UiO : Department of Informatics
University of Oslo

Object Recognition and Segmentation of Wounds

Robin Wåsjø

Master's Thesis Spring 2015

Object Recognition and Segmentation of Wounds

Robin Wåsjø

1st February 2015

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo
Oslo, Norway

The Interventional Centre
Oslo University Hospital, Rikshospitalet
Faculty of Medicine
University of Oslo
Oslo, Norway

2015

Object Recognition and Segmentation of Wounds

Robin Wåsjø

https://www.duo.uio.no/

Print: University Print Centre, University of Oslo

Abstract

Object recognition and segmentation of objects is a complex task. Our goal is to develop an algorithm that can recognize and segment wound objects in images. We attempt to solve the object recognition and segmentation problem by using a hypothesis optimization framework. This method optimizes the object segmentation by assigning objective function values to the object segmentation hypotheses. The optimization algorithm is a genetic algorithm. The objective function relies on textural and shape properties, and the textural properties relies on classification of superpixel-segments and superpixel-edges within wound images. Superpixel-segments and superpixel-edges within the same image are dependent samples. We use combined hyperparameter and feature selection methods to train classification models, and we evaluate the impact of dependent samples on these methods. To our knowledge, no study has evaluated model-selection methods when the data contains known groups of dependent samples. Our results confirm that dependent samples results in biased error estimates. Biased error estimates can cause suboptimal feature and hyperparameter selections, and therefore reduce the classification performance. Finally, we obtain promising results by using hypothesis optimization to solve object recognition and segmentation of wounds. These results are important because of the flexible nature of hypothesis optimization; they demonstrate that hypothesis optimization is a strong candidate for general-purpose machine-learnable object recognition and segmentation.

Sincere thanks to my family, friends, and thesis advisors,

Ole Jacob Elle, Kim Mathiassen, and Mats Erling Høvin.

Contents

List of Figures, Tables, and Algorithms

1 Introduction

1.1 On the Recognition and Segmentation of Wounds

The challenge underlying the wound recognition topic is the creation of an autonomous robot operating system. One of the many requirements for an autonomous robot operating system is being able to recognize and segment wounds, particularly for wound closure tasks. Having an accurate description of the wound boundary is essential for being able to plan where the place the stitches, staples, glue strips, or whatever else the system would use for wound closure.

Solving object recognition for wounds closely aligns with solving the problem of object recognition in general. While there are some successful uses of object recognition in machines, it is by far, inferior to their biological counterparts in humans and other animals. Therefore, object recognition remains a largely unsolved problem. While I focus on object recognition for wounds, I have attempted solve the problem by using an approach applicable to a large variety of objects. Any wound object recognition algorithm heavily relying on certain wound-specific cues, will probably have counterexamples. Therefore, I believe it is important to focus on a more general-purpose object recognition algorithm.

It is important to note the distinction between object recognition, where we acquire a segmentation of the object, and object classification. In the most basic case, an object classification task consists of predicting whether the image contains an object. We can describe this simple classification task as a function mapping the high dimensional image to a single binary output. For, the segmentation task, we have a function mapping the high dimensional image to some high dimensional segmentation description. Obviously, this is a more complex task.

Figure 1-1 shows a process flow diagram of the object recognition and segmentation method proposed in this thesis. Later in this thesis, I show diagrams that are more detailed. The algorithm consists of a training part, and a prediction part. By omitting the training part, the algorithm consists of the following steps:

1. Partition the image into small homogenous segments, also called superpixels.
2. Classify superpixel-based local structures.

3. Use optimization to find the optimal hypothesis. A hypothesis consists of a group of superpixels. The objective function to optimize, considers both the shape and local structures classifications of the hypothesized object.

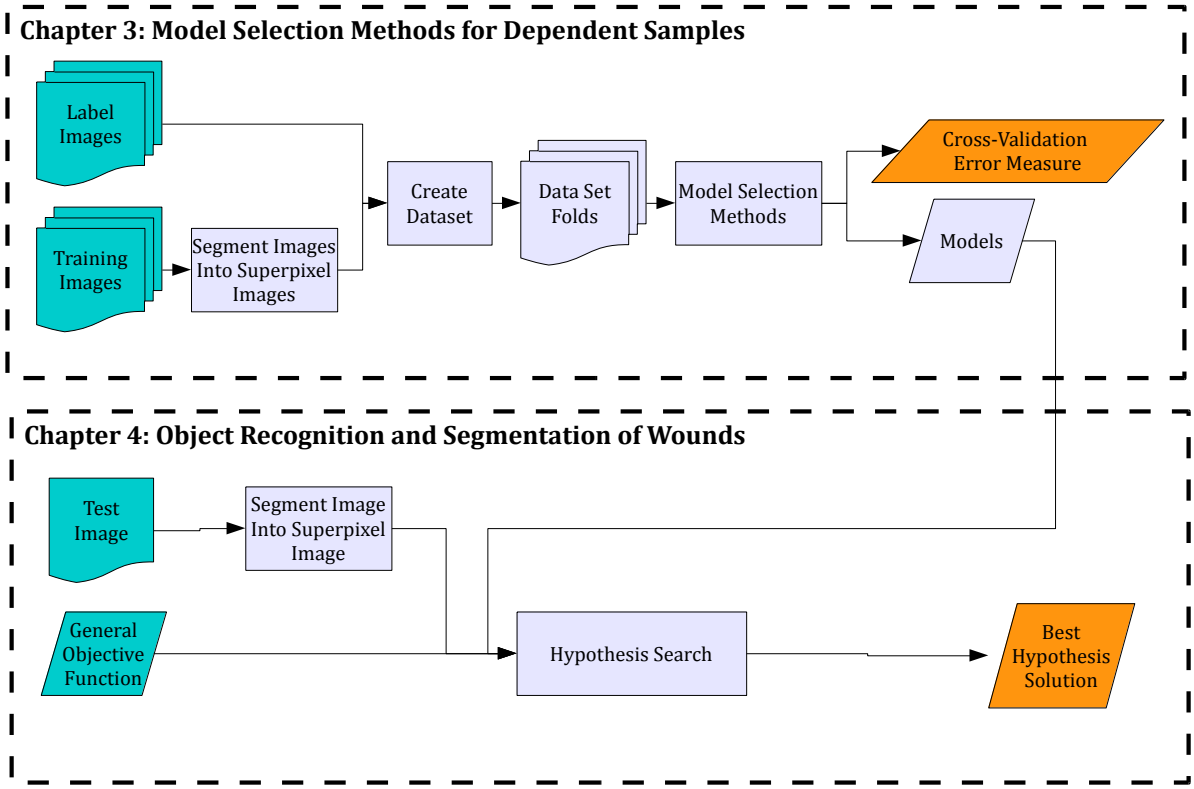

Figure 1-1 Object Recognition and Segmentation Flowchart

The flowchart shows an overview of our approach to recognize and segment wound images. Rectangles represents processing steps. Parallelograms represents objects. Curved Blocks represents data.

A big part of this thesis has been devoted to how we should classify superpixels and superpixel edges. These are local structures in images, and local structures from images of the same wounds are dependent. That is, the samples are dependent, and I study the general topic of model selection methods when faced with dependent samples.

By itself, object recognition of a single wound is of limited value. The autonomous robot operating system must be able to recognize multiple kinds of objects. This is outside the scope of this thesis. However, in section 4.6.5 I outline a multi-object recognition algorithm by using multiple single-object recognition algorithms akin to ours.

1.2 Early Work

Solving the problem of wound recognition and segmentation turned out to be a much bigger challenge than my initial hopes. I will briefly go over some of the methods I tried in the earlier stages of the thesis. My early test procedures were more flawed, and therefore results should be taken with some caution; in particular, the results were biased due to hand-tuning hyperparameters and features. I have used methods not discussed anywhere else in this thesis, but I will keep explanations to a minimum, and instead rely on references. For readers unfamiliar with machine learning and image analysis, it is advisable to read sections 2.6 and 2.8 first.

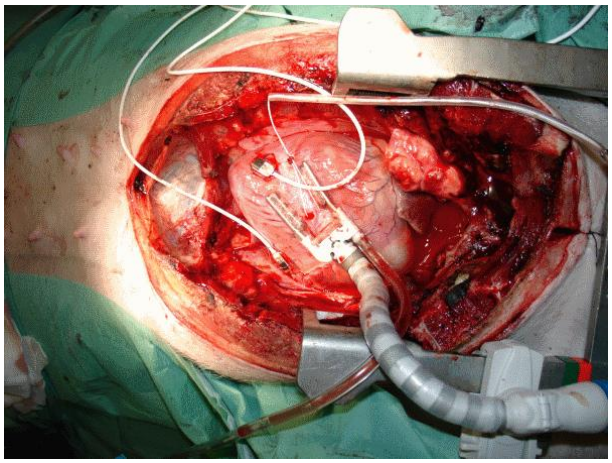

Figure 1-2 Example Wound Image

A complex wound image taken from a pig. The wound is partially occluded with medical instruments. The wound itself contains many types of tissues. Humans have no problem discerning the wound from other elements in the image; yet replicating that with a machine is a difficult task.

Most segmentation techniques would be ill suited for the wound image shown in Figure 1-2. The boundary is complex, and there are other, more pronounced edges in the image. We can find the same or very similar color pixel values in both skin, wound, and other objects. On top of that, the object is partially occluded, yet a human could easily infer the actual boundary of the wound with great accuracy. I knew the methods I used were unable to deal with images of this complexity, but they could work for simple images, and also, I saw no better option at the time.

In the initial approach, I computed the probability of every pixel to belong to a wound. I then used the Chan-Vese segmentation algorithm on the resulting probability image. The initial version of the Chan-Vese algorithm is a region based active contours model, dividing the image into two groups. The algorithm iteratively moves the boundaries of these two groups such that they minimize the internal variance of the

intensity of the image [1]. However, this technique would most likely fail; even for simple images, it had locations were the segmentation edges did not adhere to the wound edge. In an attempt to overcome this, I used the multiphase Chan-Vese segmentation algorithm, which is capable of capturing more details by dividing the image into more than two groups [2].

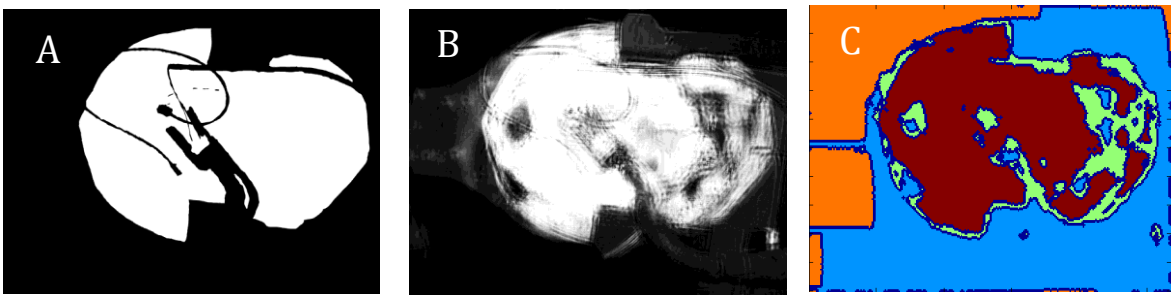

Figure 1-3 Chan-Vese Foreground Background Segmentation
(A) A wound image manually segmented into wound and background. (B) The image shows the probability of pixels to belong to the wound class. (C) The image shows a multiphase Chan-Vese segmentation of the probability image, using four groups.

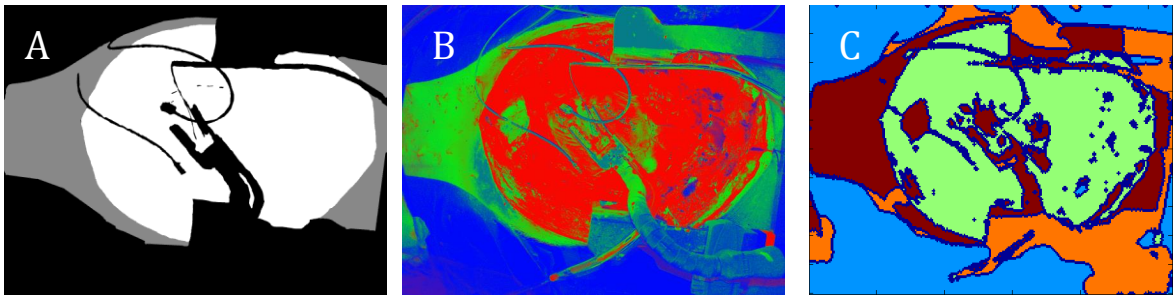

Figure 1-4 Chan-Vese Wound, Skin, and Background Segmentation
(A) A wound image manually segmented into wound, skin, and background. (B) The image shows the probability of pixels to belong to wound, skin or background. The probability of wound, skin, and background are encoded in red, green, and blue respectively. (C) The image shows a multiphase Chan-Vese segmentation of the probability image, using four groups.

In Figure 1-3, we show the correct segmentation, the probability image, and the resulting segmentation of the example pig wound image. The segments does adhere fairly well to the edges, but we now face the problem of correctly selecting which segments belongs to the wound, which is at least as big a challenge. Note that the probability image in Figure 1-3 used the image itself for training the predictor, and therefore the results are greatly optimistically biased.

Among one of the many issues with the previous technique, is that the visual properties of wound and skin are quite similar in comparison to other background objects. This tended to result in a weak edge between wound and skin, and therefore an unreliable segmentation. This led to the method of having a class for both wound, skin and background. The resulting probability image, were a 3-dimensional image containing the probability for a pixel to belong to either wound, skin or background. Figure 1-4 shows the correct segmentation, the probability image, and the resulting segmentation of this technique.

A thing not yet discussed is how I obtained the probability images. To compute the probability of any given pixel, I used a $N \times N$ region around the pixel as features. Furthermore, I used principal component analysis (PCA) to reduce the dimensionality, and then train a multilayer perceptron network using the PCA components as inputs. Principal component analysis uses a set of samples, and picks orthogonal linear projections, by iteratively selecting the component with the largest variance [3]. These orthogonal linear projections corresponds to the eigenvectors of the of the covariance matrix of the data. To evaluate the effectiveness of using PCA, I used the misclassification rate of the neural network as an error measure. A correct classification would be if the correct class and the most probable class were the same. The number of features from a region could be quite high, so using PCA did certainly improve performance over using the raw features, but the simple combination of mean and variance of every color channel had an even better performance than anything PCA could offer; only using the mean were almost as good as using PCA. Furthermore, the three first PCA components were approximately equal to a linear combination of the three, color means of the region. Based on this, the PCA did not appear as anything more than glorified region mean features.

For the three-class scenario in Figure 1-4, the multilayer perceptron network had three outputs, encoding for the probability of wound, skin, and background. To obtain these probability values, I used the softmax activation function in the output layer; this scales the outputs such that the summation of the outputs is one, and the values are between zero and one [4]. We can interpret the corresponding output values as probability values, but I have been unable to verify whether it does indicate probability, or whether it is just a pseudo measure of probability.

In the internal layers, I used the hyperbolic tangent function, and the inputs were scaled to have zero mean and a standard deviation of one. Moreover, I did use

back-propagation algorithm introduced by Rumelhart, Hinton, and Williams; note that their article refers to back-propagation as using the generalized delta rule [5].

I evaluated the usage of Gaussian mixture models with the expectation maximization algorithm as a probabilistic predictor, but the classification results I obtained were subpar, and the algorithm failed to reliably adapt to simple 2-dimensional multi-cluster datasets; Bilmes gives an account for this algorithm [6].

Predicting the probability of a pixel falls in the category of predicting the probability of local structures. However, due to the weak theoretical background and performance of probabilistic predictors, I moved on to using classification instead of probability values; this however may have been an unwise move, as classifying local structures in wound images appears to be of limited value. In section 4.6.5, we discuss using probabilistic classification in future works. Furthermore, we will be using classification of superpixel based local structure instead of classifying pixels as a tool for object recognition and segmentation.

1.3 Thesis Overview

Chapter 2 covers software, algorithms, and theory utilized in this thesis. Chapter 2.1 provides a brief literature review on object recognition and segmentation, which is the main topic of this thesis. Chapter 3 covers model selection methods for the learning problem of classifying local structures within images. Classification of local structures is an essential component of the object recognition and segmentation algorithm presented in this thesis. Chapter 4 builds upon chapter 3, and proposes an algorithm for object recognition and segmentation of wounds. Chapter 5 concludes the thesis. All abbreviations used in this thesis, are listed at the end, in the List of Abbreviations.

Our somewhat unusual thesis structure is because the thesis covers two topics, separated into the third and fourth chapter. Combining the two topics in common method, results and discussion chapters, would result in a less readable thesis. In the current structure, they share background, covered in the second chapter, and the fourth chapter skips content already covered in the third one.

The third and fourth chapters have their own abstracts. The chapter abstracts assumes a greater degree of knowledge of the background material.

2 Background

2.1 A Brief Literature Review on Object Recognition and Segmentation

In the field of computer vision, object recognition is a broad term that can refer to any technique attempting to make some prediction on objects. For instance, it may refer to object classification images containing one prominent object, or object detection, which locates an object within an image.

A primary motivation of segmentation algorithms is to segment objects, but few do directly address that. For instance, Chan Vese segmentation may result in multiple object candidates [1], and watershed segmentation parses the image into multiple segments [7]. In addition, these two algorithms rely on pre-processing of images such that the object is assigned different pixel values than the environment. From the early work discussed in section 1.2, we determined the object segmentation problem to be wholly intertwined and therefore not feasibly solved using standard segmentation algorithms.

Typical of segmentation algorithms, their goal tends to concern segmenting the most pronounced edges, or dissimilar regions in images. A problem is that the objects of interest may not have pronounced edges, or be dissimilar from other regions. Skin and wound textures have similar pixel color values, and therefore they do not have the most pronounced edges, nor the most dissimilar regions.

We have been unable to find robust segmentation algorithms directly applicable to our scenario, although it may just be that they have eluded us. The article by Andreopoulos and Tsotsos [8], and the book by Szeliski [7], are two good sources covering object recognition.

Note that the work by Levinshtein et al. shares similarities with ours [9]. What they refer to as superpixel grouping, is essentially hypothesis optimization on subsets of all superpixels. Their method attempts to find the object with the most distinct edge within an image.

2.2 Software

We have used MATLAB R2013B for all code in this thesis. MATLAB is a high-level language oriented for numerical computing. MATLAB is dynamically typed, which makes it suitable for quickly developing the functions we have had to write. MATLAB already comes with a large portion of the functionality we need, much of it via toolboxes. Most notably for our use, MATLAB has functionality for image processing, image analysis, machine learning, optimization, and creating graph plots.

Additionally, we have used the VLFeat open source library [10]. The library contains implementations for a selection of computer vision algorithms. It is written in C; but it is compatible with interfaces in MATLAB.

2.3 Data Material

Because this thesis attempts to attain recognition and segmentation of wounds, we rely a dataset of wound images. These wound images aid in developing an algorithm, and they provide the required training and testing data for the algorithm.

Unobstructed wound images suitable for this thesis were hard to come by due to restrictions concerning this type of images. Ideally, we should have a large set of wound images captured from different wounds, and from different environments. The images we have been able to acquire, originate from three separate sources. Most of these only partially display a wound, but we still make use of them for predicting the class of superpixels and other local structures.

Eleven images originate from *Trauma.org*. These are wounds resulting from injuries. These images are subject to the *Attribution-NonCommercial-ShareAlike 4.0 International* license [11]. The images from Trauma.org are in Table 2-1. Furthermore, we make use of four images from *Dr. Peter Kim*, which we have listed Table 2-2. These are wounds resulting from surgical incisions. The final image is the image of an incision of a pig, seen in Figure 1-2. We acquired this image from the *Oslo University Hospital, Ullevål*.

Table 2-1 Trauma.org Wound Images
The table lists all images from Trauma.org that we use in this thesis. The images and label images are identifiable by their unique image names. These images are in the resources attached to this thesis. The reference column references the original source of every image.

Image Name	*Label image name*	*Reference*
trauma1B.jpg	*trauma1BL.png*	*[12]*
trauma3.jpg	*trauma3L.png*	*[13]*
trauma6B.png	*trauma6BL.png*	*[14]*
trauma10B.jpg	*trauma10BL.png*	*[15]*
trauma11B.jpg	*trauma11BL.png*	*[16]*
trauma12B.jpg	*trauma12BL.png*	*[17]*
trauma13B.png	*trauma13BL.png*	*[18]*
trauma17B.png	*trauma17BL.png*	*[19]*
trauma18B.jpg	*trauma18BL.png*	*[20]*
trauma19B.jpg	*trauma19BL.png*	*[21]*
trauma20B.jpg	*trauma20BL.png*	*[22]*

Table 2-2 Dr. Peter Kim Wound Images
The table lists all images from Dr. Peter Kim that we use in this thesis. The images and label images are identifiable by their unique image names. These images are in the resources attached to this thesis. The reference column references the original source of every image.

Image Name	*Label image name*	*Reference*
peterkim_video3_im1.png	*peterkim_video3_im1L.png*	*[23]*
peterkim_video3_im2.png	*peterkim_video3_im2L.png*	*[23]*
peterkim_video3_im3.png	*peterkim_video3_im3L.png*	*[23]*
peterkim_video6_im1.png	*peterkim_video6_im1L.png*	*[24]*

The image resolution ranges from 0.15 to 1.92 megapixels, and the size of the wound portion of the image, also has a large variation. The images denoted by a capital *B* in their name, are cropped versions to balance the relative size of the wounds in the images. Cropping improves computational performance, but it also has implications for the predictive performance. All of these images are available in the data attachment to this thesis.

2.4 Color Representation

In this thesis, we use the RGB color space for feature extraction. The RGB color space simply expresses a pixel value, with its intensity in red, green, and blue. Pixel values of

similarly perceived colors have a low Euclidean distance, but the color space is not perceptually uniform.

In a perceptually uniform distance, Euclidean pixel value differences are proportional to differences perceived by humans. The international committee on calorimetry has defined several color representations, attempting to make them perceptually uniform; of them we will utilize the L^*a^*b (Lab) color space. The L-component of the Lab color space, closely matches human perception of lightness. The two other components express color. Paschos compares the RGB, HSV, and Lab color space for color texture analysis; he uses Gabor filters for feature extraction and a nearest-centroid classifier. The HSV color space has the highest performance, followed by Lab, and then RGB [25].

Drimbarean and Whelan, also compares color spaces to extract features from, namely, RGB, Lab, HSI, CIE-XYZ, and YIQ. None of the color spaces proved sufficiently superior [26].

2.5 Statistical Theory

This section briefly describes some of the basic statistical theory used. Books for introductory statistical courses usually covers these topics. We use book [27], as a reference for section 2.5.1, 2.5.2, and 2.5.2.

2.5.1 Dependent Random Variables

Let $X = [x_1, x_2, \dots, x_N]$ be a multivariate random variable with a continuous distribution. Two random variables x_i and x_j $(x_i, x_j \in X)$ are independent if and only if the following equation holds:

$$\begin{bmatrix} f_{x_i}(y) \\ f_{x_j}(y) \end{bmatrix} = \begin{bmatrix} f_{x_i}(y|x_j) \\ f_{x_j}(y|x_i) \end{bmatrix}, \qquad \forall y \in \mathbb{R} \tag{2.1}$$

Here, f_{x_i}, and f_{x_j} are probability density functions of x_i, and x_j. The unconditional probability density functions are called marginal distributions functions. Note that we could let x_i and x_j be multivariate random variables themselves, and therefore subsets of X $(x_i, x_j \subseteq X)$. Equation (2.1) would then refer to independence between these two subsets.

2.5.2 Moments

Let X be a random variable, and let μ_X be its expected value (mean). Expected values of powers of $X - \mu_X$ are called central moments. In other words, central moment K is:

$$E[(X - \mu_X)^K] \tag{2.2}$$

The second central moment is called variance. It is a measure of dispersion. The standard deviation ($SD(X)$, or σ_X) is the square root of the variance. The third central moment is a measure non-symmetry, but it is scale independent. We obtain scale independence by dividing the third central moment with σ_X^3. Kurtosis is the fourth central moment divided by σ_X^4. In other words these third, fourth, fifth … order statistics are:

$$E[(X - \mu_X)^K]/\sigma_X^K \tag{2.3}$$

These statistics provides some information on the distribution of X. The correlation coefficient of two random variables (X and Y), is a statistic for how strongly they are dependent. The correlation coefficient is:

$$\rho_{X,Y} = \frac{\sigma_{X,Y}}{\sigma_X \sigma_Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

The correlation coefficient is more formally known as the Pearson product-moment correlation coefficient.

2.5.3 Sampling Distribution of Statistics

Let $X_1, X_2, \ldots, X_n$ be a random sample from a distribution with μ_X and σ_X for mean and standard deviation. Let $\hat{\mu}_X$ be the sample mean. Then:

$$E[\hat{\mu}_X] = \mu_X \tag{2.4}$$

$$SEM(X) = SD(\hat{\mu}_X) = \sigma_X/\sqrt{n} \tag{2.5}$$

It is common to refer to $\sigma_{\bar{X}}$ as the standard error of the mean (SEM). This prevents confusion between the standard deviation (σ_X) and the standard deviation of the mean ($SD(\hat{\mu}_X)$).

An alternative to the standard error is to use confidence intervals. Due to the central limit theorem, the standard error of the mean (given a sufficient sample) is approximately normally distributed. The 95% confidence interval, assuming a normal distribution is $\pm 1.96 \times SEM(X)$. Note that the sample mean, is only approximately normally distributed, and furthermore we only have an estimate to its variance.

Therefore, there is risk that a normal distribution assumption is optimistic. For small sample sizes, we use the percentiles (critical values) of the t-distribution. The t-distribution is sample size dependent, assuming larger tails for smaller sample sizes. For instance, a sample size of $n = 30$ ($df = 29$), and a 95% (two-sided $\alpha = 0.05$) confidence interval is $\pm 2.05 \times SEM(X)$.

2.5.4 Monte Carlo Method

Monte Carlo Methods (or experiments, or simulations) are a class of algorithms that performs repeated random sampling to obtain numerical results. For example, in chapter 3 we train classifiers with small sampled datasets. For each run, the dataset is different, and hence the error values vary. By performing multiple experiments where we train classifiers on a sampled dataset, we can obtain statistics for the error values.

2.6 Image Segmentation

2.6.1 Overview

Image segmentation partitions an image into sets of pixels, called segments or superpixels. Image segmentation attempts to simplify an image into something that is easier to analyze, compared to the pixel value array of the image. Image segmentation is commonly associated with the task of separating an object from its surroundings. The result of an image segmentation may be in the form of a single region that supposedly corresponds to the object of interest, or a label for every pixel on whether they belong the objet.

Image segmentation is not limited to finding a single object, nor a binary distinction between object and non-object (foreground/background segmentation is a common binary segmentation problem). A common topic in image segmentation is the detection and localization of boundaries in natural scenes, without any prior information about the particular scene [28]. The Berkeley Segmentation dataset and benchmark [29] enables the comparison of this type of algorithms. The benchmark relies on ground truths by human subjects, on what is and what is not a boundary.

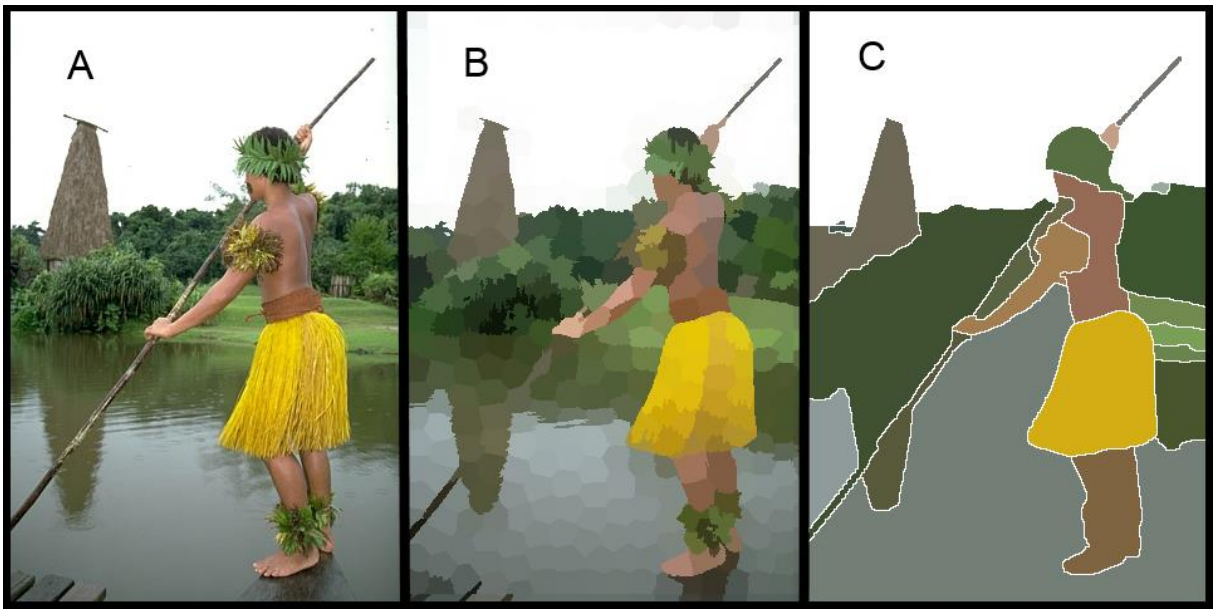

Figure 2-1 Segmentation Comparison

(A) Original Image. (B) Image segmented into 443 segments using SLIC, a superpixel segmentation algorithm. (C) Image segmented into 23 segments using "Automatic Segmentation" by Berkeley [30].

