij}}{\sum_{j \in \mathcal{N}_i} p_{ij}},$$

where $\mathcal{N}_i$ denotes the subset of all vertices of the RAG adjacent to node i and p_{ij} the probability of merging regions i and j. If $S(i)$ is below some specified threshold ψ, then the region i is added to $\mathcal{C}$. Once all nodes have been processed, the vertices in $\mathcal{C}$ constitute a coarsened version of the original RAG.

To obtain an actual image segmentation, it remains to associate each vertex in $\mathcal{C}^c$ to one node in $\mathcal{C}$. To that end, we associate each vertex i in $\mathcal{C}^c$ to the vertex j in $\mathcal{N}_i \cap \mathcal{C}$ with highest merging probability p_{ij}. Obviously, the order over which the vertices of the graph are processed to construct the coarsened graph influences the result. Here, we iterate over vertices sorted in a decreasing order according to the area of their corresponding region in the image. Other approaches could have been considered, as randomly selecting a vertex at each time.

In our study, we considered two approaches. In the one-stage approach, we train a classifier on the region adjacency graph obtained from the initial superpixel partition, before reducing the number of segments in a single stage from 500 to a number in the range $50 - 100$, which depends on the threshold specified in the Eikonal merging algorithm. In the multi-stage approach, we perform the merging more gradually, by iteratively merging regions to reduce the size of the RAG and re-training to compute new dissimilarities between regions at intermediate steps.

Region metrics

To compare the segmentation results quantitatively, we use the region metrics mentioned in [Arb11]. Different from boundary metrics such as boundary recall, these metrics are less sensitive to localization errors, are therefore more appropriate for the evaluation of region based segmentation algorithms.

— *Segmentation covering* measures the average matching between a segmentation and a ground truth.

$$SC(S, S_g) = \sum_{s_i \in S} \frac{|s_i|}{|\mathcal{P}|} \max_{s_j \in S_g} \frac{|s_i \cap s_j|}{|s_i \cup s_j|}, \tag{5.38}$$

where S is a segmentation, S_g is a ground truth segmentation, and $\mathcal{P}$ is the set of pixels. The segmentation covering is the largest overlapping ratio (intersection over union) with the ground truth segment s_j, weighted by the ratio of number of pixels within the segment s_i, summed over the segments. The maximum possible value 1 can be achieved when S is identical with S_g.

— *Rand index* [Ran71] originally measures the similarity between two clusterings. Here it is used to measure the similarity between the segmentation S with the ground truth segmentation S_g.

$$RI(S, S_g) = \frac{1}{\binom{|P|}{2}} \sum_{i<j} [\mathbb{I}(S(i) = S(j) \wedge S_g(i) = S_g(j) + \mathbb{I}(S(i) \neq S(j) \wedge S_g(i) \neq S_g(j))] \tag{5.39}$$

where i and j are pixels, $\mathbb{I}$ is the indicator function, $S(i)$, $S_g(i)$ are the segment labels of pixel i in the segmentation S and the ground truth S_g respectively. The number of pixel pairs having the same relation in the two segmentations, namely inside the same segment or in different segments, both in the segmentation S and in the ground truth segmentation S_g, is counted, and divided by the total number of pixel pairs. We take the average of the Rand index over multiple ground truths.

— *Variation of information* [Mei03] measures the distance of two clusterings through the average conditional entropy

$$VI(S, S_g) = H(S) + H(S_g) - 2I(S, S_g) \tag{5.40}$$

where entropy $H(S)$, $H(S_g)$ measure the uncertainty about S and S_g respectively, and $I(S, S_g)$ is the mutual information, showing the reduction of uncertainty. The entropy of a segmentation S is defined as

$$H(S) = -\sum_{i=0}^{K} P(i) \log P(i) \tag{5.41}$$

where K is the number of segments in S, $P(i)$ is the probability of a pixel being in segment s_i, $P(i) = \frac{|s_i|}{N}$. $H(S_g)$ can be similarly obtained. The mutual information between two segmentations is defined as

$$I(S, S_g) = \sum_{i=0}^{K} \sum_{j=0}^{G} P(i,j) \log \frac{P(i,j)}{P(i)P(j)} \tag{5.42}$$

$$P(i,j) = \frac{|s_i \cap s_j|}{N}$$

where $s_i \in S$, $s_j \in S_g$, $P(i,j)$ is the probability of a pixel being in segment s_i in S, and in ground truth segment s_j in S_g.

$$I(S, S_g) = H(S) + H(S_g) - H(S, S_g) \tag{5.43}$$

Thus we have two other representations of the variation of information:

$$VI(S, S_g) = 2H(S, S_g) - H(S) - H(S_g) \tag{5.44}$$

$$VI(S, S_g) = H(S|S_g) + H(S_g|S), \tag{5.45}$$

where $H(S|S_g)$ and $H(S_g|S)$ are conditional image entropies:

$$H(S|S_g) = \sum_{i=0}^{K} \sum_{j=0}^{G} P(i,j) log \frac{P(j)}{P(i,j)} \tag{5.46}$$

similarly for $H(S_g|S)$.

One-stage algorithm

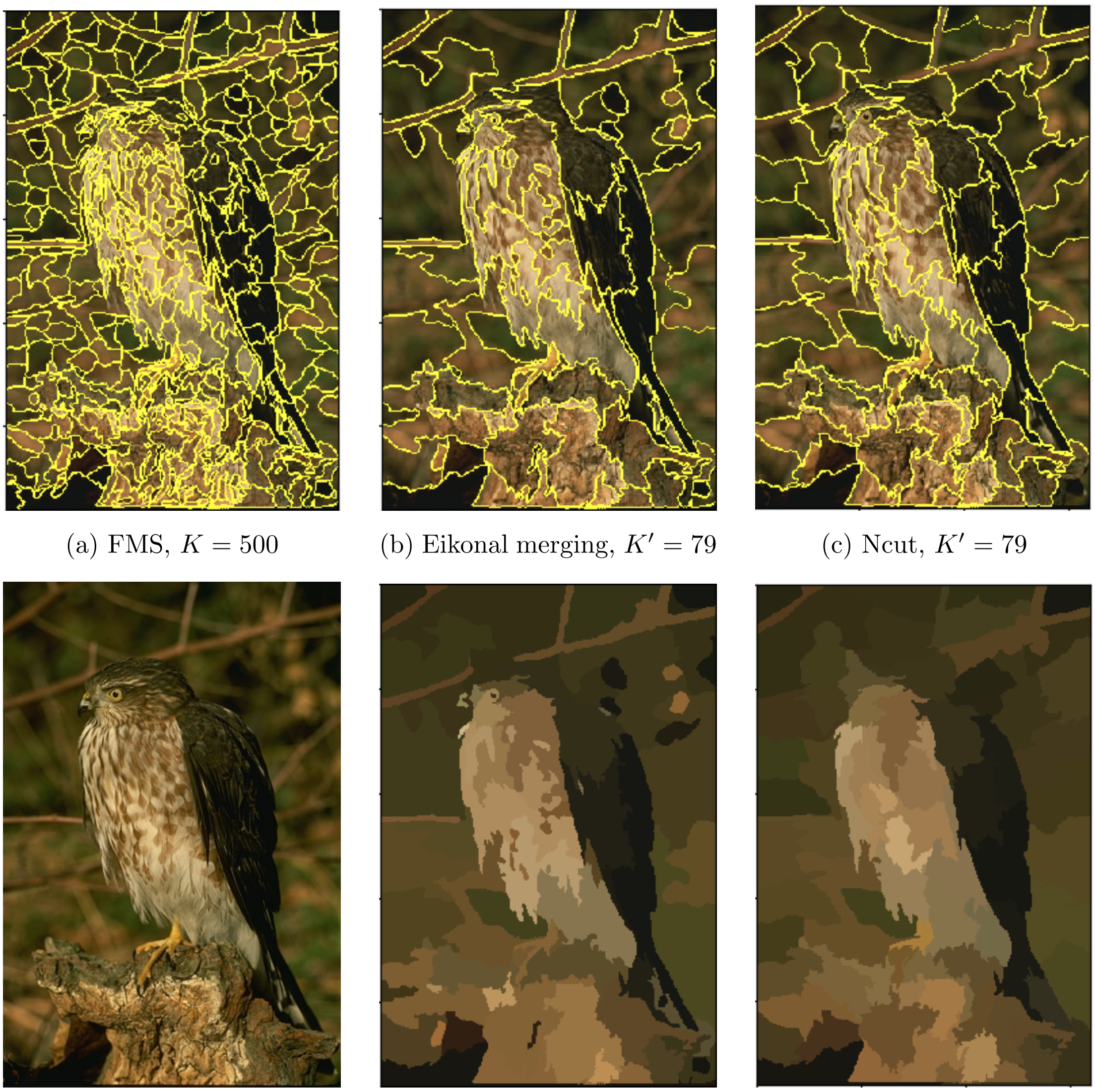
In the one-stage approach, starting from images with $K = 500$ superpixels, we decrease in a single step the number of segments to an average number of $50 - 100$ segments. Hence, the superpixels are merged according to one single round of training, conducted on the initial RAG with the training images of the Berkeley Segmentation Dataset (BSD). For this approach, we only compare the results of the Eikonal approach to the ones obtained with the normalized cut algorithm. It is indeed difficult to obtain the requested number of segments with the SWA algorithm in a single coarsening stage.

For running the Eikonal algorithm, as described in the previous section, we need to specify a threshold value t so that any region cannot have an internal edge higher than t. Here, we select t equal to the weight separating the 25% highest weights to the others 75%, before computing the segmentation. The segmentation yields a partition of the original image into K' segments. To conduct the comparison with the normalized cut algorithm, we compute a normalized cut segmentation with a specification of K' segments. We present the segmentation results in terms of boundary recall, boundary precision, covering, rand index and variation of information. The results obtained on the test set of the BSDS500 are shown in table 5.1 below.

	Eikonal	Norm. cut
boundary recall	0.79 ± 0.08	0.70 ± 0.08
Min.-Max. bound. recall	0.57-0.95	0.52-0.90
boundary precision	0.32 ± 0.11	0.28 ± 0.09
Min.-Max. bound. prec.	0.08-0.65	0.08-0.60
covering	0.23 ± 0.09	0.12 ± 0.06
Min.-Max. covering	0.06-0.52	0.03-0.41
rand index	0.75 ± 0.13	0.72 ± 0.13
Min.-Max. rand index	0.24-0.95	0.23-0.95
VoI	2.31 ± 0.44	2.90 ± 0.46
Min.-Max. VoI	1.24-3.55	1.60-4.14
number of segments	76 ± 13	75 ± 13
Min.-Max. segments	45-122	45-120

Table 5.1 – Results of the Eikonal and the normalized cut algorithm on the test images of the BSDS500.

Overall, we can note that the Eikonal approach performs significantly better than the normalized cut. A possible explanation is that the Eikonal approach is more robust to the estimation errors of the classifier used to compute the weights of the RAG. In addition, the refinement procedure introduced in the Eikonal algorithm significantly improves the results.