A family of segmentation algorithms, often denoted as superpixel algorithms partitions the image into in a large number of smaller segments [31,32,33]. Figure 2-1 illustrates a comparison of conventional segmentation and superpixel segmentation algorithms. The idea behind superpixel segmentation is to capture all structures with a spatial frequency above the region size of the individual segments. For example, the over-segmentation in Figure 2-1.B would not be able to separate the leaf-sized structures, but it does separate all individual pieces of clothing, and the building in the background. Earlier, we said segmentation partitions an image into segments. Alternatively, we could regard segmentation as grouping pixels we believe belongs to the same object. A superpixel algorithm employs a conservative grouping of pixels, thereby reducing its risk of incorrectly grouping together pixels that belongs to separate objects. Furthermore, due to the small superpixel segment sizes, any erroneous segmentation is contained to be within a small area.

2.6.2 Superpixels from Simple Linear Iterative Clustering

R. Achanta et al [33], introduced the Simple Linear Iterative Clustering (SLIC) algorithm. SLIC is a segmentation algorithm that partitions the image into a large

number of segments. These segments carry more information than individual pixels, and adhere better to edges than rectangular blocks hence the name superpixels.

A color image pixel has five values. Two space coordinates, and three color space values. SLIC is a specialized k-means algorithm [34] that finds clusters of pixels in this 5-dimensional space. The algorithm has two parameters, a superpixel size parameter, S, and a regularization parameter that weights the importance of color vs position.

The initialization places cluster centers in a grid, using S as the grid step. The clusters are relocated to the lowest gradient position in a 3×3 neighborhood. This is to prevent initialization on edge, which may be an undesired equilibrium. The regularization parameter can ensure that most superpixels are compact and of similar size, by putting a larger emphasis on the spatial coordinates.

After initialization, SLIC moves the cluster centers iteratively, where each iteration assigns pixels to the closest cluster center, and relocates the cluster center to the mean of these pixels. This iterative procedure is identical to the k-means algorithm. The difference is that SLIC only computes the distance to the pixels that are within a $2S \times 2S$ region (in spatial coordinates) of the initial cluster center locations. For N pixels, k clusters, and I iterations, the basic k-means algorithm has a $O(kNI)$ complexity. S relates to the number of clusters and pixels by $S \approx \sqrt{N/k}\, m$, and S^2 is the number of pixels evaluated per cluster. The SLIC complexity is $O(kS^2 I) \approx O(NI)$, meaning it is independent of the number of clusters.

The article claims a rule of using ten iterations suffices for most images. Furthermore, Ren and Reid [35] uses a GPU implementation of SLIC in a technical report, showing large performance gains. For the GPU implementation, a 1280×960 image clustered with 256 clusters takes 86ms, whereas the sequential implementation uses 1522ms.

2.7 Superpixel Feature Extraction

2.7.1 Overview

Classifying the class of superpixels (local regions or segments) may aid the task of object recognition. The object recognition task would be trivial if we could classify segments with 100% accuracy. We can classify superpixels by extracting a fixed set of

features from them. We are also interested in classifying edges between two superpixels, in which case we can use features from both segments. Possibly, the simplest feature one can think of is the superpixel mean color value; the remaining features are described in section 2.7.2-2.7.6. The features are primarily defined for single-channel images (greyscale images). We extend these features to color images, by extracting the same feature from all three channels separately. We call this set of three features, a *feature group*.

Drimbarean and Whelan [26] compares several methods to classify $N \times N$ regions of color texture images. They investigate local linear transforms, Gabor filtering, and co-occurrence. These methods encode spatial information. The linear transform had the highest predictive performance, followed by Gabor filter and then Co-occurrence. Among these methods, we have used Co-occurrence matrices, because the method has a simple generalization for non-rectangular regions. Concerning their method, they use the same images for classification and testing. This could have affected the conclusions.

2.7.2 Moments

A simple approach to texture description is to use statistical features that describe the distribution of the pixel color values such as mean, variance, covariance, skewness and kurtosis. We defined these statistics in section 2.5.2, but here we go in further detail, and specifically describe them for color images. These features are also known as histogram moments, because we could have derived them from the pixel histogram of the superpixel.

We assume RGB images, but the same approach applies to other color or multispectral spaces. Let R, G and B be vectors comprising the red green and blue pixel color values of a superpixel, where $= [r_1, r_2, \cdots, r_n]^T$, $G = [g_1, g_2, \cdots, g_n]^T$, $B = [b_1, b_2, \cdots, b_n]^T$, and n is the number of pixels in the superpixel. Additionally, let $X = [x_1, x_2, \cdots, x_n]^T$, and $Y = [y_1, y_2, \cdots, y_n]^T$ refer to any of the three colors. As there are three colors, there are three mean, standard deviation, skewness and kurtosis measures. They are defined as follows:

$$E[X] = \mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.6}$$

$$SD(X) = \sigma_X = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_X)^2} \tag{2.7}$$

$$Skewness(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_X)^3}{\sigma_X^3} \tag{2.8}$$

$$Kurtosis(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_X)^4}{\sigma_X^4} \tag{2.9}$$

The correlations might also be of interest. The Correlation between two different colors are:

$$Corr(X,Y) = \rho_{X,Y} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu_X)(y_i - \mu_Y)}{\sigma_X \sigma_Y} \tag{2.10}$$

Lastly, we could have considered other image moments, such as coskewness and cokurtosis moments.

2.7.3 Entropy

In addition to moments, entropy is another useful feature based on the color intensity histograms. Entropy is a measure of randomness, introduced by Shannon (1948) [36]. Let $p(z)$ be the probability of intensity value z, of color X, of the pixels in a superpixel. Further, let $\mathcal{M}$ be the set of all intensity values with non-zero probability:

$$\mathcal{M} = \{x \in \mathcal{M}_L | p(x) \neq 0\}, \qquad \mathcal{M}_L = \{0,1, \dots, L\} \tag{2.11}$$

Then, the entropy is:

$$entropy(X) = -\sum_{z \in \mathcal{M}} p(z)\log_2(p(z)) \tag{2.12}$$

The reason we ignore zero probability values, is because $\log_2(0)$ is undefined.

2.7.4 Sobel Mean

When classifying superpixel edges, the image gradient at the superpixel edge may be of value. Therefore, we have included a feature group based on the Sobel operator. The Sobel operator, named after Irwin Sobel, provides an approximation for the image gradient magnitude. We apply the Sobel operator on all three-color channels, and take

the mean value of all pixels on the edge between the two superpixels. We call this the Sobel mean feature group, and it comprises three features.

The Sobel operator convolves two 3×3 kernels with a color channel of the image, resulting in the horizontal and vertical gradient approximations, G_x and G_y. For a single-channel image I, these are defined as:

$$G_{horz} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I, \qquad G_{vert} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I \tag{2.13}$$

The gradient magnitude, for any point (x, y) in the image, is:

$$G(x,y) = \sqrt{G_{horz}(x,y)^2 + G_{vert}(x,y)^2} \tag{2.14}$$

In a comparison of image detection techniques, the Sobel operator is described as inaccurate and sensitive to noise [37]. The article recommends the Canny edge detection algorithm. However, it is important to note that our scenario differs. The superpixel edges already corresponds well to real edges, filtering out most of the false edges. In addition, using the mean of all values across the superpixel edge, adds robustness to noise. Preliminary testing by visual inspection of the Canny, Laplacian of Gaussian, Robert, Prewitt, and Sobel techniques, indicated the Sobel operator to be most suitable for our purposes.

2.7.5 Co-occurrence

Haralick et al. introduced textural features extracted from co-occurrence matrices [38]. Co-occurrence features are frequently called GLCM features, as an abbreviation for gray-level co-occurrence matrices. However, the technique is not limited gray-level images. A co-occurrence matrix is the distribution of pixel values co-occurring at a given offset. A co-occurrence matrix for a rectangular $n \times m$ region with intensity values I, and offset $[\Delta x, \Delta y]^T$, is defined as:

$$p_{\Delta x,\Delta y}(i,j) = \sum_{p=1}^{n-\Delta x} \sum_{q=1}^{m-\Delta y} \begin{cases} 1, & if\ I(p,q) = i\ and\ I(p+\Delta x, q+\Delta y) = j \\ 0, & otherwise \end{cases} \tag{2.15}$$

For a specific i and j, $p_{\Delta x,\Delta y}(i,j)$ is the probability to go from intensity value i, to intensity value j, when the offset is $[\Delta x, \Delta y]^T$. We discretize the intensity into 16 values.

To extend this definition to segments, we let the region be the bounding box of the segment, and only count pixel pairs where both pixels belong to the segment. An implementation trick is to compute the co-occurrence matrix of the bounding box,

where the values outside the segment is set to negative one. All we now have to do is remove the negative one column and row of $p_{\Delta x,\Delta y}(i,j)$.

Co-occurrence matrices are high dimensional; therefore, we extract a smaller set of features from them. Haralick et al. suggested 14 features, but we will use the four features implemented in MATLAB R2013b. These are:

$$GLCM\ Contrast = \sum_{i,j} |i-j|^2 p(i,j) \tag{2.16}$$

$$GLCM\ Correlation = \sum_{i,j} \frac{(i-\mu_i)(j-\mu_j)p(i,j)}{\sigma_i \sigma_j} \tag{2.17}$$

$$GLCM\ Energy = \sum_{i,j} p(i,j)^2 \tag{2.18}$$

$$GLCM\ Homogeneity = \sum_{i,j} \frac{p(i,j)}{1+|i-j|} \tag{2.19}$$

The MATLAB features are similar to some of the 14 features by Haralick et al. The MATLAB features, corresponds to the first, second, third and fifth Haralick et al. feature, but with some variations. For instance, GLCM Homogeneity is similar to the fifth Haralick et al. feature, defined as:

$$Inverse\ Difference\ Moment = \sum_{i,j} \frac{p(i,j)}{1+(i-j)^2} \tag{2.20}$$

They both measure the distribution's closeness to the GLCM diagonal, but with different weights.

2.7.6 Euclidean Moments

We have defined a set of features, which we have called Euclidean moments. These are features defined for multi-channel images, unlike the previous features that are defined for mono-channel images. Regular moments use the difference from the mean. Euclidean moments use the Euclidean difference from the mean.

Let $X = \left[x_{1,} x_2, \cdots, x_n\right]^T$ refer to the color values of all n pixels, where x_i is a 3×1 vector of all color values of a single pixel. Then, the mean, Euclidean standard deviation, Euclidean skewness, and Euclidean kurtosis are defined as follows:

$$E[X] = \mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.21}$$

$$Euclidean\ SD(X) = \sigma_X = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n} |x_i - \mu_X|^2} \tag{2.22}$$

$$Euclidean\ skewness(X) = \frac{\frac{1}{n}\sum_{i=1}^{n} |x_i - \mu_X|^3}{\sigma_X^3} \tag{2.23}$$

$$Euclidean\ kurtosis(X) = \frac{\frac{1}{n}\sum_{i=1}^{n} |x_i - \mu_X|^4}{\sigma_X^4} \tag{2.24}$$

The Euclidean standard deviation, skewness and kurtosis, are the second, third and fourth order Euclidean moments. Higher order moments are more sensitive to pixels far from the mean. Because we use the Euclidean distance instead of the difference, the Euclidean skewness will always be positive.

Each feature is its own feature group, and these are the only feature groups that have one, contrary to three features.

2.8 Machine Learning

2.8.1 Overview

Machine learning is an academic discipline, concerning the construction and study of systems that can learn from data. It is a multidisciplinary field, building heavily on mathematical theory, statistical theory, and computer science. Its applications vary even more widely, being useful for, forensics, physics, marketing, biology, engineering, and more. In this and the remaining subsections, we will look at utilized machine learning algorithms, and model evaluation procedures.

Supervised learning is the task of learning a model from a data set of input and output values that correctly models the relation between the input and output variables. More formally, given a training set of N input-output pairs $(\vec{x}_1, y_1)$, $(\vec{x}_1, y_1), \cdots, (\vec{x}_N, y_N)$, discover a function $f(\vec{x}) = y$, that minimizes the expected error on new examples.

If the output consists of a finite set of values, we call it a *classification problem* and we want the output to be $\arg\max_{y_j} P(y_j|\vec{x})$. We use the term *regression analysis* or *function approximation* when the output is a number. *Probabilistic classification* falls between classification and regression. It has the same data as a classification problem, but it models the outcome probabilities of the data. In other words, probabilistic classification models $P(y_j|\vec{x})$, where the output(s) are finite and constrained to sum to one. The term *soft classification* has been used in place of probabilistic classification.

The supervised learning problem may also involve multiple labels. Probabilistic classification for instance, has multiple labels in multi-class scenarios, with the constraint that they sum to one.

Unsupervised learning concerns the task of finding structures in a data set without using labels. One use of unsupervised learning is to express the data in a more compact way, either by reducing the dimensionality of the feature space, or to group data together into a set of clusters.

2.8.2 Gaussian Classifier

The Gaussian classifier assumes classes to be distributed according to a normal distribution, determined by the sample mean and sample variance of the class samples. Let μ_n and Σ_n be the mean and covariance of class n, and let the classes be denoted by the numbers $1,2,\dots,N$. Then, the probability density of sample x for class n is:

$$p(x|Class = n) = \frac{1}{\sqrt{(2\pi)^k|\Sigma_n|}} e^{-\frac{1}{2}(x-\mu_n)^T\Sigma_n^{-1}(x-\mu_n)} \tag{2.25}$$

Using this approximation of the probability density, and Bayes' theorem, we can compute the class probability of sample x:

$$p(Class = n, x) = \frac{p(x|Class = n)p(Class = n)}{p(x)} \tag{2.26}$$

The Gaussian classifier selects the most probable class, and given that all classes have the same $p(x)$ fraction, we can omit that from the equations:

$$Predicted\ Class = \underset{n}{\operatorname{argmax}}\{p(x|Class = n)p(Class = n)\} \tag{2.27}$$

The classification accuracy of the Gaussian classifier can be quite good, even if the assumed probability distribution is very inaccurate. However, the Gaussian classifier is unreliable, when the class distribution is made up of multiple clusters.

2.8.3 Gaussian Mixture Model Classifier

In section 3.2.1 we show how our data originates from a small set of sources. This motivated us to test a classifier that modeled the data of every source as a normal distribution. We refer to this as the Gaussian mixture model (GMM) classifier. This is not to be confused with training Gaussian mixture models using the expectation-maximization algorithm. Let the sources be denoted by the numbers $1,2,\dots,S$, and let $\mu_{n,s}$ and $\Sigma_{n,s}$ be the mean and covariance matrix of samples belonging to class n and source s. Then, the probability density of sample x is defined as:

$$p(x|Class = n) = \sum_{s=1}^{S} \frac{p(Source = s)}{\sqrt{(2\pi)^k |\Sigma_{n,s}|}} e^{-\frac{1}{2}(x-\mu_{n,s})^T \Sigma_n^{-1}(x-\mu_{n,s})} \tag{2.28}$$

Otherwise, the GMM classifier uses the same principles as the Gaussian classifier, by using Bayes' theorem to select the most probable class.

2.8.4 K-Nearest Neighbor

The k-nearest neighbor (k-NN) classifier uses the entire training set, and predicts the class of a new sample x, to be the mode of the classes of the k training samples with the shortest distance from x. The mode is a statistical term, and it simply means the most occurring class, of the k samples. To avoid confusing the k hyperparameter with other concepts, we will call it the H_{K-NN} hyperparameter. The k-nearest neighbor classifier can generate highly nonlinear classification results, despite of its simplicity. We use the Euclidean distance as the distance measure. The classification accuracy is highly dependent on the choice of H_{K-NN}; in section 3.3.3, we describe how we have chosen to select this hyperparameter.

Many algorithms use the principle of classifying new samples based on their proximity to the training samples. The most primitive variant is to determine the class of a sample by its single nearest neighbor, also known as the 1-NN algorithm. Our rule of selecting the mode of the k-nearest neighbors may also be known as the voting rule.

Denoeux introduced a k-nearest neighbor classification rule based on Dempster-Shafer theory [39]. He compares it to the voting rule, and the weighted k-nearest neighbor rule. On the two real world, and one artificial datasets, the Dempster-Shafer rule had the highest performance, whereas the voting rule had the lowest performance. Judging by this, we could have seen better results from something else than the voting rule.

2.8.5 Random Forests

The random forest classifier is an ensemble learning method that constructs multiple decision tree classifiers, and outputs the class that is the mode (most occurring class) of the individual classification tree predictions. Breiman developed the algorithm [40]. Classification trees are trained using random subsets of the full data, and randomly selected features. We select the features at random for every decision split. We will refer to the number of features to select by random, as the H_{RF} hyperparameter. A rule of thumb is to set H_{RF} equal to the square root of the number of features, which a study suggest is near-optimal in most cases [2].

2.8.6 Support Vector Machines

Support vector machines are binary discriminative classifiers trained using a fully labeled dataset. Cortis and Vapnik [41] introduced the support vector machines in 1995. The algorithm builds upon the max margin hyperplane algorithm introduced by Vapnik and Lerner in 1963 [42], and other intermediate work. The max margin hyperplane is the hyperplane that maximizes its distance to the closest training vector on either side of the hyperplane. The algorithm has limited use due to the constraint that the data must be linearly separable. I will refer to points in input space or mapped space as vectors, as used by the SVMs article.

SVMs use a soft margin hyperplane, which is a modified version of the max margin hyperplane, allowing some vectors to violate the margin of its class. The soft margin hyperplane minimizes the sum of deviations of these violations, and maximizes the margin for the correctly classified vectors. The soft margin hyperplane enables the algorithm to handle noise and overlap between classes.

SVMs can map input vectors to a very high dimensional space, such that the training data is linearly separable. In particular, SVMs can use the kernel trick, greatly elevating the ability to find complex nonlinear decision boundaries in the input vector space.

SVMs must calculate the distance of a vector to the hyperplane; however, we can express the hyperplane as a linear combination of training vectors. The training vectors that have nonzero weights are called support vectors and to classify a new vector we must calculate the dot products between the new vector and the support vectors. The use of support vectors is what enables SVMs to use the kernel trick. For a maximum margin hyperplane, the support vectors are the training vectors that lies on the margins. For a soft margin hyperplane, the support vectors are the training vectors that either lies on the margins, or violate them.

Although SVMs is a binary classifier, we can extend any binary classifier to classify multiple classes. One strategy is to train a classifier for each class against all the other classes. The classifier that places new data in the best position relative to the boundary labels the new data to its corresponding class.

2.8.7 Kernel Methods

One machine learning technique is to transform the input feature vectors, to another feature space. This enables linear classifiers, such as support vector machines, to form highly nonlinear decision boundaries in the original feature space. We could do this with any classifier. Kernel methods however, only uses the feature vectors in inner products between pairs of samples. This enables the usage of transformations with simple inner products, but very hard to compute transformations. Hofmann et al. published a review on kernel methods in machine learning [43].

The RBF-SVMs, discussed in section 2.8.6, uses the radial basis function kernel. Let x and y be two feature vectors, where $\varphi(x)$, and $\varphi(y)$ are the transformed feature vectors, then the kernel function is:

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle = e^{-\frac{||x-y||^2}{2H_\sigma^2}}$$

The transformed feature vectors have an infinite dimension, making it impossible actually compute the transformed feature vectors. The inner product however, is only

a simple computation using the two input feature vectors. H_σ is a free parameter, and by extension a hyperparameter of the RBF-SVMs classifier.

2.8.8 Model Selection Methods

Learning algorithms, such as support vector machines and k-nearest neighbors have hyper-parameters we must select. In the context of supervised learning, a method that selects hyper-parameters and trains a predictive model based on a dataset; is referred to as a model selection method. In addition, it can include the task of obtaining an estimate for the predictive performance. It should preferably be an unbiased estimate of the predictive performance. Guyon et al. [44] provides an up-to-date (2009) overview on the topic of model selection methods.

A common approach to select hyper-parameters is to select the hyper-parameters that minimizes the cross-validation error. Kohavi [45] compares accuracy estimation of cross-validation and bootstrap; recommending ten-fold stratified cross-validation for model selection (as a rule of thumb). In stratified cross-validation, we balance the number of samples per class for every fold. In this thesis, we always ensure every fold has an equal number of samples per class.

A problem with selecting hyperparameters that minimizes the cross-validation error is that the cross-validation error becomes optimistically biased. However, we can use an external cross-validation loop or a separate test set to estimate the out-of-sample error (predictive performance).

Cawley and Talbot [46] emphasizes the importance of a low variance, model selection criterion, to reduce over-fitting in model selection. One of their demonstrations serves as a good example on the over-fitting of cross-validation. Using a Monte Carlo simulation on a synthetic data set, they demonstrate how a kernel ridge regression classifier, over-fits the hyper-parameters. They tune hyper-parameters using an iterative procedure that minimizes the cross-validation error. The expected value of the cross-validation error is monotonically decreasing. The out-of-sample error on the other hand, first shows a decrease, but then starts to increase after a certain point. This is the exact same behavior typically seen by training neural networks, where the test data monotonically decreases, whereas the out-of-sample error (estimated using a validation set) eventually shifts and begins to increase. The difference is that, instead of over-fitting parameters (perceptron weights) we over-fit

hyperparameters. Additionally, the cross-validation error, rather than the training error shows a monotonic decrease.

The over-fitting cases in the previous paragraph are not surprising; both scenarios attempt to fit some parameters using a predictive performance criterion. To compare, a support vector machine optimizes the parameters (support vectors) that minimizes the training error. Similarly, the hyperparameter optimization optimizes the hyperparameters using the cross-validation error. In other words, the cross-validation error is the training error with respect to the optimization of the hyperparameters.

2.8.9 Feature Selection

Feature selection is a sub-problem of model selection methods. The subset of features we select from the complete feature set is a hyperparameter optimization problem. Expressed more formally, let $N_{features}$ be the number of features, and $P_{features}$ be a $N_{features} \times 1$ binary vector. A true value indicates the corresponding feature is selected, and a false value indicates the corresponding feature is not selected. The optimization problem, consist of selecting the $P_{features}$ value that maximizes the predictive performance. Guyon and Elliseef provides an overview on feature selection [47].

Kohavi and John [48] thoroughly examined using an error estimate (such as cross-validation) of the learning algorithm to search for the optimal features. They referred to this as the *wrapper approach*, and the others as *filter approaches*. Primarily, filter methods are not designed for a single learning algorithm. Note that Whitney [49] had earlier (1971) used a wrapper method to select features with a k-nearest neighbor classifier.

If training the learning algorithm is time consuming, then training the learning algorithm repeatedly for a large set of feature combinations can be too time consuming. Therefore, filter methods may still be the best choice in some scenarios. However, note that we can use a wrapper method for a fast learning algorithm as a feature selection filter for a slow learning algorithm. We could also use a hybrid method; use a filter for feature ranking, and then select the number of features that minimizes the error estimate of the learning algorithm.

For wrapper methods, there are many feature selection search strategies. In fact, we could use any binary optimization algorithm. In this thesis, we will use the greedy

forward selection algorithm. The greedy forward selection algorithm, first used by Whitney [49] for feature selection, consists of iteratively adding the best features. In the context of cross-validation, we first select the best feature according to majority vote of the cross-validation errors. Then we fix this first feature, and repeat the cross-validation for all second feature candidates. Again, we select the best feature according to majority voting. We repeat this until the performance ceases to improve.

On the opposite end from forward selection, we have exhaustive search. Testing all feature combination may be prone to over-fitting. Additionally, it has a bias to select roughly half of the features. For example, there are more ways to select five out of ten features (252) than two out of ten features (45).

The sequential forward floating selection, algorithm by Pudil et al. is popular for feature selection [50]. It follows a repeated include l exclude r ($r < l$) procedure. The algorithm is capable of removing features rendered obsolete due to high dependence with other features, yet it does not drastically increase the computation time.

2.9 Optimization

2.9.1 Overview

An optimization problem is the problem of finding the optimal solution from all candidate solutions. A candidate solution is simply a member of the set of all possible solutions. We make use a function with a real-valued output to describe the quality of a given candidate solution. We call this function, the objective function. It is most common to let lower values indicate better candidate solutions. Let f be the objective function, X^* be the optimal solution, and $\mathcal{A}$ be the set of all candidate solutions. A correct objective function must then satisfy:

$$X^* = \arg\min_{X \in \mathcal{A}} f(X) \tag{2.29}$$

In this thesis, the optimal solution will be the minimum of the objective function. A loss function specifically refers to a function we want to minimize. A function we want to maximize may be called a utility function. Some other names for objective functions are energy functions, fitness functions, and cost functions. To find the optimal solution we employ a search algorithm. Search algorithms that converges to the optimal

solution puts constraints on the objective function, and they restrict the constraints we can set for candidate solutions.

2.9.2 Genetic Algorithm Optimization

A genetic algorithm is a search heuristic, which relies on techniques inspired by natural selection. A genetic algorithm can refer to any algorithm that mimics evolution of a population of candidate solutions. Genetic algorithms have a long history with multiple contributors. We use the version implemented in Matlab R2013B to solve binary optimization problems. Adhering to the Matlab documentation, the genetic algorithm consists of the following steps:

1. Initialize a set, or population, of candidate solutions.
2. Create a sequence, using the population at one iteration, to create the next population. The steps per iteration are:
    a. Compute the objective function value for every candidate solution
    b. Select a subset of parents from the population based on their score.
    c. Select a subset of the highest-scoring (elite) members to be in the next population.
    d. Produce children from the parent's subset, by using mutation and crossover.
    e. Replace non-elite members with children.
3. Stop when a stopping criterion is met.

For a more broad and general overview on genetic algorithms we refer to a book by Mitchell, dedicated to genetic algorithms [51].

3 Model Selection Methods for Dependent Samples

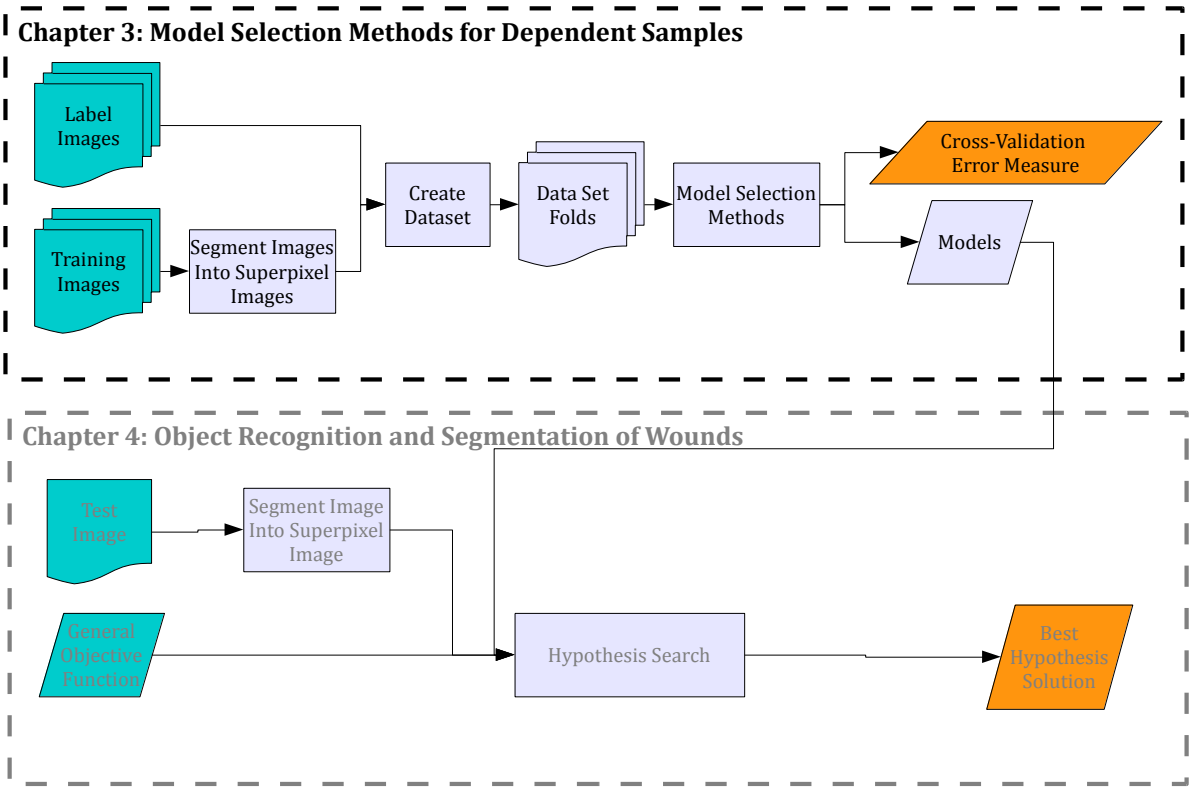

Figure 3-1 Chapter 3 Flowchart

Model selection methods for classifiers encompasses the methods we use to select features and hyperparameters. There are many previous studies on model selection methods. However, our learning problem consists of classifying groups of dependent samples. To the author's knowledge, no study has evaluated model selection methods for this specific learning problem. We evaluate hyperparameter selection for multiple classifiers on a synthetic dataset, and the results shows optimistically biased error estimates when we use dependent samples. Further, we evaluate combined feature and hyperparameter selection on classifying superpixel segments and superpixel edges from a dataset of wound images. These model-selection methods reliably selects rational hyperparameter values and features, and they should be a core part of any complete classification algorithm.

3.1 Introduction

Consider a machine-learning task where we want to create a probability model for the age and gender of drivers passing a waypoint, using only the velocity of the car as a feature. Now, also consider that the data gathering is faulty. On some occasions, the velocity of a car is measured twice. The velocity measurements of these duplicate observations are dependent samples. In our case, we have a small set of wounds, and we want to learn the class membership of local structures in images of these wounds. Figure 3-1 illustrates this learning problem.

Local structures from the same wound are dependent, and therefore our data consists of known groups of dependent samples. Unlike the duplicate observations of car velocities, we actually know which samples are dependent, allowing us to compute unbiased error estimates of classification models.

We will refer to a single wound as a *source*, and the set of local structures within the images of a wound as *samples*. Note that the term *source* has a more general concept than images of a wound. *Sources* relates to the concept of groups with within-group dependency between samples. We use the *source* term to avoid confusion with other concepts that occasionally uses the *group* term. The learning problem, is to learn a model that can classify samples from new sources when the within source samples are dependent.

A complicating factor is that we only have a small set of sources, rendering unreliable error estimates. To improve the reliability of different model-selection method comparisons, we run Monte-Carlo simulations on generated datasets. We define a synthetic two-dimensional data distribution consisting of two parts. The first part is a distribution of the sources. The second part is a within source distribution of the samples.

Our evaluation of model selection methods, are split into two parts. The first part is based on the Monte-Carlo simulation, and answers questions that does not involve feature selection. We have selected a small set of classifiers requiring hyperparameter tuning. These are the k-nearest neighbor, random forest, linear-SVM, and RBF-SVM classifiers. Furthermore, we compare these classifiers, to the Gaussian and the GMM classifiers, which do not require tuning of hyper-parameters. We also combine all of the aforementioned methods in a bucket of models method, using nested

cross-validation to provide an unbiased error estimate of each model selection method. We can pose the first evaluation part as three questions:

(1) What is the distribution of the out-of-sample error?

(2) What is the relation between the out-of-sample error and the cross-validation error?

(3) What is the effect of balancing folds by source versus splitting by source?

We have addressed these three questions in the discussion sections 3.7.1, 3.7.2, and 3.7.3 respectively.

The second evaluation part uses the results from the wound images dataset. Here we look at the out-of-sample error of the feature groups, and classifiers incorporating feature selection. The advantage of using the wound dataset is that we actually use the data we are interested in classifying. The downside is that we have a limited amount of data, and therefore the results are unreliable.

3.2 Method: Data

3.2.1 Dataset of Wound Images

The data consists of features extracted from superpixels, originating from an image set of ten wounds. These images are a subset of the images listed in section 2.3. Figure 3-2 contains a superpixel color mean feature image of every source, with varying numbers of superpixels per image. In our set, we will sample data using multiple settings for the superpixel region size.

The SLIC regularization parameter, see section 2.6.2, is 40 times the region size. Preliminary results indicated that this parameter-setting rule would result in decently formed superpixels, for both small and large region sizes. More optimal ways to decide the regularization parameter surely do exist, but we ignore that in this thesis.

We distinguish parts of the image into three categories. Wound, skin, and background. We group skin and background superpixels into class 1, and we let wound superpixels belong to class 2. Table 3-1 lists the proportion of each category. We define a superpixel to belong to wound, skin, or the background category, based on what category is most dominant among the pixels, from manually segmented ground truth images.

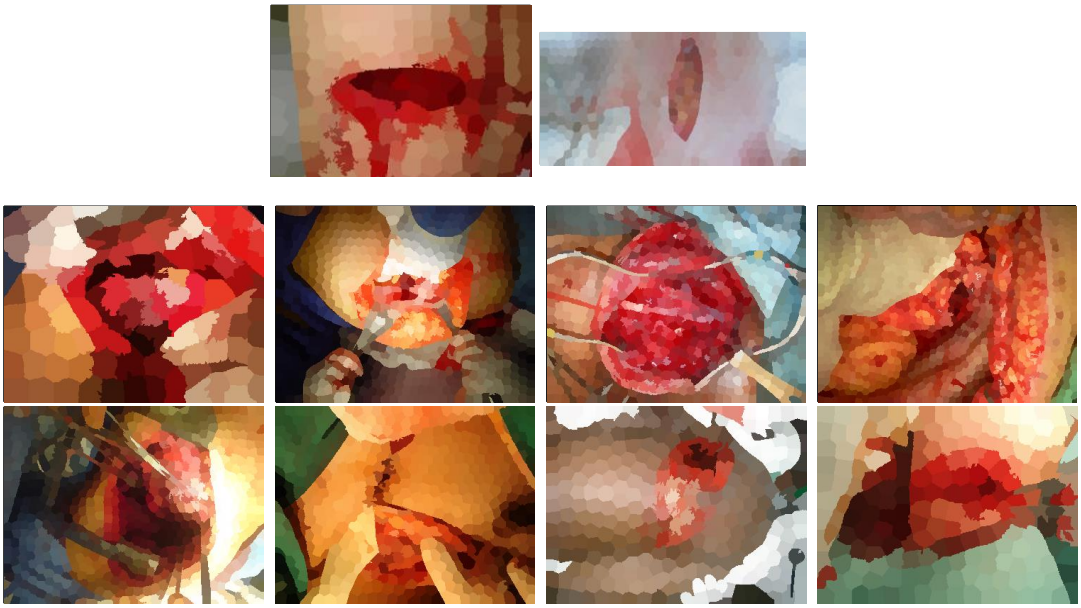

Figure 3-2 Superpixel Mean Feature Images

Mean Feature images of one image from every source in the wound dataset. There are two other images in the source that the upper right image is a part of, and the source containing the lower right image, has one other image as well. Otherwise, there is only one image per source. We call these mean feature images, because we have colored the region of each superpixel with its color mean value. The number of superpixels differs per image.

Table 3-1 Class Balance

Class 1		*Class 2*
Non-skin Background	*Skin*	*Wound*
25%	*25%*	*50%*

The class balance yields a simple two-class evenly balanced classification problem. We have not based this class proportion on the proportion observed in the images. It may be a reasonable choice of proportion in the full object recognition algorithm, but that is a very uncertain claim. We used under-sampling to adjust the class/category proportions to the desired value. With the under-sampling technique, we randomly select a subset of samples in each category such that they adhere to the desired proportion. Additionally, we under-sample each source such that they are of equal size.

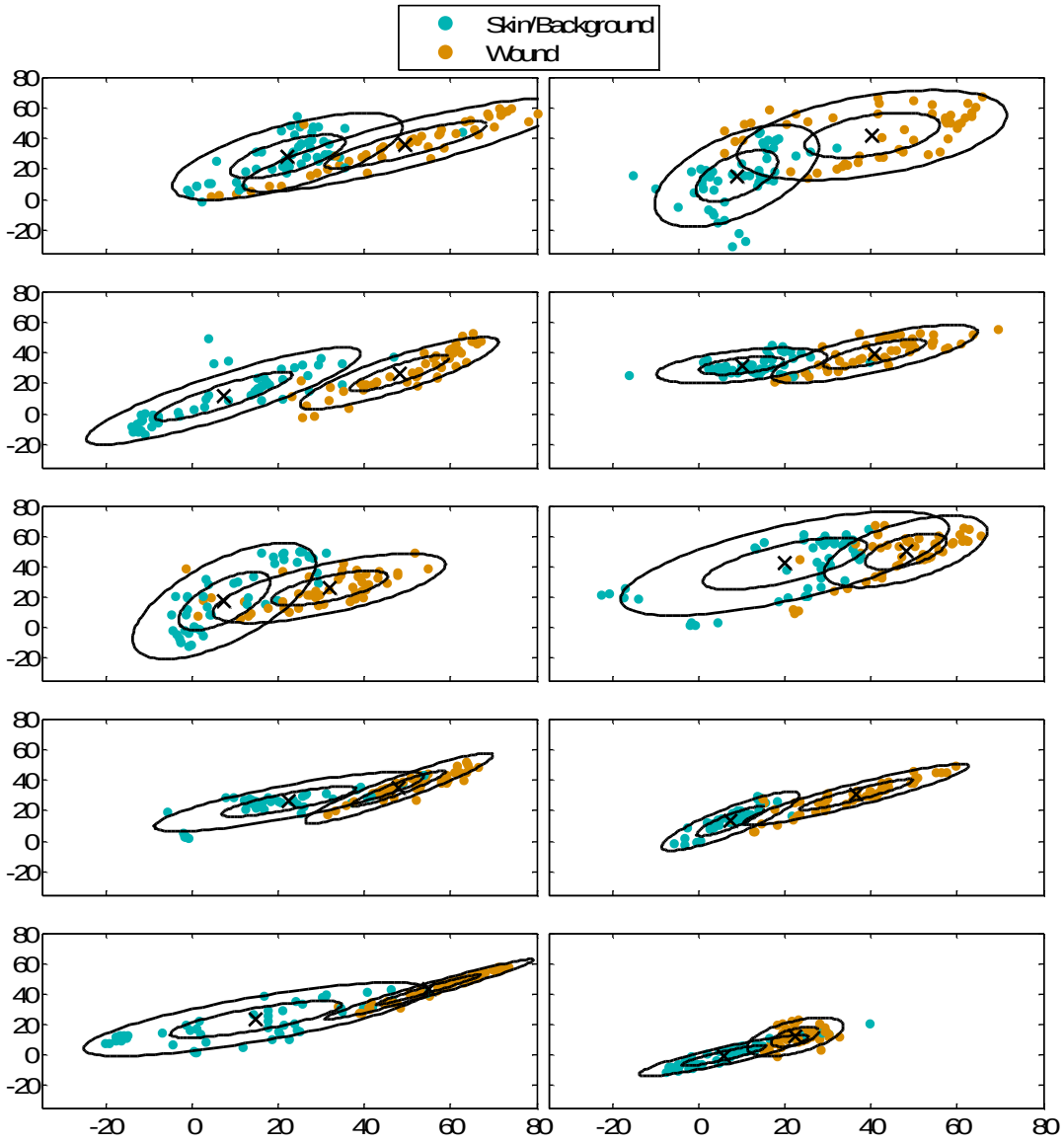

Figure 3-3 Scatter Plot of Color Segment Mean for All Ten Sources

The horizontal direction is the a component of Lab color space, and the vertical direction is the b* component of Lab color space. The circles represent the distance of one and two SD's from the class mean.*

Concerning the ideal class proportion, the goal is object recognition, not minimizing the percentage of misclassified segments in images. Presumably, class 1 and class 2 have a large overlap regardless of what feature we extract from the segments. Furthermore, say the true proportion of the wound class is 1 to 1000. Subsequently, training with severe class overlap and the 1 to 1000 class proportions would lead to always picking class 1. That would be useless for object recognition.

The superpixels are not the only local structure of interest. Other local structures may also aid the object recognition algorithm. One such type of structure is the

superpixel edge. This is the combination of two superpixels sharing borders. The shared border is the edge. We let a superpixel edge going from wound to edge, be a wound edge, and everything else be non-wound edges. We balance the proportion of both classes, although a wound edge is much rarer than the opposite. However, using the actual proportions would yield models always predicting an edge to, not be a wound edge. Again, that would be useless for object recognition.

For visualization purposes, and a basis for the synthetic data model, we use superpixel features. For feasible modelling of the data distribution, we have chosen to use no more than two features. We have selected the two color components of Lab color space (a* and b*) as the pair of features. They have an intuitive interpretation, and we were unable to find another feature pair candidate with significantly better preliminary classification results. Figure 3-3 visualizes a* and b* color mean data for all ten sources.

3.2.2 Samples Regarded as Random Variables

This section is highly theoretical. we describe how out data are dependent multivariate random samples, and what that really means.

The samples of source i, were drawn from some distribution, which we will denote f_{x_i}. The learning goal is to train and select a model that minimizes the classification error of samples drawn from unknown sources. Therefore, we should select the model that minimizes the classification error of the complete distribution of samples (f_x). Note that we let each sample be a multivariate random variable consisting of its class label, and feature values. By doing this we get one distribution for samples, rather than one for every class, thereby hiding redundant information.

We have a finite set of sources, and each source has a finite set of samples. Therefore, the set of all sources is a set of sets. The goal of the synthetic data model is the ability to generate these sets of sets. Whether the actual distribution of the synthetic data model matches the real distribution, is not so important. First, we use $|A|$ to denote the number of elements of a given set A. This is also called the cardinality of set A. Furthermore, let $x_{i,j}$ be sample j of source i. Let X be the set of all samples. Let S_i be the set of all samples from source i. We will refer to S_i as set-source i. Finally, let S be the set of all set-sources, and let $|S| = N_S$. That means S is a set of sets. Putting all this together, we can express the sets as follows:

$$S_i = \{x_{i,1}, x_{i,2}, \dots, x_{i,|S_i|}\}, \qquad 1 \leq i \leq N_S \tag{3.1}$$

$$X = \bigcup_{i=1}^{|S|} S_i \tag{3.2}$$

$$S = \{S_1, S_2, \dots, S_{N_S}\} \tag{3.3}$$

Note that the set-sources are non-overlapping, so the intersection between two set-sources is the empty set:

$$S_i \cap S_k = \{\}, \qquad i \neq k \tag{3.4}$$

In section 3.2.3, we transition from sets to develop models for the probability distributions of these sets.

In contrast to the common topic of dependence between features, we are concerned with the dependence between samples. The confusing aspect about dependence between samples is that we only have one instance of any given sample. First, let f_x be the marginal distribution of any given sample, not knowing what source it was sampled from. Let f_{x_i} be the distribution of the samples in set-source i. All the $x_{i,j}$ samples, were sampled from the distribution of their respective source, which gives us:

$$f_{x_{i,j}} = f_{x_i} \tag{3.5}$$

The key point about equation (3.5), is that knowing some $x_{i,j}$ values for a source (fixed i), infers information on the sample distribution of the source (f_{x_i}), thereby inferring information on every other $x_{i,j}$ of that source. From that follows, that the samples within a source must be dependent, and therefore the unconditional marginal distribution (f_x) of each sample, is not equal to the conditional distribution:

$$f_x(x) \neq f_x(x|x_{i,j}), \qquad x \in S_i \tag{3.6}$$

Conversely, two samples from different sources are independent:

$$f_x(x) = f_x(x|x_{i,j}), \qquad x \notin S_i \tag{3.7}$$

Note, equation (3.6) does not hold in the special case where the distribution of a source is equal to the marginal distribution ($f_x = f_{x_i}$), which we will ignore. This is fine, because we have a zero probability that these two distributions are exactly equal.

3.2.3 Synthetic Data Model

We want to build a generative model that attempts to mimic the properties of the data extracted from the wound images, as observed in Figure 3-3. Our goal is not to create an accurate model for the complete distribution, but rather to create a model that can generate sets of sources with individual within source distributions. Such a model will enable us to analyze what inferences we can make about the out-of-sample error based on a small set of sources. We can also use it to compare model selection methods.

While there are many approaches to create a distribution of distributions for two classes, we have decided to use a model consisting of four independent multivariate random variables, which we assume to be normally distributed. Normal distributions are defined by their means and covariance matrices, so when we need a distribution of sources, we really need a distribution of source means, and source covariance matrices. Figure 3-4 contains a graphical representation of the random variables.

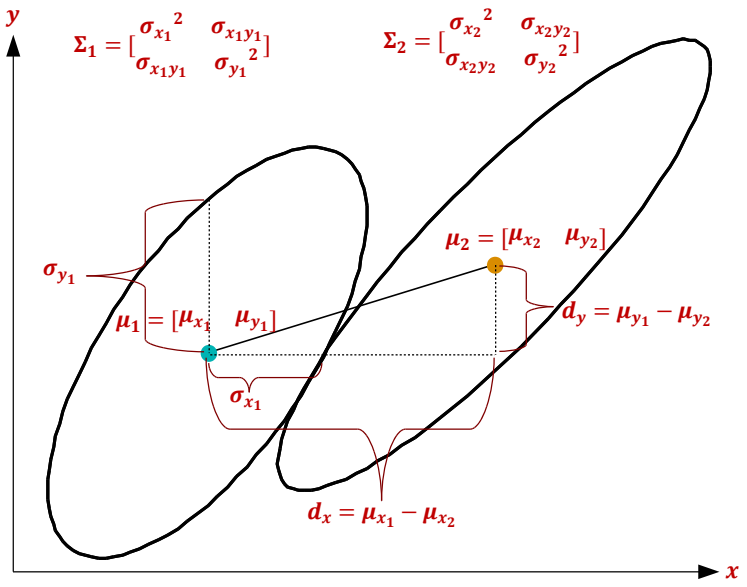

Figure 3-4 Model of Data Distribution

The figure shows the distribution of the two classes from a given source. The $[d_x, d_y]$ vector is the offset between the two class means μ_1, and μ_2.The within source distributions have covariance matrices Σ_1 and Σ_2, and the circles represent the standard deviations of the two classes.

The first multivariate variable is the source mean of the wound class ($\mu_2 = [\mu_{x_2}, \mu_{y_2}]$). The second multivariate variable is the offset from the wound class to the background class ($d = [d_x, d_y]$), which we use to calculate the mean of the other class. It is obvious that the mean of the two classes are dependent, but the same is not true

for the offset, nor is that important. By using the offset rather than the mean, we avoid having to estimate the distribution of four dependent random variables.

The third and fourth multivariate variables are the covariance matrices of the source distribution for the two classes. The 2×2 covariance matrices (Σ_1 and Σ_2) are symmetric, and therefore contains three unique parameters each of which we model as the vectors Σ_1' and Σ_2'.We can express the four multivariate random variables, and the 2×2 covariance matrices mathematically as follows:

$$\Sigma_1 = \begin{bmatrix} \sigma_{x_1}^2 & \sigma_{x_1 y_1} \\ \sigma_{x_1 y_1} & \sigma_{y_1}^2 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} \sigma_{x_2}^2 & \sigma_{x_2 y_2} \\ \sigma_{x_2 y_2} & \sigma_{y_2}^2 \end{bmatrix} \tag{3.8}$$

$$\mu_2 = [\mu_{x_2}, \mu_{y_2}] \sim \mathcal{N}\left(\widehat{\mathbb{E}}[\mu_2], \widehat{Cov}(\mu_2)\right) \tag{3.9}$$

$$d = [d_x, d_y] \sim \mathcal{N}(\hat{\mu}_d, \hat{\Sigma}_d) \tag{3.10}$$

$$\Sigma_1' = [\sigma_{x_1}^2, \sigma_{y_1}^2, \sigma_{x_1 y_1}] \sim \mathcal{N}\left(\widehat{\mathbb{E}}[\Sigma_1'], \widehat{Cov}(\Sigma_1')\right) \tag{3.11}$$

$$\Sigma_2' = [\sigma_{x_2}^2, \sigma_{y_2}^2, \sigma_{x_2 y_2}] \sim \mathcal{N}\left(\widehat{\mathbb{E}}[\Sigma_2'], \widehat{Cov}(\Sigma_2')\right) \tag{3.12}$$

By assuming the multivariate random variables to be normally distributed, we need estimates for their means and covariance matrices. We obtain these estimates from the sample means and sample covariance matrices given by the data from the wound images, as seen in Figure 3-3.

Note that covariance matrices for the within source distributions (Σ_1 and Σ_2) must have non-negative diagonal elements, and the matrix must be positive-semidefinite (non-negative eigenvalues). These conditions are not guaranteed by sampling Σ_1' and Σ_2' according to a normal distribution. We have dealt with this issue by naively resampling Σ_1' and Σ_2' until the conditions are met. This very simple approach, only works well because we use a low-dimensional feature space. For a high-dimensional feature space, we can generate covariance matrices by sampling eigenvalues, and sampling rotation values. We use these rotation values to rotate an identity matrix, and we let the result be the eigenvector matrix. This results in a valid uniquely defined covariance matrix.

The optimal solution for this model is to select the class that has the highest mean probability density for a given feature input. Because μ_2 and d are normally distributed, and μ_1 is the sum them, then μ_1 must also be normally distributed. Therefore, a Gaussian classifier with the true mean gives the optimal solution,

assuming we use the true mean and covariance matrix values for the source means. In the results section 3.6, the Gaussian classifier based method acts as a marker on what is possible to achieve for a model selection method.

3.3 Method: Model Selection Methods

3.3.1 Unbiased Error Estimates from Dependent Samples

We use cross-validation to evaluate and compare models. As described in section 3.2, our data has groups of dependent samples. Any two samples from separate groups however, are independent. Because we want an estimate for the error of new sources, a standard cross validation procedure will procure biased results. To ensure unbiased error estimates, we must ensure that samples from one source cannot exist in multiple folds.

We compare two methods to separate data into folds. The first method respects the dependency of the samples by putting all samples from one source in just one fold. This ensures that we never encounter dependent samples in both the training and validation set at the same time. The second method evenly balances samples per source for every fold. In practice, the second method is almost identical to randomly selecting data for each fold without respecting the dependency between samples.

A recent survey of cross-validation procedures for model selection [52], discussed cross-validation with dependent data. However, the survey only covered dependent data in the context of time series and dependent observations. These differ from our scenario of fully known groups of dependent samples. We were unable to find sources covering model selection methods for our specific case of dependent samples.

3.3.2 Hyperparameter Selection

One problem of model selection is that obtaining a final model, is more complex than just training a classifier on training data. Many classifiers rely on tuning hyper-parameters to perform well, and we must incorporate an automatic hyper-parameter selection procedure for the classifiers to be complete model selection methods. There are two key components required for selecting hyper-parameters, and what we choose those to be, can have a large impact on the results. The first component is the rules that

determines what parameters to initially test, and subsequent hyper-parameters to test based on the performance metrics of previously tested hyper-parameters. The second component is what performance metric we use to compare hyper-parameters.

We introduced model selection methods section 2.8.8, but it may be difficult to see exactly what our model selection methods are. To illustrate, Figure 3-5, Figure 3-6, and Figure 3-7 outlines hyperparameter selection for superpixel segment and edge classifiers. Figure 3-5 defines cross-validation when training classifiers with hyperparameters. This cross-validation function is used in Figure 3-6, which shows hyperparameter selection. Finally, Figure 3-7 shows an outer cross-validation loop around the hyperparameter selection function in Figure 3-6. Below these figures, we explain the hyperparameter selection components in more detail. The flowcharts can be a useful reference, but we will also rely on code algorithms to be more precise, using consistent mathematical definitions used throughout this chapter.

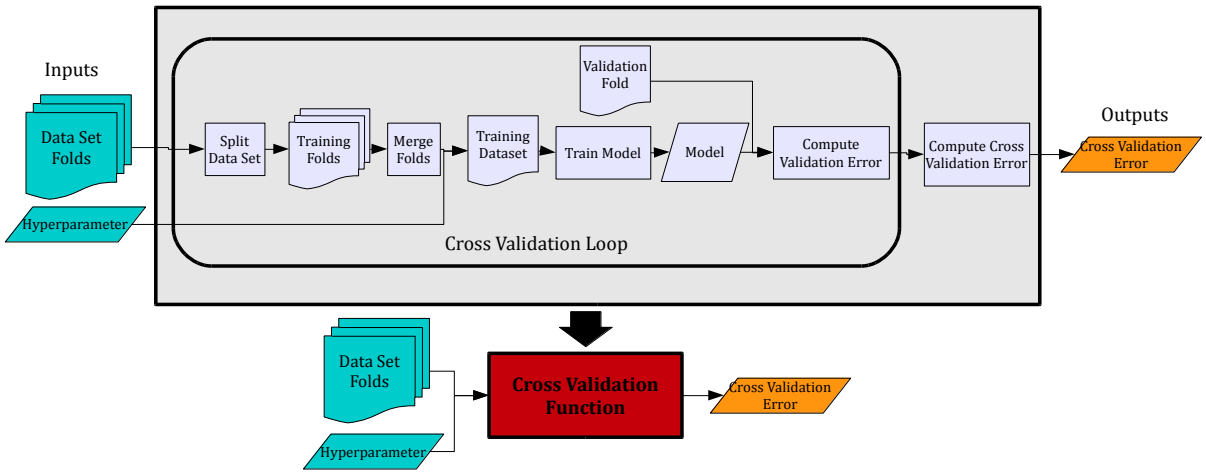

Figure 3-5 Cross Validation Flowchart

The flowchart shows cross validation for a classifier that is dependent on some parameters. Rectangles represents processing steps. Parallelograms represents objects. Curved Blocks represents data.

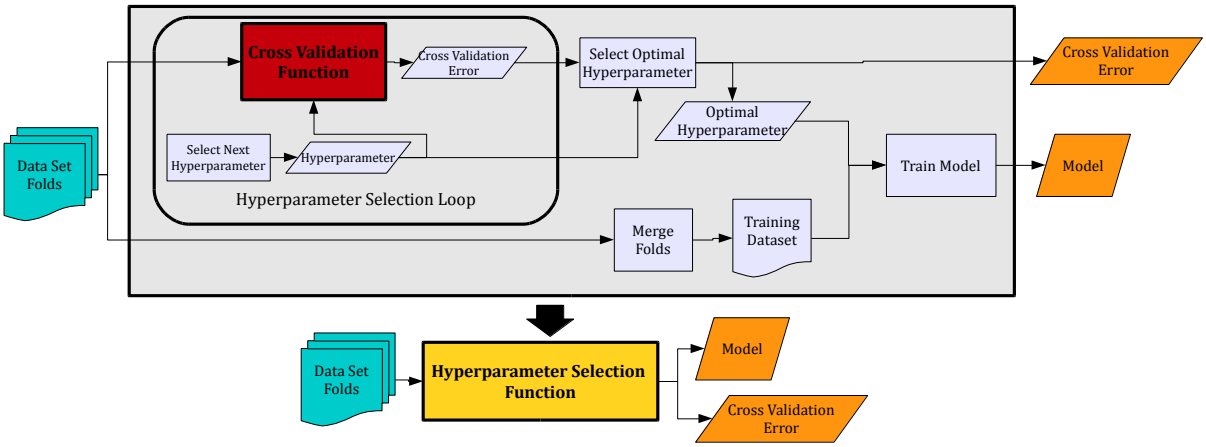

Figure 3-6 Hyperparameter Selection Flowchart

The flowchart shows hyperparameter selection. The algorithm finds the optimal hyperparameters, and then trains the classifier with these hyperparameters. The cross-validation error estimate is optimistically biased because it is used for optimization of the hyperparameter. Rectangles represents processing steps. Parallelograms represents objects. Curved Blocks represents data. We defined the red cross-validation function in Figure 3-5.