(a) FMS, $K = 500$

(b) Eikonal merging, $K' = 79$

(c) Ncut, $K' = 79$

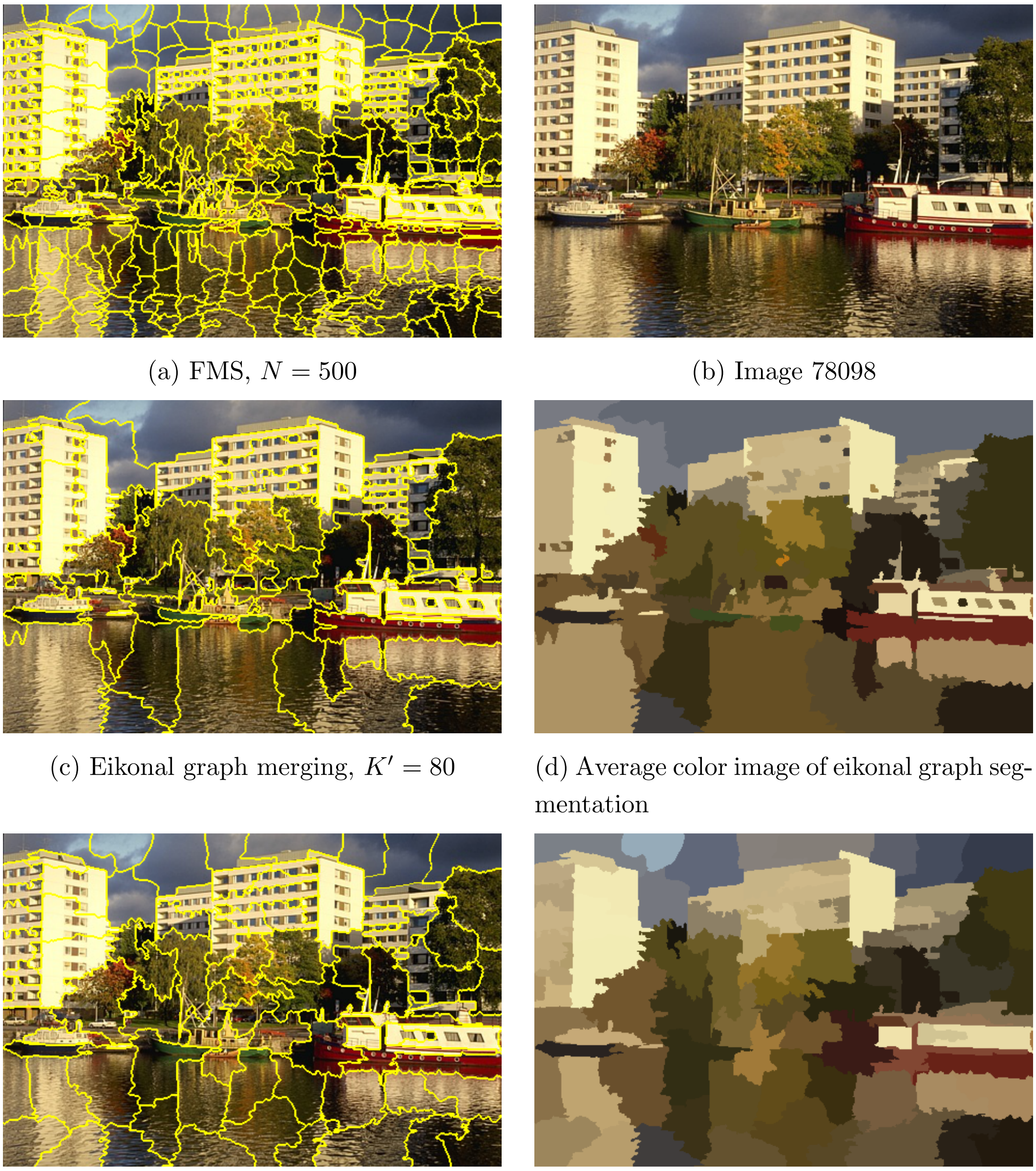
(d) Image 70090

(e) Average color image of eikonal graph segmentation

(f) Average color image of Ncut segmentation

Figure 5.1 – One-stage segmentation of a test image of the BSD using Eikonal algorithm. The boundary recall is 0.81 (Eikonal) and 0.72 (Ncut), the boundary precision is 0.17 (Eikonal) and 0.16 (Ncut).

We also display the results obtained on images of the BSD in figure 5.1 and figure 5.2. We observe from the images that, compared to the normalized cut algorithm, the Eikonal graph merging algorithm yields fewer segments for homogeneous regions, as can be seen for the sky and the water in figure 5.2, and more small and contrasted regions such as the windows on the buildings or on the boat. Firstly, the details are kept at the superpixels level due to the refinement step of the FMS. Then in the refinement step of Eikonal graph merging, seeds are added at pairs of vertices with the highest dissimilarity.



(a) FMS, $N = 500$

(b) Image 78098

(c) Eikonal graph merging, $K' = 80$

(d) Average color image of eikonal graph segmentation

(e) Ncut segmentation, $K' = 80$

(f) Average color image of Ncut

Figure 5.2 – One-stage segmentation of a test image of the BSD using Eikonal algorithm. For this image, the boundary recall is 0.69 (Eikonal) and 0.63 (Ncut), the boundary precision is 0.30 (Eikonal) and 0.26 (Ncut).

Multi-stages algorithm

We see that with a single stage, the boundary recall decreases while the precision increases when the number of segments is reduced. The aim of the multi-stages algorithm is to achieve a larger improvement of precision with the same amount of reduction of recall. In the multi-stages approach, starting from images with $K = 500$ superpixels, we decrease in several steps the number of segments to an average value of $50 - 100$ segments. The superpixels are therefore merged according to several rounds of training. Once a segmentation is obtained, the corresponding region adjacency graph is computed and a new classifier is trained to estimate the probability of merging each couple of adjacent regions. To train the classifier, we use the training images of the BSDS500. The validation images are used for hyperparameter tuning.

Here, we present a two-stage version of the algorithm. For running the Eikonal algorithm, we select at each stage a threshold t equal to the weight separating the 55% highest weights in the RAG to the other 45%. We compute a normalized cut segmentation with the same number of segments as the Eikonal algorithm. Finally, to conduct a comparison with the SWA algorithm, we apply two stages of coarsening to the original RAG representing the image. After the first stage of coarsening, a classifier is trained on the intermediate RAG of Eikonal graph merging. The threshold of SWA is adjusted to make sure that the number of segments is comparable to eikonal graph merging and the Ncut. The results obtained on the test set of the BSD are shown in tables 5.2 and 5.3 below.

	Eikonal	Norm. cut	SWA
boundary recall	0.84 ± 0.07	0.80 ± 0.07	0.82 ± 0.07
Min.-Max. bound. recall	0.67-0.96	0.61-0.94	0.63-0.95
boundary precision	0.27 ± 0.09	0.24 ± 0.08	0.24 ± 0.08
Min.-Max. bound. prec.	0.08-0.57	0.07-0.48	0.07-0.50
covering	0.16 ± 0.08	0.08 ± 0.04	0.08 ± 0.04
Min.-Max. covering	0.04-0.47	0.02-0.28	0.02-0.27
rand index	0.73 ± 0.13	0.72 ± 0.13	0.72 ± 0.13
Min.-Max. rand index	0.23-0.95	0.22-0.94	0.22-0.94
VoI	2.76 ± 0.46	3.38 ± 0.43	3.37 ± 0.41
Min.-Max. VoI	1.53-3.93	2.07-4.40	2.08-4.35
number of segments	150 ± 10	143 ± 9	151 ± 10
Min.-Max. segments	121-172	119-163	125-175

Table 5.2 – Results of the Eikonal, the normalized cut and SWA algorithm on the test images of the BSDS500.

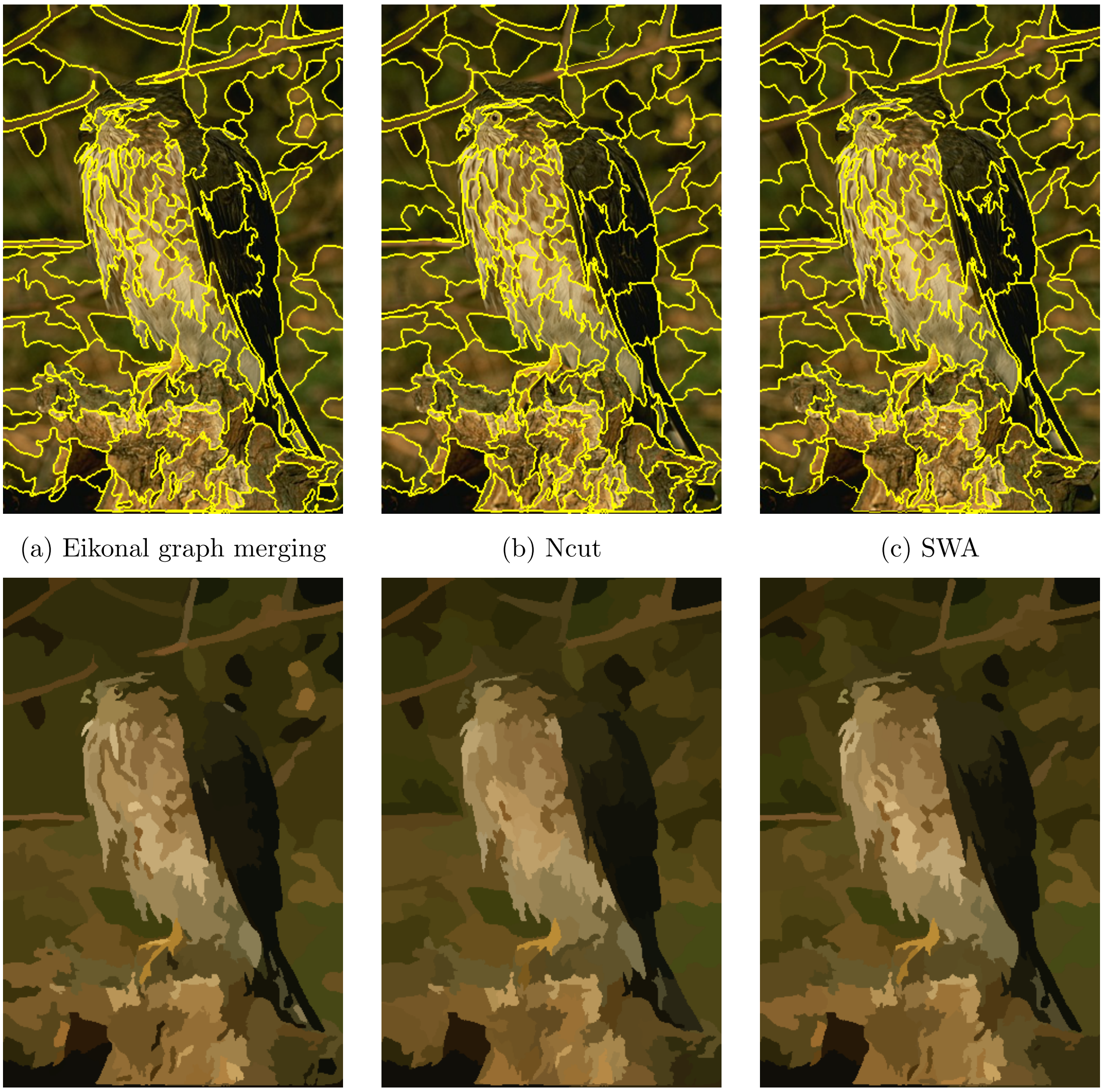
We see that the Eikonal graph merging slightly outperforms the normalized cut and SWA under both boundary metrics and region metrics. The average VoI of our algorithm is noticeable smaller than that of the other two algorithms. Compared with the single stage (F=0.46), a better F-score (0.49) and better region metrics are achieved, with a smaller number of segments.

For a multi-stages algorithm, the choice of step of segments reduction is essential. We see that the recall is reduced after one merging stage, and the wrongly merged regions can never be recovered during the subsequent stages. We observe in [RS13] that the step becomes smaller and smaller as the number of segments is reduced.

	Eikonal	Norm. cut	SWA
boundary recall	0.76 ± 0.09	0.74 ± 0.08	0.76 ± 0.09
Min.-Max. bound. recall	0.52-0.94	0.53-0.93	0.54-0.94
boundary precision	0.36 ± 0.12	0.31 ± 0.11	0.31 ± 0.10
Min.-Max. bound. prec.	0.09-0.73	0.09-0.67	0.10-0.65
covering	0.32 ± 0.12	0.23 ± 0.09	0.21 ± 0.09
Min.-Max. covering	0.09-0.71	0.07-0.52	0.06-0.50
rand index	0.77 ± 0.13	0.75 ± 0.13	0.74 ± 0.13
Min.-Max. rand index	0.26-0.96	0.24-0.95	0.24-0.95
VoI	1.95 ± 0.43	2.33 ± 0.44	2.36 ± 0.41
Min.-Max. VoI	0.80-3.11	1.25-3.49	1.36-3.49
number of segments	66 ± 10	63 ± 9	65 ± 7
Min.-Max. segments	42-95	42-94	51-87

Table 5.3 – Results of the Eikonal, the normalized cut and SWA algorithm on the test images of the BSDS500.