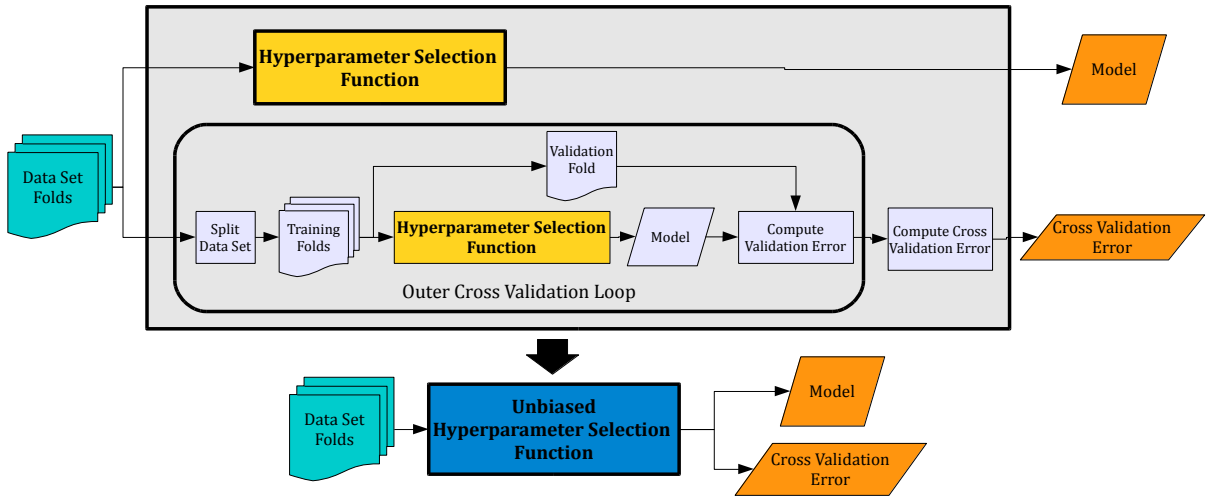

Figure 3-7 Unbiased Hyperparameter Selection Flowchart

The flowchart shows unbiased hyperparameter selection. The algorithm finds the optimal hyperparameters, and then trains the classifier with these hyperparameters. The cross-validation error estimate is considered unbiased, as we use an outer loop to compute the cross-validation error; in practice there is a small pessimistic bias due to a smaller training set for error estimation. Rectangles represents processing steps. Parallelograms represents objects. Curved Blocks represents data. We defined the yellow hyperparameter selection function in Figure 3-6.

The rules we use for selecting what parameters to test, are based on a grid-based search. This involves an initial stage of searching through the range of parameter values using a coarse grid. The following stages perform subsequent grid searches in the best performing region using finer and finer masked grids for every iteration. If the relation between the output performance metric and input hyper-parameters is a

convex function, then this grid search method will converge to the optimal hyper-parameter solution.

Although it is possible to find the optimal hyper-parameters for convex functions, it only means that the hyper-parameters are optimal for the performance metric. The hyper-parameters, however, may not be optimal for the out-of-sample error. This is a problem called *over-fitting*, and it is possible to over-fit both the training data and the validation data used to evaluate the performance of the hyper-parameters. We will use k-fold cross-validation, and let the cross-validation error be the performance metric. The cross-validation error is the mean of the mean validation error of all sources. If we let E_{CV} be the cross-validation error, E_{si} be the validation error of source i, and N_K be the number of folds, then we get:

$$E_{CV} = \frac{1}{N_K} \sum_{i=1}^{N_K} E_{si} \tag{3.13}$$

Because we want to classify new sources, we hypothesize that sources should not be split among folds. Because we only have ten folds, we assign every source to their own individual fold. This is the *split-by-source* method. We will compare this to the balance-by-source method. The *balance-by-source* method involves randomly distributing the sources in every fold, such that every folds contains an equal number of samples per class per source. In practice, this is approximately the same as a completely random selection of samples per fold.

We select the final model, by training, using the full dataset and optimal parameters. This is the result of the model selection method. Algorithm 3-1 defines the components of the general model selection method in the context of a function. The input D_{CV} is a structure of N_K folds, and it is either split-by-source or balanced by source. Note that the $optimal\_hyperparameters$ function also makes use of the cross-validation function, cv_train. Subsequently the cross-validation function makes use of the $train$ function and a $predict$ function that return the error ratio of the validation data, given a trained model.

Algorithm 3-2 specifies how we obtain an unbiased estimate of the error for a given model selection method, using nested cross-validation. We still obtain the hyper-parameters and final model in exactly the same way as the standard model selection procedure described in Algorithm 3-1. We train the unbiased validation with $K - 1$ folds, which has no difference from the standard hyperparameter selection procedure. The difference is that for the unbiased error estimate, we chose hyperparameters based

on cross-validation error of models trained with $K-2$ folds; contrarily, the biased cross validation error in Algorithm 3-1 uses $K-1$ folds.

Algorithm 3-1 Hyperparameter Selection

We use capital letters to express the variable type. We use M for models, D for datasets, and E for error ratio. The function "cv_train", computes the cross-validation error by picking training and validation sets from D_{CV} and using the optimal hyper-parameters H for training. The function "optimal_hyper-parameters" searches through different hyper-parameter values, and returns the hyper-parameter values with the lowest cross-validation error, obtained from "cv_train".

$$\boldsymbol{function}\ [E_{CV}, M_{final}] = hyperparameter\_selection(D_{CV})$$
$$H = optimal\_hyperparameters(D_{CV})$$
$$E_{CV} = cv\_train(D_{CV}, H)$$
$$D_{merged} = merge\_folds(D_{CV})$$
$$M_{final} = train(D_{merged}, H)$$
$$\boldsymbol{end}$$

Algorithm 3-2 Unbiased Hyperparameter Selection

We use capital letters to express the variable type. We use M for models, D for datasets, E for error ratio, and N for numbers. '~' indicates an unwanted return value. This function makes use of the "hyperparameter_selection" function; see Algorithm 3-1. E_{val} is a $N_{folds} \times 1$ vector, and D_{CV} is a $N_{folds} \times 1$ structure array.

$$\boldsymbol{function}\ [E_{CV}, M_{final}] = unbiased\_hyperparameter\_selection(D_{CV})$$
$$N_{folds} = length(D_{CV})$$
$$\boldsymbol{for}\ i = 1\ to\ N_{folds}$$
$$D_{val} = D_{CV}(i)$$
$$D_{train} = D_{CV}(all\ except\ i)$$
$$[\sim, M] = hyperparameter\_selection(D_{train})$$
$$E_{val}(i) = predict(M, D_{val})$$
$$\boldsymbol{end}$$
$$E_{CV} = mean(E_{val})$$
$$[\sim, M_{final}] = hyperparameter\_selection(D_{CV})$$
$$\boldsymbol{end}$$

3.3.3 Selected Hyperparameter Selection Methods

We have chosen random forests, k-NN, Linear-SVMs, and RBF-SVMs as the classifiers we base our grid-based model selection methods on. In our simulations, we will also include the Gaussian classifier and a special Gaussian mixed model classifier; both of which do not require tuning of hyper-parameters. For convenience, we will give the model selection methods, the same name as the classifier they are based on, and refer

to them by this name, when the context makes it clear that we are comparing model selection methods.

The random forests classifier described in section 2.8.5, has the H_{RF} hyper-parameter. It ranges from one to the number of features, and we select the optimal value, using an exhaustive search. Because we only have two features, the grid search simplifies into testing for the hyper-parameter to be either one or two. We fix the number of trees in the random forests classifier to 100. Otherwise, we use the default random forest options of the treebagger function in MATLAB 2013b.

The Linear-SVMs classifier has the H_C parameter. The RBF-SVMs classifier has the H_C, and H_σ parameter. We have explained what these are in section 2.8.6. In both cases, we use grid search, but for the RBF SVMs we use a 2D-grid search. We start by searching through the following geometric sequence of the parameters (the range applies to both H_C and H_σ):

$$10^{-5}, 10^{-4}, \dots, 10^{9}, 10^{10}$$

That is 16 parameter choices for the Linear-SVMs and 256 unique parameter combinations for the RBF SVMs. After having found the optimal parameter combination, we do a more fine-grained grid search in that area, and repeat that action six times to fine tune the parameters. For example, if the first stage of a Linear-SVMs grid search found 10^3 to be optimal, we would then search through the following geometric sequence:

$$10^{2}, 10^{2.5}, 10^{3}, 10^{3.5}, 10^{4}$$

Note that we only need to compute the cross-validation error of two new parameter choices. For the RBF SVMs we have a 5×5 grid and 20 new parameter combinations.

The k-NN classifier described in section 2.8.4 has the $H_{K-NN} \in \mathbb{N}$ parameter, which determines the number of nearest neighbors to evaluate. We also search through this using a grid search, but we set a hard cap of 1000. The initial search uses the following sequence (42 values):

$$1,2,3,\dots,9,10,15,20,\dots,45,50,60,\dots,140,150,180,\dots,270,300,400,\dots,900,1000$$

It makes little sense to have more neighbors, than there are samples per class, and therefore we modify this sequence by cutting off any parts, larger than the number of samples per class, and insert this value, as the maximum value. If the initial search discovers, say $H_{K-NN} = 50$, to be the optimal hyper-parameter, then we finalize with an exhaustive search in that region by testing the following parameters:

$$45, 46, \ldots, 49, 50, 51, \ldots, 59, 60$$

In the worst-case scenario, this hyper-parameter search method must test $42 + 199$ values.

3.3.4 Bucket of Models using Nested Cross-validation

We can combine all the model selection methods mentioned in section 3.3.3, by testing all of them, and picking the model with the lowest estimated error. However, we cannot ignore the bias problem of the model selection methods that requires tuning of hyper-parameters (k-NN, RF, Linear-SVM, and RBF-SVM). The cross-validation error of a model where we tuned hyper-parameters using the same folds for cross-validation is not unbiased. This is the bias problem. To circumvent this, we use nested cross-validation to evaluate the performance of these methods. The cross-validation error of this outer cross-validation loop is an unbiased estimate of the out-of-sample error.

Note that if we pick the model selection method with the lowest unbiased error estimate, that error estimate is a biased estimate of our final model. We can interpret the choice of what sub-model selection method as a hyper-parameter to the bucket of models method. However, this hyper-parameter has a very limited set of values (6); therefore, the error estimate presumably has a small bias.

An advantage of the bucket of models method is its flexibility to adapt the computational constraints of different classifiers. We may set minimum constraints on how fast the classifiers should predict new samples. This could involve reducing the number of training samples in the k-nearest neighbor classifier, or reducing the number of trees in random forests. We may also set constraints to the training time. Support vector machines does not handle very large datasets well, and so we could train the Linear-SVMs and RBF-SVMs methods with a limited dataset, but still use the full dataset for the other methods. For example, the RBF-SVMs method may have the lowest out-of-sample error with a limited dataset, yet a random forest classifier trained on the full dataset outperforms it.

We can set up automated procedures where we set training and prediction time constraints, and the bucket of models method procures the (assumed) optimal result that satisfies these constraints. All we need are approximate models for the training time and prediction time of the classifiers, accounting for the number of features, samples, number of hyper-parameters options tested, and so on. When multiple

factors can affect training or prediction time, we have to deal with the dilemma on how to balance these. For instance, in reducing training time, is it best to reduce the hyper-parameter search dataset and search through more hyper-parameter combinations, or use a larger dataset, but search through fewer hyper-parameter combinations?

3.3.5 Feature Selection

We have listed a selection of superpixel and superpixel edge features in section 2.7. Presumably, a subset of these features yields the lowest out-of-sample error. The problem of feature selection is how do we select these features? We provided a general overview of feature selection methods in section 2.8.9. Here we extend the model selection methods listed in section 3.3.3, to include feature selection. Figure 3-8, and Figure 3-9, outlines combined hyperparameter and feature selection for superpixel segment and superpixel edge classifiers.

We will treat the selection of features like an extended version of the hyperparameter optimization problem. Meaning, we select both the hyperparameters, and the features that minimize the cross validation error. To search through feature combinations, we use the greedy forward selection algorithm described in section 2.8.9. Algorithm 3-3 defines the feature and hyperparameter selection. Note that we use the same cross validation error for optimization of hyperparameters and features.

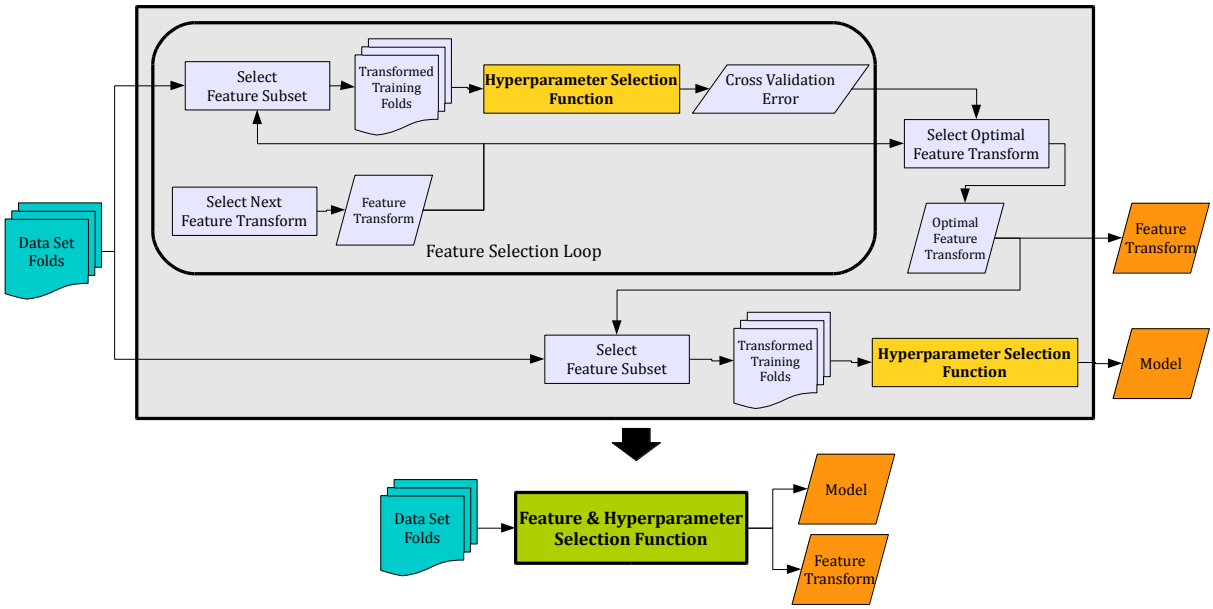

Figure 3-8 Feature & Hyperparameter Selection Flowchart

The flowchart shows feature and hyperparameter selection. The algorithm finds the optimal feature and hyperparameter subset, and then trains the classifier. The cross-validation error estimate is optimistically biased, as it is used for optimization of the hyperparameters. Rectangles represents processing steps. Parallelograms represents objects. Curved Blocks represents data. We defined the yellow hyperparameter selection function in Figure 3-6.

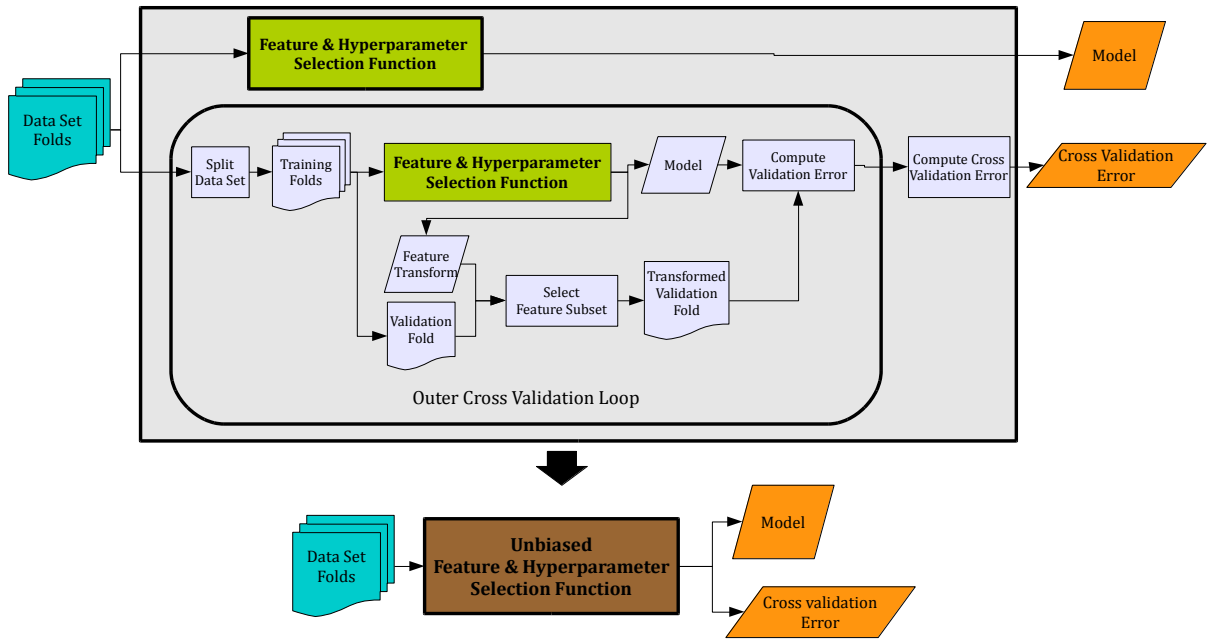

Figure 3-9 Unbiased Feature & Hyperparameter Selection Flowchart

The flowchart shows unbiased feature and hyperparameter selection. The algorithm finds the optimal feature subset and hyperparameters, and then trains the classifier. The cross-validation error estimate is considered unbiased, as we use an outer loop to compute the cross-validation error; in practice there is a small pessimistic bias due to a smaller training set for error estimation. Rectangles represents processing steps. We have defined the green feature & hyperparameter selection function in Figure 3-8.

Algorithm 3-3 Feature & Hyperparameter Selection

We use capital letters to express the variable type. We use M for models, N for numbers, D for datasets, H for hyperparameters, and E for error ratio. We defined the 'hyperparameter_selection' function in Algorithm 3-1. The 'greedy_forward_selection' function is the greedy forward selection algorithm described in section 2.8.9. It uses the error measure from the 'hyperparameter_selection' cross validation error (E_{CV}). The function "feature_subset" transforms the cross-validation data set to contain the feature subset referenced by the second argument.

$\boldsymbol{function}\ [E_{CV}, M_{final}] = feature\_and\_hyperparameter\_selection(D_{CV})$

$M^*_{features} = greedy\_forward\_selection(D_{CV})$

$D'_{CV} = feature\_subset(D_{CV}, M^*_{features})$

$[E_{CV}, M_{final}] = hyperparameter\_selection(D'_{CV})$

$\boldsymbol{end}$

3.4 Method: Monte Carlo Simulation

With a generative model of a data distribution, we can evaluate the performance of our parameter and model selection procedures using Monte Carlo simulation. We perform repeated experiments where for each experiment; draw a dataset of ten sources ($N_K = 10$). From this, we feed this to our model selection methods, who return a model and the cross-validation error. Finally, we sample a large test set containing ten-thousand sources, which we use to evaluate the out-of-sample error of the trained model. The results will differ from one run to the next, because the combined distribution from the ten training sources may significantly differ from the true distribution. We have described this algorithm in more detail, in Algorithm 3-4. We defined the model selection function in this algorithm in Algorithm 3-1. Algorithm 3-5 defines the $generate\_source$ function that randomly samples a source, and randomly samples a set of samples from the within source distribution.

The out-of-sample error is the mean error of the test sources, whereas the cross-validation error, which we defined in equation (3.13), is the mean of the N_K sources when used for validation. If we let N_{out} be the number of sources in the test set ($N_{out} = 10000$), and E'_{si} be the error of source i of the test set, then we define the out-of-sample error to be:

$$E_{out} = \frac{1}{M} \sum_{i=1}^{N_{out}} E'_{si} \tag{3.14}$$

Note that the out-of-sample error is only an estimate for the out-of-sample error, but with a large number of test sources, it should be sufficiently accurate. We are not

directly interested in the cross-validation error. What we are interested in is how well the cross-validation error estimates the out-of-sample error. Therefore, the difference in error is a more interesting variable:

$$E_{diff} = E_{CV} - E_{out} \tag{3.15}$$

The Monte Carlo simulation provides an out-of-sample error value, and difference in error value for every run.

Algorithm 3-4 Monte Carlo Simulation of Model Selection Methods

We use capital letters to express the variable type. We use M for models, D for datasets, E for error ratio, B for Boolean value, and N for numbers. E_{out}, E_{CV}, *and* E_{diff} *are* $N_{MCS} \times 1$ *vectors. We have defined the generate_source function in Algorithm 3-5.*

$M_{data} = \{\hat{E}[\mu_2], \widehat{Cov}(\mu_2), \hat{\mu}_d, \hat{\Sigma}_d, \hat{E}[\Sigma_1'], \widehat{Cov}(\Sigma_1'), \hat{E}[\Sigma_2'], \widehat{Cov}(\Sigma_2')\}$
for $i = 1$ *to* N_{MCS}
  for $j = 1$ *to* N_K
    $D_{CV}(j) = generate\_source(M_{data}, N_{samples})$
  end
  if $B_{balance\_by\_source}$
    $D_{CV} = balance\_by\_source(D_{CV})$
  end
  $[M_{classify}, E_{CV}(i)] = hyperparameter\_selection(D_{CV})$

  for $j = 1$ *to* N_{out}
    $D_{test} = generate\_source(M_{data}, N_{samples})$
    $E_{sources}(j) = predict(M_{classify}, D_{test})$
  end
  $E_{out}(i) = mean(E_{sources})$
  $E_{diff}(i) = E_{CV}(i) - E_{out}(i)$
end

We have selected five highly descriptive statistics concerning the three questions posed near the end of section 3.1. The mean and standard deviation of the out-of-sample error listed in equation (3.16), describes the distribution of the out-of-sample error, thereby providing an answer to the first question.

$$E[E_{out}], SD(E_{out}) \tag{3.16}$$

The three remaining statistics (listed in equation (3.17)) each contributes some insight into the relation between the cross-validation and out-of-sample error, and thereby providing an answer to the second question.

$$E[E_{diff}], SD(E_{diff}), Corr(E_{Out}, E_{CV}) \tag{3.17}$$

The expected value of the difference in error ($E[E_{diff}]$) describes the bias of the model selection method. A negative value implies that the cross-validation error is on average lower than the true out-of-sample error. The standard deviation of the difference in error ($SD(E_{diff})$) describes the dispersion of the cross-validation error as an estimate for the out-of-sample error. A high E_{diff} dispersion does not exclude the possibility of the cross-validation error being a reliable estimate of the out-of-sample error. It is still possible that there is a strong dependence between them, which we measure by their correlation ($Corr(E_{Out}, E_{CV})$). If there is a strong dependence, then there is also hope that the different classifiers share the same dependence, such that the cross-validation error is a reliable estimate when used to compare different classifiers.

Algorithm 3-5 Generate Source

We use capital letters to express the variable type. We use M for models, D for datasets, S for sampled values, and N for numbers. We use the → symbol to refer to values within the M_{data} structure. $M_{data} = \{\hat{E}[\mu_2], \widehat{Cov}(\mu_2), \hat{\mu}_d, \hat{\Sigma}_d, \hat{E}[\Sigma_1'], \widehat{Cov}(\Sigma_1'), \hat{E}[\Sigma_2'], \widehat{Cov}(\Sigma_2')\}$

$\boldsymbol{function}\ [D] = generate\_source(M_{data}, N_{samples})$

$S_{\mu_2} = randomly\ sample\ from\ \mathcal{N}(M_{data} \rightarrow \hat{E}[\mu_2], M_{data} \rightarrow \widehat{Cov}(\mu_2))$

$S_d = randomly\ sample\ from\ \mathcal{N}(M_{data} \rightarrow \hat{\mu}_d, M_{data} \rightarrow \hat{\Sigma}_d)$

$S_{\mu_1} = S_{\mu_2} + S_d$

$S_{\Sigma_1'} = randomly\ sample\ from\ \mathcal{N}(M_{data} \rightarrow \hat{E}[\Sigma_1'], M_{data} \rightarrow \widehat{Cov}(\Sigma_1'))$

$S_{\Sigma_2'} = randomly\ sample\ from\ \mathcal{N}(M_{data} \rightarrow \hat{E}[\Sigma_2'], M_{data} \rightarrow \widehat{Cov}(\Sigma_2'))$

$S_{\Sigma_1} = \begin{bmatrix} S_{\Sigma_1'}(1) & S_{\Sigma_1'}(3) \\ S_{\Sigma_1'}(3) & S_{\Sigma_1'}(2) \end{bmatrix}, S_{\Sigma_2} = \begin{bmatrix} S_{\Sigma_2'}(1) & S_{\Sigma_2'}(3) \\ S_{\Sigma_2'}(3) & S_{\Sigma_2'}(2) \end{bmatrix}$

$\boldsymbol{for}\ i = 1\ to\ N_{samples}/2$

$S_1(i) = randomly\ sample\ from\ \mathcal{N}(S_{\mu_1}, S_{\Sigma_1})$

$S_2(i) = randomly\ sample\ from\ \mathcal{N}(S_{\mu_2}, S_{\Sigma_2})$

$\boldsymbol{end}$

$D = merge\ S_1\ and\ S_2$

$\boldsymbol{end}$

We can answer the final question by comparing the equation (3.16 statistics, of the balance-by-source method vs the split-by-source method. We expect the cross-validation error of the balance-by-source method to be optimistically biased ($E_{diff} < 0$). We expect this, because the validation samples are sampled from sources also present in the training data. In other words, dependent groups of samples are split

among the training and validation data. In contrast, the samples used in the out-of-sample error data are not dependent with any data used for training.

Besides the split-by-source and balance-by-source method, we will also do Monte Carlo simulations where all samples are independent. We will use this to compare the effects of dependent data and independent data. We have only included the k-NN and Gaussian model selection methods in this simulation. We use them as representative examples for the other model selection methods.

Due to computational constraints, we run Monte Carlo simulations with a limited number of runs ($N_{MCS} = 200 \vee 1000$). Therefore, the statistics listed in equation (3.16) will not be entirely accurate. It is important to determine if the comparisons we make between statistics of model selection methods are statistically significant, and if the differences are large enough to be important. For the expected value statistics, we will use the standard error of the mean to. We defined the standard error of the mean in equation (2.5). For example, for E_{out}, with a standard deviation of 0.1 ($SD(E_{out}) = 0.1$), simulated with 200 Monte Carlo runs ($N_{MCS} = 200$), we get:

$$SEM(E_{out}) = SD(E[E_{out}]) = \frac{SD(E_{out})}{\sqrt{N_{MCS}}} = \frac{0.1}{\sqrt{200}} = 7.0711 * 10^{-4} \qquad (3.18)$$

3.5 Method: MATLAB Framework for Model Selection Methods

The algorithms we have represented in this chapter specifies what the model selection methods do, and they are similar to the actual MATLAB code we used for the Monte Carlo simulation results. This code however, has some issues. Since the early works of this thesis (see section 1.2), we have attempted to use different classifiers, hyperparameter values, and features. Gradually we refined upon the code, attempting to reuse code such as cross-validation methods for different classifiers. However, adding new capabilities such as feature selection of groups, deciding where to use cross-validation, or having classifiers use the source information (GMM classifier), were tedious and required alterations in many parts of the code. To address these issues, we have made a new general framework for model selection methods that can easily be extended. We used the new code implementation for the results in 3.6.3. We will discuss the most important concepts of this framework here.

For clarity, we will explain the framework using an example model-selection configuration. We have a k-NN classifier with the hyperparameter search method described in section 3.3.3. Furthermore, we use greedy forward selection for feature selection, using the same cross-validation error that we use for hyperparameter selection. Lastly, we obtain an unbiased error estimate by wrapping this in an outer cross-validation loop. Figure 3-10 illustrates this configuration.

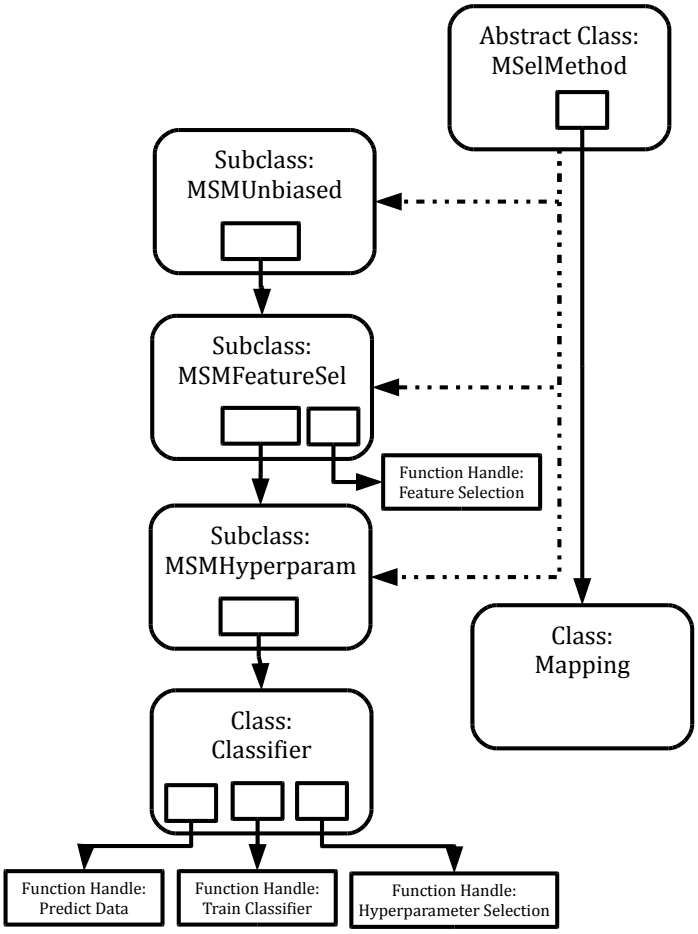

Figure 3-10 Example Setup of a Model Selection Method Algorithm
Minimalistic illustration of the model-selection method framework, training a classifier with hyperparameter selection, feature selection, and an outer cross-validation loop for unbiased error estimates. The MSM prefix of class names is an abbreviation for model selection method.