In figure 5.6, SWA achieves a better recall than Eikonal graph (EG) merging. For EG, when selecting new seeds, the region pair with the highest dissimilarity is firstly considered. Highly contrasted small regions such as the windows in the building are dissimilar with their surrounding, so they are kept during the merging. When the number of segments is limited, there remains fewer seeds to represent more important regions. Considering this, a criteria of region size can be applied when selecting new seeds.



(a) Eikonal graph merging

(b) Ncut

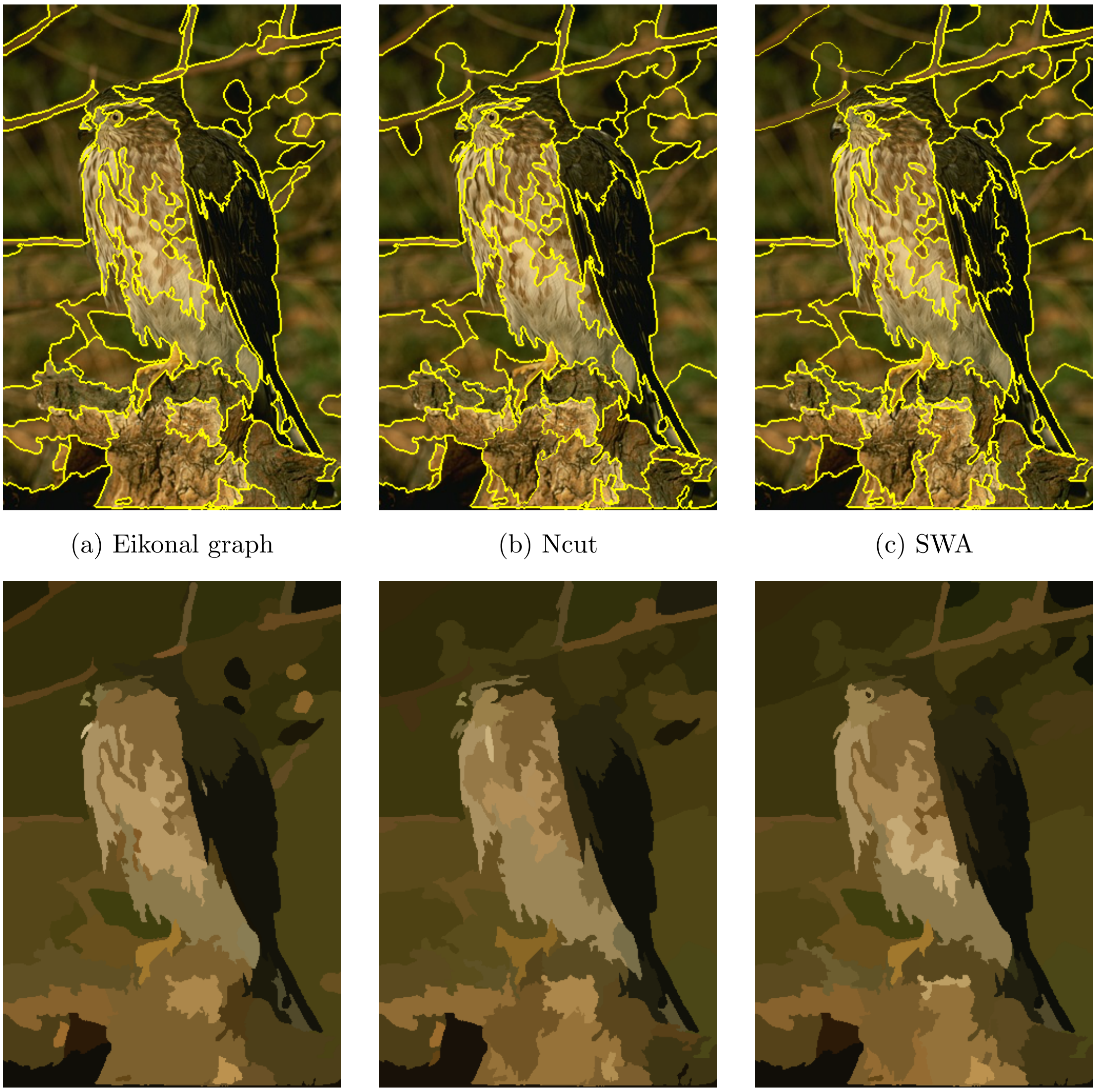
(c) SWA

(d) Average color of EG

(e) Average color of NC

(f) Average color of SWA

Figure 5.3 – Result of the first stage of the segmentation of a test image of the BSD using the Eikonal graph, normalized cut and SWA algorithm. Images in the second row show the average color of each segment. “EG” represents Eikonal graph merging, “NC” for Ncut. The boundary recall are 0.84, 0.80, 0.80; the boundary precision are 0.15, 0.13, 0.13, and the final partition comprises 153, 150, and 154 segments respectively.



(a) Eikonal graph

(b) Ncut

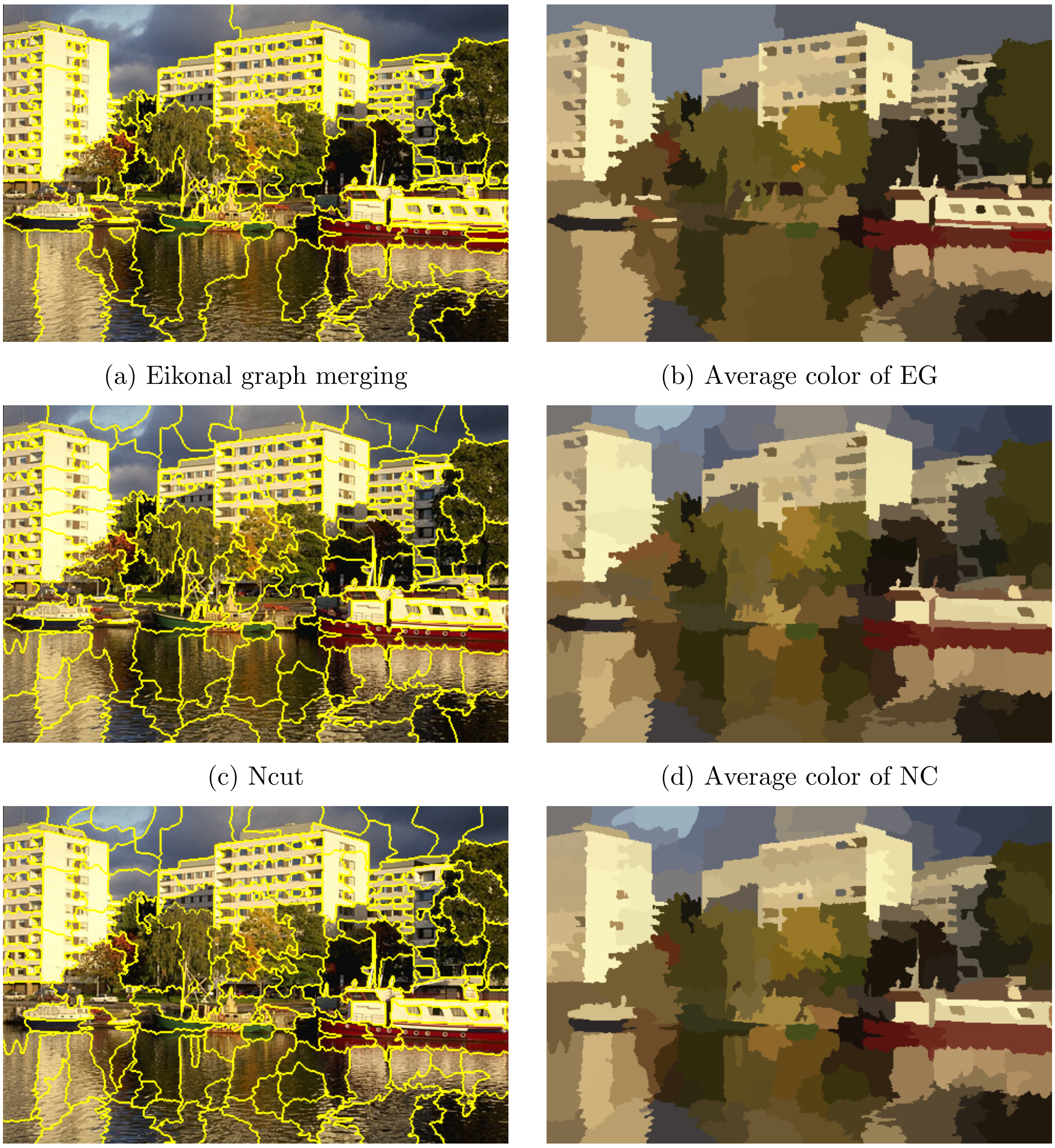
(c) SWA

(d) Average color of EG

(e) Average color of NC

(f) Average color of SWA

Figure 5.4 – Result of the second stage of the segmentation of a test image of the BSD using the Eikonal graph, normalized cut and SWA algorithm. For this image, the boundary recall are 0.76, 0.73, 0.74; the boundary precision are 0.19, 0.18, 0.17, and the final partition comprises 61, 60, and 59 segments respectively.



(a) Eikonal graph merging

(b) Average color of EG

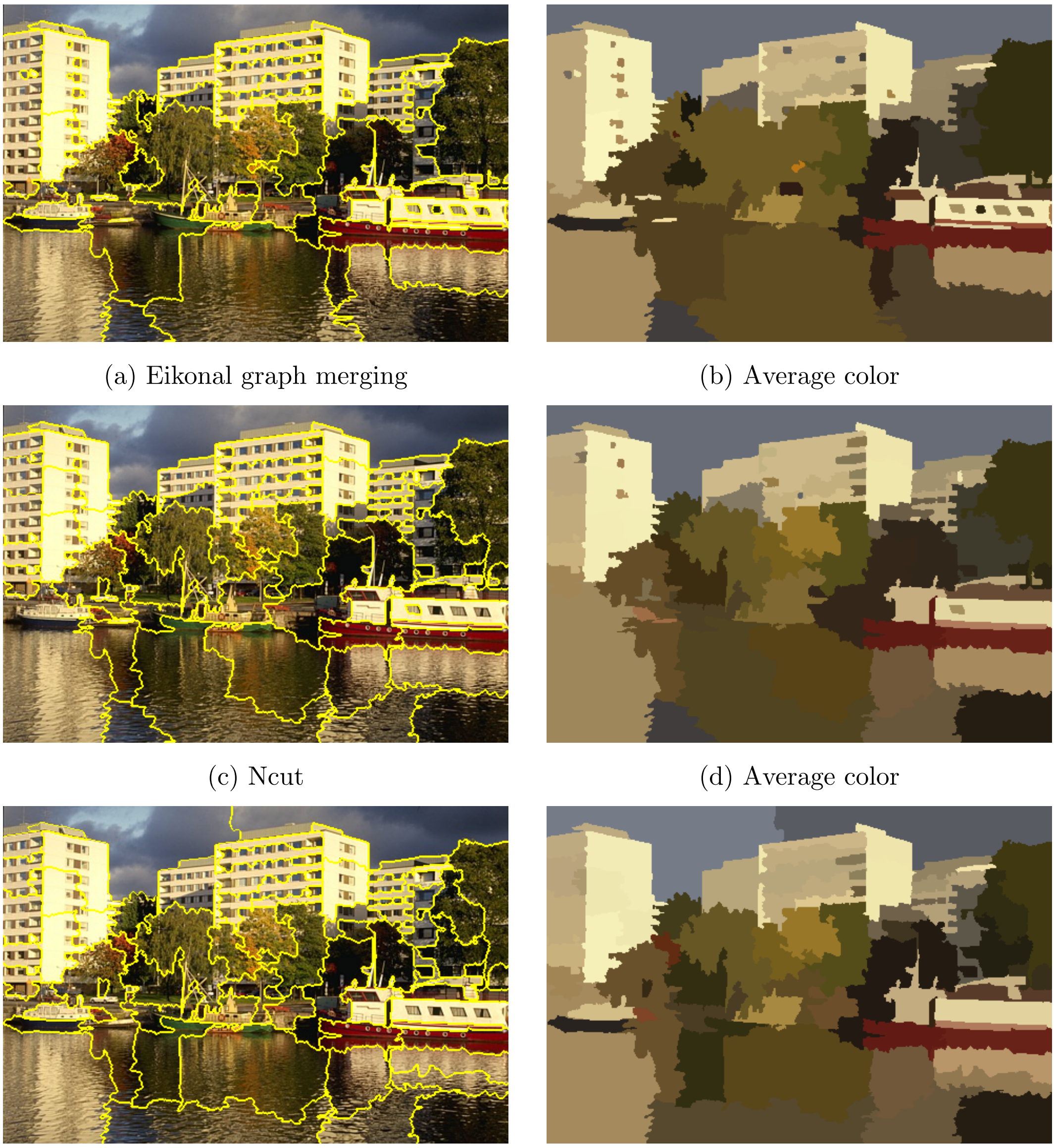
(c) Ncut

(d) Average color of NC

(e) SWA

(f) Average color of SWA

Figure 5.5 – Result of the first stage of the segmentation of a test image of the BSD using the Eikonal graph, normalized cut and SWA algorithm. The boundary recall are 0.75, 0.73, 0.71; the boundary precision are 0.26, 0.24, 0.23. The final partition comprises 152, 151, and 152 segments respectively.



(a) Eikonal graph merging

(b) Average color

(c) Ncut

(d) Average color

(e) SWA

(f) Average color

Figure 5.6 – Result of the first stage of the segmentation of a test image of the BSD using the Eikonal graph, normalized cut and SWA algorithm. The boundary recall are 0.64, 0.63, 0.69; the boundary precision are 0.33, 0.29, 0.30. The final partition comprises 70, 67, and 69 segments respectively.

Perspectives

As we have discussed in the previous subsection, a criteria of region size can be applied when selecting new seeds, to place seeds only in regions with area larger than a threshold.

In this chapter, two merging stages were applied. We also tried more stages of merging, but the improvement of precision is trivial. It might be because the number of training instances is reduced as more stages of merging are applied, while it is crucial for the accuracy of the classifier. The idea in [NI13] might be helpful for increasing the number of training instances. The main principle is to use a classifier to decide the merging order of training instances, then to train a new classifier with these training instances, and get new training instances by merging regions in the order specified by the new classifier, and so on. In the end, all training instances from each step are concatenated to train a single, final classifier. Using such an approach would allow us to update the weights of the RAG during the Eikonal propagation and could potentially improve the results of the merging algorithm.

Chapter 6

Application

In this short chapter, we present an application of our approach to perform the segmentation of a dataset of cloud images. The cloud segmentation of ground-based whole sky images is of great importance in applications such as weather prediction or to help positioning large scale solar field arrays. Here, we rely on the segmentation framework developed in the previous chapters for the segmentation of images from the Singapore Whole Sky Imaging segmentation (SWIMSEG) database [DLW17].

Introduction

Ground-based whole sky imagers equipped with fisheye lens have a field of view of about 180°, and provide high temporal and spatial resolution for sky observations. Cloud segmentation on ground-based cloud images constitutes a prerequisite for computing information such as cloud cover, that can subsequently be exploited for weather forecast or renewable energy generation applications. Human observations are subjective and somewhat inconsistent, therefore an automatic segmentation of cloud images through image analysis is desirable.

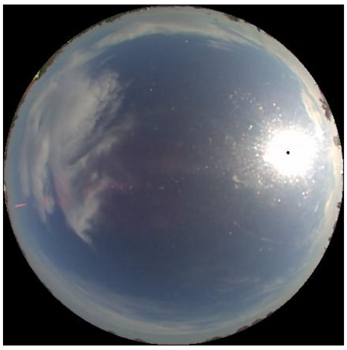


Figure 6.1 – A whole sky image.

An example of a whole sky image is given in figure 6.1. Pre-processings including color, illumination and geometry calibration have to be conducted prior to segmentation, to compensate for the vignetting and distortion induced by the camera. Figure 6.2 are examples of images in SWIMSEG database after aforementioned processings, together with ground truth segmentations from experts.

A significant difficulty of cloud sky segmentation is the color change due to illumination, due to factors including the change of sunlight during different times in a day or different weather conditions. Another difficulty is that the appearance of clouds varies, so that it is sometimes difficult to delineate a clear boundary between sky and cloud as in figure 6.2c.

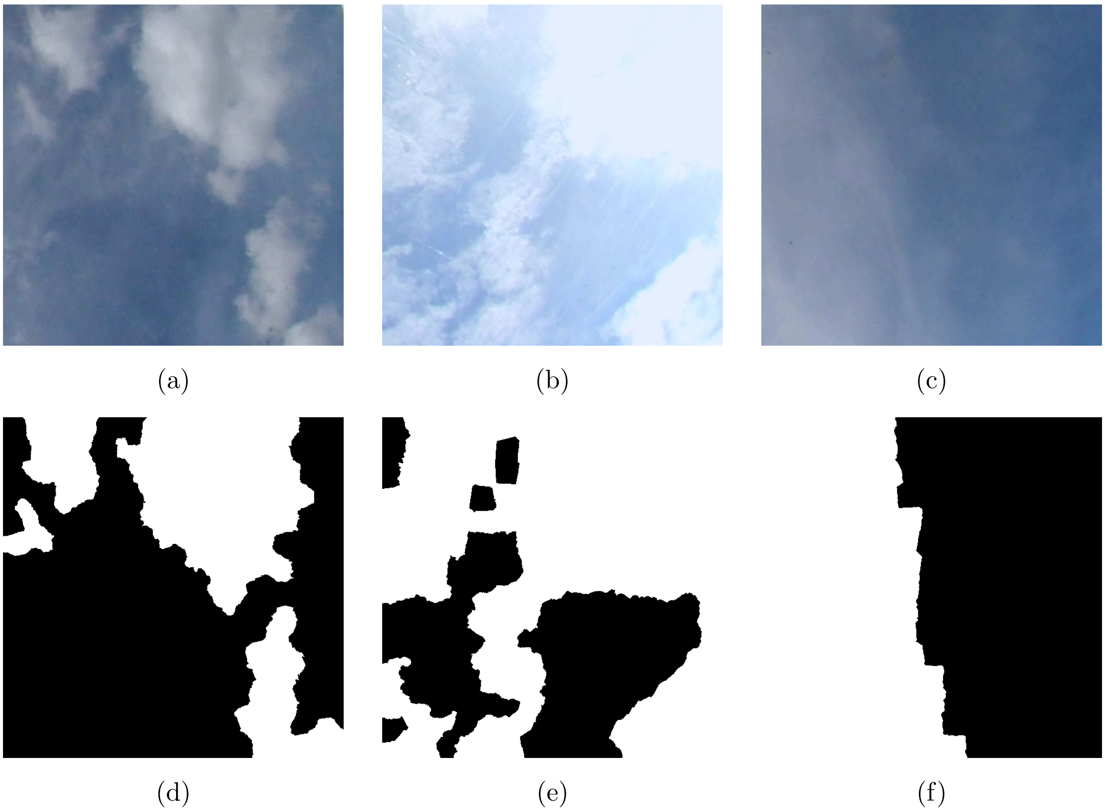

(a)
(b)
(c)
(d)
(e)
(f)

Figure 6.2 – Examples of images and ground truth from the SWIMSEG database. The first row displays the images to be segmented, while the second row corresponds to binary ground truth images with black representing sky and white representing cloud.

Early cloud segmentation algorithms often rely on a fixed thresholding operation of the ratio of blue intensity (B) over red intensity (R), or $R - B$ [Lon06; HMS10]. However, the fixed thresholding does not work well under different sky conditions. Li *et al.* [LLY11] proposed in 2011 a Hybrid Thresholding Algorithm (HYTA) for cloud detection on ground-based images, combining fixed and adaptive thresholding methods. The RGB image is firstly transformed into a single channel image with the normalized blue/red channel ratio at each pixel. The ratio images are then identified to be unimodal or bimodal images according to a standard deviation criterion. A fixed thresholding is then applied on the unimodal images, and a minimum cross entropy thresholding on the bimodal images. This algorithm is one of the state-of-the-art approaches.

A cloud segmentation algorithm based on superpixel segmentation is presented in [Liu15]. A ground-based cloud image is firstly oversegmented using the normalized cut algorithm. A global threshold on $R - B$ feature image is then computed through Otsu algorithm, and a local threshold for each superpixel is obtained by comparing the local maximal and minimal intensity with the global threshold. A threshold matrix with the same size as the feature image is constructed through bilinear interpolation, and the segmentation is obtained by pixelwise comparison between the feature image and the threshold matrix. This algorithm achieves better results than fixed threshold, global threshold and local threshold interpolation based algorithms. Superpixels are used in this algorithm as a support for local threshold calculation, but the classification is done pixelwise.

In [DLW17], Dev *et al.* conducted a systematic analysis of color spaces and components for cloud sky segmentation, and proposed a supervised segmentation approach for ground-based sky cloud images based on partial least-squares regression. An advantage of the learning based approach is that no threshold or manually defined parameters are required. Instead of returning a binary classification, a probabilistic classification of cloud pixels is provided, allowing for a soft thresholding.

In classical cloud image segmentation algorithms, a single feature is usually used, for instance the red blue ratio for classification, as in [DLW17; LLY11]. In addition, the segmentation is performed through a classification task. Here, we adopt a completely distinct approach where we apply a general segmentation algorithm to the images, with predefined features. Hence, there is no prior information given to the algorithm to perform the segmentation, and the algorithm can only rely on a limited number of training examples to fulfill its task. This allows us to reduce the oversegmentation to a total of roughly 50 regions, before applying a specific, simple classification to the images. Our objective is to see if the non specific machine learning segmentation framework developed in the previous chapters can match the performances of the specific segmentation algorithms developed to handle this classical problem.

Segmentation procedure

We test our segmentation framework on the SWIMSEG database, an image dataset with 1013 annotated whole sky images with 600×600 pixels. The images are taken by ground-based cameras, and then processed to eliminate the vignetting and the geometric distortion caused by the fish-eye lens. A binary ground truth segmented by experts is provided for each image. Figure 6.2 shows examples of cloud images from the SWIMSEG database as well as their ground truth segmentations.

The segmentation process works as follows:

1. Superpixel oversegmentation. The image is oversegmented into 800 regions through FMS with parameters $w_0 = 5$, $w_1 = 8$.
2. Labelling. An adjacent region pair is labelled to '1' if both regions belong to the same ground truth segment, or labelled to '0' if not.
3. Features calculation. A feature vector with 184 elements is extracted for each pair of adjacent regions, including geometric features (maximum and minimum of region area, the ratio between them, the maximum and minimum of region perimeter, the ratio between them, and the length of the contour separating two regions), color features (the absolute value of the difference of the average of each color channel in the RGB, LAB and HSV color space, and the earth mover's distance of histograms from RGB channels respectively), texture features (the absolute value of the difference of average responses of Gaussian filters and Gaussian Laplacian filters with different sigma, χ^2 distance of histograms of textons from aforementioned filters, and χ^2 distance of textons from RGB color channels) and gradient features (minimal, maximal, mean values and the median of the gradient on the contour).
4. Similarity learning. We train a random forest classifier with feature vectors and labels of adjacent region pairs from training images to predict the merging probability of an adjacent region pair.

   Since the images in SWIMSEG are much more homogeneous than natural images in BSDS, we expect that a model learned with a small number of representative images would generalize well to the rest of the images in the dataset. Therefore we use 20, 50 and 500 randomly selected training images respectively, and the rest images as test images.
5. Eikonal graph clustering. We construct a region adjacency graph by taking superpixel regions as nodes and linking adjacent regions by an edge. For test images, the merging probability of the adjacent regions pair is used as the edge weight in the RAG. For Eikonal based clustering, 8 initial seeds are randomly selected and 2 seeds are added at each refinement step. For each image, the weights are sorted and the threshold is set to the weight separating the 10% highest weights to the other 90%. After the merging step, we have an oversegmentation of about 50 regions.
6. Segmentation using color information. We classify in this step the segments after merging into cloud and sky regions through k-means clustering. According to [DLW17], R/B, $\frac{B-R}{B+R}$ (normalized BR ratio) and saturation channels are most relevant to cloud segmentation (R and B being red and blue channel respectively). Therefore we rely on the normalized BR ratio for the classification.