All subclasses of *MSelMethod* inherits the prediction function, and has the same interface for training. The superclass also contains other functionality not necessarily specific to a single subclass. Every *MSelMethod* object contains a sub MSelMethod object, except the MSMHyperparam object, which contains a classifier object. A, MSelMethod object is trained by training the sub MSelMethod object.

The train function of an MSelMethod serves three functions. The first is learning the optimal features or hyperparameters. The second is to train a model using the full dataset. The third function is to provide an estimate of the error, via either cross-validation, or inheriting the cross-validation error of the sub MSelMethod object.
The MSMUnbiased object has no feature or hyperparameters to optimize. It simply performs cross-validation on the sub MSelMethod object, obtaining an unbiased error estimate. Therefore, the MSMUnbiased object is always configured to provide an error estimate via cross-validation, rather than inheriting the error from the sub MSelMethod object. It only makes sense to use the MSMUnbiased object as the outer object, to provide an unbiased error estimate of the entire model-selection method training.

The MSMFeatureSel uses the feature selection function handle to select a subset of the feature groups. The feature subset is learned as a mapping stored in a *Mapping* object. The Mapping object has a function that transforms input data. In this case the function returns the data with the selected feature groups. To reduce the computational complexity, we have chosen to inherit the error from the MSMHyperparam object. Had we added a cross-validation loop here, we would get feature selection based on an unbiased estimate of the mode with optimal hyperparameters. Instead, we select feature using a biased estimate, because it were also use to select hyperparameters.

The MSMHyperparam object has some small alterations from other MSelMethod objects, as it contains a Classifier object, rather than a sub MSelMethod object. For hyperparameter selection, it extracts the hyperparameter search function stored in the classifier.

Normally when calling train on the MSMHyperparam and MSMFeatureSel objects, it initiates a feature search. However, we can configure the object to be pre-learned, in which case they will skip the hyperparameter selection and feature selection. We exploit this concept in the selection functions by taking the calling object as an argument, configuring them to be pre-learned with the hyperparameter or features we want to test, and then calling their training function. Using this, the calling object decides how to compute the error estimate, making the design of the selection functions very simple.

After training the entire model selection method, it is capable of predicting new data. The MSelMethod superclass has a predict function that takes input data, transforms it by passing it through a mapping function (if a mapping exists), calls the

predict function of the sub object, and then returns the predictions obtained by the sub object. As such, calling predict on the model of Figure 3-10 will first call the prediction function of the MSMUnbiased object. It will call the MSMFeatureSel predict function, which will transform the data to the subset of the selected features before calling the MSMHyperparam predict function. The MSMHyperparam predict function calls the predict function handle of the classifier. In this case the predict function normalizes the in accordance with the mean and standard deviation of the training set before calling the *knnclassify* function in MATLAB. Normalizing data is a functionality we should have as an option integrated in the Classifier class, which would be easy to implement.

To simplify the setup of an entire model selection method, each class has a default configuration. The configuration is a class in itself, and has an overwrite fields. As such, we can overwrite the default configuration whenever we want to make a change deviating from the default.

We only used the bucket of models, selection method for the synthetic data model. Its excessive computational time discouraged further use. We did not use the bucket of models for the wound dataset due to excessive computations. Therefore, we skipped supporting it in the new model-selection method framework; however, it would be easy to implement.

3.6 Results

3.6.1 Synthetic Data Model Parameters

The generative dataset model described in section 3.2.3 relies on mean and variance estimates of four multivariate random variables. We computed these to be the sample means and the unbiased sample variances of these variables from the dataset described in section 3.2.1. Table 3-2 is a list of these estimates. Because we derived these estimate from no more than ten samples, they are very uncertain estimates of the true mean and covariance matrix values.

Figure 3-11 visualizes the complete distribution of the synthetic data model. Note that there is a significant class overlap making it impossible to obtain low error values.

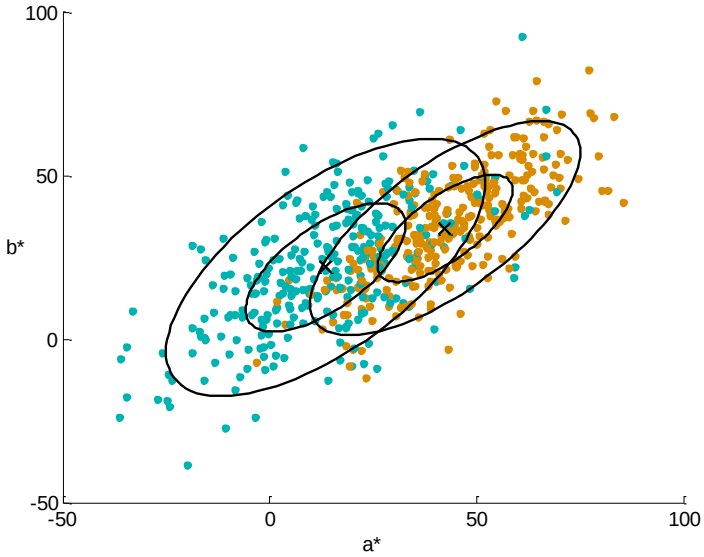

Figure 3-11 Complete Distribution of Synthetic Data

The scatter plot sampled only one sample per class from each source. The scatter plot contains 300 samples per class. The circles represent the distance of one and two standard deviations from the class mean.

Table 3-2 Estimated Parameter Values for Data Distribution Model

The parameters are the means and covariance matrices of the multivariate random variables, μ_2, d, Σ_1' and Σ_2', which we have described in section 3.2.3. These parameters define the distribution of the variables because we assume multivariate random variables to be normally distributed.

Parameters	*Estimated Values*
$\hat{E}[\mu_2]$	$[42.0878 \quad 33.7809]$
$\hat{E}[d]$	$[-29.5821 \quad -12.9007]$
$\hat{E}[\Sigma_1']$	$[195.0305 \quad 174.7942 \quad 126.3716]$
$\hat{E}[\Sigma_2']$	$[161.2562 \quad 130.9111 \quad 110.4004]$
$\widehat{Cov}(\mu_2)$	$\begin{bmatrix} 91.0864 & 73.6737 \\ 73.6737 & 112.6606 \end{bmatrix}$
$\widehat{Cov}(d)$	$\begin{bmatrix} 43.9988 & 19.7659 \\ 19.7659 & 46.8937 \end{bmatrix}$
$\widehat{Cov}(\Sigma_1')$	$10^4 * \begin{bmatrix} 1.6553 & 0.4208 & 0.8168 \\ 0.4208 & 1.1001 & 0.4446 \\ 0.8168 & 0.4446 & 0.5043 \end{bmatrix}$
$\widehat{Cov}(\Sigma_2')$	$10^3 * \begin{bmatrix} 7.2165 & 4.3271 & 4.5027 \\ 4.3271 & 3.7331 & 2.7772 \\ 4.5027 & 2.7772 & 3.4426 \end{bmatrix}$

3.6.2 Monte Carlo Simulation

Table 3-3 and Table 3-4 lists the error statistics for the model selection methods, with folds split-by-source and folds balanced by source respectively. The table description lists the numbers used for the Monte Carlo simulation, and also we randomly resampled the full training and validation sets for every method and for every run. The Monte Carlo simulation uses hyperparameter selection shown in Figure 3-6, to train classification models.

Due to the Monte Carlo simulation being quite time consuming, we only used 200 runs per model selection method. RBF-SVMs used approximately 600 seconds per run. The standard error of the mean values gives reasonable grounds to evaluate whether the difference in performance is significant. Note that the abbreviation SEM refers to the standard error of the mean, which we defined in section 2.5.3.

Table 3-3 Error Statistics of Model Sel. Methods with Folds Split-by-source

The Monte Carlo simulations used 10000 test sources per run ($N_{out} = 10000$), 10 training/validation sources per run ($N_K = 10$), and 100 samples per source ($N_{samples} = 100$). The data were split into ten folds, isolating samples from a source to their own respective fold.

Model Sel. Method	$E[E_{out}]$ (SEM)	$E[E_{diff}]$ (SEM)	$SD(E_{out})$	$SD(E_{diff})$	$Corr(E_{out}, E_{CV})$	N_{MCS}
Gaussian Classifier	0.1841 (.0002)	0.0009 (.0011)	0.0068	0.0332	0.1067	1000
GMM Classifier	0.1905 (.0003)	0.0029 (.0011)	0.0107	0.0342	0.3412	1000
Random Forest	0.2120 (.0009)	0.0018 (.0021)	0.0131	0.0303	0.7190	200
K-NN	0.1927 (.0003)	−0.0096 (.0010)	0.0108	0.0302	0.3165	1000
Linear-SVMs	0.1885 (.0003)	−0.0061 (.0010)	0.0096	0.0314	0.1967	1000
RBF SVMs	0.1885 (.0009)	−0.0104 (.0024)	0.0133	0.0333	0.0965	200
Bucket of Models	0.1862 (.0005)	−0.0009 (.0022)	0.0075	0.0316	0.0061	200

Table 3-4 Error Statistics of Model Sel. Methods with Folds Balanced by Source

The Monte Carlo simulation used 10000 test sources per run ($N_{out} = 10000$), 10 training/validation sources per run ($N_K = 10$), and 100 samples per source ($N_{samples} = 100$). The data were split into ten folds containing an even number of samples from every source. (*) The Gaussian and GMM classifiers have no hyper-parameters and therefore there is no difference in how the out-of-sample errors are computed compared to Table 3-3.

Model Sel. Method	$E[E_{out}]\ (SEM)$	$E[E_{diff}]\ (SEM)$	$SD(E_{out})$	$SD(E_{diff})$	$Corr(E_{out}, E_{CV})$	N_{MCS}
Gaussian Classifier	0.1839 (.0002)*	−0.0156 (.0009)	0.0068*	0.0287	0.0840	1000
GMM Classifier	0.1906 (.0003)*	−0.0244 (.0009)	0.0106*	0.0274	0.2663	1000
Random Forest	0.2132 (.0010)	−0.0229 (.0018)	0.0140	0.0261	0.7315	200
K-NN	0.1931 (.0003)	−0.0253 (.0009)	0.0110	0.0284	0.2196	1000
Linear-SVMs	0.1875 (.0003)	−0.0135 (.0009)	0.0081	0.0295	0.0865	1000
RBF SVMs	0.1893 (.0007)	−0.0330 (.0020)	0.0101	0.0279	0.1702	200

The expected E_{diff} values in Table 3-3, suggests that the cross-validation error is negatively biased for Linear-SVMs and RBF-SVMs, meaning it underestimates the out-of-sample error. The bias for the random forest is too small to evaluate. The other classifiers have near zero bias values. For folds balanced by source, we see similar out-of-sample error statistics.

For both fold-split methods, the expected out-of-sample error has a positive correlation with its standard deviation. Additionally, E_{diff} has a large standard deviation (≈ 0.03). It is likely to both greatly overestimate and greatly underestimate the out-of-sample error. The standard deviation of the learned models out-of-sample error is more moderate (≈ 0.01).

Figure 3-12 contains the E_{out} histograms of the Gaussian and random forest classifier using the split-by-source method. These are also representative for the other model selection methods.

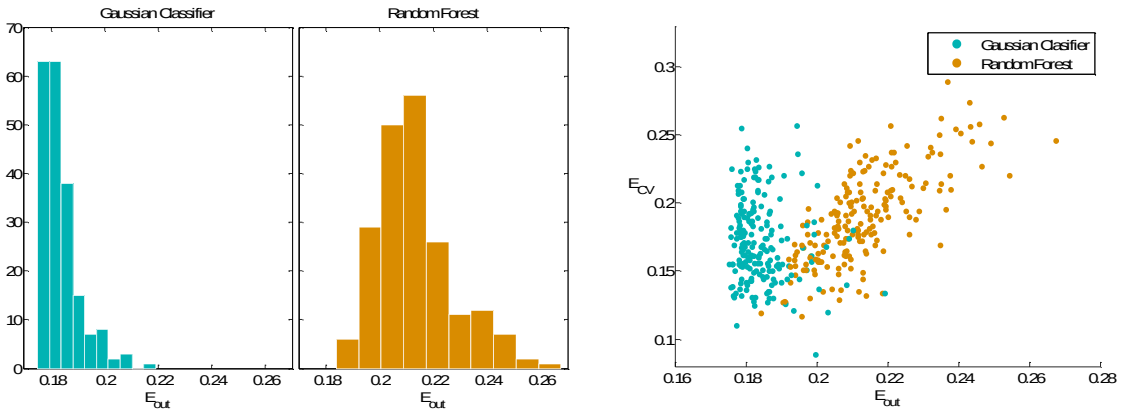

Figure 3-12 E_{out} and E_{CV} of Gaussian and Random Forest Classifier
The two histograms to the left were computed from the MCS results using the split-by-source method. The dots in the scatter plot to the right shows the out-of-sample error and cross-validation error for each run. We choose the Gaussian and Random forest classifiers because they are at both ends of the spectrum in mean and dispersion of the out-of-sample error. Additionally, the random forest classifier has the most extreme correlation.

Table 3-5 contains error statistics when all samples are independent. We calculated the statistics for the Gaussian and k-NN classifier as representative examples of all the model selection methods. The learned models have a lower expected out-of-sample error, including a lower dispersion. Additionally, the error estimate has a lower dispersion ($SD(E_{diff})$).

Table 3-5 Error Statistics of Model Sel. Methods with Independent Samples
The Monte Carlo simulation used 10000 test sources per run ($N_{out} = 10000$). Training/validation data consists of 10 folds ($N_K = 10$), and 100 samples per fold ($N_{samples} = 100$). Every sample were drawn from a new source. SEM is the standard error of the mean of E_{out} and E_{diff}.

Model Sel. Method	$E[E_{out}]$ (SEM)	$E[E_{diff}]$ (SEM)	$SD(E_{out})$	$SD(E_{diff})$	$Corr(E_{out}, E_{CV})$	N_{MCS}
Gaussian Classifier	0.1769 (.0000)	0.0001 (.0004)	0.0011	0.0122	0.0472	1000
K-NN	0.1835 (.0001)	−0.0088 (.0004)	0.0043	0.0124	0.0801	1000

3.6.3 Wound Dataset

For every picture of all ten sources, we extracted features from superpixel segments and superpixel edge samples, using multiple settings for the superpixel region size. We chose superpixel region sizes based on visual inspection, choosing an upper and lower bound to ensure superpixels accurately followed the edges, and that they were not too fine grained. We added smaller step sizes between the upper and lower bound on images that generated few samples, which primarily were due to a low resolution. The

superpixel scales shown in Figure 3-2 is representative for the variation we have used to extract samples. After balancing the number of samples such that we have an equal number of samples per class per source, we end up with 8400 superpixel samples, and 3200 superpixel edge samples. We omitted the bucket of models, model selection method due computational limitations. It would take weeks to compute on a laptop with an Intel i7 4500U processor.

To recap, the greedy forward selection algorithm iteratively adds the feature group that improves the performance the most. The algorithm stops if no additional feature group improved the error.

The error estimates for a specific feature group and classifier in Table 3-6 to Table 3-9, are very unreliable. Although we use 1000 samples for cross-validation, we only have 10 sources, and the variance between sources is large. Also note that the results of Table 3-7, and Table 3-9 highly relies on how well the classifier happened to perform on the mean feature group, and therefore it is a poor indicator to compare different classifiers; the intention of these two tables, is to see how good results we can achieve when we combine two feature groups. The results indicate a large classification error. We could have used more samples, but this would be very computationally expensive, in particular for the RBF-SVMs classifier.

Table 3-6 Error Estimates of Segment Feature Groups

Error estimates of segment feature groups from unbiased cross-validation. Estimates were computed using a subset of 1000 samples from the original 8400-sample dataset.

Feature Group	*Gaussian*	*GMM*	*K-NN*	*Random Forest*	*Linear-SVMs*	*RBF-SVMs*
Mean	*0.342*	*0.149*	*0.229*	*0.227*	*0.166*	*0.220*
St. Dev.	*0.320*	*0.263*	*0.316*	*0.312*	*0.324*	*0.285*
Covariance	*0.380*	*0.347*	*0.355*	*0.324*	*0.340*	*0.361*
Correlation	*0.411*	*0.375*	*0.336*	*0.404*	*0.371*	*0.348*
Skewness	*0.332*	*0.243*	*0.242*	*0.231*	*0.231*	*0.200*
Kurtosis	*0.453*	*0.348*	*0.390*	*0.333*	*0.394*	*0.331*
Entropy	*0.353*	*0.257*	*0.259*	*0.277*	*0.425*	*0.266*
Euclidean St. Dev.	*0.337*	*0.328*	*0.302*	*0.424*	*0.344*	*0.326*
Euclidean Skewness	*0.345*	*0.304*	*0.306*	*0.391*	*0.343*	*0.300*
Euclidean Kurtosis	*0.475*	*0.434*	*0.423*	*0.493*	*0.456*	*0.411*
GLCM Contrast	*0.355*	*0.327*	*0.304*	*0.333*	*0.323*	*0.349*
GLCM Correlation	*0.523*	*0.508*	*0.488*	*0.465*	*0.507*	*0.532*
GLCM Energy	*0.385*	*0.323*	*0.271*	*0.297*	*0.422*	*0.288*
GLCM Homogeneity	*0.337*	*0.317*	*0.334*	*0.352*	*0.331*	*0.394*

Table 3-7 Error Estimates of Segment Feature Groups with Mean

Error estimates of segment feature groups from unbiased cross-validation. Estimates were computed using a subset of 1000 samples from the original 8400-sample dataset. Here we trained models using the feature groups listed in the left column, and the mean feature group.

Feature Group	*Gaussian*	*GMM*	*K-NN*	*Random Forest*	*Linear-SVMs*	*RBF-SVMs*
St. Dev.	0.261	0.166	*0.235*	*0.202*	*0.157*	*0.181*
Covariance	0.278	0.184	*0.235*	*0.201*	*0.171*	*0.175*
Correlation	0.276	0.176	*0.235*	*0.222*	*0.199*	*0.173*
Skewness	0.269	0.176	*0.209*	*0.201*	*0.145*	*0.139*
Kurtosis	0.354	0.210	*0.323*	*0.228*	*0.173*	*0.176*
Entropy	0.308	0.175	*0.251*	*0.199*	*0.197*	*0.179*
Euclidean St. Dev.	0.294	0.158	*0.232*	*0.202*	*0.210*	*0.191*
Euclidean Skewness	0.313	0.147	*0.215*	*0.213*	*0.149*	*0.152*
Euclidean Kurtosis	0.336	0.166	*0.336*	*0.231*	*0.174*	*0.191*
GLCM Contrast	0.311	0.217	*0.247*	*0.208*	*0.153*	*0.163*
GLCM Correlation	0.342	0.203	*0.328*	*0.239*	*0.167*	*0.221*
GLCM Energy	0.344	0.192	*0.240*	*0.211*	*0.182*	*0.188*
GLCM Homogeneity	0.293	0.217	*0.243*	*0.220*	*0.184*	*0.177*

Table 3-8 Error Estimates of Edge Feature Groups

Estimates were computed using a subset of 1000 samples from the original 3200-sample dataset. The Feature groups combines the feature groups extracted from both superpixels of the edge.

Feature Group	*Gaussian*	*GMM*	*K-NN*	*Random Forest*	*Linear-SVMs*	*RBF-SVMs*
Mean	*0.399*	*0.302*	*0.299*	*0.288*	*0.206*	*0.204*
St. Dev.	*0.320*	*0.330*	*0.362*	*0.350*	*0.300*	*0.312*
Covariance	*0.366*	*0.363*	*0.346*	*0.345*	*0.333*	*0.347*
Correlation	*0.382*	*0.453*	*0.380*	*0.424*	*0.386*	*0.346*
Skewness	*0.401*	*0.339*	*0.297*	*0.298*	*0.262*	*0.253*
Kurtosis	*0.499*	*0.472*	*0.420*	*0.401*	*0.515*	*0.397*
Entropy	*0.377*	*0.367*	*0.335*	*0.317*	*0.410*	*0.330*
Euclidean St. Dev.	*0.333*	*0.330*	*0.342*	*0.346*	*0.318*	*0.365*
Euclidean Skewness	*0.366*	*0.390*	*0.356*	*0.355*	*0.330*	*0.320*
Euclidean Kurtosis	*0.495*	*0.506*	*0.462*	*0.489*	*0.499*	*0.468*
GLCM Contrast	*0.482*	*0.386*	*0.353*	*0.334*	*0.363*	*0.419*
GLCM Correlation	*0.442*	*0.483*	*0.492*	*0.488*	*0.469*	*0.493*
GLCM Energy	*0.396*	*0.377*	*0.429*	*0.359*	*0.413*	*0.352*
GLCM Homogeneity	*0.368*	*0.352*	*0.404*	*0.378*	*0.373*	*0.449*
Sobel Mean	*0.357*	*0.370*	*0.346*	*0.374*	*0.350*	*0.359*

Table 3-9 Error Estimates of Edge Feature Groups with Mean

Estimates were computed using a subset of 1000 samples from the original 3200-sample dataset. Here we trained models using the feature groups listed in the left column, and the mean feature group. The Feature groups combines the feature groups extracted from both superpixels of the edge.

Feature Group	*Gaussian*	*GMM*	*K-NN*	*Random Forest*	*Linear-SVMs*	*RBF-SVMs*
St. Dev.	*0.292*	*0.342*	*0.319*	*0.267*	*0.202*	*0.241*
Covariance	*0.335*	*0.337*	*0.300*	*0.277*	*0.204*	*0.231*
Correlation	*0.376*	*0.368*	*0.296*	*0.286*	*0.216*	*0.250*
Skewness	*0.377*	*0.344*	*0.265*	*0.284*	*0.204*	*0.191*
Kurtosis	*0.443*	*0.330*	*0.460*	*0.294*	*0.208*	*0.220*
Entropy	*0.351*	*0.365*	*0.317*	*0.268*	*0.236*	*0.224*
Euclidean St. Dev.	*0.317*	*0.331*	*0.311*	*0.268*	*0.226*	*0.224*
Euclidean Skewness	*0.371*	*0.316*	*0.306*	*0.260*	*0.181*	*0.166*
Euclidean Kurtosis	*0.395*	*0.319*	*0.410*	*0.285*	*0.252*	*0.165*
GLCM Contrast	*0.400*	*0.388*	*0.326*	*0.286*	*0.237*	*0.235*
GLCM Correlation	*0.345*	*0.388*	*0.298*	*0.292*	*0.225*	*0.228*
GLCM Energy	*0.361*	*0.368*	*0.318*	*0.273*	*0.225*	*0.211*
GLCM Homogeneity	*0.356*	*0.409*	*0.332*	*0.292*	*0.204*	*0.252*
Sobel Mean	*0.350*	*0.318*	*0.342*	*0.293*	*0.201*	*0.234*

Table 3-10 and Table 3-11, lists unbiased cross-validation errors using combined hyperparameter and feature selection shown in Figure 3-9. The feature selection part uses the greedy forward selection algorithm. We have chosen to train the models using multiple sample sizes; this provides information on the effect of sample size. We excluded some estimates due to the excessive time required to compute them.

Table 3-10 Error Estimates of Segments using Feature Selection
The table shows unbiased cross-validation errors of segment classes. The models were trained using greedy selection algorithm, for feature selection.

Sample Size	*Gaussian*	*GMM*	*K-NN*	*Random Forest*	*Linear-SVMs*	*RBF-SVMs*
200	*0.249*	*0.256*	*0.240*			
400	*0.246*	*0.210*	*0.196*			
1000	*0.240*	*0.166*	*0.201*	*0.183*	*0.144*	*0.147*
3000	*0.248*	*0.181*	*0.217*			
8400	*0.251*	*0.203*	*0.181*			

Table 3-11 Error Estimates of Edges using Feature Selection
The table shows unbiased cross-validation errors of edge classes. The models were trained using greedy selection algorithm, for feature selection. The Feature groups combines the feature groups extracted from both superpixels of the edge.

Sample Size	*Gaussian*	*GMM*	*K-NN*	*Random Forest*	*Linear-SVMs*	*RBF-SVMs*
200	*0.373*	*0.357*	*0.323*			
400	*0.309*	*0.322*	*0.332*			
1000	*0.323*	*0.316*	*0.295*	*0.258*	*0.202*	*0.226*
3200	*0.324*	*0.274*	*0.300*			

Table 3-12 and Table 3-13 lists the features selected by the greedy forward selection, feature selection algorithm. Here the all folds (and therefore all sources) were used for feature and hyperparameter selection. In contrast, in Table 3-10 and Table 3-11, we had an outer cross-validation loop, so one fold were used for validation. Due to high computation time for training, we reduced the sample size of the random forest, linear-SVMs, and RBF-SVMs classifiers.

With multiple independent tests, we could have obtained the likelihood that a feature is selected. It may be tempting to create multiple tests by varying the number of samples, or count the selected features for every model in the outer cross-validation loop; the problem is that these results have a strong dependence, and therefore, counting the occurrence of feature groups may be misleading.

Table 3-12 Feature Group Occurrence in Segments

Selected feature groups marked with ×. Sample sizes are in parentheses under the classifier name.

Feature Group	*Gaussian (8400)*	*GMM (8400)*	*K-NN (8400)*	*Random Forest*	*Linear-SVMs (1000)*	*RBF-SVMs*
Mean	×	×	×	×	×	×
St. Dev.				×		×
Covariance						
Correlation	×			×		
Skewness	×		×	×	×	×
Kurtosis						
Entropy		×				
Euclidean St. Dev.						
Euclidean Skewness		×				
Euclidean Kurtosis						×
GLCM Contrast				×		
GLCM Correlation						
GLCM Energy			×			×
GLCM Homogeneity	×		×	×		

Table 3-13 Feature Group Occurrence in Edges

Selected feature groups marked with ×. Sample sizes are in parentheses under the classifier name.

Feature Group	*Gaussian (3200)*	*GMM (3200)*	*K-NN (3200)*	*Random Forest*	*Linear-SVMs (1000)*	*RBF-SVMs*
Mean	×	×	×	×	×	×
St. Dev.	×		×			
Covariance					×	
Correlation	×		×	×		
Skewness	×		×		×	×
Kurtosis						×
Entropy				×		×
Euclidean St. Dev.			×			
Euclidean Skewness	×		×			×
Euclidean Kurtosis					×	
GLCM Contrast						
GLCM Correlation						
GLCM Energy						
GLCM Homogeneity						
Sobel Mean					×	×

3.7 Discussion

3.7.1 Out-of-sample Error Distribution

In section 3.1, we defined three questions, which were to be used to compare the model selection methods. The first question concerned the distribution of the out-of-sample error. For the synthetic data model used to test the model selection methods, the Gaussian classifier has the lowest expected out-of-sample error, whereas the random forests classifier has the highest. The remaining classifiers falls somewhere in between, but the differences are not large. We have listed the results in Table 3-3. Table 3-4 contains similar results, only using the balance-by-source method. We will discuss the differences between the two methods in section 3.7.3.

The dispersion of the out-of-sample error correlates positively with the expected out-of-sample error. Again, the Gaussian and random forests classifier are at the extreme ends of the spectrum. We did expect that a slightly larger mean out-of-sample error had a slightly larger standard deviation, but the difference is quite drastic. The Random forest standard deviation is double that of the Gaussian classifier (0.0148 vs 0.0077), whereas the Random forest classifier only has a 1.165 times larger expected out-of-sample error. The Gaussian classifier makes correct assumptions about the distribution of the data, and is therefore able to infer near optimal models from training sets. The Gaussian classifier is only slightly off the optimal decision boundary, but in the immediate region around the decision boundary, both classes are still somewhat evenly balanced. If we venture further from the optimal decision boundary though, one class will be more and more dominant. The random forest classifier tends to lie further off the optimal decision boundary than the Gaussian classifier and therefore small deviations of this boundary will lead to larger differences in the out-of-sample error. This is the most plausible explanation for the large differences in standard deviations of the out-of-sample error that we can think of. Another factor that could have explained the higher standard deviation of the random forest classifier would have been outliers, or a higher skewness towards higher values. However, the data contained no such anomalies; see Figure 3-12.

3.7.2 Cross-validation Error and Out-of-sample Error Dependency

The results in Table 3-3 agrees with the intuition that the cross-validation error tends to have a larger optimistic bias the more extensive the hyper-parameter search is. A larger optimistic bias is conveyed by an increasingly more negative expected error difference ($E[E_{diff}] < 0$). There are two factors contributing to the bias. The first factor comes from optimizing the hyper-parameters on the cross-validation error, which contributes to an optimistically biased estimate. The second factor is that we train the final model using all N_K folds, whereas the cross-validation uses models trained on $N_K - 1$ folds. This contributes to a pessimistic bias, as more data will improve the expected out-of-sample performance.