   We calculate the normalized BR ratio with the average color in each region, and then use k-means clustering to find two clusters. These clusters are then classified to be cloud cluster or sky cluster according to their respective ratio. The cluster with larger ratio is classified as cloud cluster, and the regions associated with it are classified as cloud, since the cloud always has a larger red component than sky.

Results and discussion

Three examples of the segmentation obtained with 20 training images are provided in figure 6.3. We see that our segmentation resembles the ground truth except for a few regions. Quantitative evaluations, including pixelwise precision, recall and F-score are shown in table 6.1.

Num of training images	precision	recall	F-score
0	0.902	0.919	0.892
20	0.905	0.919	0.896
50	0.903	0.922	0.897
500	0.907	0.918	0.899
500 Dev	0.92	0.90	0.90

Table 6.1 – Results of cloud sky segmentation with 800 superpixels. The first row are the results of direct application of k-means on the superpixels without learning and merging. The last row are results of the algorithm proposed in [DLW17].

Num of training images	precision	recall	F-score
0	0.890	0.918	0.882
20	0.896	0.907	0.883
500	0.902	0.908	0.890

Table 6.2 – Results of cloud sky segmentation with 200 superpixels. The first row are the results of direct application of k-means on the superpixels without learning and merging.

We can see from table 6.1 that our approach achieves state-of-the-art segmentation performance on the SWIMSEG database with only 20 training images. The performance does not improve apparently with additional training images. We also notice that the direct segmentation on superpixels achieves good results, and the involvement of learning and merging process does not apparently improve the F-score. A possible reason is that the largest difference between cloud and sky pixels lies in the red and blue channels. Therefore, in the next step, it is interesting to run random forest with only color related features for instance $R-B$, R/B, $(B-R)/(B+R)$, and saturation. A feature selection could be done to find the most effective features to reduce the calculation time in real applications.

We also tested the algorithm with 200 FMS superpixels, the results of which are shown in table 6.2. We can see that the number of initial superpixels has an influence on the segmentation performance. With 200 superpixels we can still achieve a F-score of 0.890.

The reduction of superpixel numbers reduce both the time of superpixel generation and the time of feature calculation. The number of superpixels can be chosen relying on the trade off between segmentation accuracy and the calculation time.

More complicated features such as the texture features could potentially be useful for the classification of different clouds (cumuliform, stratiform, etc.), which is generally the next step of the cloud segmentation.

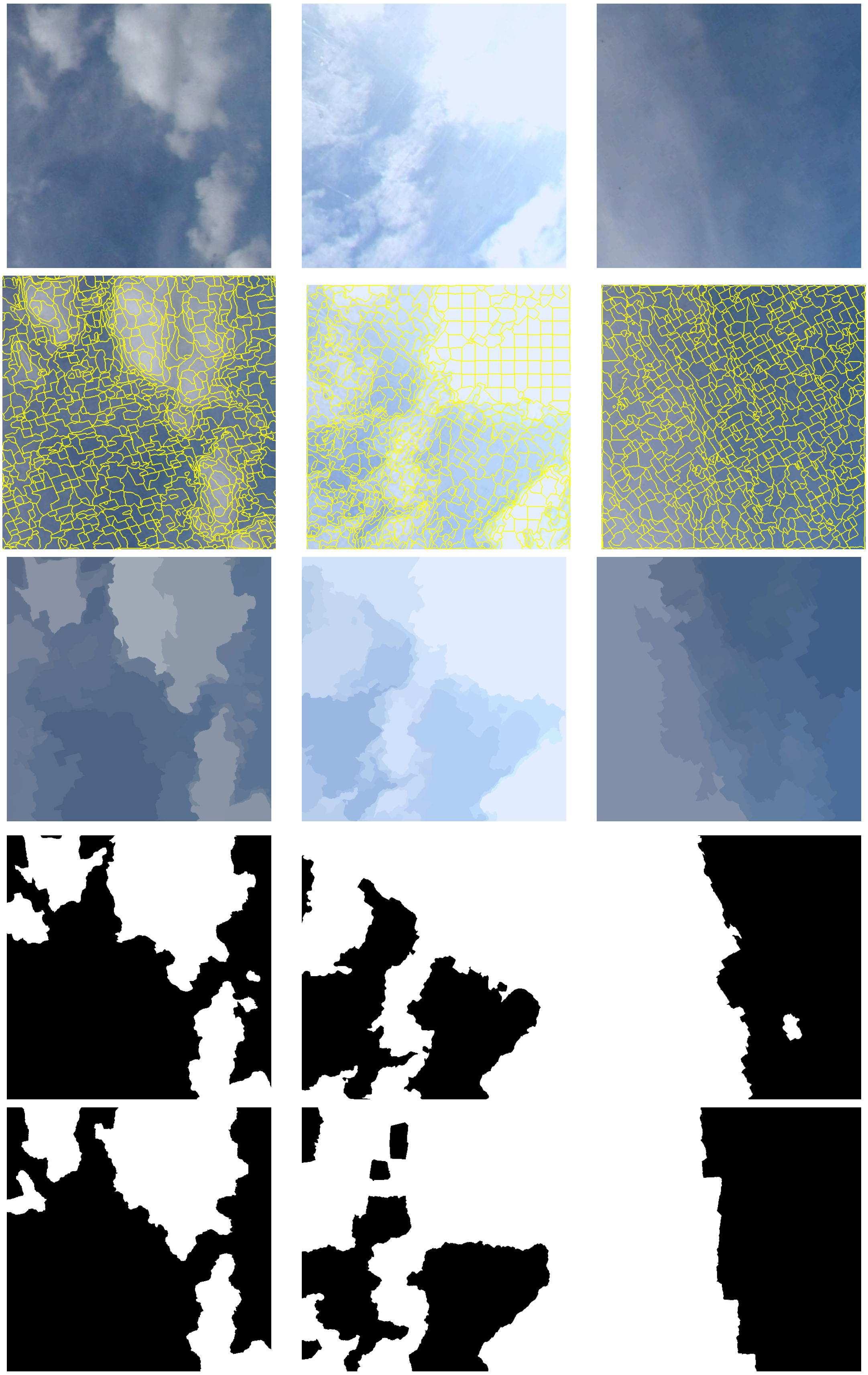


Figure 6.3 – Examples of cloud sky segmentation. The second row shows superpixels images, the third row shows the average color images after superpixel merging, the fourth row shows segmentation results, the last row shows ground truth images.

Conclusion and perspectives

Conclusion

In this manuscript, we presented a general methodology for performing the segmentation of a dataset constituted of similar images with only a few annotated images as training examples. This methodology is intended to be applied to images gathered in earth observation or materials science applications, where there is not enough annotated segmentation examples to use state-of-the-art deep learning algorithms.

The proposed methodology starts from a superpixel partition of the image and uses a merging algorithm which gradually merges the initial regions until an actual segmentation is obtained. The main contributions described in this PhD manuscript are the development of a new superpixel algorithm which makes use of the Eikonal equation and of the fast marching resolution method, and the development of a superpixel merging algorithm steming from the adaptation of the Eikonal equation to the setting of weighted graphs.

In the first part of the manuscript, corresponding to chapters 2 and 3, we introduced two novel algorithms that can be used to compute a superpixel partition and a hierarchical partition of the image, respectively. The superpixel algorithm presented in chapter 2 makes use of the Eikonal equation to generate the superpixel partition, in a way relatively similar to the algorithm introduced by Buyseens *et al.* [Buy14b]. This algorithm compares favorably to other state-of-the-art superpixel algorithms on the Berkerley Segmentation Dataset.

In the second part of the manuscript, corresponding to chapters 4 and 5, we presented an algorithm performing superpixels merging based upon learned similarities between adjacent regions. The superpixels merging approach makes use of a region adjacency graph computed from the superpixel partition. The edges are weighted by a dissimilarity measure corresponding to the probability that superpixels belong to distinct segments of the final segmentation. A machine learning algorithm based upon random forests and XGboost classifiers was specifically designed for evaluating inter-region dissimilarity depending on low-level cues computed on the superpixels. A procedure using an adapted version of the Eikonal equation to the setting of graphs was finally used to perform the clustering of the region adjacency graph. When compared to classical approaches for performing region mergings, including the normalized cut algorithm and the segmentation by weighted aggregation, the proposed approach yielded overall better results on most classical metrics used to evaluate segmentations.

The global approach to image segmentation was finally evaluated on a dataset of sky cloud images. On this dataset, using only a limited amount of images for training our algorithm, we were able to obtain segmentation results similar to the ones obtained with state-of-the-art algorithms.

Perspectives

Several directions of research can be thought of after the present work.

— For the Eikonal based superpixel algorithm, the weights of color and texture distance terms are chosen at dataset level, which is not optimal for each image. In the next step, we could try to determine the weights locally, adaptive to the image content.

— Although we aim to develop a general methodology of segmentation, we should notice that for the segmentation of images in a specific domain, features should be selected to adapt to the characteristics of the images for the sake of computational efficiency. For example, for cloud images, features in Lab color space is less important than those in RGB color space, and could therefore be eliminated.

— Another potential direction of research is directly related with the merging algorithm using to combine the superpixels into actual segments. In its current state, a similarity measure between adjacent regions is computed before merging, and the weights are kept constant during the merging procedure. Following the idea in [NI13], it might be interesting to try to update the weights of the region adjacency graph dynamically during the merging process.

Appendix A

Appendices

Proof of proposition 2.3.3

We recall equation (2.23):

$$\max(t_{ij} - t_A, 0)^2 + \max(t_{ij} - t_B, 0)^2 = \frac{1}{u_{ij}^2}$$

where $t_A = \min(t_{i-1,j}, t_{i+1,j})$ and $t_B = \min(t_{i,j-1}, t_{i,j+1})$. By construction, (i, j) is adjacent to at least one frozen pixel. Without loss of generality, we can assume that A belongs to the frozen set and that $t_A \leq t_B$. B can be frozen, in the narrow band, or far away (in the latter case, $t_B = +\infty$).

Proof. When the condition $t_A \leq t_{ij} \leq t_B$ is fullfilled, equation (2.23) becomes

$$\max(t - t_A, 0)^2 = \frac{1}{u_{ij}^2}. \tag{A.1}$$

Since $t \geq t_A$, the only possible solution is

$$t = t_A + \frac{1}{u_{ij}} \tag{A.2}$$

This solution is only valid if

$$t_A + \frac{1}{u_{ij}} \leq t_B, \tag{A.3}$$

which is satisfied automatically, since we follow the front propogation, $t_A \leq t_{ij} \leq t_B$ means the front reaches (i, j) before it reaches $(i, j-1)$ or $(i, j+1)$.

When this condition is not satisfied, namely $t_A \leq t_B \leq t_{ij}$, equation (2.23) reads

$$(t - t_A)^2 + (t - t_B)^2 = \frac{1}{u_{ij}^2}, \tag{A.4}$$

and has two distinct, real solutions. The discriminant of (A.4) is indeed

$$\Delta = \frac{2}{u_{ij}^2} - (t_A - t_B)^2 \tag{A.5}$$

and is strictly positive since

$$\frac{1}{u_{ij}} \geq t_B - t_A. \tag{A.6}$$

Hence, the two solutions of (A.4) are

$$t_{\pm} = \frac{t_A + t_B}{2} \pm \frac{1}{2}\sqrt{\Delta}. \tag{A.7}$$

Among these two solutions, only one satisfies the condition $t \geq t_A$. □

Numerical scheme for the Eikonal equation

We discuss in this section the numerical scheme used to solve the Eikonal equation. The presentation of this topic is inspired by the classical article published by Sethian in 1999. Here, our aim is to show that the numerical scheme proposed to solve the Eikonal equation with the fast marching method on a discretized domain $\Omega \subset \mathbb{R}^2$ is stable.

Convergence of the fast marching algorithm

In the following, let us consider a 1D domain Ω, discretized on a grid $\{x_0, x_1, x_2, .., x_N\}$ with uniform spacing Δx. On the domain, the Eikonal equation takes the form

$$u(x)\|\nabla T(x)\| = 1.$$

We set $u(x_0) = 0$ as boundary condition, and we assume that there exists some constant $C > 0$ such that $u(x) > C, \forall x \in \Omega$. To compute the arrival at some point $x_j, j \in \{1, ..., N\}$ of the grid, we rely on the discretization formula

$$\max\left(\frac{T_{j+1} - T_j}{\Delta x}, \frac{T_{j+1} - T_j}{\Delta x}, 0\right)^2 = \frac{1}{u(x_j)^2}.$$

In this expression, T_j denotes the arrival time $T(x_j)$ at point x_j. The discretization formula can be recasted as:

$$\frac{u(x_j)}{\Delta x}\Big(T_j - \min(T_{j+1}, T_{j-1}, T_j)\Big) - 1 = 0.$$

When the discretization step Δx tends to 0, we obtain

$$u(x_j)\max(T'(x_j), -T'(x_j), 0) - 1 = 0,$$

so that

$$u(x_j)|T'(x_j)| - 1 = 0,$$

since $\max(T'(x_j), -T'(x_j), 0) = |T'(x_j)|$. When the discretization step decreases to zero, we recover the original 1D Eikonal equation: this proves that the numerical scheme is *consistent*. Proposition A.2.1 below proves the stability of the fast marching algorithm.

Proposition A.2.1. *For any $j > 0$, we have*

$$T_j = \sum_{k=1}^{j} \frac{\Delta x}{u_k}.$$

Proof. Due to the topology of the considered domain, the propagation front visits the points $x_k, k \in \{0, ..., N\}$ by increasing indices. Therefore, for $j \in \{0, N-1\}$, we have

$$\frac{T_{j+1} - T_j}{\Delta x} = \frac{1}{u_{j+1}}.$$

according to the discretization formula. Summing between O and j therefore yields

$$T_j = \sum_{k=1}^{j} \frac{\Delta x}{u_k}.$$

□

A direct consequence of proposition A.2.1 and of the assumption $u(x) > C, \forall x \in \Omega$ is stated in the following corollary:

Corollary A.2.1.1. *For all $j \in \{0, ..., N\}$, we have*

$$0 \leq T_j \leq \frac{j\Delta x}{C} < \frac{L}{C},$$

where $L := N\Delta x$ is an upper bound on the size of the domain.

Corollary A.2.1.1 states that, in the 1D case, the arrival times are uniformly bounded. In addition, the bound is independant from the discretization step Δx when the discretization domain is finite. This proves that the fast marching algorithm is stable in the 1D case. Being both stable and consistent, the fast marching algorithm converges toward a solution of the Eikonal equation on a 1D domain.

Stability of the fast marching algorithm

In this subsection, our aim is to prove that the fast marching algorithm as defined in the framework of graphs is stable. To that end, we consider a weighted undirected graph $\mathcal{G} : (V, E)$. We assume that the weights of the graph are lower bounded by some constant $w^* > 0$.

Let us consider an arbitrary step of the fast marching algorithm. We assume that a vertex v with associated time t_v has been extracted from the narrow band and frozen. We denote by t_v^{-1} the time that has been accepted in the narrow band at the previous step of the algorithm. Note that the corresponding vertex of the graph might not be adjacent to v. Since v is in the narrow band, one of its neighbors has already been associated a finite time and has been frozen. Since the fast marching algorithm freezes the points by increasing order of arrival time, for u in the narrow band, we necessarily have

$$\min_{u \in U \setminus v} t_u \geq t_v^{-1},$$

where U is the ensemble of vertices in the narrow band. Therefore, we have the inequality

$$0 \leq t_v - t_v^{-1} \leq \frac{1}{w^*}.$$

We can iteratively apply this inequality until we reach a time $t_v^{-K} = 0$ corresponding to a point located on the initial front, to obtain:

$$0 \leq t_v \leq \frac{K}{w^*}.$$

Finally, K is necessarily bounded by the number $|E|$ of edges in the graph, so that

$$0 \leq t_v \leq \frac{|E|}{w^*}.$$

This result proves the arrival times are uniformly bounded, and consequently that the proposed algorithm is stable.

Resumé

Dans cette thèse de doctorat, notre objectif est d'établir une méthodologie générale permettant d'effectuer automatiquement la segmentation de bases d'images de contenu similaire à partir d'un faible volume d'images annotées. Cette méthode vise à être appliquée à des images issues d'observations satellitaires, ou de campagnes expérimentales menées en science des matériaux. Pour ce type d'application, il est en effet difficile d'obtenir un volume suffisant d'annotations pour entraîner les algorithmes d'apprentissage profond qui constituent à l'heure actuelle l'état de l'art en matière de segmentation d'images.

Les algorithmes récents de segmentation d'image s'appuient généralement sur des outils d'apprentissage supervisé. Les réseaux de neurones convolutionnels nécessitent en particulier d'être entrainés à partir d'un nombre conséquent d'exemples de segmentations. Dans de nombreuses applications, qui incluent notamment l'imagerie satellitaire ou les images de microstructures de matériaux, le manque d'images préalablement annotées pour la segmentation constitue ainsi un frein significatif à l'utilisation de ces méthodes. Des algorithmes supervisés plus traditionnels, basées sur des caractéristiques extraites manuellement des images, peuvent cependant être entraînés sur des bases d'image de taille plus restreinte. L'algorithme gPb de Arbelaez *et al.* [Arb11] est ainsi entraîné sur une base de 200 images.

La méthodologie présentée dans cette thèse est basée sur une approche par régions. La méthode s'appuie sur une partition préliminaire de l'image en superpixels, et vise à fusionner les superpixels jusqu'à l'obtention d'une segmentation de l'image. Les principales contributions de ce travail de thèse sont le développement d'un algorithme de génération de superpixels basé sur l'équation eikonale, et le développement d'un algorithme de fusion de superpixels basé sur une adaptation de l'équation eikonale au contexte des graphes. Dans cette approche, l'image est representée sous forme d'un graphe d'adjacence entre superpixels, dont les arêtes sont pondérées par une mesure de dissimilarité entre superpixels voisins. Cette mesure de dissimilarité est apprise par un algorithme d'apprentissage artificiel à partir de caractéristiques statistiques extraites de chaque superpixel.

L'approche développée est illustrée sur une base d'image contenant des images satellitaires du ciel décrivant la présence de nuages, la base *Singapore Whole Sky Imaging segmentation* (SWIMSEG) [DLW17]. Sur cette base d'image, en utilisant uniquement un nombre limité d'images d'entraînement, nous obtenons avec notre approche des résultats similaires à l'état de l'art.

Le travail présenté dans ce manuscrit de thèse se focalise sur deux aspects de la méthodologie proposée: la génération de superpixels à partir de l'équation eikonale et la fusion de superpixels. Dans le chapitre 1, nous détaillons la vaste littérature scientifique qui existe autour de la question de la segmentation d'image et nous présentons plus en détail les objectifs de la présente étude. Dans la première partie du manuscrit, qui correspond aux chapitres 2 et 3, nous introduisons deux nouveaux algorithmes, qui peuvent respectivement être utilisés pour calculer la partition en superpixels d'une image et une version multi-échelle de cette même partition.

L'algorithme de superpixels présenté dans le chapitre 2 s'appuie sur l'équation eikonale et l'algorithme de fast marching [Set96] pour générer une partition en superpixels de l'image étudiée, s'inspirant d'une approche initialement proposée par Buyssens *et al.* [Buy14a] en 2014. Partant d'un ensemble de germes initiaux, la partition est calculée en simulant la propagation d'ondes issues de ces germes sur un champ de vitesse dépendant des propriétés locales des pixels, comme leur couleur ou leur texture. Le calcul du champ de vitesse utilisé par l'équation eikonale est un aspect essentiel de l'algorithme. Dans leurs travaux de 2014 [Buy14a], Buyssens *et al.* considèrent un champ de vitesse qui dépend de la différence entre la couleur du pixel où le champ est calculé et la couleur moyenne des pixels traversés par l'onde incidente. Dans notre approche, nous prenons également en compte la texture locale en ajoutant un terme de texture dans le calcul de la vitesse locale. Un raffinement itératif de la partition en superpixels est également implémenté, qui consiste à implanter itérativement de nouveaux germes pour regénérer la partition là où les temps d'arrivée des ondes sont les plus importants. L'algorithme développé se compare favorablement aux algorithmes SLIC [Ach12] et ERGC [Buy14a] sur la Berkerley Segmentation Dataset [Mar01]. Les travaux présentés dans ce chapitre ont été présentés et publiés dans la conférence ISMM 2019 [CF19].

Dans le chapitre 3, relativement indépendant du reste du manuscrit, nous décrivons un algorithme de segmentation multi-échelle basé sur la transformée en ondelettes et la ligne de partage des eaux morphologiques, qui permet de construire une partition en superpixel multi-échelle d'une image. Le principe de la méthode est de s'appuyer sur la transformée en ondelettes orthogonale pour obtenir des approximations multi-résolution de l'image à segmenter. Les minima du gradient de chaque approximation de l'image sont ensuite repropagés sur l'image originale, et sélectionnés comme marqueurs pour la ligne de partage des eaux, ce qui conduit à l'obtention de contours hiérarchiques. L'approche proposée est évaluée de manière quantitative sur des images de la Berkeley Segmentation Dataset, et a été publiée dans les proceedings de la conférence ISMM 2017 [FCF17].

Dans la seconde partie du manuscrit, qui correspond aux chapitres 4 et 5, nous présentons un algorithme de fusion des superpixels basé sur un algorithme d'apprentissage qui estime une mesure de similarité entre superpixels voisins. La fusion des superpixels est déterminée dans le contexte des graphes. Plus précisément, un graphe d'adjacence est

calculé à partir de la partition en superpixels. Chaque superpixel de la partition initiale correspond à un noeud du graphe d'adjacence, et les frontières entre superpixels adjacents sont représentées par les arêtes du graphe. Les arêtes sont pondérées par une mesure de dissimilarité entre superpixels, qui s'interprète comme étant la probabilité que les superpixels appartiennent au même segment dans la segmentation finale. Un algorithme d'apprentissage artificiel est utilisé pour estimer cette probabilité, qui se base sur des caractéristiques statistiques calculées sur les superpixels, comme leur couleur, leur texture, ou encore l'intensité du gradient sur la frontière qui sépare les superpixels.

Dans le chapitre 4, nous décrivons en détail la méthode utilisée pour apprendre la mesure de similarité entre superpixels voisins, ainsi que les critères statistiques utilisés en entrée du modèle d'estimation. La méthode permet de combiner différents critères de similarité entre eux pour en extraire une unique mesure. En effet, des critères de similarité univariés, comme par exemple la différence en termes de niveau de gris moyen entre superpixels, ou l'intensité moyenne du gradient sur la frontière qui sépare les superpixels restent trop simples pour capturer la complexité du problème. Dans ce chapitre, nous définissons un certain nombre de critères statistiques permettant de caractériser la similarité entre superpixels, et nous utilisons deux familles d'algorithmes d'apprentissage, les forêts aléatoires et le gradient boosting (XGBoost), pour combiner ces différents critères entre eux en une mesure univariée de similarité.

Dans le chapitre 5, nous présentons une nouvelle méthode de clustering permettant de partitionner le graphe d'adjacence associé à la partition en superpixels de l'image. Cette méthode est basée sur un algorithme de Fast marching sur le graphe d'adjacence et sur la mesure de dissimilarité introduite dans le chapitre 4. De manière similaire à l'approche proposée dans le chapitre 2 pour calculer une partition en superpixels, nous utilisons l'équation eikonale pour calculer une partition du graphe d'adjacence. L'équation eikonale est en particulier généralisée au contexte spécifique des graphes. La partition résultante permet l'obtention d'une segmentation de l'image à partir de ses superpixels en regroupant efficacement les superpixels adjacents. L'approche proposée est comparée aux approches classiques pour effectuer des fusions de régions, y compris l'algorithme de coupe normalisée [SM00] (NCUT) et la segmentation par agrégation pondérée (SWA) [Alp12]. La comparaison est effectuée à partir de la Berkeley Segmentation Dataset. Nous présentons également des résultats obtenus en appliquant de manière itérative l'algorithme de fusion et en recalculant à chaque étape la mesure de similarité employée. L'algorithme proposé permet d'obtenir de meilleurs résultats que l'algorithme de coupe normalisée et la segmentation par agrégation pondérée.