The Gaussian and GMM classifiers have no hyper-parameter search, so they should have a small pessimistic bias due to the second factor. The Monte Carlo simulations does show a small bias, but it is roughly one standard error of the mean, and therefore not enough to confirm such an argument.

The bias of the random forest method is close to zero, and is as likely to be optimistic as pessimistic. We only test two hyper-parameter options, and so that should only contribute to a small optimistic bias. The pessimistic bias due to the difference in folds used for training may contribute to an overall pessimistic bias.

The k-NN, Linear-SVM and RBF-SVM model selection methods have more extensive hyper-parameter searches, and they all have an optimistic bias ($E[E_{diff}]$ equals -0.0119, -0.0090, and -0.0104 respectively). It is important to account for this bias, when comparing model selection methods; this is why we chose to use nested cross-validation to obtain unbiased error estimates in our bucket of models method.

Somewhat discouraging, are the results on how poor the error estimate (E_{CV}) is. The difference in error (E_{diff}) has a large dispersion ($SD(E_{diff}) \in [0.297, 0.374]$). Additionally, the model selection methods only have weak correlations between the error estimate and the out-of-sample error. The random forest method has a moderate correlation, but unfortunately, it is also the worst performing classifier. The scatter plot to the right of Figure 3-12 shows the dependence between the error estimate and the out-of-sample error. Although the out-of-sample error has a lower dispersion, we do not know whether our obtained error estimate has an optimistic or pessimistic bias. For example, we may obtain an optimistic bias ($E_{CV} = 0.13$), but the out-of-sample error is as likely to be 0.08, as 0.21. The encouraging part is that even though the error

estimate may be ways off, the learned model is not so far off the optimal model. That is for our 2-dimensional synthetic data model.

In Table 3-5, we computed the error statistics for model selection methods when all samples were independent. We used the same amount of training data as with the split-by-source method in Table 3-3, where the within-fold data were dependent. The variation in error statistics indicates that training with within-fold dependent data is similar to training with a reduced set of independent data. With independent samples, we obtain better models. The mean and dispersion of the out-of-sample error is lower. Additionally, the error estimates are more accurate. The k-NN model selection method has a slightly lower bias, indicating it is less able to over-fit the validation data, but more importantly, the error estimate has a lower dispersion. For example, the standard deviation of the difference in error for the Gaussian classifier is 2.4 times lower ($SD(E_{diff}) = 0.0122$ for independent data, and otherwise $SD(E_{diff}) = 0.0297$).

3.7.3 Split-by-source Versus Balance-by-source

When folds are balanced by source (Table 3-4), we get an optimistically biased cross-validation error, regardless of model selection methods. However, the out-of-sample errors seems to be unaffected, meaning we could still use the biased cross-validation error for optimization of hyper-parameters. A bigger problem is that the bias differs between model selection methods. The Gaussian and GMM classifier are both unbiased using the split-by-source method, but when we balance samples by source they have very different biases (-0.0123 and -0.0278 respectively). Using nested cross-validation would not fix this problem, because the fault lies with splitting dependent samples across several folds. We would be unable to pick among model selection methods without bias. In other words, we would tend to pick one model over another, because it has a more optimistic bias, not because it has the lowest out-of-sample error. That is something we do not want.

3.7.4 Wound Dataset and Feature Groups

The classification results of superpixel segments and superpixel edges in section 3.6.3 are not very promising. No combination feature group and classifier combination achieves a low classification error for neither segment nor edge. Note that the error

results have a large degree of uncertainty, and so we limit our discussion to the most significant trends.

For segments (Table 3-6), the GLCM correlation has the weakest classification error; it roughly has a 50% classification error, indicating zero discriminative power. The next worst is the Euclidean kurtosis with a classification error in the 41-49% range. None of the Euclidean feature groups achieves a low error, but they also only consist of one feature, in contrast to three. The mean feature group has the lowest classification error, and skewness the second lowest.

An interesting trend is that the Gaussian classifier performs poorly. This is in contrast to the synthetic data model, where the Gaussian classifier performed very well. Given how the Gaussian classifier is based on making strong assumptions on the data, it is not surprising to see its unfavorable results on data, which obviously does not follow that distribution.

When we combine the mean feature group with another feature group (Table 3-7), we see results similar to the only using the mean feature group. Combining mean with another strong feature group tends to slightly improve performance, whereas combining mean with a weak feature group tends to slightly reduce performance or have no impact. Combining the mean and skewness feature groups has the overall best performance, whereas combining the mean and the Euclidean skewness feature group has the second best performance.

The k-NN classifier has the largest performance deterioration when we add a weak feature group. This is not surprising, as the algorithm uses the nearest neighbors to classify, and feature with bad discriminative power are given equal importance to determine who the nearest neighbors are. Another explanation is that the classifier performs more poorly for higher-dimensional data.

For edges (Table 3-8 and Table 3-9), we see the same trends as for segments, but the overall classification errors are higher. The edge-specific Sobel mean feature group has an average performance.

The spatial-based Sobel and GLCM feature groups does not achieve any particularly good performance, but the resolution, and scale of the wound images have a large variation. These feature groups have the potential to be much more reliable with a fixed feature group and scale.

We conclude this section by discussing an alternate feature selection method. Classifiers such as the RBF-SVMs, are time consuming to train. It already has two real-

valued hyperparameters to optimize, and is notoriously slow to train for large sample training sets. To reduce the computational complexity, we can use filter methods to first rank all features. Instead of optimizing the entire feature set using a wrapper method, we only have to use a wrapper method to select the number of features to include. Algorithm 3-3 shows the step of this algorithm. One special case where this algorithm could be useful is for random forest classifiers. The hyperparameter of the random forest classifier ranges from one to the number of features, therefore more models are trained for larger feature sets. When more models are trained for data with many features, it leads to a bias towards selecting many features. A feature-ranking filter avoids this dilemma.

4 Object Recognition and Segmentation of Wounds

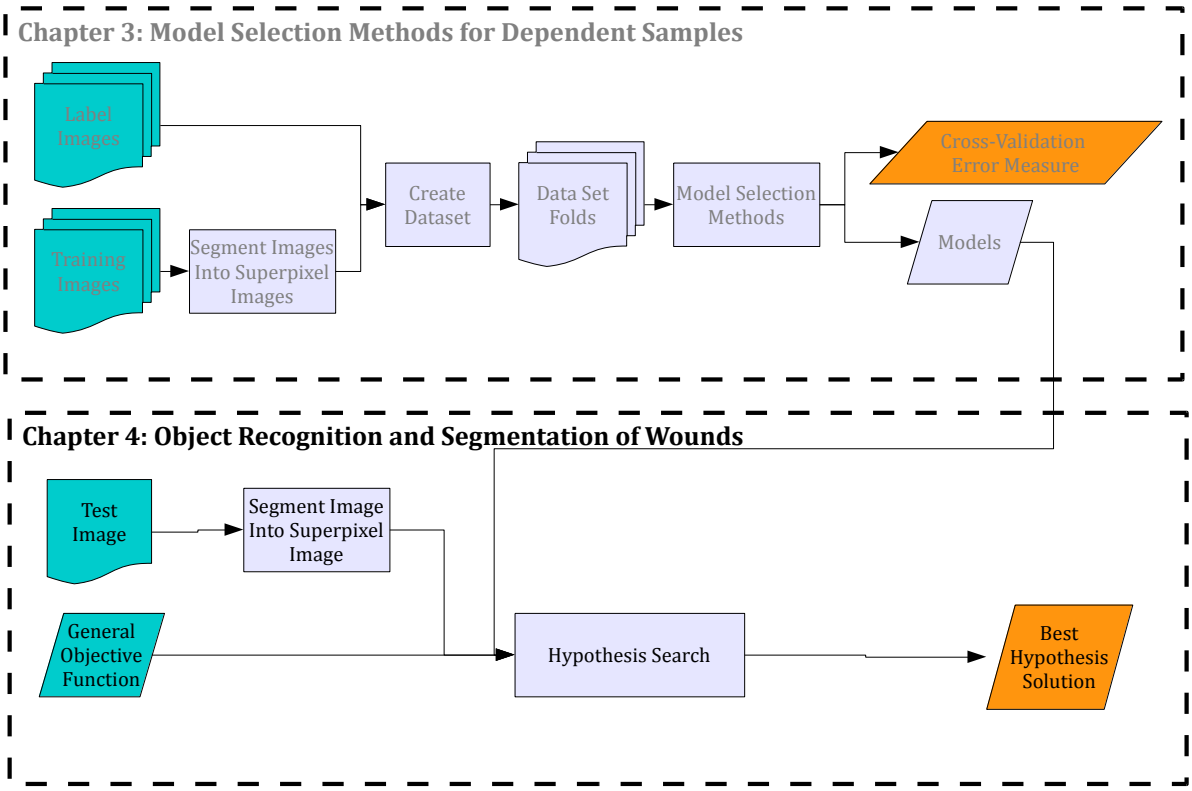

Figure 4-1 Chapter 4 Flowchart

Object recognition and segmentation of wounds, is a complex task. The input is an image, and the output is a description of the object outline, both of these are high dimensional. There have been some success in learning object class labels directly from raw image data using convolutional neural networks, but that problem has a single discrete output value. For segmentation problems, the norm is to use segmentation algorithms. However, most segmentation algorithms rely on strong assumptions, such as segmenting the most salient structure, yet these assumptions are rarely definitive. We attempt to solve object recognition and segmentation using a hypothesis optimization framework. We constrain the optimization problem to search hypotheses that are subsets of superpixels obtained from SLIC superpixel segmentation. We define an objective function using textural and shape properties characteristic of wounds, and genetic algorithm to search for the optimal hypothesis. Our results demonstrates that a hypothesis optimization framework can solve object recognition and segmentation of wounds. These results are important because, given the flexible nature of hypothesis

optimization they demonstrate that hypothesis optimization is a strong candidate for general-purpose machine-learnable object recognition and segmentation.

4.1 Introduction

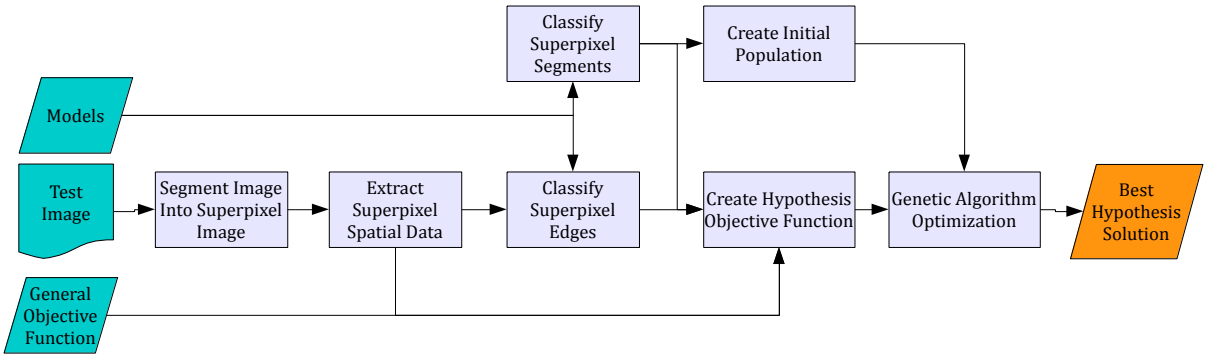

Figure 4-2 Detailed Chapter 4 Flowchart

The flowcharts highlights the subject of this chapter. Our object recognition algorithm relies on hypothesis optimization. To enable solving the segmentation problem as an optimization problem, we restrict the hypothesis to be a subset of all superpixels in a test image. From the classified segments, classified edges, superpixel spatial data, and general objective function we make a specific objective function that is only dependent on the hypothesis. To obtain a hypothesis solution, we use a genetic algorithm to optimize the specific objective function. We use the superpixel segment classifications to provide a reasonable initial population.

In the previous chapter, we studied model selection methods for dependent samples. In this chapter, we use those superpixel-segment and superpixel-edge models as a component in the complete object recognition and segmentation algorithm. In the previous chapter, we used Monte Carlo simulations to obtain strong comparative results. In this chapter, we use a small set of test images. The exact details of our proposed algorithm are simply initial proposals. In no way do we claim them to be optimal. The important results, is that we can use hypothesis optimization to solve object recognition and segmentation of wounds, and why this is so important.

Figure 4-2 shows a flowchart of our proposed algorithm. The algorithm takes an input image, and outputs a predicted hypothesis solution of the outline of the wound. The algorithm relies on classification models for classifying superpixel-segments and superpixel-edges, and a general objective function separating wounds with a low value, from non-wounds with a high value. The figure description explains the main components of the algorithm. We covered the superpixel segmentation and classification in the previous chapter; the superpixel objective function and genetic algorithm optimization are covered in sections 4.2, and 4.3 respectively.

In the chapter abstract, we mentioned how this hypothesis optimization framework is learnable. It is theoretically possible to learn a function that maps the raw high dimensional data of an image to an explicit outline of the wound object. There are multiple ways to express the outline, but all of them are high-dimensional outputs. This function would be as follows:

$$f: \mathbb{R}^N \rightarrow \mathbb{R}^M, \qquad N, M > 1 \tag{4.1}$$

Learning this function directly is not very feasible for either machines or humans. There is one method, which shifts the complexity to the input side of the object recognition and segmentation problem without making any assumptions. Instead of learning a function, that explicitly expresses the object outline we can learn a function that implicitly expresses the object outline by being the minimum of an objective function. The input is now the image data, and the hypothesis. The output is a single-dimensional hypothesis score value, see equation (4.2).

$$\Gamma: \mathbb{R}^{N+M} \rightarrow \mathbb{R}, \qquad N, M > 1 \tag{4.2}$$

This shift in complexity is what characterizes hypothesis optimization, and we believe that hypothesis optimization is a feasible approach to solve the object segmentation problem. We claim that it is easier to learn the objective function (Γ) in equation (4.2). Having learned the objective function, we solve the object segmentation problem by finding its global minimum as shown in equation (4.3). The input data is x, and h denotes hypotheses.

$$hypothesis^* = f(x) = \underset{h}{\operatorname{argmin}}\, \Gamma(x, h) \tag{4.3}$$

Note that many segmentation algorithms do pose segmentation as an optimization problem, like active contours for instance [1]. In addition, we can pose other segmentation algorithms in terms of what objective function they optimize, and how they attempt to optimize it.

4.2 Method: Hypothesis Objective Function

The objective function assign a high value to hypotheses that are not wounds, and a low value to hypotheses that are wounds. That is the intention of the objective function, but constructing such an objective function is not an easy task.

A strictly correct objective function must always return a better score for true hypotheses than for false hypotheses. Let $\mathcal{H}_{True}$ be the set of all true hypotheses, which also means that the complementary set $\overline{\mathcal{H}}_{True}$ is the set of all false hypotheses. A strictly correct objective function must satisfy the following criterion:

$$\Gamma(h_{True}) < \Gamma(h_{False}), \qquad \forall \begin{matrix} h_{True} \in \mathcal{H}_{True} \\ h_{False} \in \overline{\mathcal{H}}_{True} \end{matrix} \tag{4.4}$$

Even humans ability to recognize wounds adhere to this criterion, and we do not expect our objective function to be strictly correct. In fact, wound images are varied, and complex; our objective function does perform, but not to the level required of an autonomous robot operating system.

The objective function consist of simple shape and textural properties, typical of wounds. The objective function uses a combined weighted sum and weighted product model of these properties. The weakest aspect of our approach is that we rely on hand-hand tuning these weights. In section 4.6.3, we discuss a method to optimize the objective function automatically.

4.2.1 Shape Properties

We make use of five simple shape properties in the objective function. These prioritize large non-jagged hypotheses, without hulls. They punish hypotheses with too many disconnected components, and hypotheses that collide with the image edge. First, we explain what we mean by components and hulls. Then, we define sub-properties we make use of, and finally we define the shape properties.

A component refers the number of connected components. Superpixels in the hypothesis that are connected comprise one component. We can consider superpixels to be nodes in a graph, and let these nodes have undirected edges towards all its neighbors. Then a connected component is simply a strongly connected component of that graph. We exploit this by using Matlab graph functions. Further, a hull is an interior part of one of the components of the hypothesis. These hulls are components of the inverse hypothesis. The inverse hypothesis components may also contain background components, containing superpixels on the image border. We can easily separate the hull and background components, by testing if any of their superpixels are on the image edge.

We derive the shape properties from even simpler properties. These sub-properties are:

- A_{hull}, is the total area of all the hulls of the hypothesis.
- A_{hyp}, is the area of the hypothesis.
- $N_{components}$, is the number of components of the hypothesis.
- P_{hyp}, is the outer perimeter of the hypothesis. The outer perimeter excludes the hull perimeter.
- $P_{hyp-border}$, is the outer perimeter of the hypothesis that also lies on the image border.
- A_{wound}, is the total area of the wound-labeled superpixels. This is not a shape property, but the derived property in equation (4.7), is.

The shape property $\gamma_{hullRatio}$, is the ratio of hulls. When there are no hulls, its value is zero. γ_{size}, is optimal when the hypothesis is as large as possible, and ranges between zero and one. $\gamma_{jaggedness}$, is optimal when the perimeter is small compared to the filled hypothesis size. This should prevent jaggedness. γ_{border}, is optimal when no superpixels in the hypothesis lies on the image edge. $\gamma_{connectivity}$, aims to prevent too many components and starts to increase exponentially when the number of components is surpasses five. These shape properties are defined as follows:

$$\gamma_{hullRatio} = \frac{A_{hull}}{A_{hyp} + A_{hull}} \tag{4.5}$$

$$\gamma_{connectivity} = 1.2^{\max(0, N_{components} - 5)} \tag{4.6}$$

$$\gamma_{size} = 1 - \frac{A_{hyp}}{A_{hyp} + A_{wound}} \tag{4.7}$$

$$\gamma_{jaggedness} = \frac{P_{hyp}^2}{A_{hyp} + A_{hull}} \tag{4.8}$$

$$\gamma_{border} = \frac{P_{hyp-border}^2}{A_{hyp} + A_{hull}} \tag{4.9}$$

4.2.2 Textural Properties

Our objective function uses two textural properties. The first property γ_{wound}, indicates how well the hypothesis matches the predicted wound-labeled superpixels. The second property $\gamma_{woundPerimRatio}$, indicates how well the hypothesis matches the predicted wound-labeled superpixel edges. Note that we use the term textural, because we obtain the labels by classifying textural features of superpixels. The first property relies on two sub-properties. These two sub-properties are:

- $\gamma_{wprecision}$, is the ratio of the hypothesis area predicted to be a wound.
- $\gamma_{wrecall}$, is the ratio of the wound-predicted area selected by the hypothesis.

Now, we can combine these two sub-properties as defined in equation (4.10).

$$\gamma_{wound} = 1 - \sqrt{\gamma_{wprecision}\gamma_{wrecall}} \tag{4.10}$$

It is not obvious why we define γ_{wound} as follows, but this definition has some attractive traits. Obviously, the property should be at its maximum when the hypothesis simultaneously covers all wound area, and only consists of wound area. Contrarily the property should be at its minimum when the hypothesis covers none of the wound area, and consists of no wound area. The property should have an intermediate value when the sub-properties have intermediate values. In addition, the two sub properties are also symmetrical. More formally, these traits are as follows:

- $\gamma_{wound} \in [0,1]$
- $\begin{matrix}\gamma_{wprecision} = 0 \\ \gamma_{wrecall} = 0\end{matrix} \Leftrightarrow \gamma_{wound} = 1$
- $\begin{matrix}\gamma_{wprecision} = 1 \\ \gamma_{wrecall} = 1\end{matrix} \Leftrightarrow \gamma_{wound} = 0$
- $\begin{matrix}\gamma_{wprecision} = 0.5 \\ \gamma_{wrecall} = 0.5\end{matrix} \Rightarrow \gamma_{wound} = 0.5$
- $\gamma_{wound} = f(\gamma_{wprecision}, \gamma_{wrecall}) \Rightarrow f(a,b) = f(b,a)$

It is important that the γ_{wound} property require both of the sub-properties to have decent values. If we had just added the two sub-properties together, then one of them would take priority over the other property.

The second textural property is simply the ratio of the hypothesis perimeter that classifies as wound edges. Note that every superpixel edge are classified for either of the two directions, and that the second property, $\gamma_{woundPerimeterRatio}$, is the ratio of wound edge along the hypothesis perimeter, when we use the correct direction.

4.2.3 Combining the Object Properties

In equation (4.12), we combine the shape properties in section 4.2.1 to the Γ_{shape} property using a weighed sum model. Similarly, we use a weighted sum model to combine the textural properties in section 4.2.2 to the $\Gamma_{textural}$ property, as seen in equation (4.11). We create objective function by combining Γ_{shape} and $\Gamma_{textural}$ using a weighted product model that we define in equation (4.13).

$$\Gamma = \Gamma_{fit}^{\beta_1}\Gamma_{shape}^{\beta_2} \tag{4.11}$$

$$\Gamma_{textural} = \beta_3\gamma_{wound} + \beta_4(1 - \gamma_{woundPerimRatio}) \tag{4.12}$$

$$\Gamma_{shape} = \beta_5\gamma_{hullRatio} + \beta_6\gamma_{connectivity} + \beta_7\gamma_{size} + \beta_8\gamma_{jaggedness} + \beta_9\gamma_{border} \tag{4.13}$$

We hand tuned the weights using example images. We used the image *W2*, from results section 4.4. The few other images used to hand-tune the weights are not present in this thesis, as we were unable to verify the source of the images, and we were therefore unable to obtain permission to use them. The weights we used in the section 4.4 results are listed in Table 4-1.

Table 4-1 Objective Function Weights
The table lists the objective function weights used for the results in section 4.4.

Weights	*Weight Values*
β_1	*1.0*
β_2	*4.0*
β_3	*1.0*
β_4	*1.0*
β_5	*1.0*
β_6	*0.01*
β_7	*0.7*
β_8	*0.01*
β_9	*0.1*

Note that the objective function is undefined for empty hypotheses, and therefore, we simply return a penalty value of 10.

Unfortunately, we made a small error when computing the results in section 4.4. In the MATLAB code, our $\Gamma_{textural}$ property had two other terms, which were both set to 0.5, when they should have been set to zero. We show this $\Gamma_{textural}$ version in equation (4.14). We do not show comparisons here, but the two $\Gamma_{textural}$ versions are

still similar. The biggest difference between them is that the added terms renders the segment classifications more important than the edge classifications.

$$\Gamma_{textural} = \beta_3 \gamma_{wound} + \beta_4 (1 - \gamma_{woundPerimRatio}) + 0.5(1 - \gamma_{wprecision}) + 0.5(1 - \gamma_{wrecall}) \quad (4.14)$$

4.3 Method: Genetic Algorithm Optimization

4.3.1 Overview

The wound-hypothesis optimization problem requires an optimization algorithm capable optimizing high-dimensional binary search problems with arbitrary objective functions. There are competing algorithms, such as particle swarm optimization, but Matlab has a reasonable documented genetic algorithm out of the box. We found no strong indications of genetic algorithms yielding inferior solutions when compared to other algorithms. Therefore, we decided to use genetic algorithms.

Besides using a custom initial population, we stick to the default settings. The population size is 200, which is default for high-dimensional binary search problems. We produce new members using a scattered crossover function, which means that we randomly select a gene value from either of the parents. For mutation, we use uniform mutation at a mutation rate of 0.01. This means that after crossover, any gene has a 1% chance of having its binary value flipped.

Although more sensible genetic algorithm options exists, we refrained from such algorithm option optimization. Our dataset is limited, and any further optimization would only add to the bias problem. In section 4.5.3, we discuss how these options affected the results, and propose improvements to the optimization algorithm.

4.3.2 Initial Population

A good strategy to reduce generations required to find a good solution, is employ a specialized initialization procedure. Our initialization procedure uses information from the superpixel segment classifications. We can think of the superpixels classified as wound, as an initial wound hypothesis. We create the initial population by repeating

20 base members. These 20 base members are themselves split into four groups, consisting of five members each. The four groups are as follows:

1. We pick the five largest components.
2. At random, we pick half of all components, for each of the five members in this group.
3. At random, we pick half of all wound-labeled superpixels, for each of the five members in this group.
4. We exclude on the components for each of the five members, starting with the largest.

Note that by components, we refer to sets of wound-predicted superpixels that are connected. The results shown in Figure 4-4 are examples of wound-predicted superpixels. Also, note that if there are less than five components, then group one and four will randomly replicate its own current members to attain five members.

4.4 Results

We base the results section on five example images that shows the entire wound and have minimal obstruction. The first image is the example image of a pig, which we used in section 1.2. The other four are the least obstructed wounds in our image dataset. For every image, we show the results using both a small and a large superpixel region size. The actual region size depends on the image, as resolutions between them vary. We place results using large superpixels in the left column, and results using small superpixels in the right column. For convenience, we name the images, W1, W2, W3, W4, and W5.

4.4.1 Superpixel Segmentation and Classification

This section contains the superpixel segmentation, and superpixel segment and edge classification results. Here, we build on the results shown in the previous chapter. We train superpixel segment, and superpixel edge classifiers using linear support-vector machines with feature and hyperparameter selection.

Figure 4-3 shows the segmentation results. We selected the superpixel sizes manually, and to minimize bias we chose them before viewing their classification results (W2 is an exception). We ended up with the following region sizes:

W1: 40, and 20
W2: 25, and 15
W3: 60, and 30
W4: 40, and 20
W5: 40, and 20

We selected these region sizes based on two primary principles. Firstly, the number of superpixels for the wound should not be too few. Otherwise, a single correctly classified superpixel may outcompete the true hypothesis, as single superpixels has naturally good shape properties. The second principle is to avoid a very small region size. If the number of pixels within a superpixel falls below a certain measure, then its derived features become unreliable, and the classification error rises. Initially we used 10 and 20 region size for W2, but the region size of 10, resulted in much poorer classification performance. The SLIC regularization parameter is 40 times the region size, which we have already discussed in section 3.2.1.

Figure 4-4 shows classification of the superpixel segments. The error rate for linear-SVMs were at about 15% with classes balanced according to Table 3-1. The error rates here should be about the same, but one thing to note is that the error rate varies greatly for each image. The most distinguishing pattern of the wound-labelled superpixels, displayed in white, is that they have a high recall rate of the true wound superpixels. However, their precision is low. About half of the wound-predicted superpixels in W2, W3, and W4 are misclassifications, which is caused by a background to wound ratio that is much higher in the Figure 4-4 images, compared to the balanced datasets used in the previous chapter.

Figure 4-5 show the superpixel edge classifications. The wound edge to non-wound edge ratio is much higher than for the balanced datasets that we used to train the classification models. The results shows many edges misclassified as wound-edges, something we did expect. The disappointing result is that misclassifications correlate strongly with the superpixel segment misclassifications. For instance, the W3 superpixel-segment and superpixel-edge classifications both indicates a structure to the top left of the wound, to be a better wound hypothesis candidate.

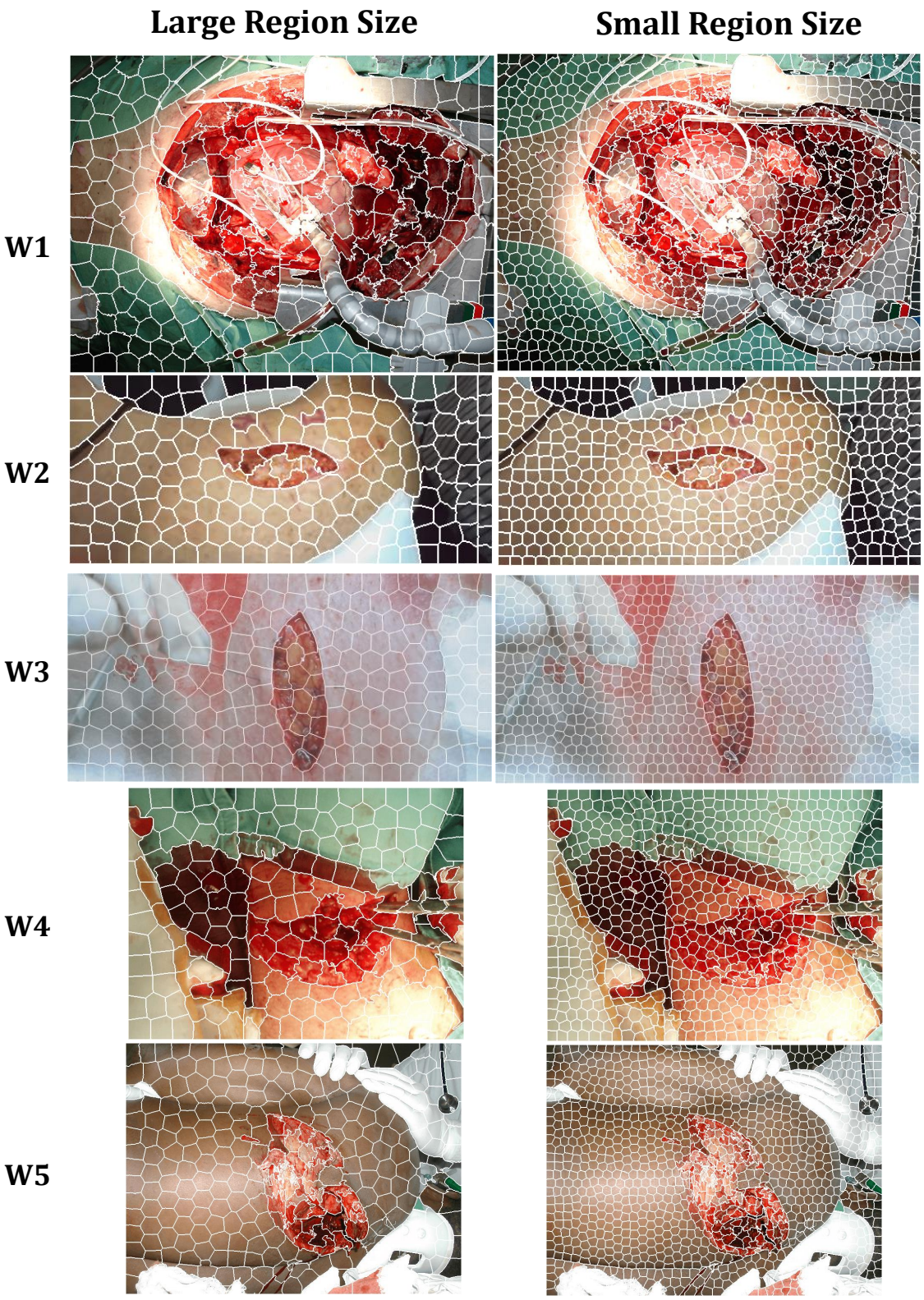

Figure 4-3 Superpixel Segmentation

Superpixel segmentation of five wound images using both a large and small region size.

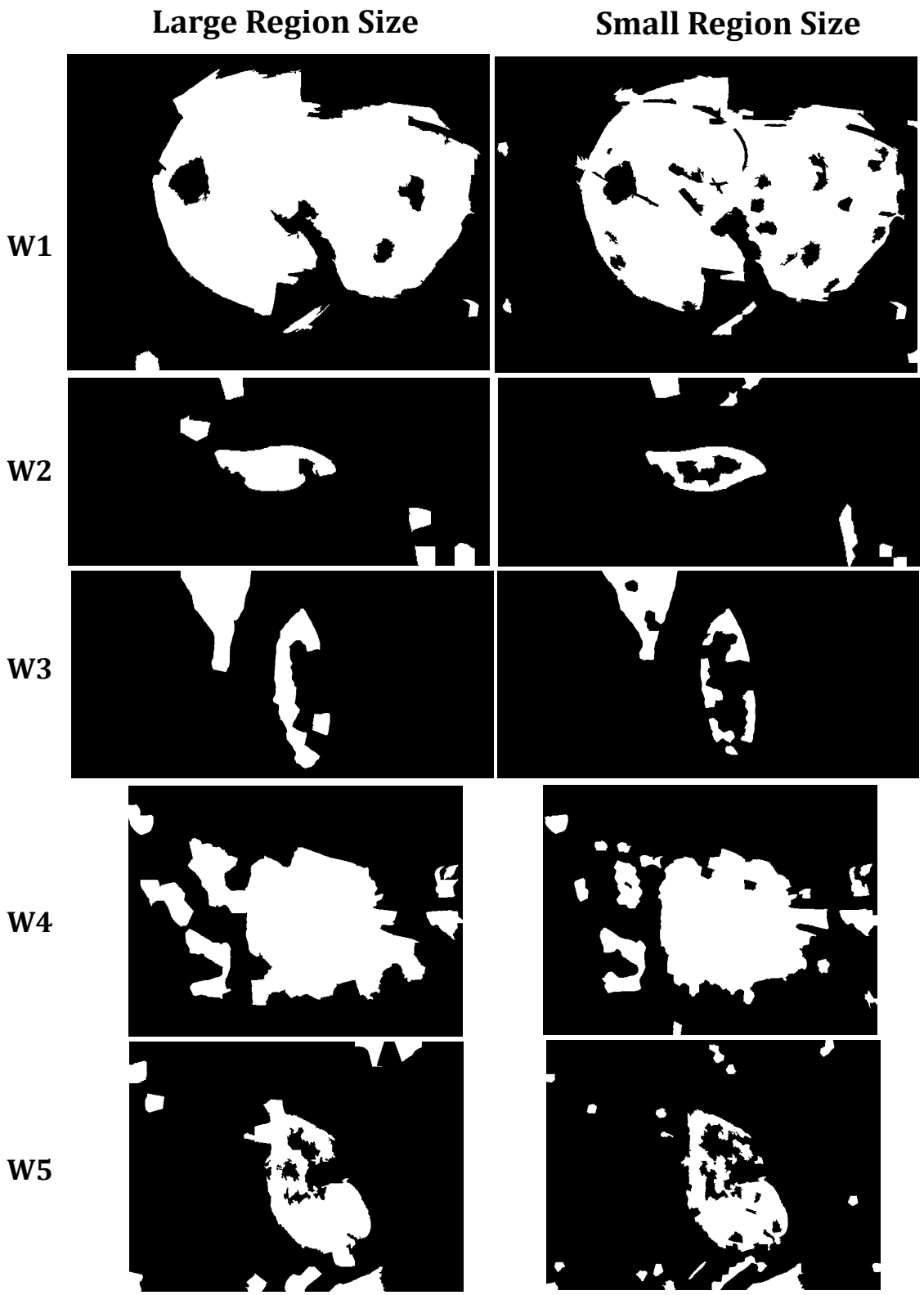

Figure 4-4 Superpixel Segment Classification

The figure shows a superpixel segment classification of five wound images. White superpixels were classified as wound superpixels, and black superpixels were classified as non-wound superpixels.

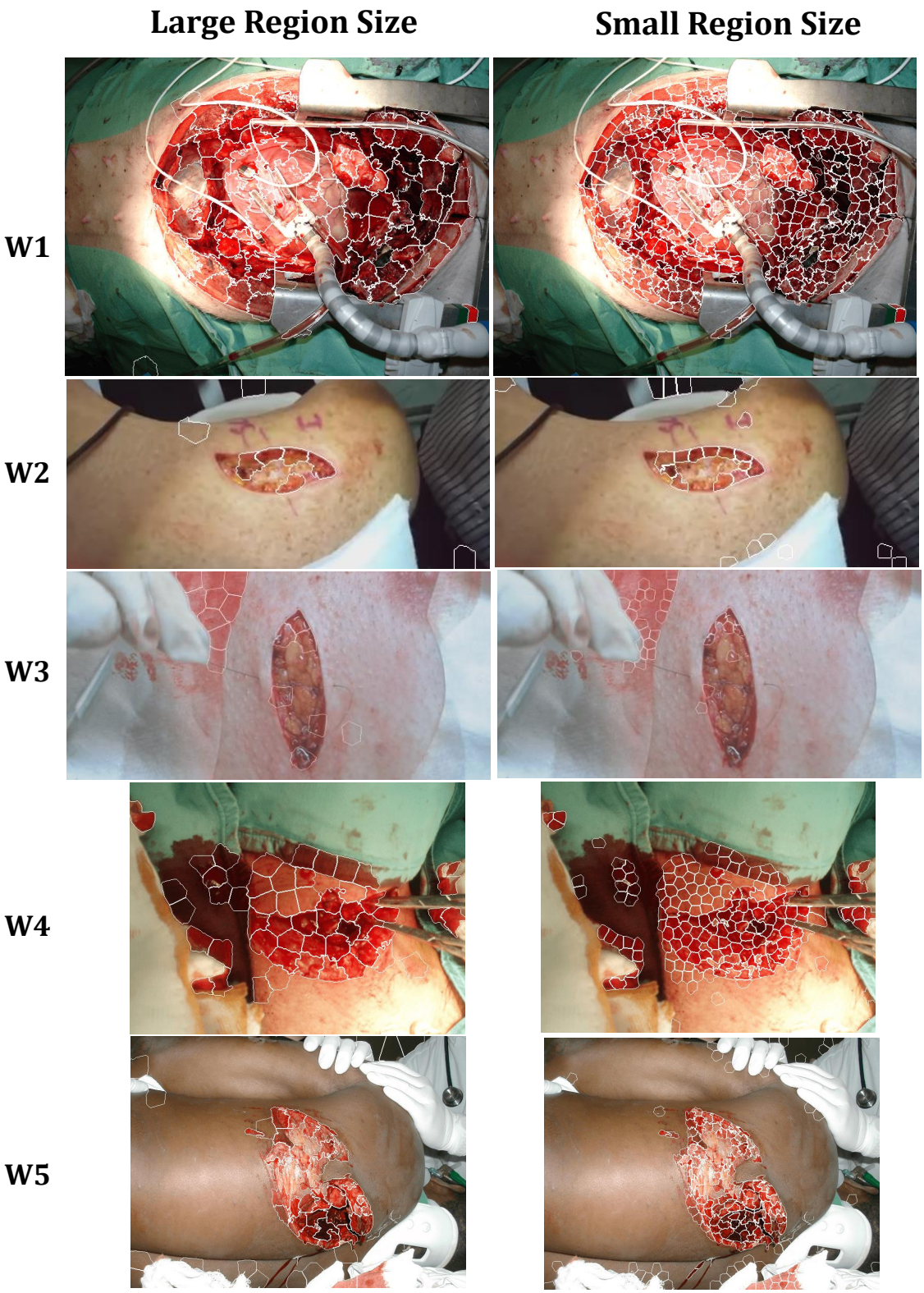

Figure 4-5 Superpixel Edge Classification

The figure shows a superpixel edge classification of five wound images. Superpixel edges classified as wound edges are indicated by drawing the superpixel edge perimeter in white. The thick white lines indicates that both superpixel edge directions were classified as a wound edge. This figure may be inaccurate in paper version, and sadly, the direction of the wound edge is not displayed.

4.4.2 Genetic Algorithm Optimization

Figure 4-6 displays the true hypotheses, which we use as a reference, to compare them with the predicted hypotheses in Figure 4-7. The predicted hypotheses are the best results obtained from genetic algorithm optimization using the configuration described section 4.3.

Figure 4-8 shows the genetic algorithm optimization progress per iteration. Of particular importance are objective function scores for the predicted hypothesis, and the true hypothesis. The predicted hypotheses can be incorrect due to failing to converge to the optimal value, or because the objective function itself, is incorrect. When the true hypothesis (black curve), is below the predicted hypothesis (orange curve), then the predicted hypothesis converged to a bad solution. In the reverse scenario, the objective function yields a better value to the incorrect predicted hypotheses, than the true hypotheses.

The predicted hypotheses for W3, and for W5 using a large region size, have a worse objective function value, than the true hypotheses. Here, the genetic algorithm fails. Even a perfect objective function will not help, if the optimization algorithm fails to find the optimal hypothesis. W1 is a case for the reverse scenario. The predicted hypothesis of W1 has a much better hypothesis value, than the true hypothesis. W1 is a partially obstructed image, and the true hypothesis contains multiple components, which is the cause for the poor objective function values.

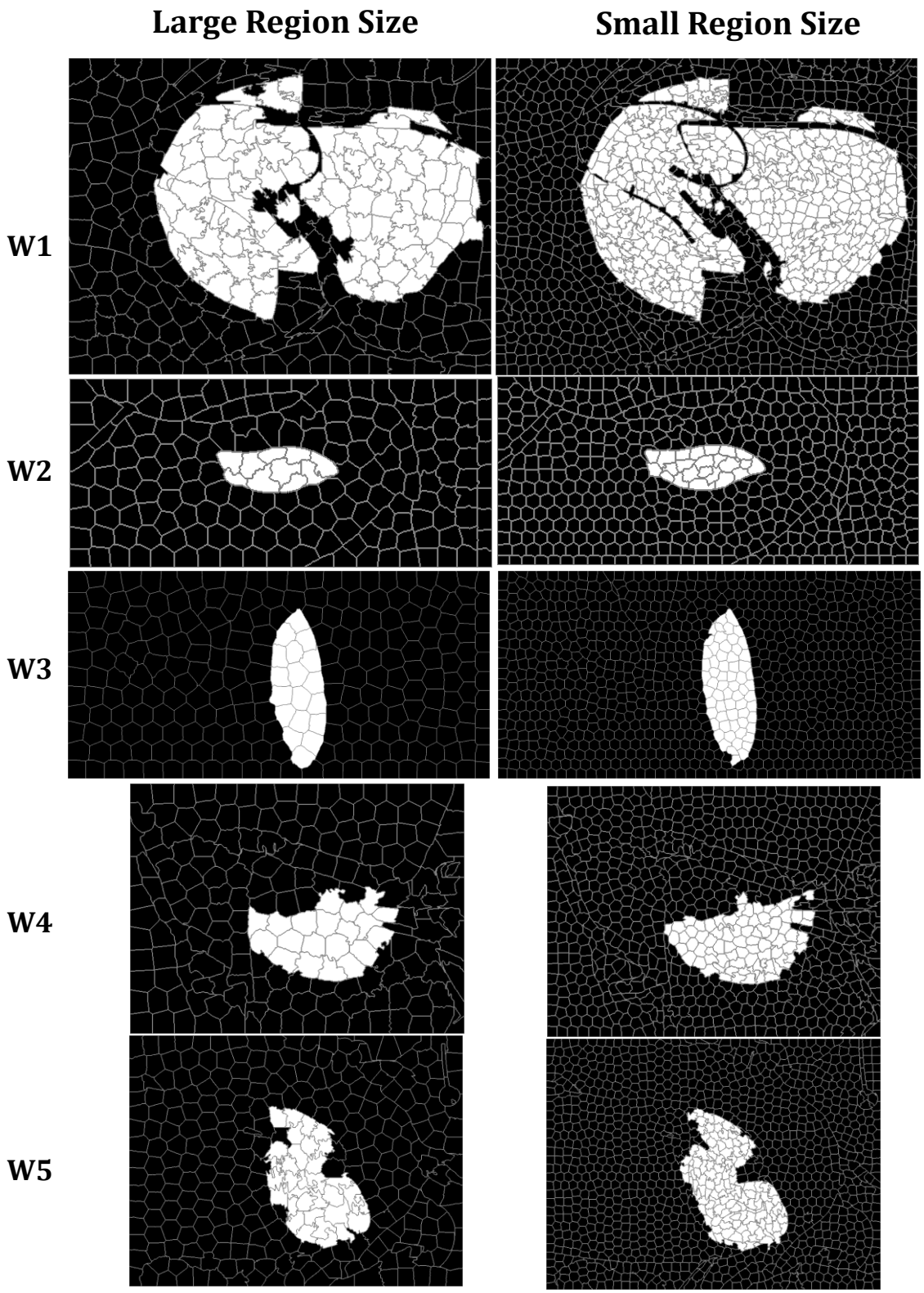

Figure 4-6 True Hypotheses

The figure shows the true hypotheses for five wound images. The true hypotheses were derived from hand-labelled images, and the label of the superpixel is the majority class among the pixels within a superpixel. The true hypotheses are indicated by the white superpixels.

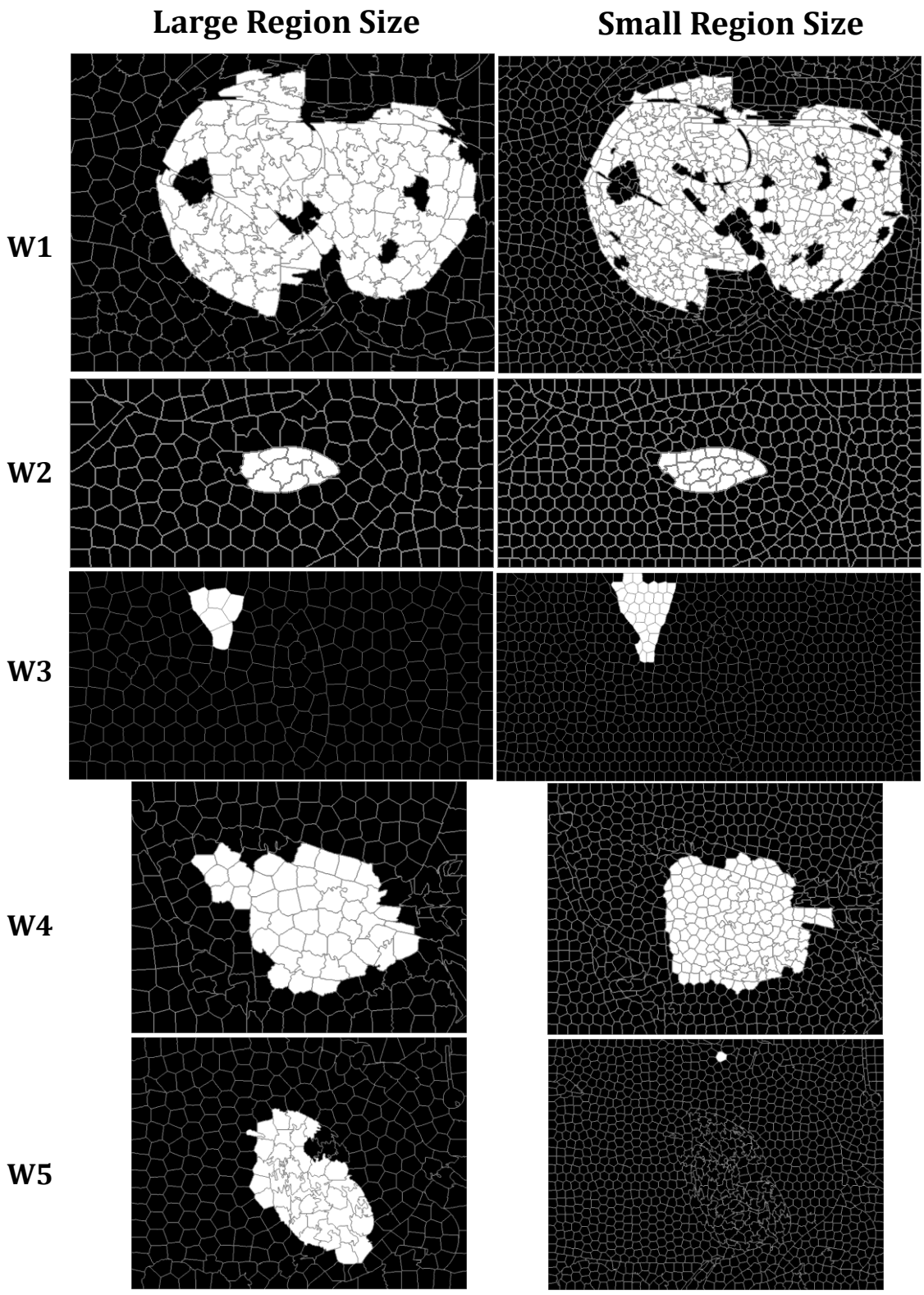

Figure 4-7 Predicted Hypotheses

The figure shows the predicted hypotheses for five wound images. The hypotheses were computed using 1000 iterations of genetic algorithm optimization, on the the objective function defined in section 4.2.

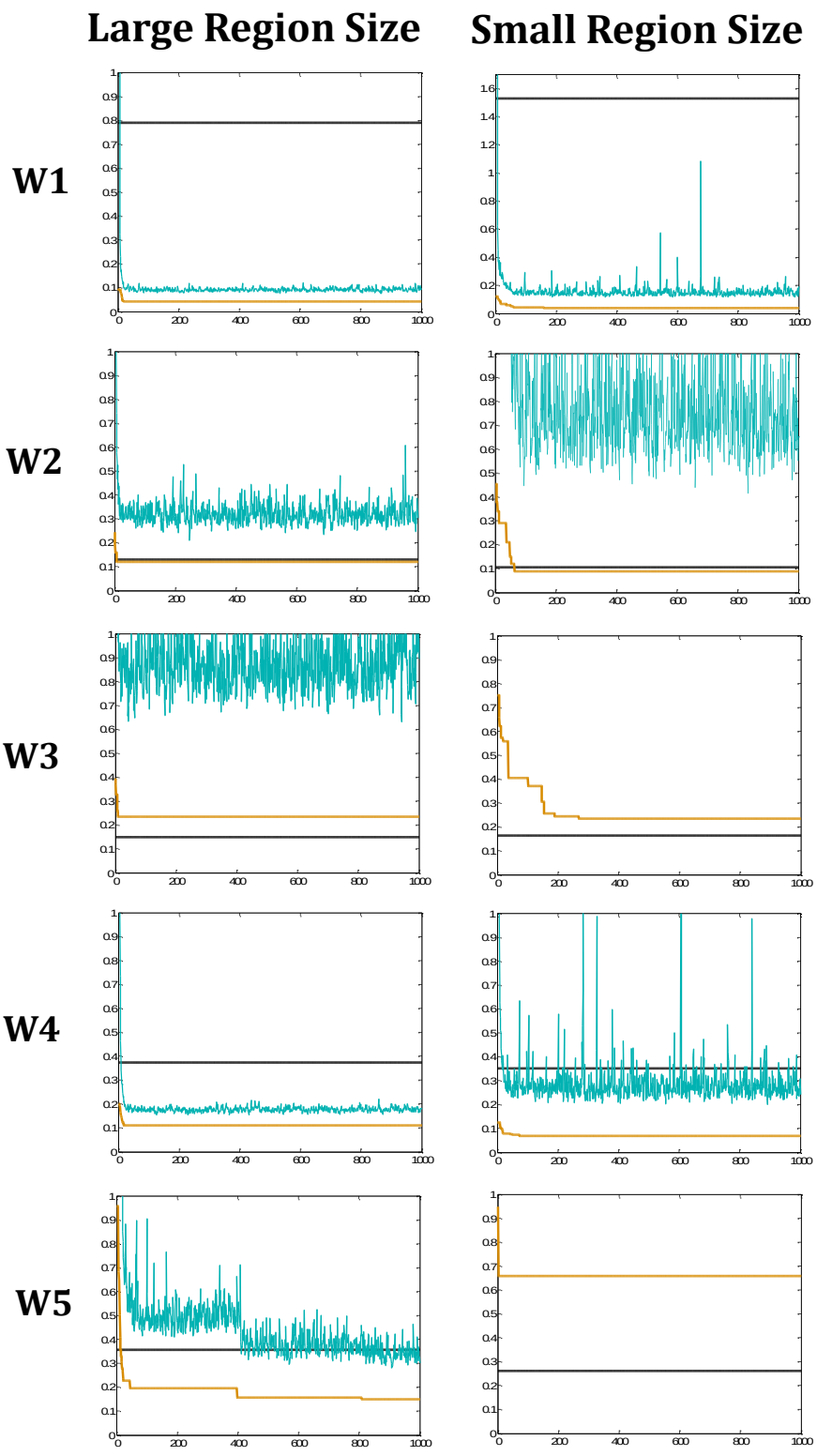

Figure 4-8 Hypothesis Optimization

The figure shows genetic algorithm statistics per iteration for the five wound images. The straight black line shows the true hypothesis value. The orange curve shows the best currently found hypothesis for every iteration of the genetic algorithm. The turquoise curve shows the mean objective function values per iteration. The mean value is sometimes out of bounds of the [0,1] *graph range.*

4.5 Discussion

This chapter lacks quantitative results. We provided quantitative results for the classification performance in the previous chapter. However, quantitatively evaluating the correctness of object hypotheses is more complicated. There are proposed segmentation correctness measures, but the small dataset would render any estimate unreliable. Therefore, our results are limited to a set of examples, and that limits our ability to make conclusive statements. Because of the unreliable results, we warn the reader, that our statements should be interpreted as indications.

4.5.1 Results Evaluation

In this section, we analyze the results of the five wound images from section 4.4. We describe how well superpixel segmentation adheres to edges. We describe the superpixel-segment and superpixel-edge classification results. Further, we compare the predicted hypotheses to the true hypotheses, and finally we compare their objective function values. If the objective function value for the true hypothesis is below the predicted hypothesis, then that means that the optimization algorithm failed as it found a suboptimal value. Contrarily, if the objective function value for the true hypothesis is above the predicted hypothesis, then that means that the objective function is incorrect, as the optimization algorithm found a better yet false hypothesis. We devote a paragraph to each image covering the evaluations we just listed.

The W1 image shows open abdominal surgery on a pig. The superpixel images reliably capture the wound edge, but they do not properly capture some of the small medical instruments obscuring the wound. The superpixel-segment classifications are very accurate, and it does classify most wound-edges as such. However, it also classifies most superpixel edges within the wound as a wound edge, and that is not an intended behavior. We labeled the true hypothesis to be the wound, excluding any obstructing medical instruments. The predicted hypotheses adhere well to the edges, but they struggle with obstructed parts. Some obstructed parts belong to the hypothesis, whereas others do not. As we have not addressed obstructed wounds, we did expect this type of behavior. Some of the hulls in the hypotheses may be due to un-optimized hypotheses. In particular, W1 with a small region size has many superpixels and 1000 iterations may not have been enough. In addition, illogical hulls are characteristic of

not having performed enough iterations. The true hypothesis has a bad objective function score, but this is probably because the true hypothesis has a split from a medical instrument, and therefore counts as multiple components, which increases the objective function value.

The W2 image shows a simple unobstructed incision. Superpixel segmentation adheres well to edges. Segment and edge classification have some errors. In particular, the edges only partially classify wound edges as such, and they classify other superpixel edges as wound-edges. The predicted hypotheses are decent, but they lack some wound superpixels on the left side, which were probably a result of the incorrect edge classifications. The objective function value is slightly higher than the predicted value, and so the fault does not lie with the optimization algorithm.

The W3 image shows a simple unobstructed incision. The image contains white mostly white objects, but also a bloodied cloth piece resembling the textural properties of a wound. Superpixel edge segmentations have a high error rate, only recalling about half the wound object and classifying superpixels of the bloodied piece of cloth as wound-superpixels. The edge classifications appear to be useless. For both superpixel sizes, the algorithm predicts the bloodied piece of cloth as the wound object. What is interesting is that the predicted hypotheses have a higher (worse) value than the true hypothesis. Therefore, the objective function may be correct for this instance, but the genetic algorithm converges to a sup-optimal local minima.

The W4 image shows a wound with some obstruction. Superpixels adhere well to edges, and it recalls most wound-superpixels, but it also labels many other superpixels as wound-superpixels. Similarly, it classifiers most wound edges correctly, but also classifies many non-wound edges as such. The resulting hypotheses covers the wound, but also include many additional superpixels in the wound hypothesis. Here the predicted objective function value is lower than the objective function value for the true hypothesis, and therefore the objective function must be incorrect for this instance.

The W5 image shows an unobstructed wound with an irregular wound edge. The superpixel segmentation adheres well to the wound edges. The superpixel classifications recalls most of the wound, and has a good precision. The superpixel-segment classifications recalls most of the wound edge, but also falsely classifies many internal edges in the wound as wound-edges. The predicted hypothesis with a large region size is a smoothed version of the true hypothesis, and it does have a lower

objective function value due to the smooth properties. The predicted hypothesis with a small region size fails completely. It predicts a single superpixel to be the most optimal hypothesis, and here the optimization algorithm that fails because the predicted hypothesis is far from optimal. We have seen similar results in preliminary tests, and we suspect that the mutation and crossover function causes the optimization to stagnate if the hypotheses in the population at any point reaches a stage where the hypotheses are close to null.

4.5.2 Comments on the Hypothesis Objective Function

The performance of the objective function, it is not affected by the number of pixels, but rather by the number of superpixels. To calculate the hypothesis area for instance, we only need a vector containing the area values for each superpixel, and sum the value of the superpixels flagged in the hypothesis. To find the components of the hypothesis, we can use an array to express superpixels pairs by flagging neighboring superpixels. This array is a graph of the edges, and we just need to find the connected components of this graph. Note that the performance decreases with larger hypothesis sizes. This is because there are more calculations concerning superpixel values in the hypothesis, than outside the hypothesis.

With a quadratic function, we could in theory, always find the global optimum, but it is not possible to compute the number of components, and the hulls, using a quadratic function. Therefore, we have to resort to optimization algorithms that do not guarantee the global minimum. In general, it may be impossible to construct a valid quadratic objective function. Being restricted to quadratic functions, may hinder the discovery of crucial object properties.

4.5.3 Comments on the Genetic Algorithm

We treat the genetic algorithm in Matlab like a black box. Therefore, we mostly stick to the default settings, which depends on the data type and dimensionality of candidate solutions. Presumably, the Matlab staff found these settings to be suitable for a wide variety of optimization problems independent from ours. They may have relied on results from earlier research, or hand tuned the settings using multiple optimization problems. We might achieve better results by tuning the settings to our optimization

problem, instead of handling the algorithm as a black box. However, we already have to decide the superpixel region size, the objective function, and the classification algorithms. We want to evaluate the objective function, and therefore we seek to isolate it from improving the optimization algorithm. For instance, a wound hypothesis is typically a blob of multiple neighboring superpixels. Therefore, it seems logical to modify mutation and crossover, such that they are more likely to produce candidate solutions with blobs. Assume this strategy proves successful. We certainly optimized the genetic algorithm for the training set. However, we may have fallen into the trap of optimizing the data, rather than the general problem. To properly compare the default and custom optimization algorithms, we would need a large independent test set. Our dataset does not satisfy those requirements.