Pour conclure le manuscrit, nous présentons dans le chapitre 6 l'application de notre méthodologie à la segmentation d'images satellitaires de nuages. La segmentation de ces images est d'une grande importance dans des applications telles que la prévision météorologique ou pour aider à positionner des réseaux de panneaux solaires à large

échelle. Ici, nous nous appuyons sur le cadre de segmentation développé dans les chapitres précédents pour la segmentation des images de la base de données SWIMSEG. Notre approche donne des résultats similaires à l'état de l'art en n'utilisant qu'un nombre restreint d'exemples d'apprentissage.

Bibliography

[AS17] R. Achanta and S. Susstrunk. “Superpixels and Polygons Using Simple Non-iterative Clustering”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Honolulu, HI: IEEE, July 2017, pp. 4895–4904.

[Ach12] R. Achanta et al. “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.11 (Nov. 2012), pp. 2274–2282.

[Alp12] S. Alpert et al. “Image Segmentation by Probabilistic Bottom-Up Aggregation and Cue Integration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34.2 (Feb. 2012), pp. 315–327.

[AJ07] J. Angulo and D. Jeulin. “Stochastic watershed segmentation”. In: *Proc. of the 8th Int. Symposium on Mathematical Morphology.* 2007, pp. 265–276.

[Arb09] P. Arbelaez et al. “From contours to regions: An empirical evaluation”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* June 2009, pp. 2294–2301.

[Arb12] P. Arbeláez et al. “Semantic segmentation using regions and parts”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition.* 2012, pp. 3378–3385.

[Arb11] P. Arbeláez et al. “Contour Detection and Hierarchical Image Segmentation”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 33.5 (May 2011), pp. 898–916.

[Arb14] P. Arbeláez et al. “Multiscale Combinatorial Grouping”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition.* June 2014, pp. 328–335.

[Ber12] M. Van den Bergh et al. “SEEDS: Superpixels Extracted via Energy-Driven Sampling”. In: *Computer Vision – ECCV 2012.* Ed. by Andrew Fitzgibbon et al. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 13–26.

[Ber13] K. B. Bernander et al. “Improving the stochastic watershed”. In: *Pattern Recognition Letters* 34.9 (2013), pp. 993–1000.

[Beu94] S. Beucher. “Watershed, Hierarchical Segmentation and Waterfall Algorithm”. In: *Mathematical Morphology and Its Applications to Image Processing.* Ed. by Max A. Viergever, Jean Serra, and Pierre Soille. Vol. 2. Dordrecht: Springer Netherlands, 1994, pp. 69–76.

[BL79] S. Beucher and C. Lantuéjoul. “Use of Watersheds in Contour Detection”. In: Sept. 1979.

[Bor18] V. Bortolussi et al. “Morphological modeling of cold spray coatings”. In: (2018).

[Bre01] L. Breiman. “Random Forests”. In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32.

[BA83] P. Burt and E. Adelson. “The Laplacian Pyramid as a Compact Image Code”. In: *IEEE Transactions on Communications* 31.4 (Apr. 1983), pp. 532–540.

[BGR14] P. Buyssens, I. Gardin, and S. Ruan. “Eikonal based region growing for superpixels generation: Application to semi-supervised real time organ segmentation in CT images”. In: *IRBM* 35.1 (Dec. 2014), pp. 20–26.

[Buy14a] P. Buyssens et al. “Eikonal-based region growing for efficient clustering”. In: *Image and Vision Computing* 32.12 (Dec. 2014), pp. 1045–1054.

[Buy14b] P. Buyssens et al. “Eikonal-based vertices growing and iterative seeding for efficient graph-based segmentation”. In: *IEEE International Conference on Image Processing (ICIP 2014).* Paris, France, Oct. 2014, 5 pp.

[CJ19] P. Cettour-Janet et al. “Watervoxels”. In: *Image Processing On Line* 9 (2019), pp. 317–328.

[CF19] K. Chang and B. Figliuzzi. “Fast Marching Based Superpixels Generation”. In: *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing.* Springer. 2019, pp. 350–361.

[CG16] T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD ’16* (2016). arXiv: 1603.02754, pp. 785–794.

[CR09] J. Cheng and J. C. Rajapakse. “Segmentation of Clustered Nuclei With Shape Markers and Marking Function”. In: *IEEE Transactions on Biomedical Engineering* 56.3 (Mar. 2009), pp. 741–748.

[Che16] M Cheng et al. “HFS: Hierarchical Feature Selection for Efficient Image Segmentation”. In: *Computer Vision – ECCV 2016.* Ed. by B. Leibe et al. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 867–882.

[CM02] D. Comaniciu and P. Meer. “Mean shift: A robust approach toward feature space analysis”. In: *IEEE Transactions on pattern analysis and machine intelligence* 24.5 (2002), pp. 603–619.

[DLW17] S. Dev, Y. H. Lee, and S. Winkler. “Color-Based Segmentation of Sky/Cloud Images From Ground-Based Cameras”. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 10.1 (Jan. 2017), pp. 231–242.

[DD03] E. Dokladalova and P. Dokladal. “A parallel architecture for curve-evolution PDEs”. In: *Journal of Image Analysis and Stereology* 22.2 (2003), pp. 121–132.

[DTB06] P. Dollár, Z. Tu, and S. Belongie. “Supervised Learning of Edges and Object Boundaries”. In: *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*. CVPR ’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 1964–1971.

[DZ15] P. Dollár and C. L. Zitnick. “Fast Edge Detection Using Structured Forests”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.8 (2015), pp. 1558–1570.

[Eve10] M. Everingham et al. “The pascal visual object classes (voc) challenge”. In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[Far13] C. Farabet et al. “Learning Hierarchical Features for Scene Labeling”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.8 (2013), pp. 1915–1929.

[FH04] P. F. Felzenszwalb and D. P. Huttenlocher. “Efficient graph-based image segmentation”. In: *International journal of computer vision* 59.2 (2004), pp. 167–181.

[Fig19] B. Figliuzzi. “Eikonal-based models of random tessellations”. In: *Image Analysis and Stereology* 38.1 (2019), pp. 15–23.

[FCF17] B. Figliuzzi, K. Chang, and M. Faessel. “Hierarchical segmentation based upon multi-resolution approximations and the watershed transform”. In: *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. Springer. 2017, pp. 185–195.

[Fig16] B. Figliuzzi et al. “Modelling the microstructure and the viscoelastic behaviour of carbon black filled rubber materials from 3D simulations”. In: *Technische Mechanik* 32.1-2 (2016), pp. 22–46.

[FLZ09] S. Fomel, S. Luo, and H. Zhao. “Fast sweeping method for the factored eikonal equation”. In: *Journal of Computational Physics* 228.17 (2009), pp. 6440–6455.

[FA15] G. Franchi and J. Angulo. “Bagging Stochastic Watershed on Natural Color Image Segmentation”. In: *Mathematical Morphology and Its Applications to Signal and Image Processing*. Ed. by J. A. Benediktsson et al. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 422–433.

[FVS09] B. Fulkerson, A. Vedaldi, and S. Soatto. “Class segmentation and object localization with superpixel neighborhoods”. In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 670–677.

[Gal03] Galun et al. "Texture segmentation by multiscale aggregation of filter responses and shape elements". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. Nice, France: IEEE, 2003, 716–723 vol.1.

[Gho19] Swarnendu Ghosh et al. "Understanding Deep Learning Techniques for Image Segmentation". In: *arXiv:1907.06119 [cs]* (July 2019).

[Gir19] R. Giraud et al. "Texture-Aware Superpixel Segmentation". In: *arXiv:1901.11111 [cs]* (Jan. 2019).

[Gir14] R. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. June 2014, pp. 580–587.

[Gó19] J. V. Gómez et al. "Fast Methods for Eikonal Equations: An Experimental Survey". In: *IEEE Access* 7 (2019), pp. 39005–39029. ISSN: 2169-3536.

[HTF09] Tr. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2009.

[HG09] H. He and E. A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (Sept. 2009), pp. 1263–1284.

[HMS10] A. Heinle, A. Macke, and A. Srivastav. "Automatic cloud classification of whole sky images". In: *Atmospheric Measurement Techniques* 3.3 (May 2010), pp. 557–567.

[HEH05] Derek Hoiem, Alexei A. Efros, and Martial Hebert. "Geometric Context from a Single Image". In: *Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1 - Volume 01*. ICCV '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 654–661.

[Hsv] *HSL and HSV*. https://en.wikipedia.org/wiki/HSL_and_HSV.

[JF91] A. K. Jain and F. Farrokhnia. "Unsupervised texture segmentation using Gabor filters". In: *Pattern recognition* 24.12 (1991), pp. 1167–1186.

[Jai11] V. Jain et al. "Learning to Agglomerate Superpixel Hierarchies". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS'11. Granada, Spain: Curran Associates Inc., 2011, pp. 648–656.

[Jun07] C. R. Jung. "Combining wavelets and watersheds for robust multiscale image segmentation". In: *Image and Vision Computing*. SIBGRAPI 25.1 (Jan. 2007), pp. 24–33.

[JS05] C. R. Jung and J. Scharcanski. "Robust watershed segmentation using wavelets". In: *Image and Vision Computing* 23.7 (July 2005), pp. 661–669.

[KWT88] M. Kass, A. Witkin, and D. Terzopoulos. "Snakes: Active contour models". In: *International Journal of Computer Vision* 1.4 (1988), pp. 321–331.

[KK03] J. B. Kim and H. J. Kim. “Multiresolution-based watersheds for efficient image segmentation”. In: *Pattern Recognition Letters* 24.1-3 (Jan. 2003), pp. 473–488.

[Kru56] Joseph B. Kruskal. “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem”. In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50.

[Lab] *Lab color space*. https://en.wikipedia.org/wiki/CIELAB_color_space.

[LM98] T. Leung and J. Malik. “Contour continuity in region based image segmentation”. In: *Computer Vision — ECCV’98*. Ed. by H. Burkhardt and B. Neumann. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, pp. 544–559.

[Lev09] A. Levinshtein et al. “Turbopixels: Fast superpixels using geometric flows”. In: *IEEE transactions on pattern analysis and machine intelligence* 31.12 (2009), pp. 2290–2297.

[LLY11] Q. Li, W. Lu, and J. Yang. “A Hybrid Thresholding Algorithm for Cloud Detection on Ground-Based Color Images”. In: *Journal of Atmospheric and Oceanic Technology* 28.10 (May 2011), pp. 1286–1296. (Visited on 04/24/2019).

[LC15] Z. Li and J. Chen. “Superpixel segmentation using Linear Spectral Clustering”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 1356–1363.

[Lin14] T. Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[Liu11] M. Y. Liu et al. “Entropy rate superpixel segmentation”. In: *CVPR 2011*. June 2011, pp. 2097–2104.

[Liu15] S. Liu et al. “Automatic Cloud Detection for All-Sky Images Using Superpixel Segmentation”. In: *IEEE Geoscience and Remote Sensing Letters* 12.2 (Feb. 2015), pp. 354–358.

[LST16] T. Liu, M. Seyedhosseini, and T. Tasdizen. “Image Segmentation Using Hierarchical Merge Tree”. In: *IEEE Transactions on Image Processing* 25.10 (Oct. 2016), pp. 4596–4607.

[Lon06] C. N. Long et al. “Retrieving Cloud Characteristics from Ground-Based Daytime Color All-Sky Images”. In: *Journal of Atmospheric and Oceanic Technology* 23.5 (May 2006), pp. 633–652.

[LSD15] J. Long, E. Shelhamer, and T. Darrell. “Fully convolutional networks for semantic segmentation”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 3431–3440.

[Low04] D. G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110.

[Lux07] U. von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and Computing* 17.4 (Dec. 2007), pp. 395–416.

[MDW14] V. Machairas, E. Decenciere, and T. Walter. “Waterpixels: Superpixels based on the watershed transformation”. In: *2014 IEEE International Conference on Image Processing (ICIP)*. Oct. 2014, pp. 4343–4347.

[Mac15] V. Machairas et al. “Waterpixels”. In: *IEEE Transactions on Image Processing* 24.11 (2015), pp. 3707–3716.

[Mal01] J. Malik et al. “Contour and texture analysis for image segmentation”. In: *International journal of computer vision* 43.1 (2001), pp. 7–27.

[MSV95] R. Malladi, J. A. Sethian, and B. C. Vemuri. “Shape modeling with front propagation: a level set approach”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.2 (Feb. 1995), pp. 158–175.

[Mal99] S. Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[MH14] F. Malmberg and Cris L L. Hendriks. “An efficient algorithm for exact evaluation of stochastic watersheds”. In: *Pattern Recognition Letters* 47 (2014), pp. 80–84.

[MLN09] F. Malmberg, J. Lindblad, and I. Nyström. “Sub-pixel segmentation with the image foresting transform”. In: *International Workshop on Combinatorial Image Analysis*. Springer. 2009, pp. 201–211.

[MB05] B. Marcotegui and S. Beucher. “Fast Implementation of Waterfall Based on Graphs”. In: *Mathematical Morphology: 40 Years On*. Ed. by Christian Ronse, Laurent Najman, and Etienne Decencière. Computational Imaging and Vision. Springer Netherlands, 2005, pp. 177–186.

[MM97] B. Marcotegui and F. Meyer. “Segmentation de bas en haut de sÉquences d’images en vue du codage”. In: *Annales Des Télécommunications* 52.7 (July 1997), pp. 397–407.

[Mar01] D. Martin et al. “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics”. In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 2. July 2001, 416–423 vol.2.

[MFM04] D. R. Martin, C. C. Fowlkes, and J. Malik. “Learning to detect natural image boundaries using local brightness, color, and texture cues”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.5 (May 2004), pp. 530–549.

[Mei03] M. Meilă. “Comparing Clusterings by the Variation of Information”. In: *Learning Theory and Kernel Machines*. Ed. by Bernhard Schölkopf and Manfred K. Warmuth. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 173–187.

[Mey98] F. Meyer. “From connected operators to levelings”. In: *Computational Imaging And Vision* 12 (1998), pp. 191–198.

[Mey04] F. Meyer. “Levelings, Image Simplification Filters for Segmentation”. In: *Journal of Mathematical Imaging and Vision* 20.1 (Jan. 2004), pp. 59–72. (Visited on 09/24/2019).

[Mey15] F. Meyer. “Stochastic watershed hierarchies”. In: *2015 Eighth International Conference on Advances in Pattern Recognition (ICAPR)*. 2015, pp. 1–8.

[MB90] F. Meyer and S. Beucher. “Morphological segmentation”. In: *Journal of Visual Communication and Image Representation* 1.1 (1990), pp. 21 –46.

[MLC86] O. J. Morris, M. d J. Lee, and A. G. Constantinides. “Graph theory for image analysis: an approach based on the shortest spanning tree”. In: *IEE Proceedings F - Communications, Radar and Signal Processing* 133.2 (Apr. 1986), pp. 146–152.

[NP12] P. Neubert and P. Protzel. “Superpixel benchmark and comparison”. In: *Forum Bildverarbeitung 2010*. 2012, pp. 205–218.

[NI13] J. Nunez-Iglesias et al. “Machine Learning of Hierarchical Clustering to Segment 2D and 3D Images”. In: *PLOS ONE* 8.8 (Aug. 2013), e71715.

[PD07] S. Paris and F. Durand. “A Topological Approach to Hierarchical Segmentation using Mean Shift”. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. June 2007, pp. 1–8.

[Pas07] N. Passat et al. “Watershed and multimodal data for brain vessel segmentation: Application to the superior sagittal sinus”. In: *Image and Vision Computing* 25.4 (2007), pp. 512–521.

[Ped11] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[PZZ13] B. Peng, L. Zhang, and D. Zhang. “A survey of graph theoretical approaches to image segmentation”. In: *Pattern Recognition* 46.3 (Mar. 2013), pp. 1020–1038.

[Ran71] W. M. Rand. “Objective Criteria for the Evaluation of Clustering Methods”. In: *Journal of the American Statistical Association* 66.336 (1971), pp. 846–850.

[RFM05] X. Ren, C. C. Fowlkes, and J. Malik. “Scale-invariant contour completion using conditional random fields”. In: *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*. Vol. 2. 2005, pp. 1214–1221.

[RM03] X. Ren and J. Malik. “Learning a classification model for segmentation”. In: *Proceedings Ninth IEEE International Conference on Computer Vision*. Oct. 2003, 10–17 vol.1.

[RS13] Z. Ren and G. Shakhnarovich. “Image Segmentation by Cascaded Region Agglomeration”. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition.* June 2013, pp. 2011–2018.

[Rez00] M.R. Rezaee et al. “A multiresolution image segmentation technique based on pyramidal segmentation and fuzzy clustering”. In: *IEEE Transactions on Image Processing* 9.7 (July 2000), pp. 1238–1248.

[SFS12] A. Schick, M. Fischer, and R. Stiefelhagen. “Measuring and evaluating the compactness of superpixels”. In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012).* Nov. 2012, pp. 930–934.

[Set96] J. A. Sethian. “A fast marching level set method for monotonically advancing fronts”. In: *Proceedings of the National Academy of Sciences* 93.4 (1996), pp. 1591–1595.

[Set99a] J. A. Sethian. “Fast marching methods”. In: *SIAM review* 41.2 (1999), pp. 199–235.

[Set99b] J. A. Sethian. *Levet set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science.* Vol. 3. Cambridge University Press, 1999.

[SBB00] E. Sharon, A. Brandt, and R. Basri. “Fast multiscale image segmentation”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662).* Vol. 1. June 2000, 70–77 vol.1.

[SBB01] E. Sharon, A. Brandt, and R. Basri. “Segmentation and boundary detection using multiscale intensity measurements”. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001.* Vol. 1. Dec. 2001, pp. I–I.

[She16] J. Shen et al. “Real-Time Superpixel Segmentation by DBSCAN Clustering Algorithm”. In: *IEEE Transactions on Image Processing* 25.12 (Dec. 2016), pp. 5933–5942.

[SM97] J. Shi and J. Malik. “Normalized cuts and image segmentation”. In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition.* June 1997, pp. 731–737.

[SM00] J. Shi and J. Malik. “Normalized cuts and image segmentation”. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905.

[Skl78] J. Sklansky. “Image Segmentation and Feature Extraction”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 8.4 (1978), pp. 237–247.

[Soi13] P. Soille. *Morphological Image Analysis: Principles and Applications.* Springer Science & Business Media, Mar. 2013.

[SHL17] D. Stutz, A. Hermans, and B. Leibe. “Superpixels: An Evaluation of the State-of-the-Art”. In: *Computer Vision and Image Understanding* (Apr. 2017).

[Sze11] R. Szeliski. “Segmentation”. In: *Computer Vision*. Texts in Computer Science. Springer London, 2011, pp. 235–271.

[Urq82] R. Urquhart. “Graph theoretical clustering based on limited neighbourhood sets”. In: *Pattern Recognition* 15.3 (Jan. 1982), pp. 173–187.

[VS91] L. Vincent and P. Soille. “Watersheds in digital spaces: an efficient algorithm based on immersion simulations”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13.6 (June 1991), pp. 583–598.

[Wal14] S. van der Walt et al. “scikit-image: image processing in Python”. In: *PeerJ* 2 (June 2014), e453.

[WLW01] J. Z. Wang, J. Li, and G. Wiederhold. “SIMPLIcity: semantics-sensitive integrated matching for picture libraries”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 23.9 (2001), pp. 947–963.

[Wan17] M. Wang et al. “Superpixel segmentation: A benchmark”. In: *Signal Processing: Image Communication* 56 (Aug. 2017), pp. 28–39.

[Wan11] S. Wang et al. “Superpixel tracking”. In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 1323–1330.

[WL93] Z. Wu and R. Leahy. “An optimal graph theoretic approach to data clustering: theory and its application to image segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.11 (Nov. 1993), pp. 1101–1113.

[XGZ17] X. Xiao, Y. Gong, and Y. Zhou. “Adaptive superpixel segmentation aggregating local contour and texture features”. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mar. 2017, pp. 1902–1906.

[XT15] S. Xie and Z. Tu. “Holistically-Nested Edge Detection”. In: *arXiv:1504.06375 [cs]* (Apr. 2015).

[XM08] X. Xie and M. Mirmehdi. “A Galaxy of Texture Features”. In: *Handbook of Texture Analysis*. Imperial College Press, 2008. Chap. 13, pp. 375–407.

[Yan15] J. Yan et al. “Object detection by labeling superpixels”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 5107–5116.

[YLY14] F. Yang, H. Lu, and M. Yang. “Robust Superpixel Tracking”. In: *IEEE Transactions on Image Processing* 23.4 (Apr. 2014), pp. 1639–1651.

[Zah71] C. T. Zahn. “Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters”. In: *IEEE Transactions on Computers* C-20.1 (Jan. 1971), pp. 68–86.

[Zha04] H. Zhao. “A Fast Sweeping Method for Eikonal Equations”. In: *Mathematics of computation* 74.250 (2004), pp. 603–627.

[ZK07] C. L. Zitnick and S. Kang. “Stereo for image-based rendering using image over-segmentation”. In: *International Journal of Computer Vision* 75.1 (2007), pp. 49–65.

RÉSUMÉ

La présente thèse vise à développer une méthodologie générale basée sur des méthodes d'apprentissage pour effectuer la segmentation d'une base de données constituée d'images similaires, à partir d'un nombre limité d'exemples d'entraînement. Cette méthodologie est destinée à être appliquée à des images recueillies dans le cadre d'observations de la terre ou lors d'expériences menées en science des matériaux, pour lesquelles il n'y a pas suffisamment d'exemples d'entraînement pour appliquer des méthodes basées sur des techniques d'apprentissage profond.

La méthodologie proposée commence par construire une partition de l'image en superpixels, avant de fusionner progressivement les différents superpixels obtenus jusqu'à l'obtention d'une segmentation valide. Les deux principales contributions de cette thèse sont le développement d'un nouvel algorithme de superpixel basé sur l'équation eikonale, et le développement d'un algorithme de fusion de superpixels basé sur une adaptation de l'équation eikonale au contexte des graphes. L'algorithme de fusion des superpixels s'appuie sur un graphe d'adjacence construit à partir de la partition en superpixels. Les arêtes de ce graphe sont valuées par une mesure de dissimilarité prédite par un algorithme d'apprentissage à partir des caractéristiques de bas niveau calculées sur les superpixels.

A titre d'application, l'approche de segmentation est évaluée sur la base de données SWIMSEG, qui contient des images de nuages. Pour cette base de données, avec un nombre limité d'images d'entraînement, nous obtenons des résultats de segmentation similaires à ceux de l'état de l'art.

MOTS CLÉS

Segmentation d'image, superpixel, apprentissage automatique, équation eikonale, fast marching, graphe.

ABSTRACT

In this PhD thesis, our aim is to establish a general methodology for performing the segmentation of a dataset constituted of similar images with only a few annotated images as training examples. This methodology is directly intended to be applied to images gathered in Earth observation or materials science applications, for which there is not enough annotated examples to train state-of-the-art deep learning based segmentation algorithms.

The proposed methodology starts from a superpixel partition of the image and gradually merges the initial regions until an actual segmentation is obtained. The two main contributions described in this PhD thesis are the development of a new superpixel algorithm which makes use of the Eikonal equation, and the development of a superpixel merging algorithm steaming from the adaption of the Eikonal equation to the setting of graphs. The superpixels merging approach makes use of a region adjacency graph computed from the superpixel partition. The edges are weighted by a dissimilarity measure learned by a machine learning algorithm from low-level cues computed on the superpixels.

In terms of application, our approach to image segmentation is finally evaluated on the SWIMSEG dataset, a dataset which contains sky cloud images. On this dataset, using only a limited amount of images for training our algorithm, we were able to obtain segmentation results similar to the ones obtained with state-of-the-art algorithms.

KEYWORDS

Image segmentation, superpixel, machine learning, Eikonal equation, fast marching, graph.