To find additional wound objects, we can rerun the genetic algorithm optimization while also omitting superpixels belonging to the previously discovered hypotheses. We must also verify if a proposed hypothesis is satisfactory. We could determine an objective function threshold-value, but a binary classifier acting as a hypothesis-verifier may prove more successful.

4.5.4 Comments on Hypothesis Optimization

Our most important result is not the effectiveness of the proposed algorithm. We know it to be flawed. However, the object segmentation approach we use shows great opportunity. Here we summarize some of the strengths and weaknesses this method has. Some of the strengths are:

- In the general form, hypothesis optimization does not put any limitations on what object segmentation tasks it can solve. It simply shifts the output complexity of the object segmentation function to the input side, see equation (4.2).
- Hypothesis optimization evaluates hypotheses of the object, and we can seamlessly combine global and local object properties in the objective function.
- Arguably, discovering the objective function of an object category is simpler than discovering a segmentation algorithm for that object category. Therefore, if it is feasible to discover a correct segmentation algorithm, then it should also be feasible to discover a correct objective function. However, if it is feasible to

discover an objective function, then it may still not be feasible to discover a correct segmentation algorithm.

Lacking a clear comparative algorithm makes defining weaknesses an obscure prospect. However, we list some of the important obstacles faced when implementing a hypothesis-based algorithm that:

- Discovering the correct objective function, does not imply that we can find the optimal hypotheses within satisfactory computational constraints.
- The hypothesis search space of all pixels is large. Hypothesis optimization requires some method to reduce the search space complexity. We solve this by constraining the hypothesis to subsets of SLIC superpixels SLIC superpixels would not be suitable for objects with thin structures, so they would require some other algorithm to reduce the search space complexity.
- Hypothesis optimization requires an optimization algorithm, and optimizing non-convex objective functions is an NP-hard problem. Presumably, any correct objective function is non-convex for most object categories, and therefore these segmentation problems themselves are NP-hard.

4.5.5 Preventing Bias

It is important to ensure that the results we obtain are valid, and therefore we must be aware decisions that can produce biased results. In particular, we must be aware of optimistic bias. A separate independent test set would be the most suitable means to obtain unbiased results. In chapter 3, we relied on outer cross-validation loops, with completely separate wound sources per fold. This is valid, but hyperparameter and feature selection, are not the only decisions we had to make. Here we list decisions that could have introduced bias:

- The wound images in this thesis were retrieved by attempting to find suitable images from the internet. With an autonomous surgical system, we have some control over picture quality and environment, but we have to analyze images continuously, rather than a select few, chosen for publication. Additionally, we looked for unobstructed open wounds, and an autonomous surgical system will have to deal with nearly closed wounds, and obstructed wounds.

- We chose to use the SLIC algorithm. It performs well with dense objects, but SLIC superpixels does not align well with very thin objects, and wounds or incisions are typically thin objects when they are nearly closed.
- We chose the SLIC regularization parameter based on the wound images in this thesis. It is possible that this regularization rule gave unusually good results for our wound images.
- We chose to use linear-SVMs in the section 4.4 results. We chose this based on the results in the model-selection method results in the previous chapter. However, we used two test images from here as part of those error results.
- We only used one of the test images (W2) to design the objective function and select its weights, but we did know what our test images looked like, and presumably, they indirectly influenced our perception of what a wound object is.

It is important to note that this is still in an experimental stage. The algorithm proposed in this thesis is in no way finalized, and there are no direct comparisons with other algorithms. Therefore, bias that could yield slightly optimistic results are tolerable. Still, one should minimize bias when possible. Here we summarize some of the methods we used to reduce bias:

- We used outer cross validation loops to obtain unbiased error measures of classifiers with feature and hyperparameter selection. We did choose a good candidate among multiple classifiers (linear-SVMs), but this should only introduce a small, expected bias.
- In the classifications for the chapter 4 results, we trained multiple models. For every image (source) used in the model selection method, we trained models that excluded those sources throughout the entire model selection method process. At no point were the classification results for an image trained using the data from that image. Note that by training we also consider data used to select hyperparameters and features as training data.
- We decided region sizes before viewing their classification results.
- We used both a small and large region size per image to limit the possibility of results with an overly optimistic bias. For instance, we could obtain near-perfect results for W2 and W3 if we were to fine-tune the region size.

- We chose the final genetic algorithm configuration and the objective function before computing the test results. An exception is the already discussed W2 image, which we used in developing the objective function, genetic algorithm configuration.

4.6 Future Work

This section extends the discussion, proposing how to improve the algorithm, extend the algorithm, or include it as one part of a more complete vision system. We have not clarified how the proposed algorithm is useful by itself, and currently and the algorithm does not perform to the level required in an autonomous surgical system. However, our method treats object recognition and segmentation as an optimization problem nearly void of assumptions. Consequently, this approach has high potential, and we outline some proposals on how to improve upon object recognition and segmentation.

4.6.1 Better Object Properties

The shape and textural properties described in section 4.2.1 and 4.2.2 are merely initial proposals. Surely, more fitting object properties must exist. An interesting direction is object properties making use of similarity measures. One type is similarity measures between superpixels either within a hypothesis, on the hypothesis edge, or outside the edge. Another type of similarity measure is comparing the hypothesis to a database of wound objects. There are many possible similarity measures. We propose a histogram-based similarity measure, uniquely defined as follows:

- The similarity measure computes is a measure of difference between two segments.
- For each color channel, we compute a similarity measure between the histograms of the two segments. The total similarity measure is the mean of the three individual histogram similarity measures.
- Consider every pixel value in a histogram to be individual objects. The dissimilarity between two histograms is the minimum energy required to move objects within the first histogram such that they resemble the other histogram.

Moving an object one, unit pixel-value, requires one, unit energy. Moving an object two, unit pixel-values, requires two, unit pixel-values, and so on.

We can compute the histogram similarity measure directly, so we do not rely on optimization. For superpixel-based similarity measures, you only have to compute the similarity measures once, as they are not dependent on the object hypothesis.

4.6.2 Post Processing the Hypothesis Boundary

Restricting hypotheses to superpixels reduce the optimization complexity, but the superpixels may not have correct boundaries. However, we can refine the predicted hypothesis by optimizing the object boundary using finer superpixels, or even directly optimizing on pixels. To make this possible we constrain the search to only optimize the hypothesis on the current object boundary.

4.6.3 Learning the Hypothesis Objective Function

So far, we have manually tune the objective function weights. We used textural and shape properties in a combined weighted sum and product model. Here we present a proposal to learn the objective function automatically.

A hypothesis is a candidate solution of the optimization problem. A true hypothesis is the optimal solution, and a false hypothesis is a non-optimal candidate solution. The grouping of superpixels that corresponds to the actual wound object is the true hypothesis (H^*). Any other grouping of superpixels are false hypotheses (H'). We denote true and false hypotheses of image i as H_i^* and H_i', and we let N be the number of images. Further we let $\mathcal{H}_i$ be the set of all hypotheses in image i. To find the optimal hypothesis, we make use of an objective function and a genetic algorithm to search through hypotheses of an image, and find the optimal solution according to the objective function. We let $f(H)$ be the objective function, taking a hypothesis as an input. Obviously, this function must be minimal for the optimal solution:

$$H_i^* = \arg\min_H f(H \in \mathcal{H}_i) \tag{4.15}$$

We pick intuitive shape and textural properties, and combine these in a combined weighted sum and weighted product model. The problem is that we must learn these weights from a small set of images. We may also want to test other property combinations that do not fit within a weighted sum of weighted product model.

Let $f(H,W)$ be the objective function, where the second argument, W, is the weights. Further, we create a dataset of false hypotheses, where H'_{ij} denotes false hypothesis j of image i. For simplicity, select an equal amount of false hypotheses from every image (M). We make pairwise binary comparisons between true and false hypotheses using a classification function with zero for successful comparison and one for an unsuccessful comparison, as shown in (4.16). The optimization problem itself is defined in equation (4.17). We can solve the optimization problem using a search algorithm (such as a genetic algorithm) to find satisfactory solutions.

$$g\left(H_i^*, H'_{jk}, W\right) = \begin{cases} 0, & f(H_i^*, W) < f\left(H'_{jk}, W\right) \\ 1, & f(H_i^*, W) \geq f\left(H'_{jk}, W\right) \end{cases} \tag{4.16}$$

$$W^* = \arg\min_W \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{M} g\left(H_i^*, H'_{jk}, W\right) \tag{4.17}$$

Note that there are also alternative comparison functions, such as the one defined in equation (4.18).

$$g\left(H_i^*, H'_{jk}, W\right) = f\left(H'_{jk}, W\right) - f(H_i^*, W) \tag{4.18}$$

An unresolved challenge is how to select the false hypotheses. A possible method would be to pick an initial set, and then as the search algorithm learns new weight, we test what solutions the genetic algorithm finds. When these hypotheses are false, we use them in our set of false hypotheses. We could implement a ranking between false hypotheses, assuming that later generation (false) hypotheses are better than early generation (false) hypotheses. The logic is that the weights are supposed to improve, thereby finding continually better hypotheses.

To recap, training an objective function contains the following steps:

1. Create an initial set of false hypotheses (the true hypotheses are fixed).
2. Create a sequence of weights and false hypotheses. The steps per iteration are:
    a. Optimize the objective function weights based on the current set of hypotheses. We defined this optimization problem in equation (4.17).
    b. Compute optimal hypotheses according to the new objective function weights.
    c. Verify if the current optimal hypotheses are true or false hypotheses.
    d. Create a new set of false hypotheses, using the previous false hypotheses, and the new optimal yet false hypotheses.

3. Stop upon reaching some criteria. If training were successful, then the objective should have global minimums for true hypotheses.

4.6.4 Hypothesis Optimization using a Comparison Function

We have so far shown that we can solve object segmentation as a hypothesis optimization problem. The method takes a hypothesis as an input, and outputs a hypothesis score. The goal is that the true hypothesis should have the globally minimal hypothesis score. Still, object segmentation, is a complex problem. We could consider the possibility of using a comparison function instead, as it may be more feasible to create a comparison function than an objective function.

With a comparison function, we take two hypotheses as an input, but we only need to output a single binary value. A comparison function does have issues, especially if it does not consistently rank hypotheses. A comparison function would also require a modified optimization algorithm.

4.6.5 Probabilistic Classification of Local Structures

In this section, we propose a probabilistic approach to handle prediction of local structures such as superpixel-segments and superpixel-edges. Classifying local structures is of limited value when there is a class overlap. The object recognition algorithm uses these classifications to infer the segmentation of the object, but that does not help, when the classifications contradict the truth.

It is important that the predictor can discriminate between wound-related and other structures co-occurring in wound images. At the same time, it must also be able to predict that something is non wound-related structures, when the data is extremely sparse.

We propose to combine outlier detection, and regular probabilistic classifiers. For every class we train a conservative outlier detection model. We also train a probabilistic classifier. Predicting a sample contains the following logical steps:

1. We check if the sample is an outlier of any of the classes.
2. We compute the class probabilities of the sample using the probabilistic classifier.

3. We combine the results in step 1, and 2. We set the outlier classes to zero probability, and we scale up the probabilities of the other classes such that they sum to one.

4.6.6 Context Based Multi-object Recognition

Object segmentation of a single object, is by itself of limited value. An autonomous surgical robot system will require image analysis algorithms capable of recognizing and segmenting multiple objects simultaneously. Here we propose how to use our object-specific algorithm as a component of a multi-object recognition and segmentation algorithm.

The multi-object algorithm contains sub-algorithms, including object-specific segmentation algorithms for every object category, and an object detection algorithm. The proposal is a high-level example of a multi-object recognition algorithm, but other variations could also work. The proposal is as follows:

1. An efficient object detection algorithm detects objects of every category within the image, and estimates their location. False hypotheses are tolerated as the intention is to provide initial hypotheses.
2. Perform object segmentation to obtain object outlines for every initial hypothesis. Filter away unfavorable hypotheses.
3. Initiate high-level object reasoning algorithm, starting a sequence of image hypotheses. Stops when results are deemed satisfactory.
    a. Detect possible conflicts between object hypotheses. Object overlap can indicate a conflict.
    b. Determine if current hypotheses infers the expectation of other specific objects, possibly in certain locations. One utilization is when objects are expected to co-occur.
    c. Using knowledge obtained in steps 3a, and 3b, initiate new object hypothesis searches. The new searches accept weaker conditions, but may also contain regional constraints, and other constraints.
    d. Pick a subset of the object hypotheses found so far, to form the currently best image hypothesis.

4. Output the final image hypothesis, containing possibly multiple object outlines and multiple object categories.

The object-specific algorithm essentially hypothesizes that the object has a high chance of existing in the image, and then evaluates how well the observed data matches that hypothesis. For instance, this justifies why we can assume wound segments to have a 50% probability, whereas in a real surgical setting, this probability may be much lower.

Note that the individual object segmentation and verification algorithms could result in duplicate object categories for the exact same object outline. This is acceptable, as the multi-object recognition algorithm can resolve such conflicts. In fact, this can be an advantage. It may be impossible to determine an object category without considering the context, and then the best an object segmentation algorithm can hope to accomplish, is to determine whether the object category could have produce the observed visual properties. For instance, one object can act as both a chair and a table. We can solve this problem type if both the chair and table object segmentation algorithms recognize the object, because the high-level algorithm may be able to resolve the conflict.

4.6.7 Tracking

An autonomous robot operating system will not just need to recognize objects. It will also need to track these objects with a short response time. The objective with object recognition is to recognize an object instance of a general class; with tracking, the objective is to re-recognize the same object in other instance of time.

Our object recognition algorithm is currently not fast enough for tracking purposes. An unrelated tracking algorithm however, does not share the objective function of the recognition algorithm. It might be best to first pursue a robust object recognition algorithm, and then convert it to a tracking algorithm. A logical step is to modify the objective function by adding an object similarity measure, and the likelihood that the object can be observed in the hypothesized location. These modifications will strengthen the current optimal hypothesis, something which we ca not guarantee with an unrelated tracking algorithm. Inconsistent objective functions between recognition and algorithms, does not only mean that false hypotheses might

be more optimal. The algorithms may also disagree on exactly where the object boundaries are

Tracking can make greater assumptions about the object, which we can exploit to improve both predictive and computational performance:

- We can limit hypothesis search to a small area around where we expect the wound to be.
- We can reduce image resolution to whatever is required for the specific task.
- We can increase superpixel size, as long as it captures the structures of the wound outline.
- We can make assumptions on where the object may be located from one image to the next.
- If wounds overlap between consecutive images, we can use a search algorithm that takes advantage of this assumption.

A more general method to improve performance is to use an efficient parallel search algorithm, and throw powerful hardware at it.

5 Conclusions

5.1 Model Selection Methods for Dependent Samples

In chapter 3 we studied model selection methods for dependent samples. The results are mostly typical of model selection methods with independent samples, but there are also some anticipated exceptions. Simultaneously we use this theory to train classification models required for chapter 4.

Our datasets consists of superpixel segments, and superpixel edges within a small set of images. The samples within these images are dependent, which we argued for in section 3.2.2. You will get biased error results when dealing with dependent samples. Potentially this could also negatively affect classifier performance through poor hyperparameter, feature, or classifier choices. We did not observe that in our results, but it is a theoretical possibility. Luckily, these dependent samples are known groups, and by placing these groups within their own folds (Referred to as the Split-by-Source method), we obtain unbiased error results. One other anticipated exception, is that a large sample size does not mean the same as a large sample size when you have independent samples. Combined, the number of sources and the number of samples per source influence the generalizability of the data. Few sources with many samples per source will behave as if the number of samples were less. We can clearly observe this phenomenon in the error estimates. Although the total number of samples are high, our error estimates have an uncertainty as if the number of samples were significantly lower.

We evaluate combined feature and hyperparameter selection for superpixel segments and superpixel edges using the wound image dataset. No classification algorithm achieves very low results. We chose linear-SVMs because it had the lowest error results for both superpixel segment classification (0.144), and superpixel edge classification (0.202). The exact error estimates are unreliable, and another test may have put another classifier ahead. Having optimized classifier choice on these error measures, we can expect them to be optimistically biased. Therefore, we will not list some final error measure for our classifiers. We have no final unbiased error measures, and that would not by itself a very important result, because it is completely dependent on the specific dataset we used.

The limited wound dataset puts some limitation of how strongly we can interpret those results, but in agreement with the Monte Carlo results, they do indicate that the model-selection methods reliably selects rational hyperparameter values and feature subsets. We argue that these model selection methods should be a core part of any complete classification algorithm. When we use automated hyperparameter and feature selection procedures (model selection methods), we train classifiers in a controlled environment. This reduces risk of biased results. If a human were to select these, that human should strictly have no idea of what the test data are. This is simply not feasible when we use cross-validation.

5.2 Object Recognition and Segmentation of Wounds

We solve object recognition and segmentation using a hypothesis optimization framework. The object recognition and segmentation task consists of mapping a high-dimensional image to a high-dimensional segmentation description. Without restricting ourselves to any assumptions, we can reduce the complexity of the segmentation problem, by instead making a function that implicitly expresses the object outline as the minimum of the objective function. This reduces the output complexity to a single real number. In practice, implicitly expressing the object outline through an objective function comes at the cost of relying on an optimization algorithm. For that, we can exploit the already existing literature, and we end up using a genetic algorithm.

In this conclusion, the most obvious question is, were our algorithm successful? That is a question lacking a simple answer. One consideration is that the dataset limits our results to a set of examples. Another consideration is that we have no comparative references. Finally, the wound object-segmentation task is an ill-posed problem. We have not addressed partially obstructed wound objects, yet some of our data contains partially obstructed wounds. What we can conclude is that the algorithm does not always give the correct answer. Both the optimization algorithm and the objective function can be the cause of incorrect hypothesis predictions.

Current segmentation algorithms are limited in what we can achieve with them. We require robust algorithms that we can use for object segmentation applications. The results of the proposed algorithm are not the most interesting aspects of this thesis. The most interesting aspect of this thesis is the utilization of hypothesis optimization

to solve object segmentation, given the opportunities this approach might have. We use rudimentary properties for the objective function. We hand tune objective function weights instead of automatically learning them. We use a genetic algorithm with default crossover and mutation settings. In the future works, section 4.6, we have proposed multiple improvements to our base algorithm. We can use better object properties. We can learn the objective function weights. Replacing classification with probabilistic classification may have some advantages. We also propose how to use our object segmentation algorithm as a component in a high-level multi-object segmentation algorithm. Finally, we also discuss how to combine object segmentation and tracking.

Because of the flexible nature of hypothesis optimization and our results, it places hypothesis optimization as a strong candidate for general-purpose machine-learnable object segmentation. Our algorithm acts as an initial proposal, demonstrating the advantages of hypothesis optimization.

List of Abbreviations

CI	Confidence Interval
CV	Cross-validation
GMM	Gaussian Mixture Model
k-NN	K-Nearest Neighbors
Linear-SVMs	Support Vector Machines with no Kernel
MCS	Monte Carlo Simulation
MLP	Multilayer Perceptron Network
NN	Neural Network
RBF	Radial Basis Function
RBF-SVMs	Support Vector Machines with a Radial Basis Function Kernel
RF	Random Forest
SD	Standard Deviation
SEM	Standard Error of the Mean
SLIC	Simple Linear Iterative Clustering
SVMs	Support Vector Machines
W-REC	Wound Object Recognition Algorithm
Eq.	Equation
Sel.	Selection
Perim.	Perimeter

References

[1] T.F. Chan and L.A. Vese, "Active contours without edges," *Image Processing, IEEE Transactions on*, vol. 10, no. 2, pp. 266-277, Feb 2001.

[2] LuminitaA. Vese and TonyF. Chan, "A Multiphase Level Set Framework for Image Segmentation Using the Mumford and Shah Model," *International Journal of Computer Vision*, vol. 50, no. 3, pp. 271-293, 2002. [Online]. http://dx.doi.org/10.1023/A%3A1020874308076

[3] Di-Yuan Tzeng and Roy S. Berns, "A review of principal component analysis and its applications to color technology," *Color Research & Application*, vol. 30, no. 2, pp. 84-98, 2005. [Online]. http://dx.doi.org/10.1002/col.20086

[4] Christopher M Bishop and others, "Neural networks for pattern recognition," 1995.

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, , David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, Eds. Cambridge, MA, USA: MIT Press, 1986, ch. Learning Internal Representations by Error Propagation, pp. 318-362. [Online]. http://dl.acm.org/citation.cfm?id=104279.104293

[6] Jeff Bilmes, "A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models," Tech. rep. 1998.

[7] Richard Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. New York, NY, USA: Springer-Verlag New York, Inc., 2010.

[8] Alexander Andreopoulos and John K. Tsotsos, "50 Years of object recognition: Directions forward ," *Computer Vision and Image Understanding* , vol. 117, no. 8, pp. 827-891, 2013. [Online]. http://www.sciencedirect.com/science/article/pii/S107731421300091X

[9] Alex Levinshtein, Cristian Sminchisescu, and Sven Dickinson, "Optimal Contour Closure by Superpixel Grouping," in *Proceedings of the 11th European Conference on Computer Vision: Part II*, Berlin, Heidelberg, 2010, pp. 480-493. [Online]. http://dl.acm.org/citation.cfm?id=1888028.1888066

[10] Andrea Vedaldi and Brian Fulkerson, "Vlfeat An Open and Portable Library of Computer Vision Algorithms," in *Proceedings of the International Conference on Multimedia*, New York, NY, USA, 2010, pp. 1469-1472. [Online]. httpdoi.acm.org10.11451873951.1874249

[11] Trauma.org. (2015, January) Terms of Use. [Online]. http://www.trauma.org/index.php/main/general/21

[12] Victor Joel Garza Silva. (2011, September) Hepatic injury caused by a firearm. Image. [Online]. http://www.trauma.org/index.php/main/image/1305/C13

[13] Gabriel Mejia Consuelos. (2010, March) Intraperitoneal view of a stab wound injury. Image. [Online]. http://www.trauma.org/index.php/main/image/1004/C13

[14] Carlos Zavaleta. (2008, October) Liver and Kidney 06. Image. [Online]. http://www.trauma.org/index.php/main/image/762/C13

[15] Herb Phelan and Brian Eastridge. (2007, July) Gunshot wound to carotid artery - operative - 03. Image. [Online]. http://www.trauma.org/index.php/main/image/588/C1

[16] Carlos Zavaleta. (2008, June) Gunshot wound to the kidney. Image. [Online]. http://www.trauma.org/index.php/main/image/682/C13

[17] Fernando Joglar. (2008, January) Right sided diaphragmatic hernia secondary to blunt abdominal trauma. Image. [Online]. http://www.trauma.org/index.php/main/image/637/C13

[18] Nícolas Leal, Maurício Mentz, and Jorge Mentz. (2007, April) Penetrating Neck Injury with Oesophageal Laceration 04. Image. [Online]. http://www.trauma.org/index.php/main/image/531/C13

[19] Vicente Scopel and Jorge Carlotto. (2009, August) Open pneumothorax. Image. [Online]. http://www.trauma.org/index.php/main/image/902/C11

[20] Trauma.org. (2011, January) Open pneumothorax. Image. [Online]. http://www.trauma.org/index.php/main/image/1196/C11

[21] Antonio Muria. (2010, April) A stab injury to the neck 02. Image. [Online]. http://www.trauma.org/index.php/main/image/1018/

[22] Antonio Muria. (2010, April) A stab injury to the neck 03. Image. [Online]. http://www.trauma.org/index.php/main/image/1019/

[23] Peter Kim. (2009, July) Live Demo - Vertical Mattress Suturing. Video. [Online]. https://www.youtube.com/watch?v=XPILD2O9ZUY

[24] Peter Kim. (2012, June) Deep suture for Deep wound closure. Video. [Online]. https://www.youtube.com/watch?v=TqzSHxJuLiY

[25] G. Paschos, "Perceptually uniform color spaces for color texture analysis: an empirical evaluation," *Image Processing, IEEE Transactions on*, vol. 10, no. 6, pp. 932-937, Jun 2001.

[26] A. Drimbarean and P.F. Whelan, "Experiments in colour texture analysis ," *Pattern Recognition Letters* , vol. 22, no. 10, pp. 1161-1167, 2001. [Online]. http://www.sciencedirect.com/science/article/pii/S0167865501000587

[27] Jay L Devore and Kenneth N Berk, *Modern mathematical statistics with applications*.: Cengage Learning, 2007.

[28] D.R. Martin, C.C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 5, pp. 530-549, May 2004.

[29] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour Detection and Hierarchical Image Segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 33, no. 5, pp. 898-916, May 2011.

[30] Contour Detection and Image Segmentation. [Online]. http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html

[31] A.P. Moore, S. Prince, J. Warrell, U. Mohammed, and G. Jones, "Superpixel lattices," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008, pp. 1-8.

[32] A. Levinshtein et al., "TurboPixels: Fast Superpixels Using Geometric Flows," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 12, pp. 2290-2297, 2009.

[33] R. Achanta et al., "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 11, pp. 2274-2282, Nov 2012.

[34] J. MacQueen, Some methods for classification and analysis of multivariate observations, 1967.

[35] Carl Yuheng Ren and Ian Reid, "gSLIC: a real-time implementation of SLIC superpixel segmentation," *University of Oxford, Department of Engineering, Technical Report*, 2011.

[36] Claude Elwood Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3-55, 2001.

[37] Raman Maini and Himanshu Aggarwal, "Study and comparison of various image edge detection techniques," *International Journal of Image Processing (IJIP)*, vol. 3, no. 1, pp. 1-11, 2009.

[38] R.M. Haralick, K. Shanmugam, and Its'Hak Dinstein, "Textural Features for Image Classification," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-3, no. 6, pp. 610-621, Nov 1973.

[39] T. Denoeux, "A k-nearest neighbor classification rule based on Dempster-Shafer theory," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 25, no. 5, pp. 804-813, May 1995.

[40] Leo Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001. [Online]. http://dx.doi.org/10.1023/A%3A1010933404324

[41] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273-297, 1995. [Online]. http://dx.doi.org/10.1007/BF00994018

[42] Vladimir Vapnik, "Pattern recognition using generalized portrait method," *Automation and remote control*, vol. 24, pp. 774-780, 1963.

[43] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola, "Kernel methods in machine learning," *The annals of statistics*, pp. 1171-1220, 2008.

[44] Isabelle Guyon, Amir Saffari, Gideon Dror, and Gavin Cawley, "Model Selection: Beyond the Bayesian/Frequentist Divide," *J. Mach. Learn. Res.*, vol. 11, pp. 61-87, #mar# 2010. [Online]. http://dl.acm.org/citation.cfm?id=1756006.1756009

[45] Ron Kohavi, "A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, San Francisco, CA, USA, 1995, pp. 1137-1143. [Online]. http://dl.acm.org/citation.cfm?id=1643031.1643047

[46] Gavin C. Cawley and Nicola L.C. Talbot, "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation," *J. Mach. Learn. Res.*, vol. 11, pp. 2079-2107, #aug# 2010. [Online]. http://dl.acm.org/citation.cfm?id=1756006.1859921

[47] Isabelle Guyon and André Elisseeff, "An Introduction to Variable and Feature Selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157-1182, #mar# 2003. [Online]. http://dl.acm.org/citation.cfm?id=944919.944968

[48] Ron Kohavi and George H. John, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1–2, pp. 273–324, 1997, Relevance. [Online]. http://www.sciencedirect.com/science/article/pii/S000437029700043X

[49] A.W. Whitney, "A Direct Method of Nonparametric Measurement Selection," *Computers, IEEE Transactions on*, vol. C-20, no. 9, pp. 1100-1103, Sept 1971.

[50] P. Pudil, J. Novovicová, and J. Kittler, "Floating search methods in feature selection," *Pattern Recognition Letters*, vol. 15, no. 11, pp. 1119-1125, 1994. [Online]. http://www.sciencedirect.com/science/article/pii/0167865594901279

[51] Melanie Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[52] Sylvain Arlot, Alain Celisse, and others, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40-79, 2010.

[53] Michael Kass, Andrew Witkin, and Demetri Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321-331, 1988. [Online]. http://dx.doi.org/10.1007/BF00133570

[54] Juha Reunanen, "Overfitting in Making Comparisons Between Variable Selection Methods," *J. Mach. Learn. Res.*, vol. 3, pp. 1371-1382, #mar# 2003. [Online]. http://dl.acm.org/citation.cfm?id=944919.944978